

В.Н. Гололобов



Умный дом. Виртуальная лаборатория.

Москва 2005

## Содержание

|  |            |
|--|------------|
| <b>СОДЕРЖАНИЕ</b> .....  | <b>2</b>   |
| <b>ПРЕДИСЛОВИЕ</b> .....   | <b>4</b>   |
| Работа в среде MULTISIM.....                                     | 7          |
| Усилительный каскад на транзисторе .....                         | 11         |
| <b>ЧАСТЬ 1. «КУКОЛЬНЫЙ УМНЫЙ ДОМИК»</b> .....                    | <b>18</b>  |
| <b>Знакомство с профессиональными решениями</b> .....            | <b>18</b>  |
| Система Landmark .....   | 18         |
| Система «StarGate» - X10.....                                    | 31         |
| <b>Постановка задачи</b> .....                                   | <b>38</b>  |
| Релейный модуль .....  | 40         |
| <b>Основы работы в среде MPLAB</b> .....                         | <b>78</b>  |
| <b>Релейный модуль, версия программы на языке «С»</b> .....      | <b>84</b>  |
| <b>Вернулся я из магазина – грабли «железные» и мягкие</b> ..... | <b>96</b>  |
| <b>Модуль приема инфракрасных кодов</b> .....                    | <b>108</b> |
| <b>Technical Info</b> .....                                      | <b>109</b> |
| <b>Борьба с собственной гениальностью</b> .....                  | <b>134</b> |
| <b>Модуль излучения инфракрасных кодов</b> .....                 | <b>145</b> |
| <b>Модуль считывания инфракрасных кодов</b> .....                | <b>175</b> |
| <b>Первая версия основной программы</b> .....                    | <b>182</b> |
| <b>Завтра</b> .....  | <b>196</b> |
| <b>И немного назад</b> .....                                     | <b>205</b> |
| <b>Подведем итоги</b> .....                                      | <b>215</b> |
| <b>ЧАСТЬ 2. «ДЕТСКИЙ УМНЫЙ ДОМИК»</b> .....                      | <b>217</b> |
| <b>Многословное предисловие</b> .....                            | <b>217</b> |
| <b>Почему ASPLinux?</b> .....                                    | <b>218</b> |
| <b>Работа в среде Electric</b> .....                             | <b>226</b> |
| <b>Модуль цифровых вводов</b> .....                              | <b>233</b> |

|  |            |
|--|------------|
| Работа с программой piclab .....                         | 235        |
| Результаты тайм-аута.....                                | 237        |
| Модуль с триаком.....                                    | 250        |
| Модуль с триаком и плавной регулировкой яркости .....    | 251        |
| Модуль последовательного интерфейса .....                | 264        |
| Модуль видео коммутатора.....                            | 265        |
| Модуль управляемого усилителя .....                      | 266        |
| Модуль системного ИК пульта управления.....              | 267        |
| Модуль аналогового ввода для термометра.....             | 268        |
| Замена проводного канала RS485.....                      | 268        |
| Усовершенствование базовых модулей.....                  | 269        |
| Вторая версия основной программы.....                    | 270        |
| Основная программа в Kdevelop .....                      | 270        |
| Последние замечания.....                                 | 292        |
| <b>ЧАСТЬ 3. СПРАВКИ И КОНСПЕКТЫ .....</b>                | <b>294</b> |
| gpsim.....   | 295        |
| Как работают компоненты ИК-датчика движения .....        | 346        |
| СВЧ - датчик движения для охранной сигнализации .....    | 349        |
| Таблица основных команд микроконтроллера PIC16F628A..... | 352        |
| С.Липпман. Основы C++ .....                              | 355        |

## Предисловие

Своими успехами электроника во многом обязана современным технологиям. В частности, компьютерным программам, позволяющим осуществлять полную разработку устройств за компьютером. И самим компьютерам, поддерживающим работу этих сложных программ. С другой стороны, как компьютеры, так и программы к ним – дети электроники. Получается, что электроника совершенствуется сама себя. И, хотя работа с САМ/CAD-программами больше подходит профессионалам с их ориентацией на производство, и что любителям, я почти убежден, ни в коем случае не следует замыкаться в этом виртуальном мире, однако, как средство решения своих задач, любитель не только может, но и не минует работы за компьютером. Каждого увлечет свой портал в виртуальный мир.

Предыдущая книга «Хобби-электроникс» содержит большое количество иллюстраций, сделанных с помощью программы Electronics Workbench (версия 5.12) фирмы Interactive Image Technologies Ltd. В этой книге будет больше рассказано о работе с программой Multisim (более поздняя версия Electronics Workbench).

При работе с микроконтроллерами в первой части книги используются две программы – MPLAB и PonyProg2000. В качестве базового микроконтроллера, используемого на протяжении всей книги, я выбрал контроллер PIC16F628A. В настоящий момент микросхему можно купить в магазине «Чип и Дип» по цене вполне доступной. Забегая немного вперед, хочу отметить, что я не пожалел о выборе микросхемы – при налаживании устройств мне пришлось сотни раз перепрограммировать ее; теряя бдительность, я несколько раз устанавливал ее в панельку неправильно, но она как работала, так и работает. Это одно, как мне кажется, достойно и восхищения, и уважения.

Основой сюжета станет система «Умный дом». Краткий рассказ о профессиональных системах автоматизации жилья основан на моем знакомстве, за что я очень благодарен компании «ЭРКО», с двумя системами - «Landmark» корпорации AMX (PHAST), и «StarGate» (JDS). Последняя система работает по протоколу X10. С ее компонентами можно ознакомиться на сайте [www.smarthome.com](http://www.smarthome.com) или на российском сайте фирмы «Умный дом» [www.ydom.ru](http://www.ydom.ru). Сайт корпорации AMX – [www.amx.com](http://www.amx.com) ([www.archelect.ru](http://www.archelect.ru) сайт российского представительства).

Об «Умном доме». В это название можно вложить различный смысл. Будем понимать его, как систему автоматизации и управления бытовыми устройствами в доме или квартире.

Я согласен с точкой зрения фирм, представляющих на российском рынке продукцию интеллектуальной автоматизации быта, что они представляют не столько разные концепции или системы, или разные электронно-программные комплексы, сколько стиль жизни нового века. Это так. Но в качестве представителя стиля жизни «Умный дом» пока ближе Дому Моделей, чем любителям электроники.

Автоматизацией быта сегодня никого не удивишь – автоматическая стиральная машина, программируемая СВЧ печь, робот-пылесос. Даже полюбившийся всем чайник –

## Хобби-электроникс 2. Умный дом.

электрический автоматический кипятильник – настолько удобен в быту, что иного, казалось бы и не надо. Тем выше заслуга разработчиков автоматизированных производств – они первыми увидели, что пора все домашние автоматы соединить вместе, подчинив их работу единой цели, и пора дать им возможность проявить разные свойства в разных обстоятельствах, сообразуясь с собственным разумом. Так появился «Умный дом». Попробуем, если не воспроизвести его во всей красе и величии, то хотя бы создать скромную копию, некий «Детский Умный домик».

Какой на данном этапе мне представляется структура книги? Затяжное и многословное предисловие плавно, если получится, переходит в Часть первую, в которой воспроизводится миниатюрная, но базовая, версия – «Кукольный умный домик» (Барби достаточно богатая кукла). Именно в этой части будут описаны все проблемы, которые появились у меня при работе с программами MPLAB и PonyProg, при разработке первых конструкций. Здесь же, по ходу работы, должна появиться некоторая определенность в планировании содержания следующей, части второй – «Детский умный домик».

Часть вторая в меньшей мере будет описывать работу с программами для программирования микроконтроллеров, в меньшей мере будет посвящена «разбору полетов». Больше внимания я постараюсь уделить обоснованию появления новых модулей в системе, таких, как модуль цифровых вводов, модуль с триаком, модуль аудио коммутатора, модуль системного ИК пульта управления и т.д. В этой же части я планирую изменить среду программирования «Детского умного домика». Это будет либо KDevelop – среда программирования, работающая с операционной системой ASPLinux, либо программа, написанная в среде KDevelop. Существующие дистрибутивы операционной системы Linux имеют почти все необходимое для работы – текстовые процессоры (редакторы), графические редакторы, среду разработки на огромном количестве языков программирования – я сам знакомился и с «С++», и с php и Haskell, используя KDevelop. И все это великолепие можно честно, в плане сегодняшних спекуляций на тему пиратства, приобрести за 300 рублей, что доступно всем владельцам компьютеров. Есть версия PonyProg2000 для Linux. Есть версия MPLAB, но пока я с ней не вполне ознакомился.

Часть третья предполагается справочной. Я постараюсь включить свои конспекты, не хочу называть их переводами, книги С. Липпмана по языку «С++», описания работы с MPLAB и языку «С» для микроконтроллеров. И, полагаю, справочные данные по применяемой элементной базе. Цель этой книги, как и предыдущей, не в описании готовых любительских разработок, хотя все устройства я проверю в макетном варианте, а показать один из путей подхода к разработке. В этом мне должны помочь и мои собственные ошибки (базовая часть), и те проблемы, которые всегда возникают при работе. Книга мне видится не посланием гуру народу, а, скорее, многословной инструкцией к конструктору «Собери сам».

Прежде, чем окончательно свести счеты с реальностью, и погрузиться в виртуальный мир, позволю себе привести фрагменты статьи, случайно попавшей в мой архив, о взгляде на реальность одного из светил в области современной физики Дэвида Бома. К сожалению, в моем архиве статья имеет пометку: «Автор неизвестен». Прошу извинить эту оплошность, которая может поставить под сомнение и принадлежность высказываемой автором статьи точки зрения именно Д. Бому. Но, что я точно знаю, так это то, что результаты опытов А. Аспека действительно вызвали живейший интерес у самых крупных физиков нашего времени.

## «Существует ли объективная реальность, или Вселенная - голограмма?»

В 1982 году произошло замечательное событие. Исследовательская группа под руководством Алана Аспека при университете в Париже представила эксперимент, который может оказаться одним из самых значительных в 20 веке.

Аспек и его группа обнаружили, что в определенных условиях элементарные частицы, например, электроны, способны мгновенно сообщаться друг с другом независимо от расстояния между ними.

Каким-то образом каждая частица всегда знает, что делает другая. Проблема этого открытия в том, что оно нарушает постулат Эйнштейна о предельной скорости распространения взаимодействия, равной скорости света. Поскольку путешествие быстрее скорости света равносильно преодолению временного барьера, эта пугающая перспектива заставила некоторых физиков пытаться объяснить опыты Аспека сложными обходными путями. Но других это вдохновило предложить более радикальные объяснения.

Физик лондонского университета Дэвид Бом считает, что согласно открытию Аспека, реальная действительность не существует, и что, несмотря на очевидную плотность, Вселенная в своей основе - фикция, гигантская, роскошно детализированная голограмма.

Принцип голограммы “все в каждой части” позволяет нам принципиально по-новому подойти к вопросу организованности и упорядоченности. Почти на всем своем протяжении западная наука развивалась с идеей о том, что лучший способ понять явление, будь то лягушка или атом, - это рассеять его и изучить составные части. Голограмма показала нам, что некоторые вещи во вселенной не могут это нам позволить. Если мы будем рассекать что-либо, устроенное голографически, мы не получим частей, из которых оно состоит, а получим то же самое, но меньше размером.

Эти идеи вдохновили Д. Бому на иную интерпретацию работ Аспека. Бом уверен, что элементарные частицы взаимодействуют на любом расстоянии не потому, что они обмениваются таинственными сигналами между собой, а потому, что их разделенность есть иллюзия. Частицы - не отдельные “части”, но грани более глубокого единства, которое в конечном итоге голографично и невидимо подобно объекту, снятому на голограмме. И поскольку все в физической реальности содержится в этом “фантоме”, Вселенная сама по себе есть проекция, голограмма.

Вдобавок к ее “фантомности”, такая Вселенная может обладать и другими удивительными свойствами. Если разделение частиц - это иллюзия, значит, на более глубоком уровне все предметы в мире бесконечно взаимосвязаны. Электроны в атомах углерода в нашем мозгу связаны с электронами каждого лосося, который плывет, каждого сердца, которое стучит, и каждой звезды, которая сияет в небе.

Все взаимопроникает со всем, и хотя человеческой природе свойственно все разделять, расчленять, раскладывать по полочкам, все явления природы, все разделения искусственны и природа в конечном итоге есть безразрывная паутина».

## ***Работа в среде MULTISIM***

Хотя основная часть книги касается программирования контроллера PIC16F628A, как основы построения модулей системы, некоторые аспекты удобнее рассматривать, используя программу MULTISIM.

После запуска программы на экране появляется стандартное для среды Windows рабочее поле и меню:

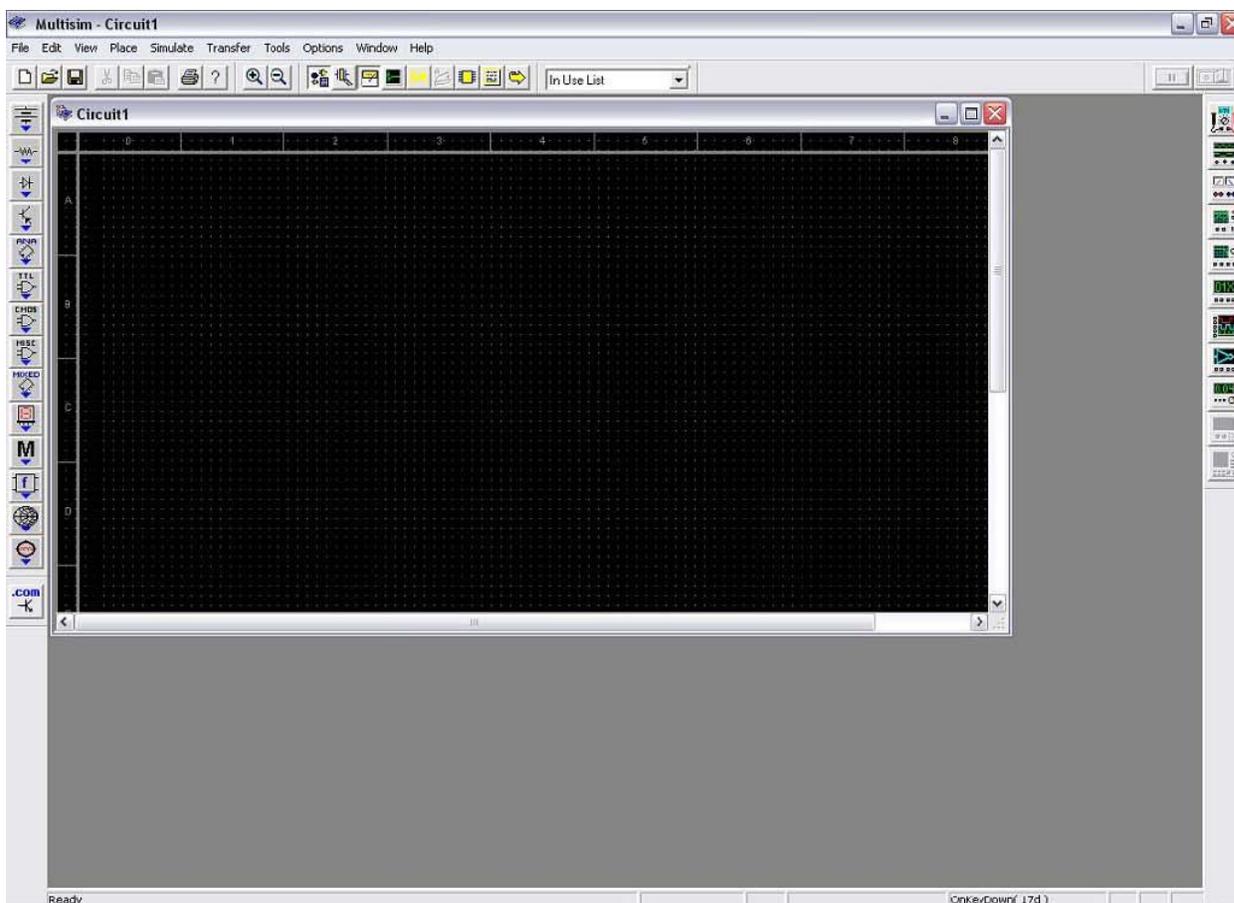


Рис.1

Раскрывая меню компонентов, можно выбрать необходимые в данный момент и мышкой перенести их на рабочее поле. В разных меню разные электронные компоненты, индикаторы и приборы. Для перетаскивания следует нажать соответствующую кнопку меню курсором, затем, отпустив кнопку мыши, перенести компонент в поле чертежа.

## Хобби-электроникс 2. Умный дом.

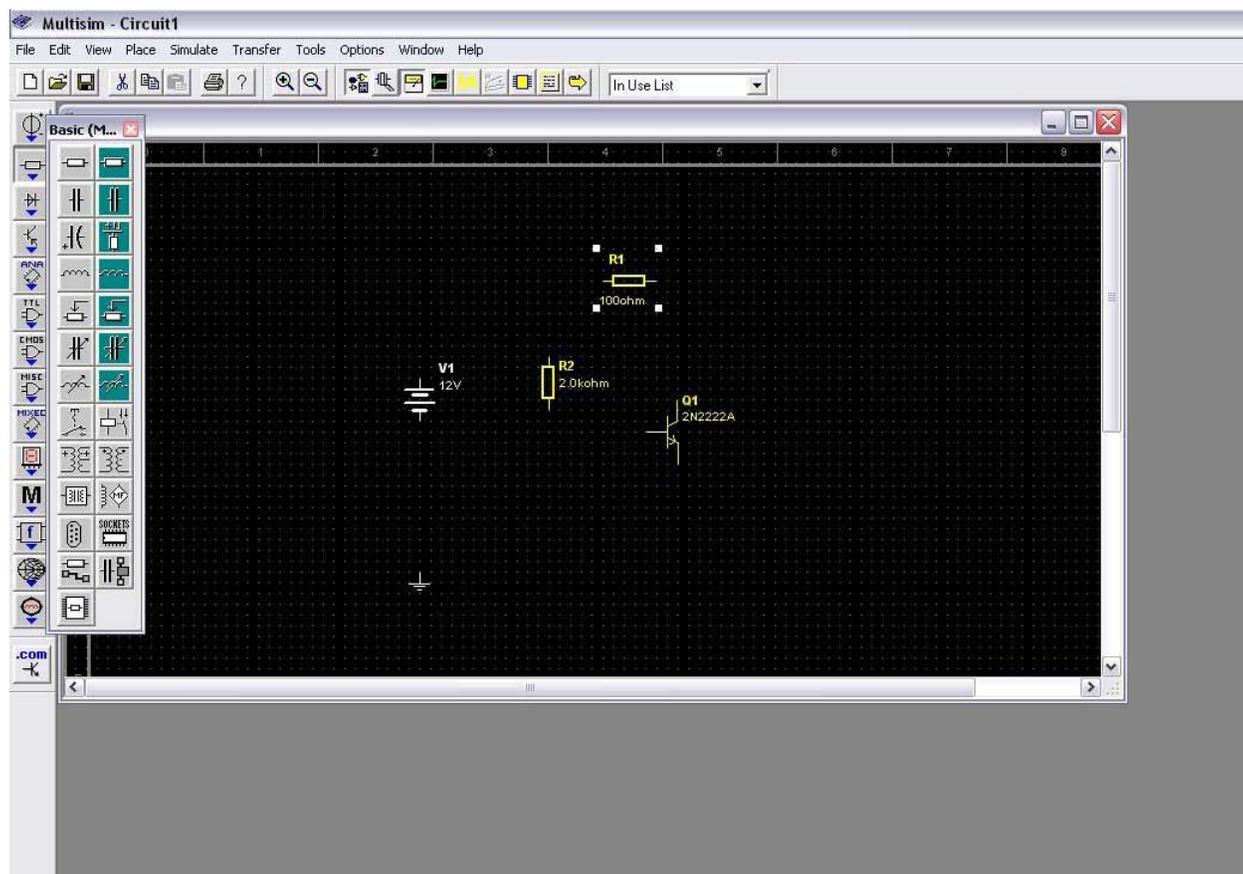


Рис.2

Выбрав все компоненты, можно разместить их наиболее наглядным образом, используя при необходимости подмену поворота и отражения, которые появляются при нажатии правой кнопки мышки на выделенном компоненте. При размещении следует иметь в виду, что позже к схеме добавятся приборы, с помощью которых вам можно будет проверить работу схемы, настроить ее, привести к виду, готовому для воплощения, т.е. сборки готовой платы.

Кроме того, можно сразу позаботиться о наглядности схемы, придав ей тот вид, который вам легче понять. Этому моменту, как мне кажется, зачастую совсем не уделяется внимания. Дело в том, что профессиональные чертежи оформляются в соответствии с принятыми нормами и рекомендациями ГОСТов, которые базируются на старых технологиях. В частности стремятся к наиболее полному заполнению поля чертежа (экономия бумаги), стремятся максимально упростить чертеж, чтобы облегчить жизнь чертежникам, которым приходится многократно воспроизводить один и тот же чертеж. В результате чертеж, который легко воспроизводится, трудно читается. Ситуация схожа с книгоизданием, где применение аббревиатуры настолько общепринято, что сделав перерыв в чтении, ты бываешь вынужден вернуться к началу – иначе не вспомнить значение многочисленных АБВГД, ВБЛГ и т.п.

За компьютером вам нет смысла придерживаться этих стандартов, полезнее сделать чертеж легко понимаемым, чтобы, возвращаясь к нему, вы могли сразу понять схему со всеми ее особенностями.

## Хобби-электроникс 2. Умный дом.

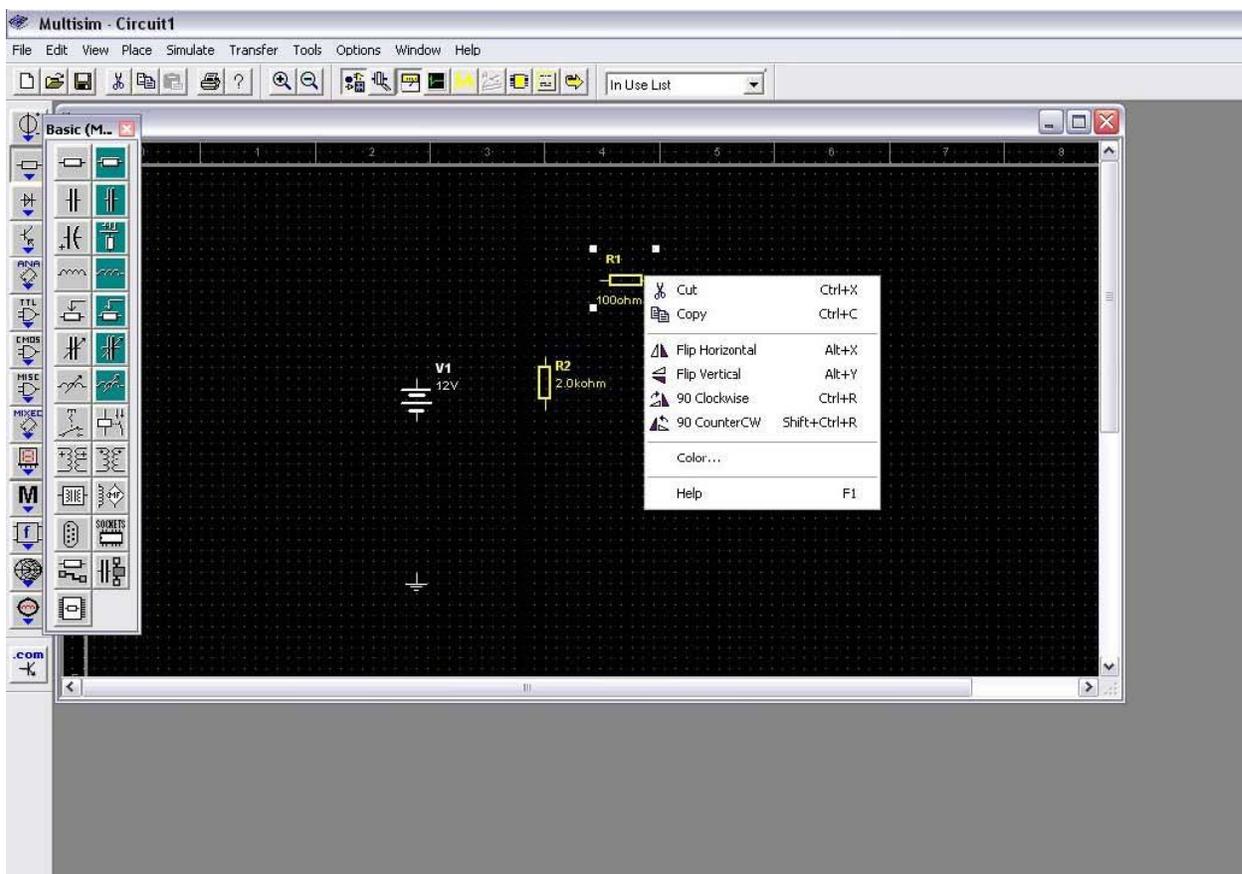


Рис. 3

После размещения всех необходимых компонентов следует соединить их, используя левую кнопку мышки. Когда курсор находится на краю любого из компонентов, щелкнув левой кнопкой мышки, нужно провести соединение к следующему элементу схемы до появления пересечения курсора с линией в точке соединения, где следует повторно щелкнуть левой кнопкой мышки.

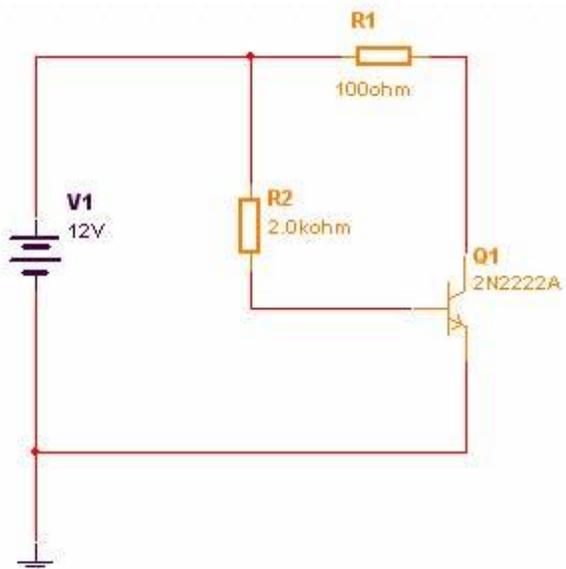


Рис. 4

На схеме можно разместить приборы, выбрав их из меню приборов. Вольтметр и амперметр находятся в меню индикаторов, представленном иконкой семисегментного цифрового индикатора.

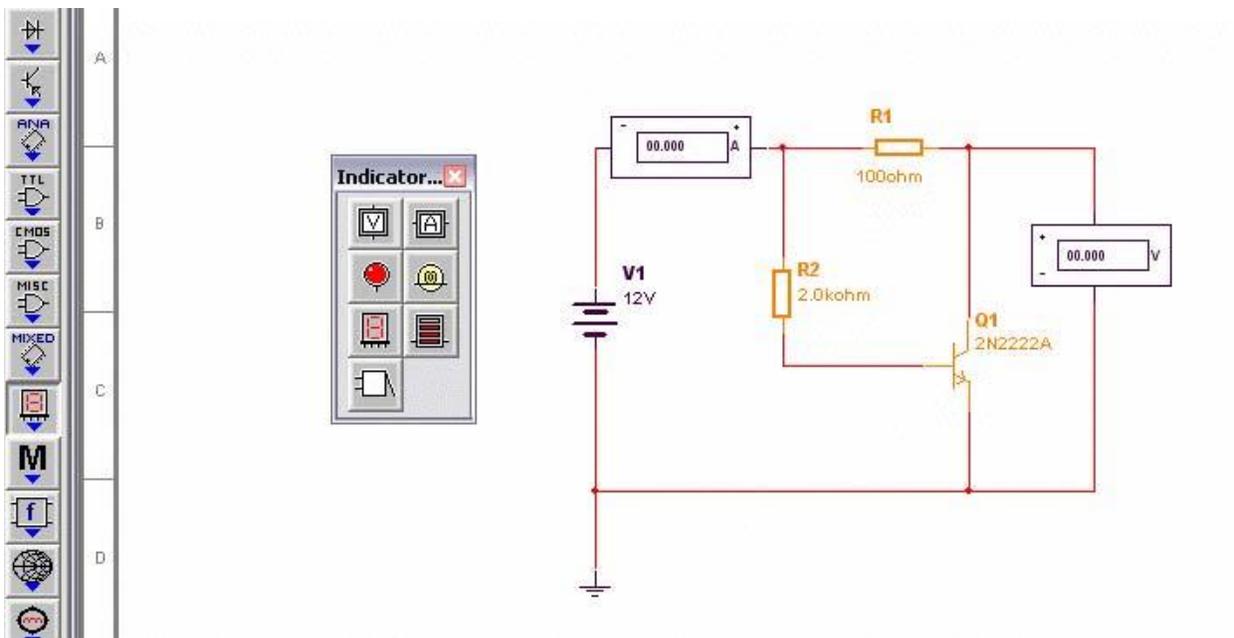


Рис. 5

После включения схемы (в правом верхнем углу есть клавиша включения) приборы покажут измеряемые величины.

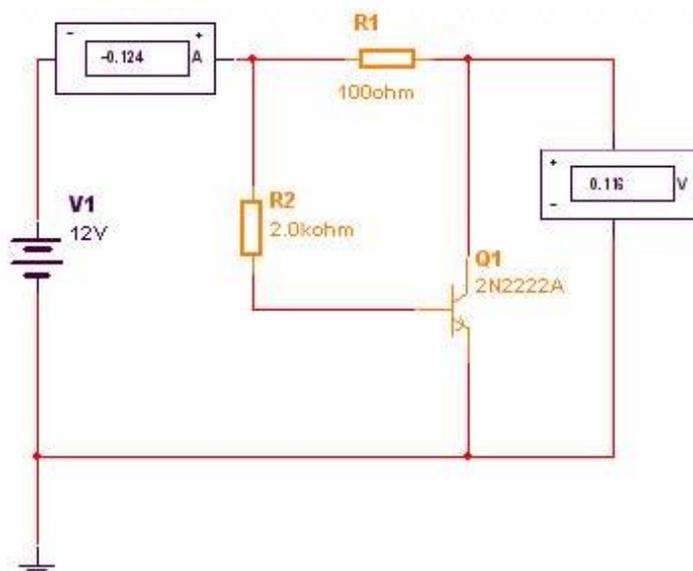


Рис. 6

Если появляется необходимость изменить значения параметров компонентов, их следует выделить, дважды щелкнув левой кнопкой мышки. В появившемся диалоговом окне «component properties» (свойства компонент) следует выбрать клавишу Replace (заменить) в нижнем левом углу, а затем выбрать новое значение компонента.

Следует отметить, что для работы схемы необходимо обязательно использовать заземление, даже если схема не требует его. Проект, который мы осуществим для знакомства с программой, может показаться скучным – усилительный каскад на транзисторе.

## Усилительный каскад на транзисторе

Если открыть любой учебник по схемотехнике усилителей, можно там же найти и методики расчета транзисторных усилителей. В учебнике обязательно будет приведена классификация усилительных каскадов по способу включения транзистора, как активного элемента, и свойства каскадов, при разных способах включения транзисторов. Однако чаще всего применяют включение транзистора с общим эмиттером. Это означает, что эмиттер служит общим выводом для входной и выходной цепи. Тем, кто интересуется теоретическими аспектами вопроса, кому хотелось бы методично во всем разобраться, я могу назвать несколько книг:

- 1• Герасимов, Мигулин, Яковлев. Расчет полупроводниковых усилителей и генераторов. Киев, 1961г.
- 2• Воронков, Овечкин. Основы проектирования усилительных и импульсных схем на транзисторах. Москва, 1973г.
- 3• Шкритек. Справочное руководство по звуковой схемотехнике. Москва, 1991г.

Мы же воспользуемся теми преимуществами, которые получаем от использования

компьютера.

Несколько слов о том, почему мне хочется начать с усилительного каскада на транзисторе. Мне кажется, что знание работы транзистора закладывает основу понимания и современной аналоговой техники, и современной цифровой, включая микропроцессорную, техники. Конечно, сегодня при разработке электронных устройств, или ремонте, использование микросхем подразумевает знание не того, как устроена микросхема, а того, какими свойствами она обладает (ее параметров), для чего она предназначена, и какие сигналы используются. В этом смысле сегодняшняя работа с электроникой ближе к программированию на объектно-ориентированных языках, в отличие от программирования с использованием процедурных языков. Некоторые преподаватели информатики считают, что знания процедурных языков даже мешает быстрому освоению современного программирования. Возможно, так. Но быстрое освоение в узкой области знаний рано или поздно может завести в тупик. Я так думаю, но не готов спорить.

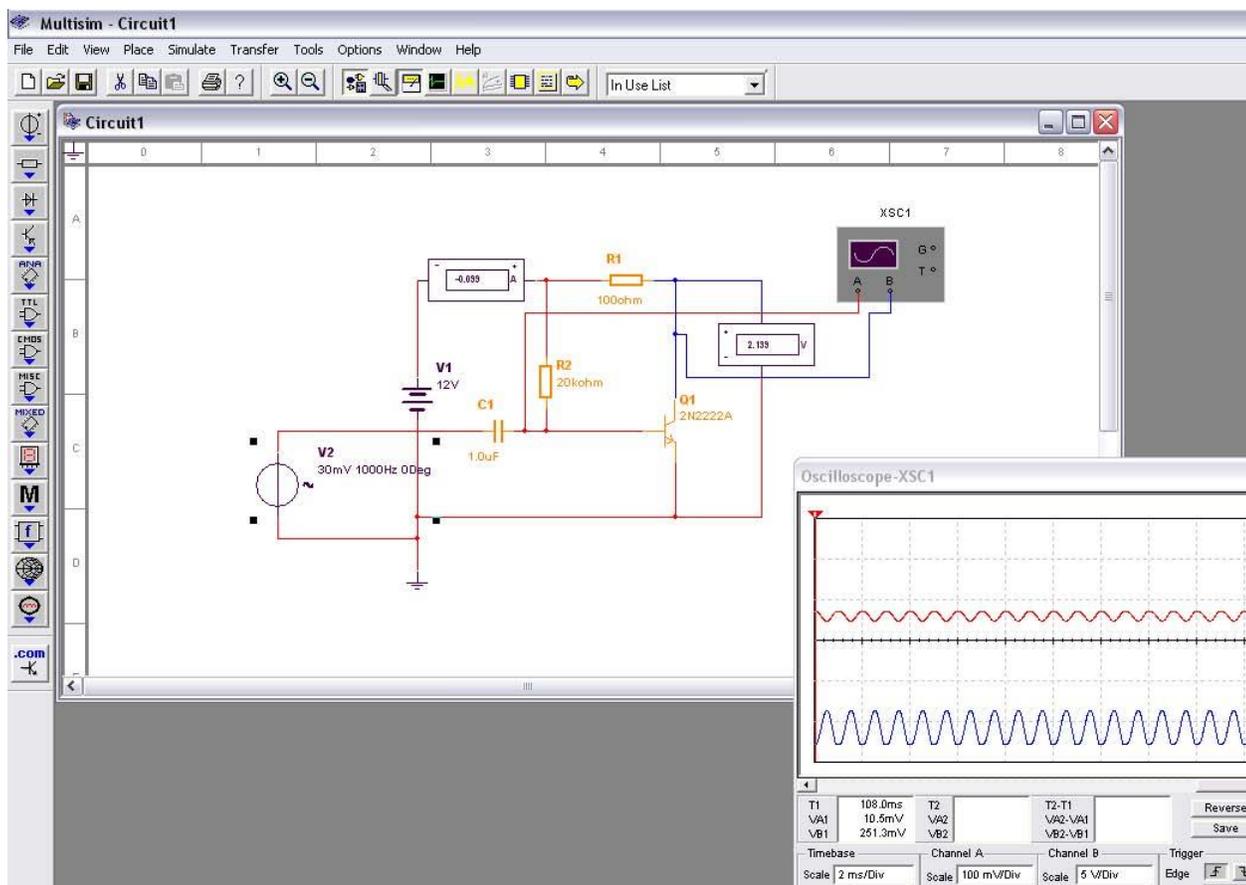


Рис. 7

Вот так будет выглядеть усилительный каскад с общим эмиттером. В качестве источника сигнала использован генератор синусоидального напряжения, а сигналы на входе и выходе усилителя мы наблюдаем с помощью осциллографа. Немного изменим нашу схему, как показано на следующем рисунке:

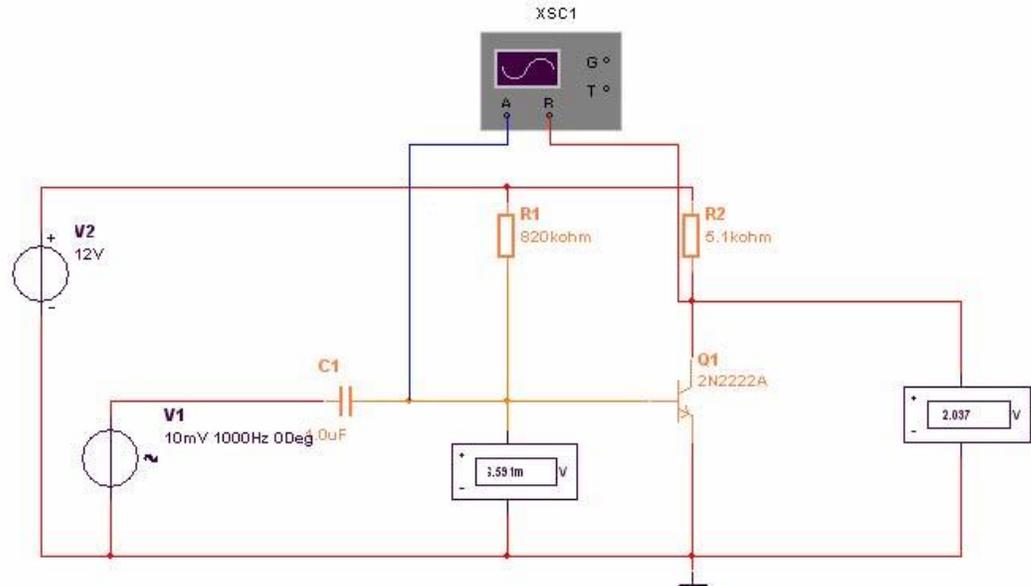


Рис.8

В схеме добавились два вольтметра переменного тока для измерения величины сигнала. Как видно из рисунка (по показаниям вольтметров), сигнал на входе усилителя около 6 мВ, а на выходе 2 В. Отношение выходного сигнала к входному - это коэффициент усиления каскада по напряжению. Разделив 2 В на 6 мВ, мы получим, что  $K_n=333$ . Усиление по напряжению интересует нас довольно часто.

Здесь уместно пояснить один момент, касающийся единиц измерения. Усиление мы измеряем в относительных единицах, как отношение выходного напряжения к входному:

$$K_u = U_{\text{вых}} / U_{\text{вх}}$$

Есть и другие единицы измерения - децибелы. Формула перехода выглядит так:

$$K_u, \text{ дБ} = 20 \lg K_u \text{ или } K_u, \text{ дБ} = 20 \lg (U_{\text{вых}} / U_{\text{вх}}).$$

Зачем нужны другие единицы? Общее усиление двух каскадного усилителя равно произведению коэффициентов усиления. А мы знаем из свойств логарифмов, что логарифм произведения равен сумме логарифмов. Таким образом, вместо сложной операции умножения можно применить более простую операцию – сложение. Порой это удобно.

Вернемся к схеме. Для чего служат ее элементы? Сопротивление R1 определяет базовый ток транзистора, ток коллектора которого равен некоторому параметру транзистора, называемому статическим коэффициентом усиления транзистора, умноженному на его базовый ток. Расчет каскада «на вскидку» можно провести так – мы определяем, что в отсутствие сигнала напряжение на коллекторе должно быть равно половине напряжения питания (для того, чтобы симметричный сигнал мог усиливаться максимально, но без искажений). Зная величину сопротивления  $R_2 = 5 \text{ кОм}$  (сопротивления нагрузки транзистора) и величину напряжения ( $12\text{В}/2 = 6\text{В}$ ), можно легко определить необходимый ток коллектора.

Ток коллектора равен половине напряжения питания, деленному на сопротивление нагрузки ( $6\text{В}/5\text{ кОм} = 1,2\text{ мА}$ ). Ток коллектора связан с током базы величиной статического коэффициента усиления транзистора по току в схеме с общим эмиттером ( $\beta_{ст} = 200$  для данного транзистора 2N2222):

$$I_k = \beta_{ст} \cdot I_b, \text{ где}$$

$I_k$  – ток коллектора,  $I_b$  – ток базы.

Таким образом, ток базы должен быть в 200 раз меньше, т.е. равен 6 мкА. Через резистор R1 должен протекать ток в 6 мкА, а напряжение на нем должно быть равно напряжению питания минус напряжение база-эмиттер транзистора. Для расчетов «на вскидку» последним можно пренебречь, приняв напряжение на резисторе равным напряжению питания. Тогда величина резистора R1 будет равна отношению напряжения питания к току базы транзистора –  $12\text{В}/6\text{мкА}$  (т.е. 2 Мом). Что мы и сделали. Остался вопрос, почему величина резистора R2 выбрана равной 5 кОм? Эта величина обычно определяется параметрами каскада, следующего за данным. Например, можно считать, что наш усилитель будет подключен к усилителю мощности. Линейный вход усилителя мощности имеет, примерно, такие параметры – входное напряжение 250 мВ, входное сопротивление 47 кОм. Если выходное сопротивление нашего каскада будет на порядок ниже, т.е. 4,7 кОм, то входное сопротивление усилителя мощности не будет мешать работе нашего каскада. Выходное же сопротивление нашего каскада не будет больше сопротивления нагрузки транзистора, равного 5 кОм.

Усиление каскада в 300 единиц по напряжению, казалось бы достаточно для многих целей. Например, сигнал обычного динамического микрофона, который будет включен вместо генератора, имеет порядок 1 мВ, а выходной сигнал в 300 мВ соответствует уровню стандартного линейного входа усилителя мощности. Чего бы нам еще желать?

Но если мы посмотрим на реальные схемы, то увидим, что они имеют несколько иной вид:

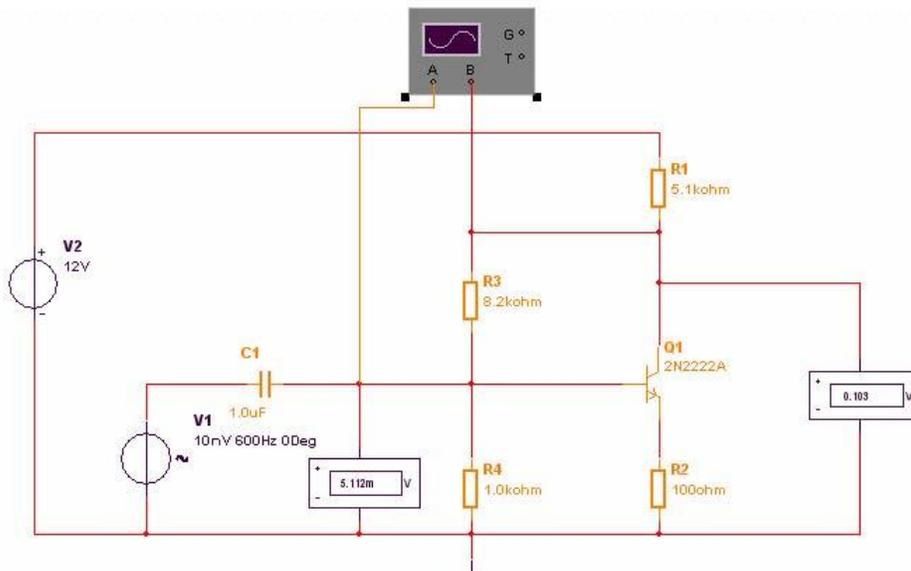


Рис.9

Что получилось с усилением?  $K_n = 100\text{мВ}/5\text{мВ}$  ( $K_n = 20$ ). Усиление снизилось более, чем в 10 раз. Зачем же нам нужны лишние детали – резисторы R3 и R4, и потеря усиления?

Дело в том, в первую очередь, что полупроводники очень чувствительны к изменению температуры. Это их свойство часто используют, изготавливая из полупроводников датчики температуры. У транзистора с изменением температуры окружающей среды изменится ток коллектора, что сместит напряжение на коллекторе и может привести к искажению сигнала. Аналогично действует изменение напряжения питания. Попробуйте повлиять на ток коллектора в схеме рис.8, меняя величину сопротивления резистора R1, и наблюдайте за формой сигнала на экране осциллографа. Этому же можно добиться, меняя температуру в диалоговом окне условий моделирования (Меню Simulate, раздел Analyses, подраздел Temperature Sweep). Для этого нужно проставить новое значение температуры в строке Values открывающегося диалогового окна, и нажать клавишу Accept.

Дополнительные элементы схемы рис.9 призваны стабилизировать параметры усилительного каскада. Если обратиться к рис.7, то можно заметить, что входной и выходной сигналы находятся в противофазе. В нашей схеме (рис.9) резистор R3 в сочетании с входным сопротивлением каскада образует параллельную обратную связь, а резистор R2 – резистор последовательной обратной связи. Обе обратные связи – отрицательные по постоянному току - стабилизируют все параметры усилительного каскада.

Посмотрим, что произойдет, если ток коллектора возрастет. Напряжение на коллекторе транзистора падает, уменьшается ток через резистор R3, что приводит к уменьшению базового тока транзистора, а, следовательно, и тока коллектора, равного величине базового тока, умноженного на статический коэффициент усиления транзистора.

Похожие изменения происходят и из-за наличия резистора R2. Увеличение тока коллектора приводит к увеличению падения напряжения на этом резисторе. Ток базы зависит от напряжения между базой и эмиттером транзистора, которое, при увеличении падения напряжения на резисторе R2, уменьшается (оно равно разности напряжения между базой и общим проводом и падением напряжения на резисторе R2). С уменьшением напряжения между базой и эмиттером уменьшается ток базы, а, следовательно, ток коллектора.

Есть еще один из достаточно важных параметров каскада, на который отрицательная обратная связь влияет благотворно.

Верхняя граничная частота усилителя на рис.8 составляет примерно 2.3МГц (частота, на которой коэффициент усиления уменьшается на 3 децибела). Верхняя граничная частота усилителя на рис.9 – более 16МГц.

Для определения этой частоты можно построить частотную характеристику усилителя – зависимость усиления от частоты. В средней части частотной характеристики график идет горизонтально, а после верхней граничной частоты происходит спад усиления в 20 дБ на декаду (т.е. изменение частоты в 10 раз приводит к спаду усиления на 20 дБ или в 10 раз). Частотную характеристику можно строить по точкам. Изменив частоту, определить выходное напряжение, вновь изменить частоту и измерить выходное напряжение, и т.д. Программа Multisim предлагает воспользоваться для этой цели прибором – плоттером Боде. При работе с прибором не следует забывать, что он работает тогда, когда на входе схемы включен

генератор переменного напряжения (его параметры не существенны). Кроме того, включив схему, можно не обнаружить частотной характеристики на экране плоттера Боде. Скорее всего, причина в том, что она выше выделенного окна. Изменяя параметр F вертикального отклонения, можно найти пропажу.

Кому-то подобные простые объяснения могут показаться излишне простыми. Но почему-то в сложных ситуациях, именно об этих простых вещах вспоминаешь в последнюю очередь. Впрочем, это дело вкуса.

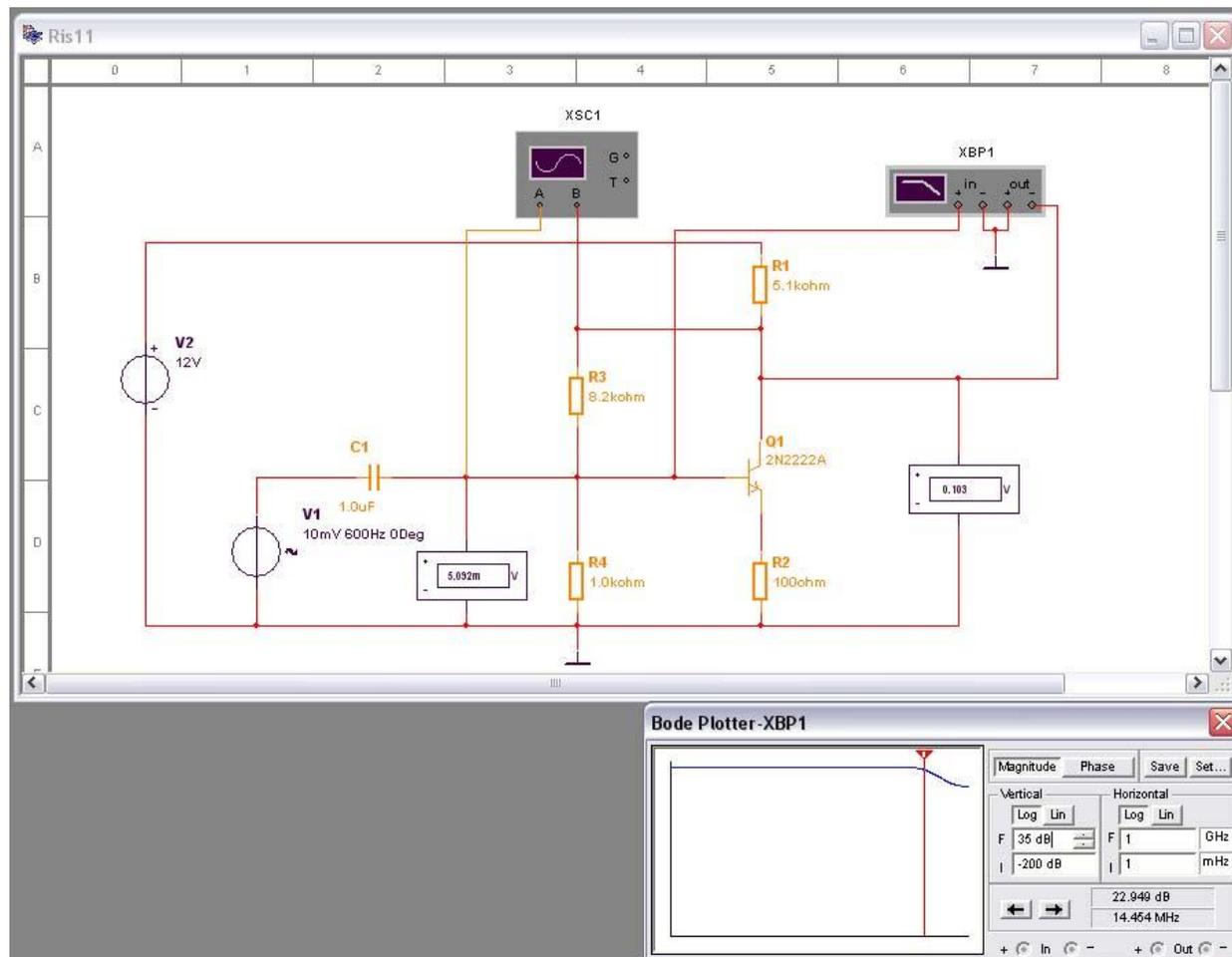


Рис.10

Так выглядит частотная характеристика усилительного каскада.

Для получения фазовой характеристики на плоттере Боде следует включить клавишу Phase в верхней части прибора.

Знание фазовой характеристики усилительного каскада позволяет определить устойчивость усилителя после введения отрицательной обратной связи. На граничной частоте, фаза сигнала изменяется на 45 градусов и продолжает изменяться со скоростью 45 градусов на декаду (относительно фазы на входе). При этом отрицательная обратная связь на некоторой частоте превращается в положительную. Если при этом усиление больше единицы, то усилитель превращается в генератор. Часть сигнала с выхода усилителя

приходит на вход в фазе с входным сигналом, складывается с ним, увеличивая выходной сигнал... часть которого складывается с входным...

Пожалуй, этого краткого введения будет достаточно. В книге я не так часто буду обращаться к программе Multisim, и не только потому, что использую demo-версию, работающую без сохранения файла – многие из программ, с которыми предстоит познакомиться, имеют то же свойство, а скорее по причине акцента на работе с микроконтроллером. Но, когда это понадобится, мы вернемся к Multisim или ее аналогу.

## Часть 1. «Кукольный умный домик»

### Знакомство с профессиональными решениями

Сколько специалистов, столько мнений. Я часто повторяю это себе, поскольку решений может существовать множество, даже после применения всех критериев отбора. Дальнейший выбор происходит на основе личных предпочтений. Системы автоматизации быта отнюдь не исключение. Можно до бесконечности спорить, делать ли систему централизованной или децентрализованной, или какую сеть использовать - компьютерную, силовую или специализированную. За основу выбора можно взять надежность, или стоимость, или доступность готовых устройств, за счет которых в будущем система может расширяться, совершенствоваться, развиваться.

Полный обзор существующих систем может занять не одну книгу, поэтому я лишь вкратце расскажу о двух системах, с которыми знаком, и которые находятся, в какой-то мере, на ценовых полюсах систем бытового назначения. Но вначале немного о том, из чего состоит любая система «Умный дом», хотя можно по-разному подойти и к этому вопросу. Я разделю систему на центральное управляющее устройство, исполняющие модули, средства управления, системную сеть и среду программирования (она же средство отладки). Базовые исполняющие модули – релейные модули, модули цифровых входов, диммеры, трансляторы ИК команд, коммутаторы сигналов. Средства управления – универсальные пульты ИК (инфракрасный спектр) команд, специализированные клавишные пульты и сенсорные панели. Системная сеть – системный интерфейс и среда передачи системных команд. Это не полный перечень, а, скорее, произвольная выборка, которая понадобится в дальнейшем, чтобы определиться с реализацией.

### Система Landmark

В настоящее время система поддерживается корпорацией AMX (PHAST, Panja).

Почти все модули выполнены в виде печатных плат, предназначенных для установки в конструктив:



Рис.11

С перечнем модулей можно ознакомиться на сайте производителя, я приведу названия и назначение некоторых модулей из списка, который есть в моем архиве.

- PLC-MCU - модуль центрального процессора системы.
- PLB-AMP8 – 8и-канальный аудио усилитель (обслуживание акустических зон).
- PLB-AS16 - 16и-канальный аудио коммутатор (матрица 16x16).
- PLC-IN7 - модуль с 7ю цифровыми входами (в основном для подключения датчиков).
- PLC-IRIN - модуль с 3я входами для подключения приемников ИК управляющих кодов.
- PLC-IROUT - модуль для подключения 4х ИК излучателей для трансляции ИК кодов.
- PLC-RL8 - релейный модуль с 8ю исполняющими реле.
- PLL-MLC - управляемый диммер - выключатель.

В конструктиве (CardFrame) расположен центральный процессор. Конструктив имеет встроенный хаб – сетевой концентратор, к которому подключаются сетевые системные устройства (например, все выключатели света). Аналогично компьютерным хабам, он имеет вход и выход для объединения всех сетевых концентраторов.

В составе системы средства управления представлены двумя базовыми решениями. Это настенный клавишный пульт с дисплеем для отображения информации и переносной пульт для ИК (инфракрасного) управления. Кроме них система поддерживает сенсорные панели AMX. Позже появилось множество универсальных ИК пультов управления, способных запомнить большое количество кодов от пультов управления бытовыми устройствами. Некоторые, например фирмы Philips, могут полностью программироваться на компьютере, а затем загружаться через COM или USB-порт.

Несомненно, и сенсорные панели AMX, и пульты Philips очень красивы.

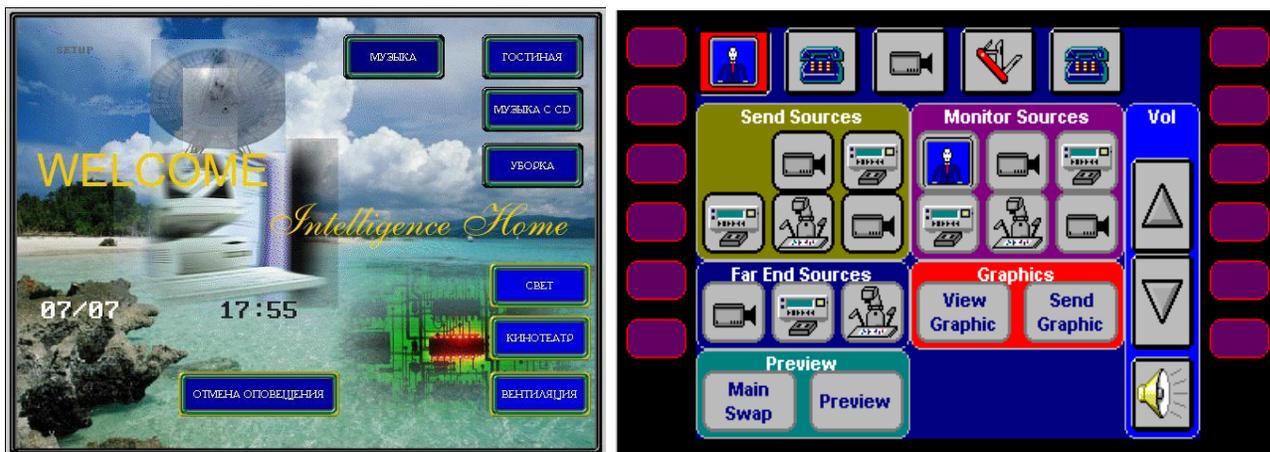


Рис.12 Панели AMX.

Вид и работа сенсорных панелей определяются с помощью специальной программы (или программ), а затем загружаются в панель. Для загрузки панелей, уже установленных в систему, используется существующая системная сеть.



Рис.13 Пульты Pronto фирмы Philips.

Универсальные пульты для ИК управления могут применяться, практически, с любой системой. Некоторые модели могут поддерживать как инфракрасное управление, так и управление по радио каналу. Так пульты ProntoPro позволяют одни страницы использовать в режиме IR (инфракрасное управление), другие в режиме RF (управление по радио каналу).

Я не готов утверждать, что любительская разработка подобного устройства столкнется с непреодолимыми трудностями, но могу сказать, что, если серийное устройство оценивается в розничной продаже в тысячу (или несколько тысяч) долларов, то в любительской разработке оно может стоить не дешево. По этой причине разработку средств управления подобного типа лучше пока оставить за профессионалами.

Система Landmark имеет удобную и эффективную среду программирования, позволяющую работать, как с проектом автоматизации обычной квартиры, так и с автоматизацией многоэтажной постройки. Каждый этаж в последнем случае может быть разбит на помещения, в которых будут располагаться устройства и датчики. Можно использовать строительные поэтажные планы, что дает полное представление о помещениях.

Программа, по сути, набирается из необходимых событий, вызываемых пультами управления, датчиками, или временем суток, и ответных действий системы. Программа оснащена достаточным количеством условий.

Приведу пример работы с системой Landmark. После запуска и ввода пароля новый проект выглядит следующим образом:

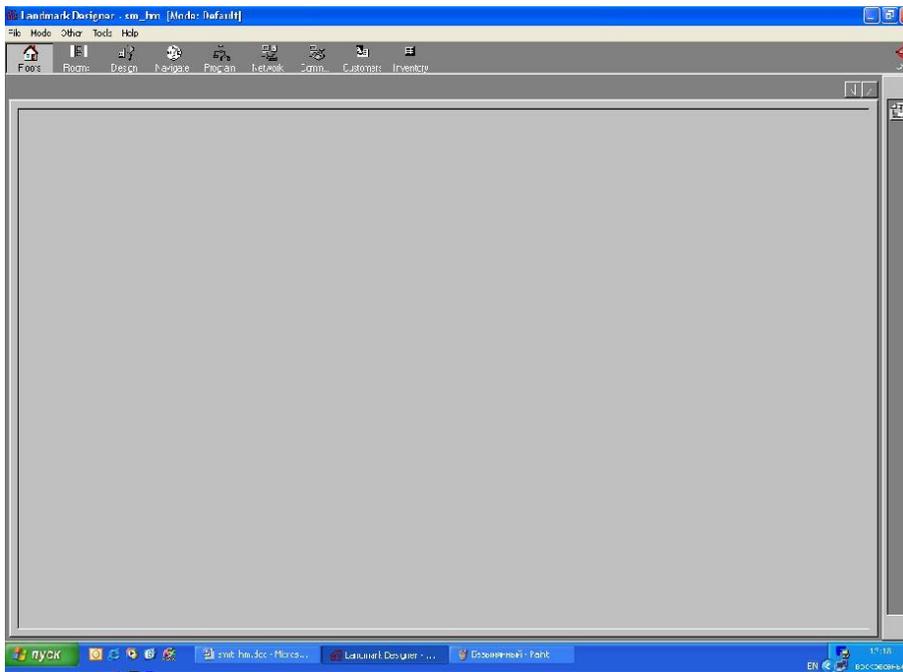


Рис.14

Первые клавиши инструментальной панели, ниже основного меню, вполне ясно подскажут вам, что делать дальше – определиться с вашим домом – этажами и комнатами. Рисунок этажа можно взять из проектной архитектурной документации, можно изобразить в редакторе Paint, встроенном в Windows, в AutoCAD или CorelDraw. Landmark поддерживает множество форматов изображений. Нажав клавишу в правом углу инструментальной панели, выбрав имя для этажа (поддерживается русский для задания названий, но лучше использовать английский, чтобы не иметь больше трудностей, чем следует, при печати), и, указав рисунок этажа, получаем:

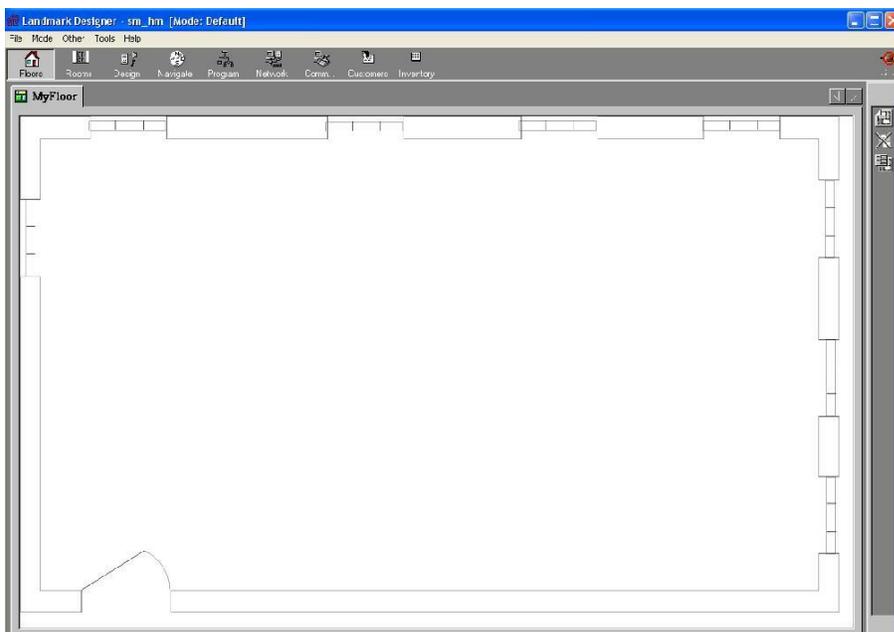


Рис.15

На этом рисунке мы расположим помещения проекта, перейдя на вторую закладку основной панели инструментов «Rooms». Для добавления помещения используем клавишу «Add» меню страницы. Впишем название комнаты. Можно изменить цвет, а можно оставить цвет, предлагаемый программой. Затем нарисуем помещения, используя левую клавишу мышки для фиксации положения линии. Если впоследствии понадобится поправить положение зафиксированной точки, то можно установить курсор на эту точку, нажать левую кнопку мышки, и, удерживая ее, переместить опорную точку в нужное место.

Продвигаясь дальше по основной инструментальной панели, можно понять, что следует сделать дальше, поскольку следующая клавиша носить название «Design».

А пока, вот вид нашего проекта в плане помещений:

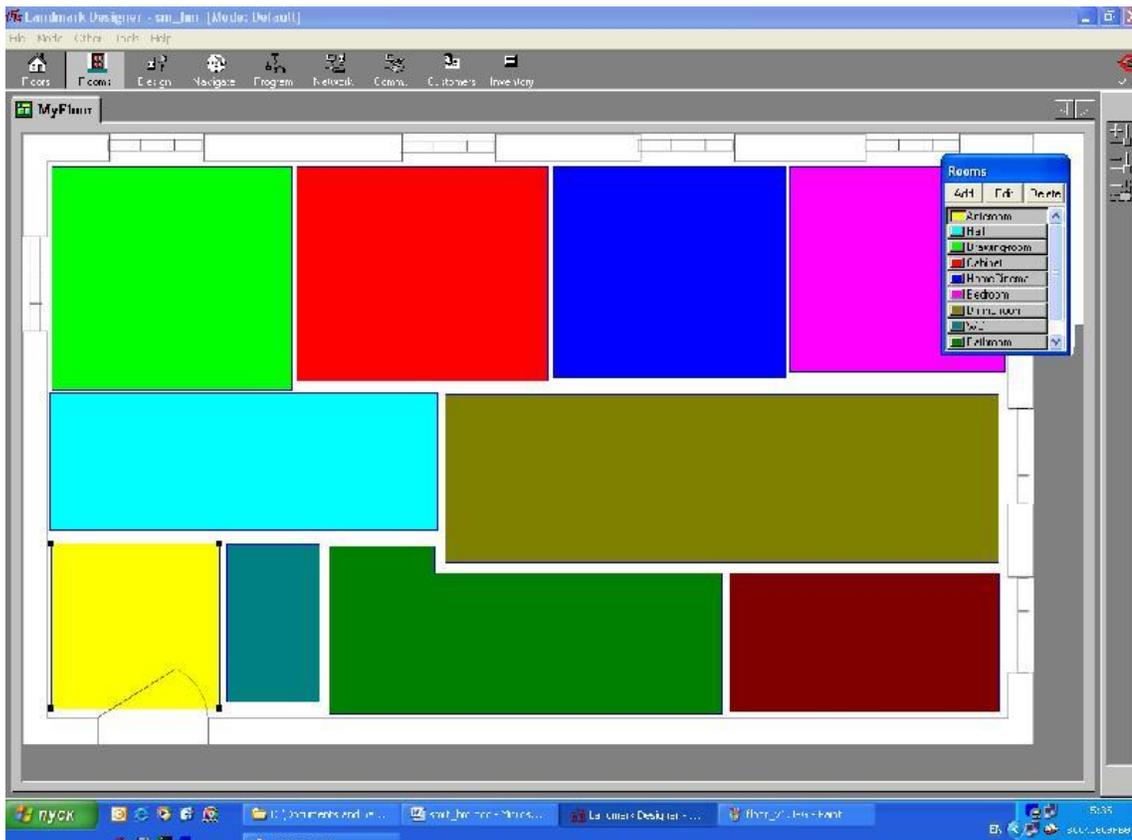


Рис.16

Если теперь перейти на вкладку «Design», то получим следующий вид проекта:

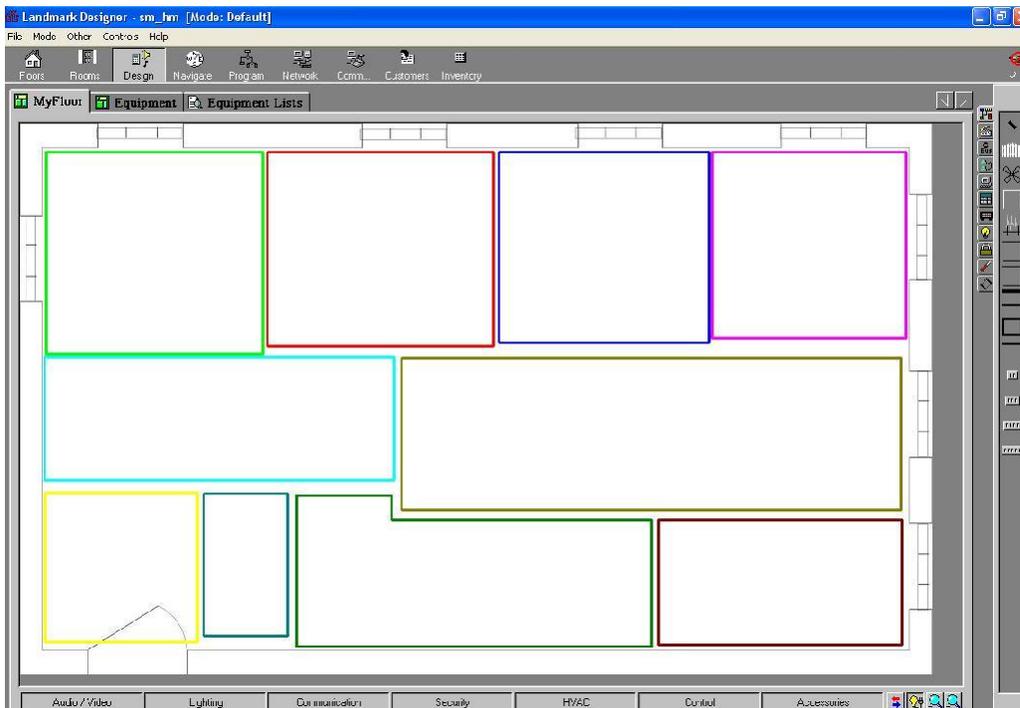


Рис.17

На инструментальной панели страницы (справа) все устройства, поддерживаемые системой. Их нужно расположить по комнатам, соответственно с реальным положением дел. Для начала расставим свет (клавиша с лампочкой). Нажимаем, отыскиваем на подменю лампочку, обозначаем помещение, например, гостиная, и, нажав на лампочку и удерживая левую клавишу мышки, переносим в нужное место:

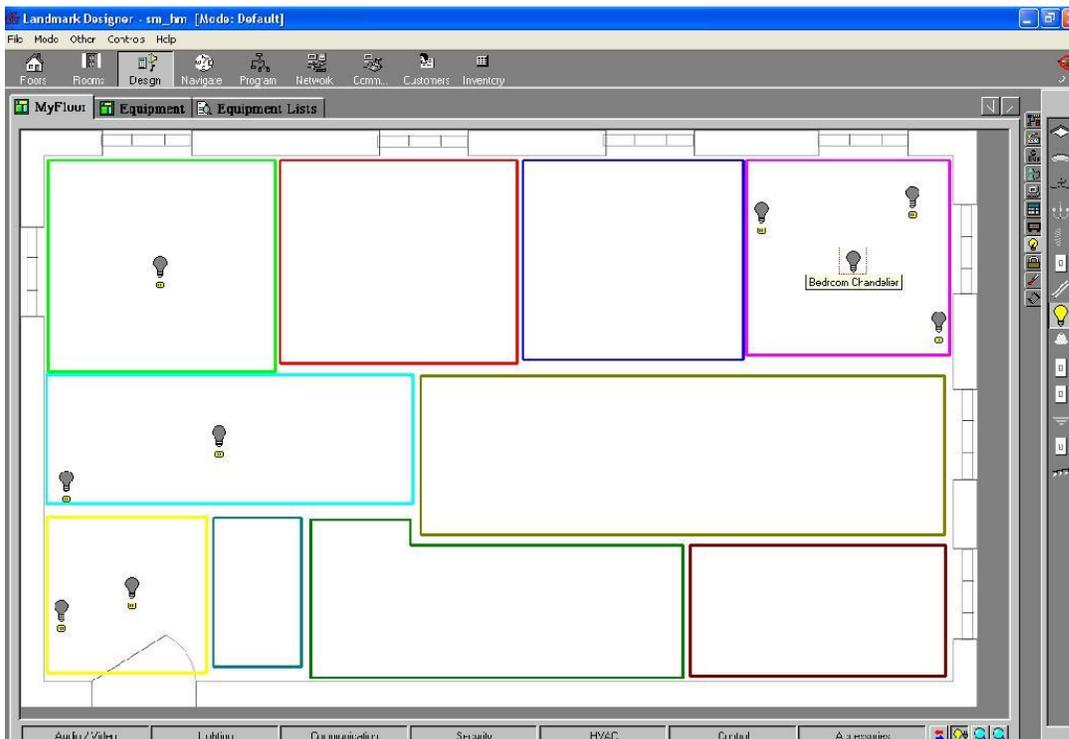


Рис.18

После расстановки источников света (люстр, бра, групп светильников), задаем нужные свойства и меняем, если это нужно, обозначения. При использовании диммеров следует обратить внимание на опцию «Ramp to level gradually» слева в рубрике On/Off Rate. Если не установить эту опцию, то плавное включение будет недоступно в программе. Остальные опции позволяют выбрать скорость нарастания яркости – Dim rate, установить уровень яркости в ночное и дневное время, яркость свечения индикаторов в ночное и дневное время и т.д. На первой вкладке «Control» после того, как источник света идентифицирован, можно управлять им в интерактивном режиме, когда компьютер подключен к системе. На других вкладках можно построить сцены управления светом (вместе с люстрой, нажав на клавишу ее включения, можно включать потолочную подсветку в спальне и т.п.). Включается меню настроек (свойств - properties) либо двойным нажатием левой клавиши мышки, либо нажатием правой клавиши при позиционировании на объекте и выборе в выпадающем меню раздела свойства (properties):

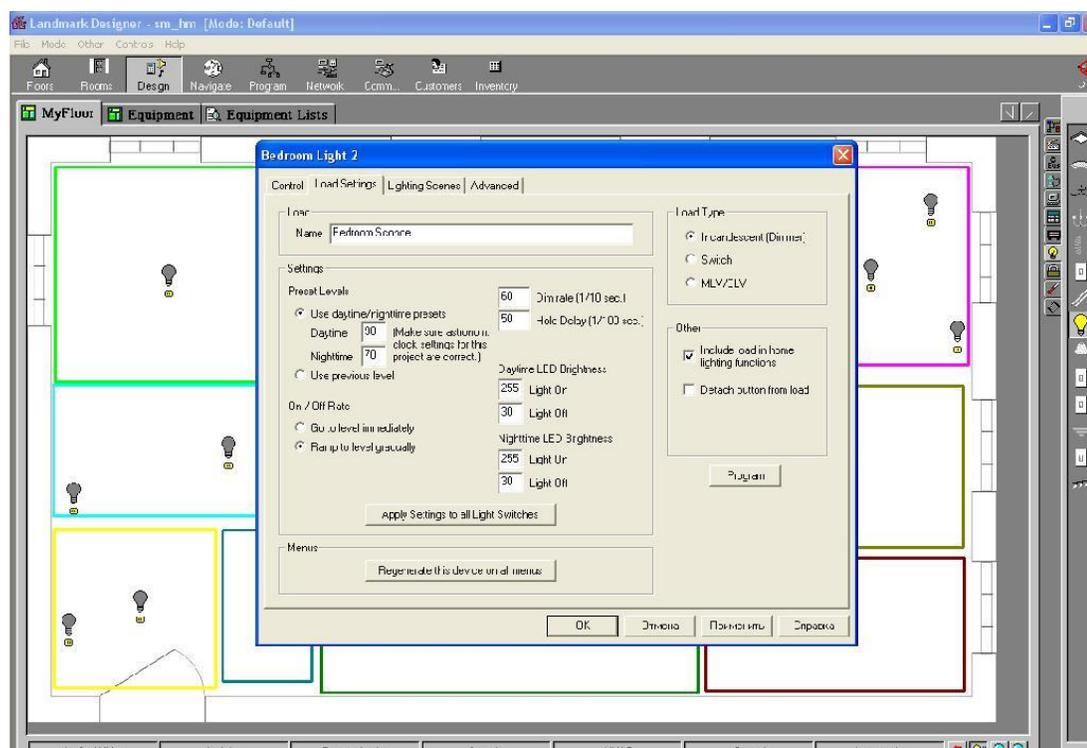


Рис..19

Расставив источники света, задав все необходимые свойства, пора заглянуть в раздел основной инструментальной панели озаглавленный Program. В данном случае я раскрыл подменю спальни (bedroom):

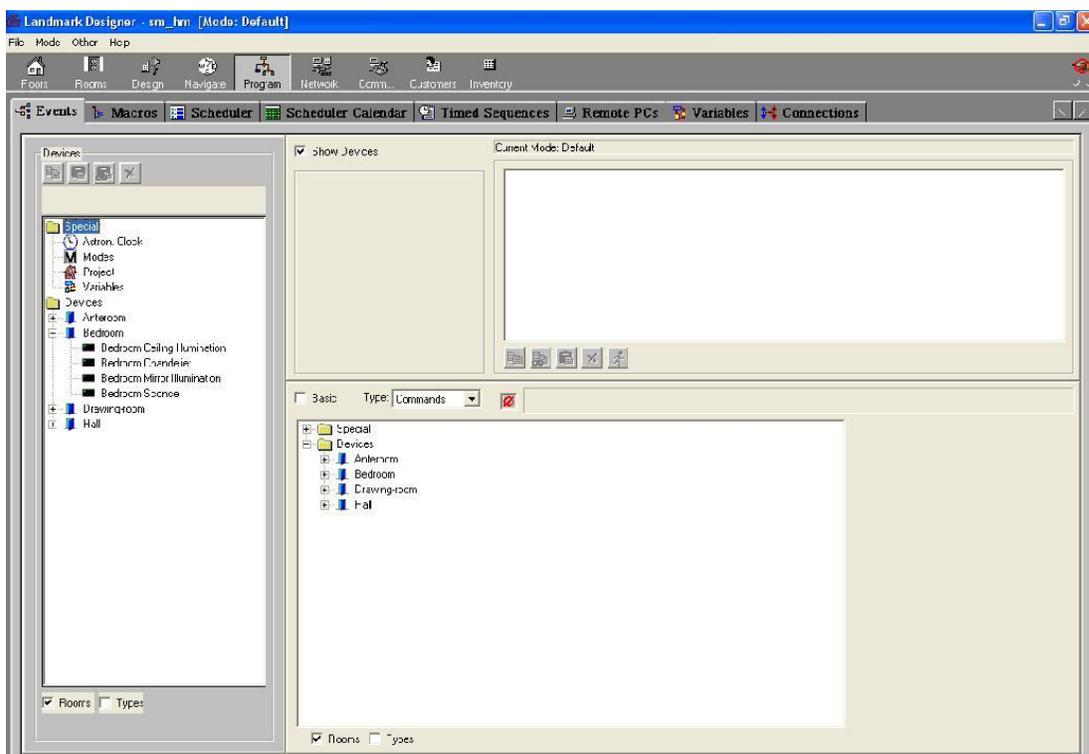


Рис.20

В левом окне вкладки событий (events) в спальне обозначены все светильники по их функциональному назначению. Обязательно ли это? Нет. Для профессионалов, возможно, даже удобнее иной подход с обозначением помещений 1, 2, 3, а источников света 1 Light1, 1 Light 2 и т.д., поскольку в рабочей документации будут таблицы соответствия. Однако я считаю удобным сразу обозначать функциональное назначение помещений и оборудования. В дальнейшем это позволит осмысленнее подходить к программированию событий – события в кухне и спальне, мне кажется, могут сильно различаться.

Аналогично в проект добавляются устройства управления, аудио и видео оборудование, датчики и т.п. На этом подготовка к работе с проектом закончена, можно начинать программирование.

Отвлечемся немного от технических деталей. Поскольку все системы, с которыми мне приходилось сталкиваться, в равной мере позволяют осуществить, хотя бы формально, любую программу, самое время определиться, а что мы хотим видеть в программе?

В качестве примера приведу простенькую программу, отображающую мои привычки. По возвращении домой я привык первым делом приготовить чашечку кофе, и выпить ее, просматривая программу новостей по телевизору.

Посмотрим, насколько просто подобную программу можно реализовать в системе Landmark. Но вначале я несколько изменю предыдущий вариант конфигурации квартиры:

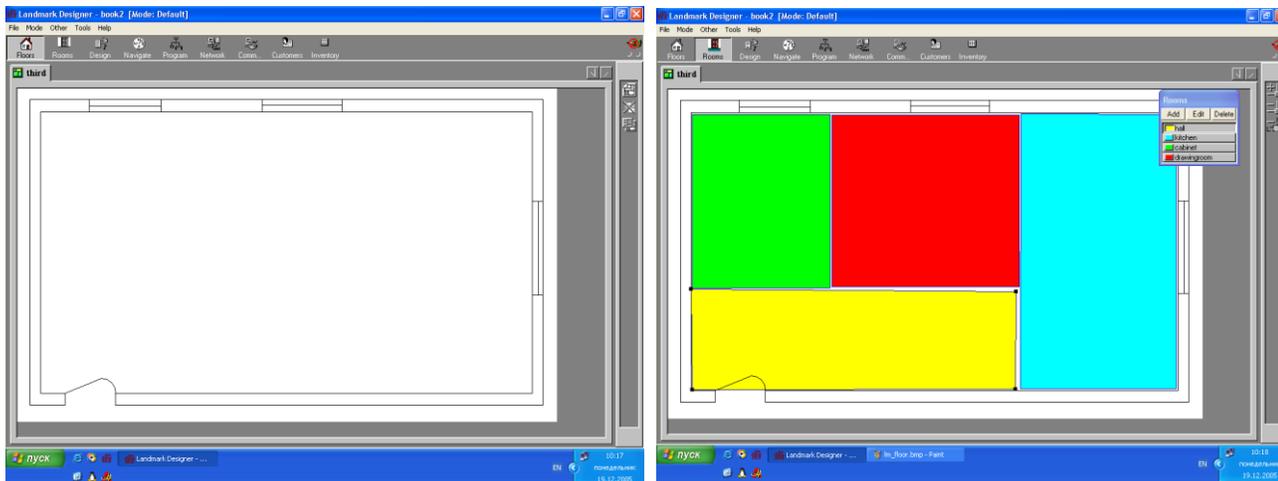


Рис.21

Здесь желтым цветом обозначена прихожая, зеленым кабинет, красным гостиная, а бирюзовым кухня.

Расставим необходимые элементы:

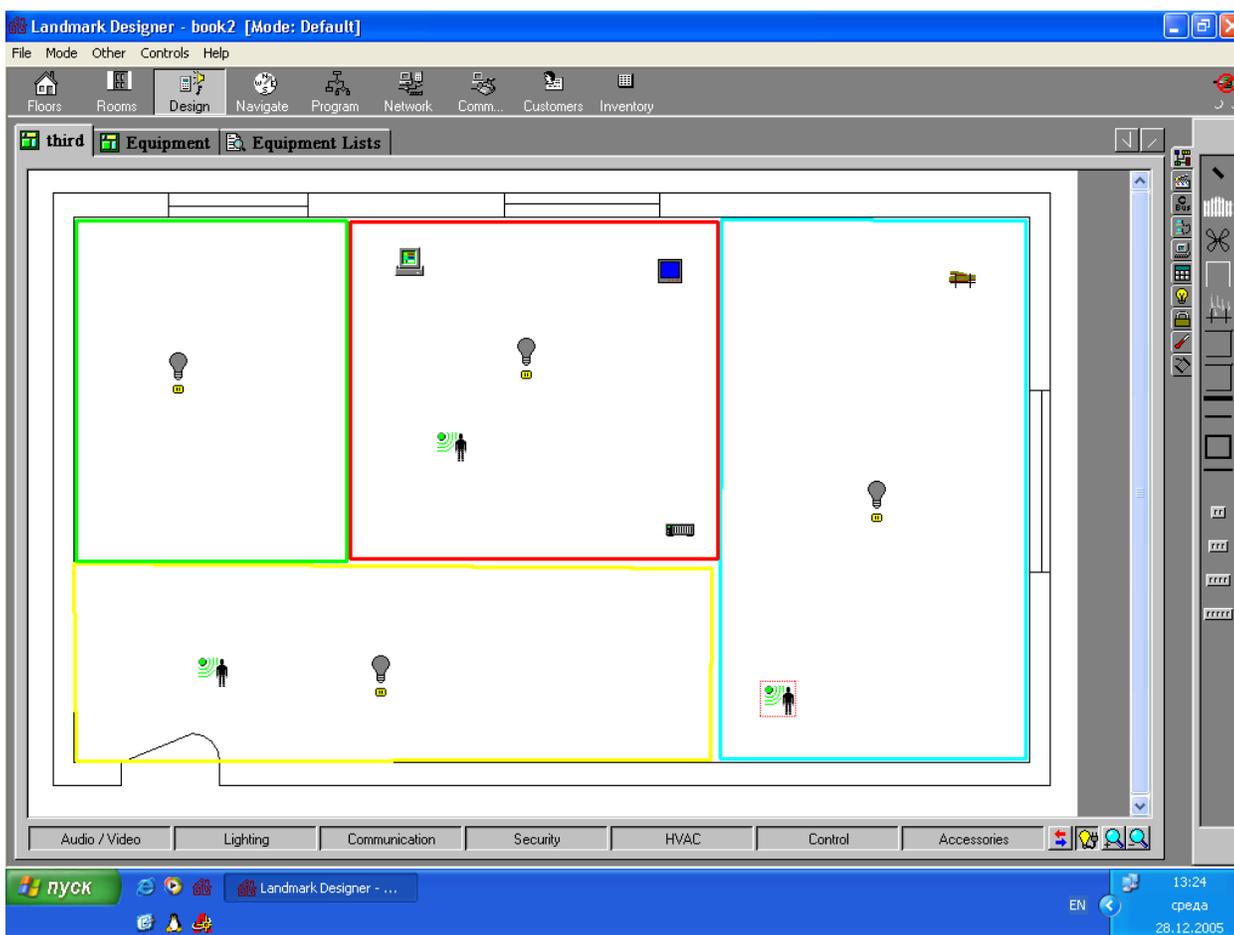


Рис.22

Во всех комнатах я устанавливаю свет. Везде, кроме кабинета, добавляю датчики движения, в гостиной будет телевизор и сенсорная панель управления, а на кухне будет плита. В гостиной я поместил и CardFrame (конструктив, в котором расположатся модули). По команде «спецификации» оборудования (Spec) программа заполнит CardFrame необходимыми модулями:

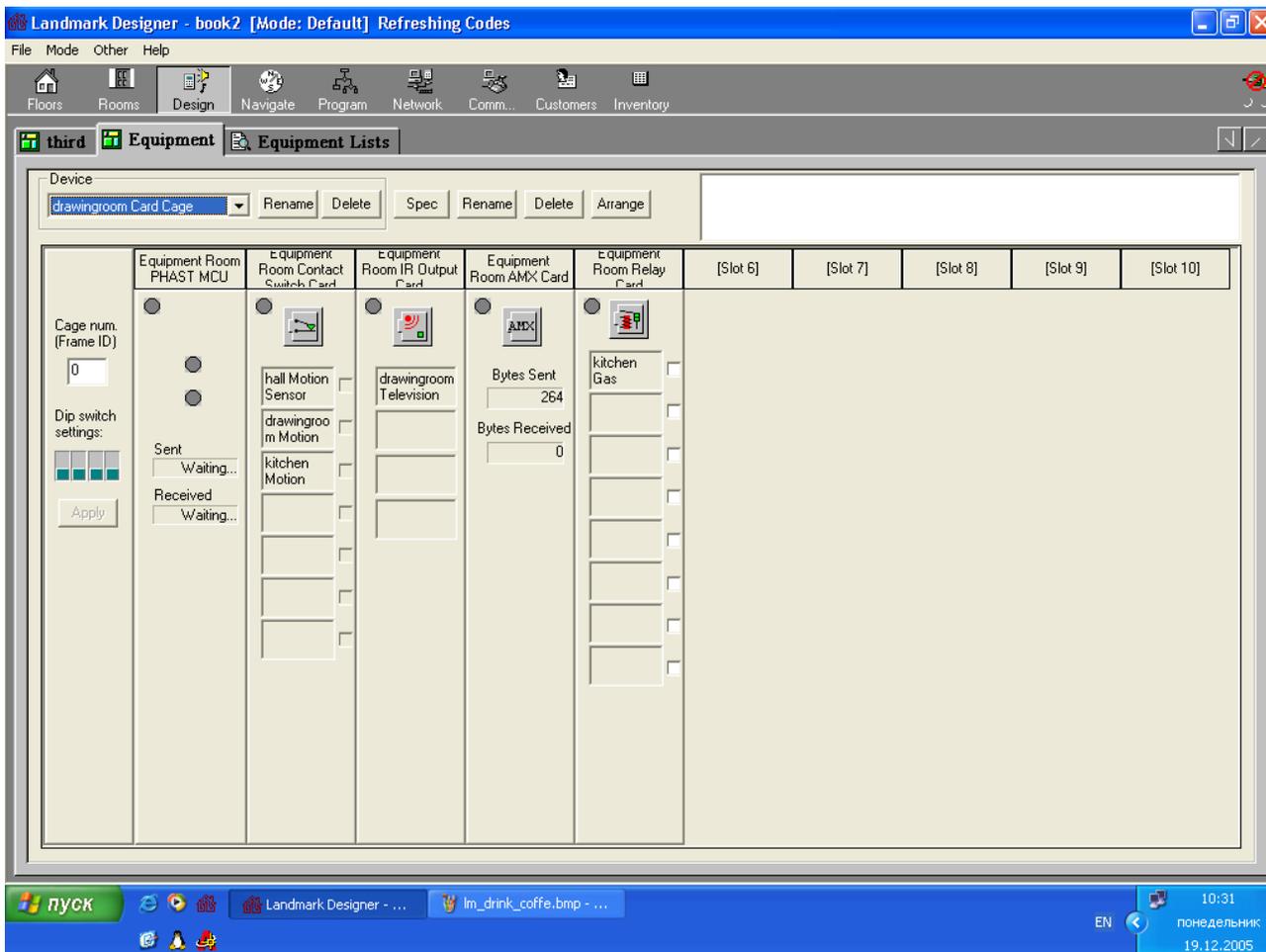


Рис.23

Теперь можно перейти к программированию. Опишем события: я прихожу домой, сработавший датчик движения в холле зажигает свет в холле, включает мой электрочайник, чтобы, пока я переоденусь, вода для кофе вскипела. Конечно, чайник я наполняю водой и ставлю на подставку перед уходом на работу:

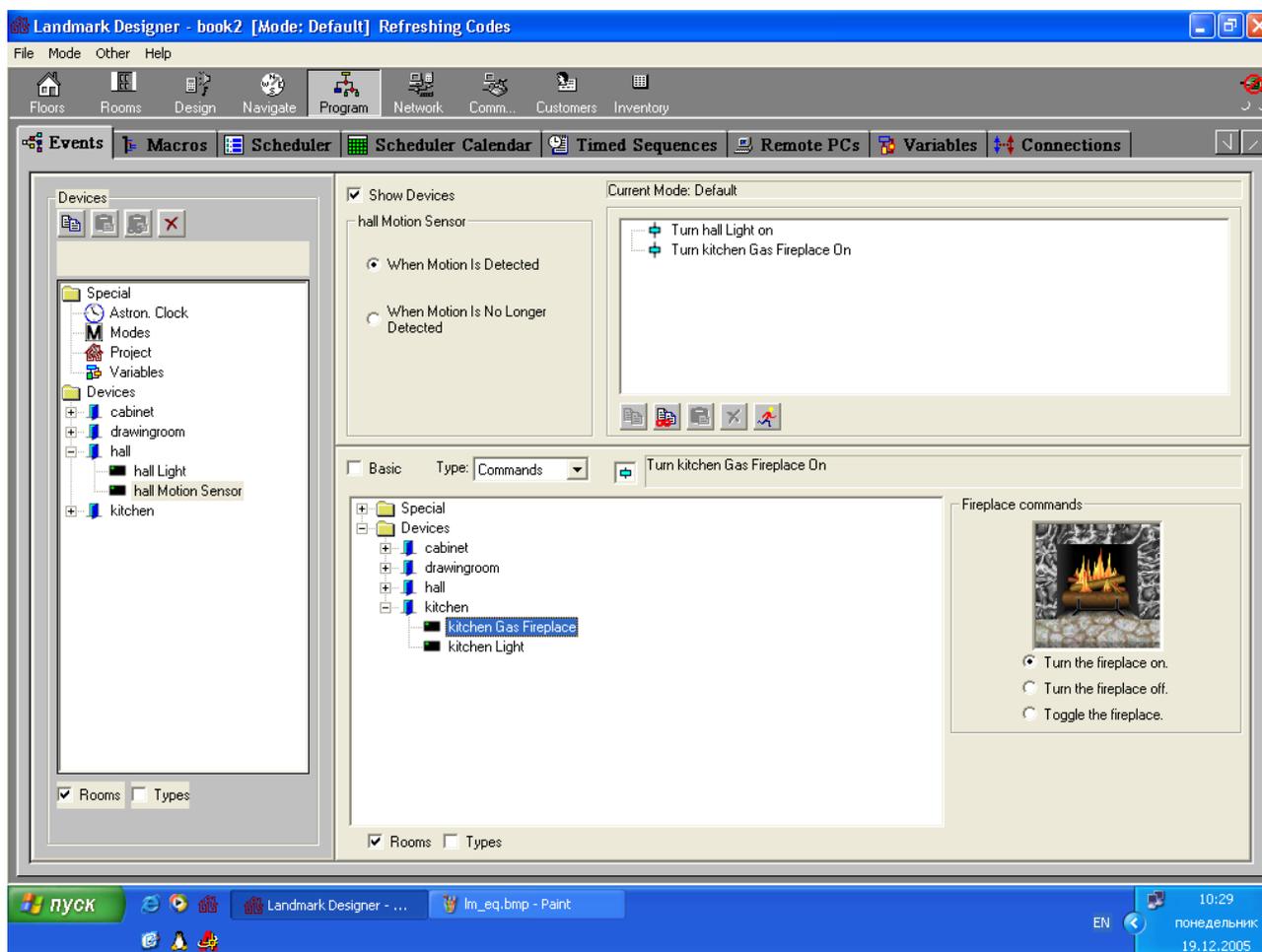


Рис.24

Переодевшись, я перехожу в кухню, наливаю кофе, и иду в гостиную. Программа выключает свет в холле, выключает электрочайник на кухне, включает телевизор в гостиной, едва датчик движения в гостиной отметил мое появление:

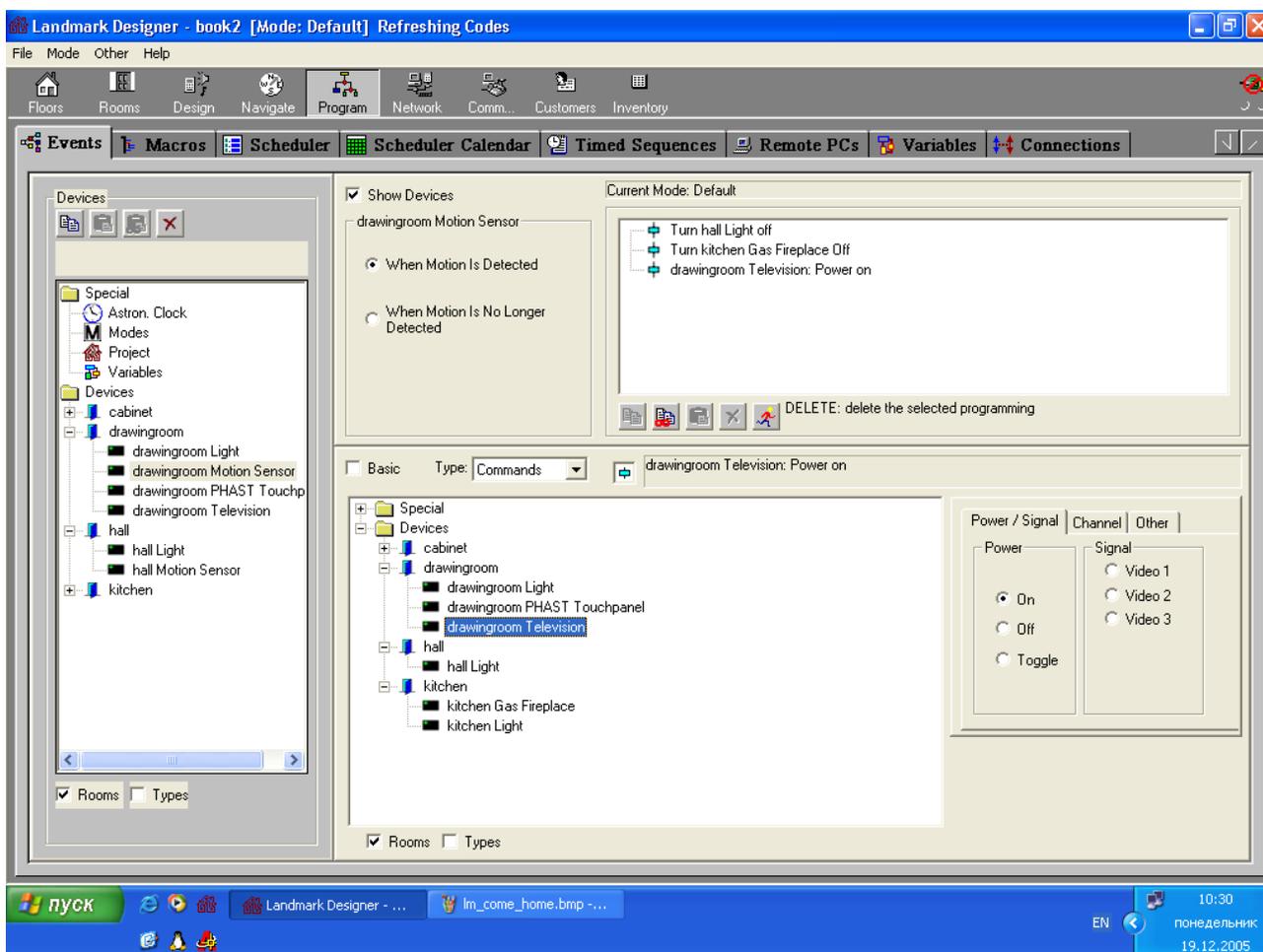


Рис.25

Модули, которые программа задействовала самостоятельно, это, конечно, модуль процессора, модуль, поддерживающий сенсорную панель, модуль цифрового ввода. К этому модулю мы подключим датчики движения. Еще система добавила релейный модуль. С помощью этого универсального модуля можно включить и выключить такой бытовой прибор, как электрический чайник (выключается он сам, но модуль дополнительно отключит его и от сети). Модуль трансляции ИК кодов займется управлением телевизора. С помощью сенсорной панели я могу легко управлять всеми бытовыми устройствами квартиры, например, открыть входную дверь и включить свет в холле, если сосед (или соседка) зашел в гости на чашку кофе.

Система Landmark централизованная. Она может работать под управлением компьютера, под управлением собственного процессора, в который программа загружается, и переходить от управления с помощью компьютера к управлению от собственного процессора, если компьютер по какой-то причине выключается.

Конечно, она имеет средства организации распределения аудио и видео сигналов, организации охраны дома, управления климатическими устройствами и т.д. Под «и т.д.» я подразумеваю дальнейшее описание работы с системой Landmark, что, однако, не является в данном случае целью повествования.

А вот как среда программирования выглядит в NetLinx Studio – программе, предназначенной для работы с процессорами и модулями AMX. После запуска программы можно создать новый проект обычными для Windows средствами (Меню, файл, новый), или открыть существующий:

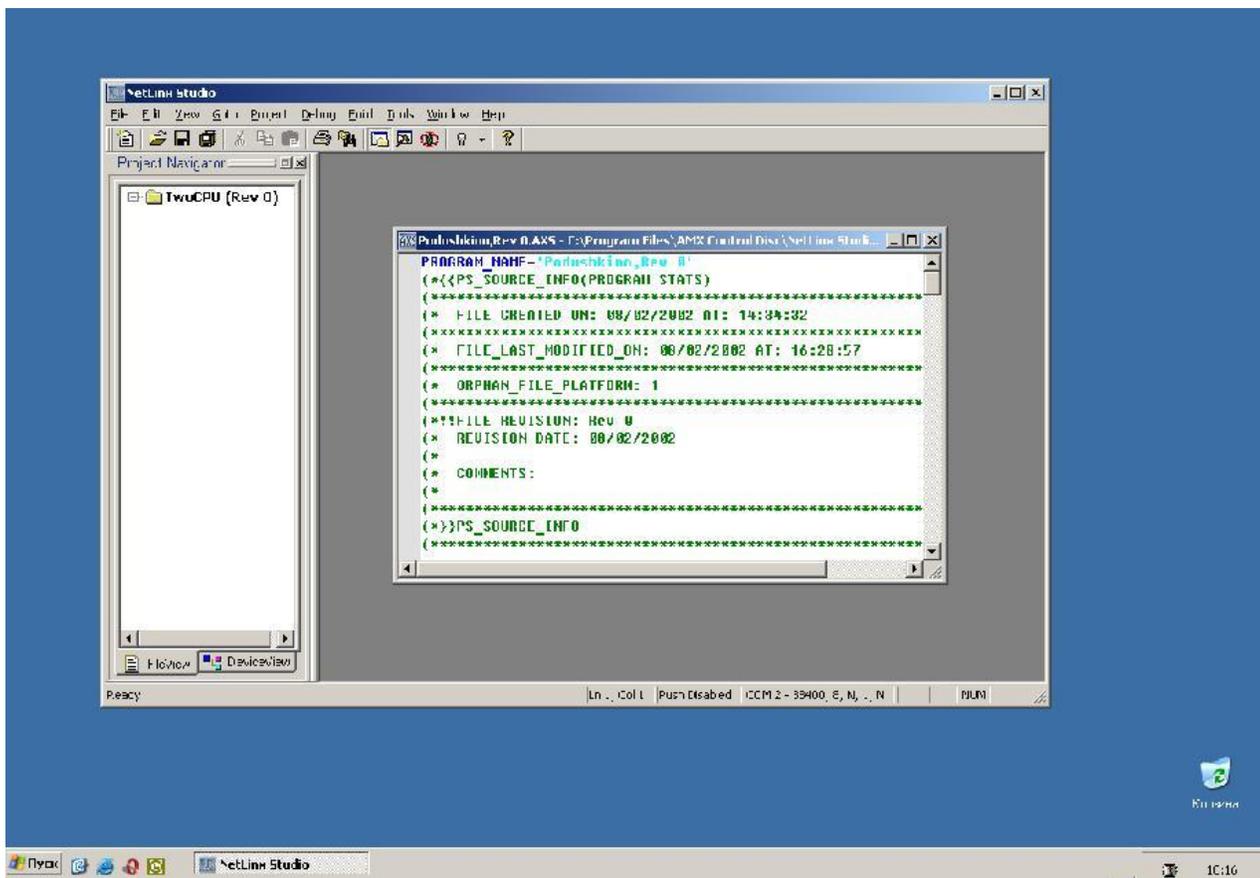


Рис.26

Язык программирования, что отражается и на среде программирования, больше похож на процедурно-ориентированный язык, хотя не следует забывать, что он остается специализированным языком программирования. Встроенный редактор позволит скопировать программу в блокнот из Windows и продолжить программирование в блокноте. При наличии готовых программных блоков они легко добавляются в программу. Таким образом, даже достаточно сложную программу можно быстро скомпоновать из рабочего материала предыдущих разработок. Если вы специализируетесь на работе с системой NetLinx, то кроме программных блоков, предлагаемых производителем, у вас много своих, надеюсь, систематизированных, проверенных и отлаженных. Вот с установкой устройств проблем несколько больше. Программа, после начального процесса установки устройств, будет выглядеть, примерно, так:

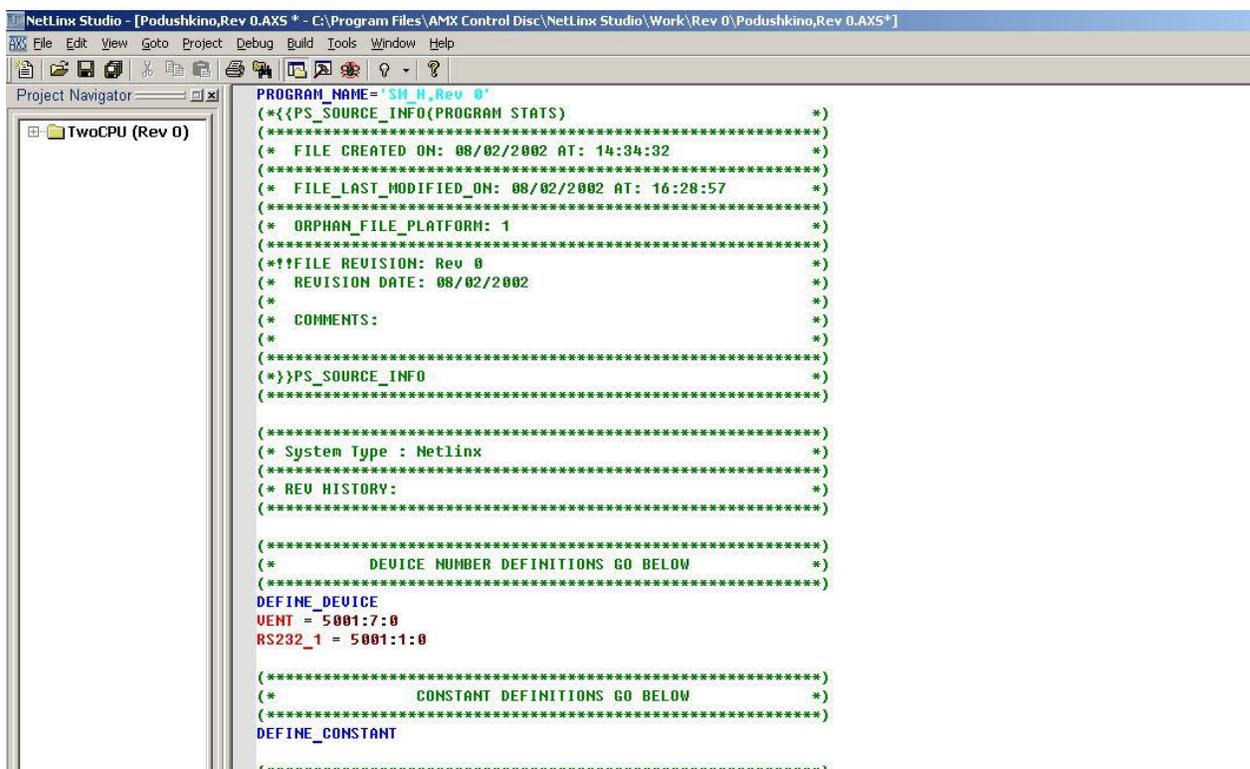


Рис.27

Во всех системах после начального подготовительного процесса, который никак нельзя назвать очень сложным, мы готовы приступить к программированию.

## Система «StarGate» - X10

Системы, работающие по протоколу X10 - это другой ценовой полюс. Множество производителей выпускают, как отдельные компоненты, так и законченные системы автоматизации. Можно использовать централизованное построение, можно создать децентрализованную систему. В качестве примера рассмотрим систему StarGate (JDS).

Как и предыдущая, она позволяет быстро объединить события с реакцией системы, создав программу обслуживания дома. Имеет и схожий набор устройств, в котором, как и в других системах, есть все необходимое для создания подсистем управления светом, управления аудио и видео оборудованием, подсистем климатической и охранной, подсистемы управления, например, садовым оборудованием или гаражом.

О перечне модулей, составляющих физическое наполнение системы, можно судить по меню, которое открывается в программе после обращения к разделу Define основного меню:

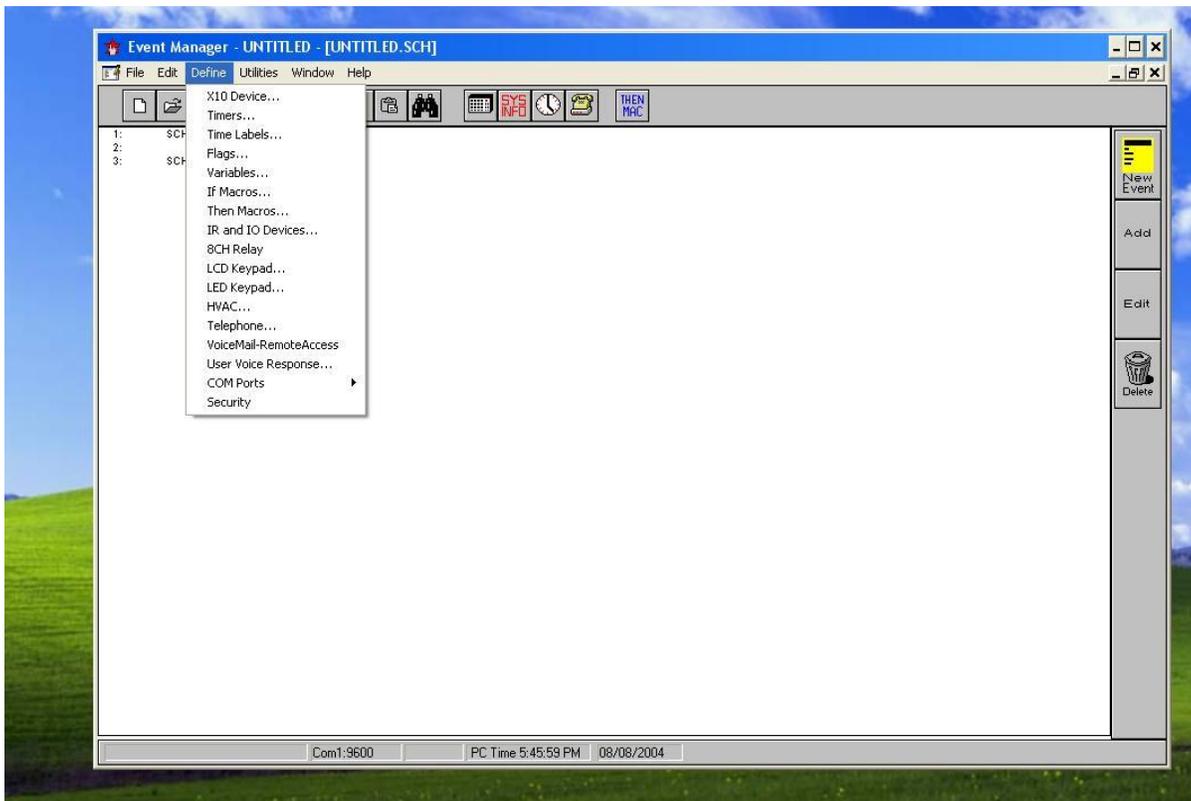


Рис.28

При создании нового проекта программа добавляет начало и конец программы:

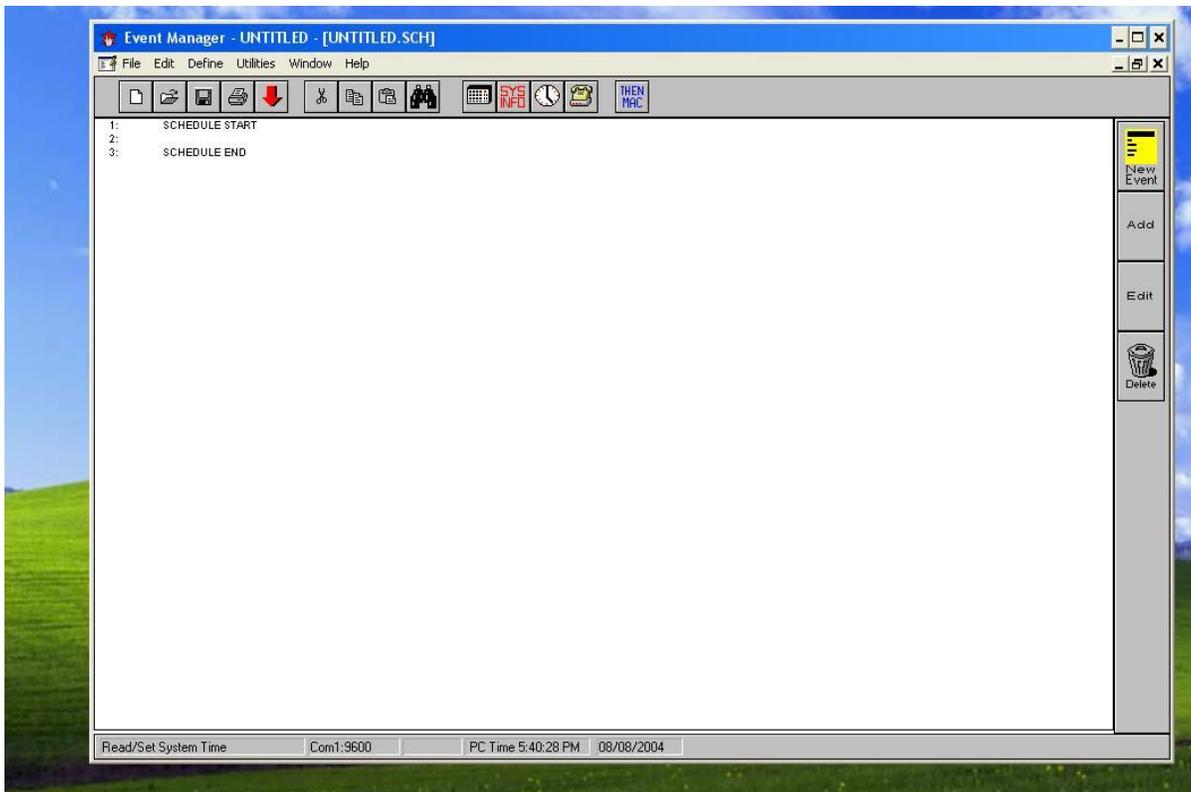


Рис.29

После запуска программы и выбора New Schedule (через основное меню File или инструментальную панель – первая клавиша) в оглавлении, мы готовы перечислить оборудование нашего проекта. Как и в случае с системой Landmark, начнем со света.

В основном меню выберем раздел Define, в котором выберем первую строку X10 Device...

Открывается таблица, в которой можно обозначить все источники света (но не только). В системах, работающих с сетевым протоколом X10, адресация устройств состоит из двух знаков: латинской буквы от А до Р и цифры от 1 до 16. Всего, таким образом, можно адресоваться к 256 устройствам, что достаточно для довольно большой системы. Чуть выше я упоминал таблицы соответствия, так вот в данном случае, именно с подобной таблицей мы и имеем дело:

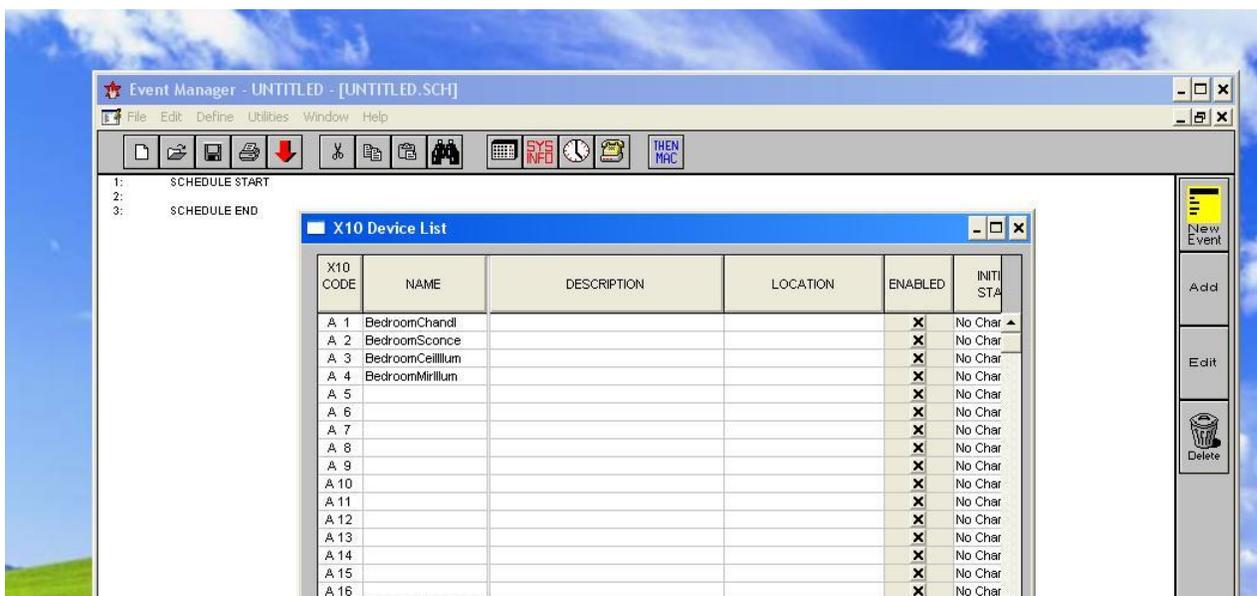


Рис.30

Расположение выключателей света можно задать в описании – Location, назначение - Description.

Я хочу повторить предыдущую маленькую программку, реализованную в системе Landmark. Поэтому выберу светильники и добавлю релейный модуль для включения электрического чайника:

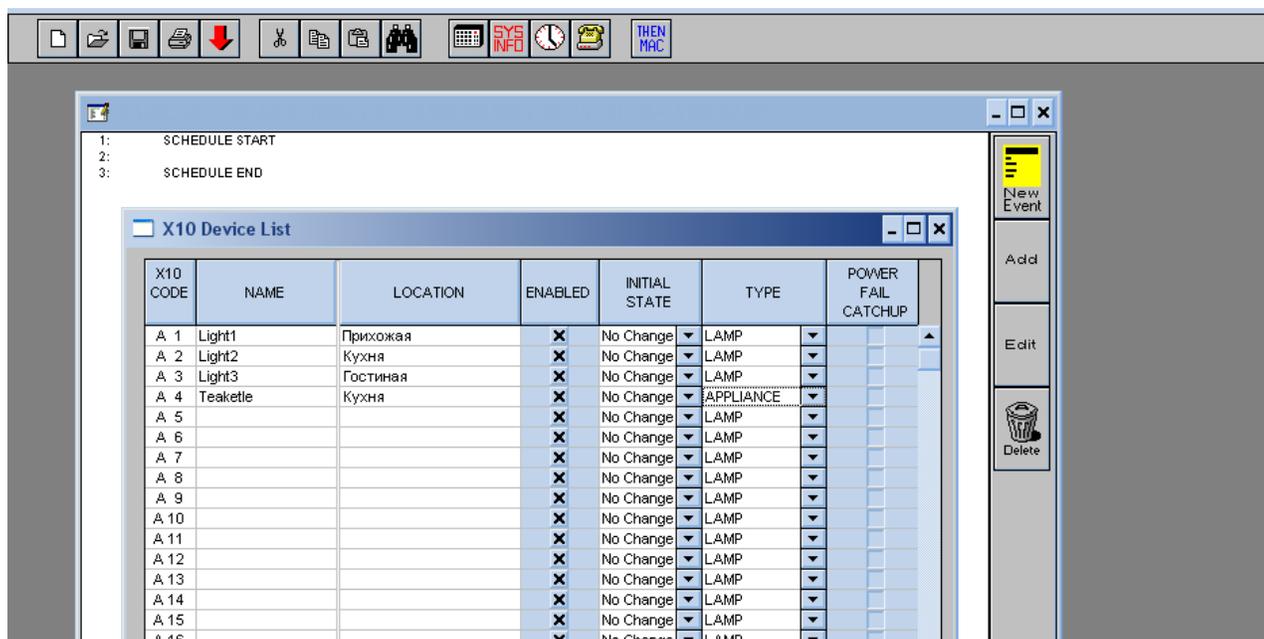


Рис.31

Осталось добавить в проект датчики движения, которые будут работать со встроенным модулем цифрового ввода (Define - IR and IO Devices):

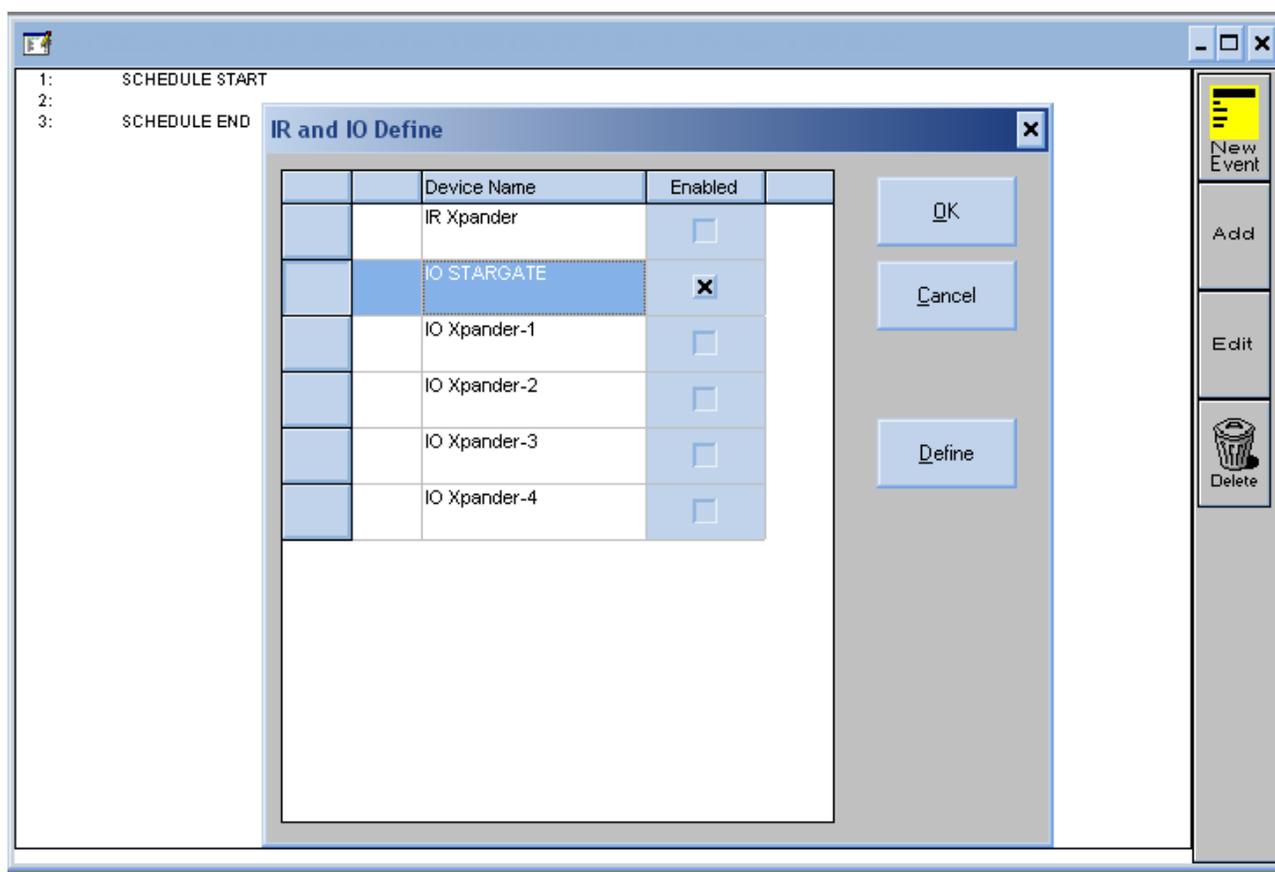


Рис.32

Я выбираю модуль ввода-вывода, нажимаю клавишу Define и определяю свои датчики движения:

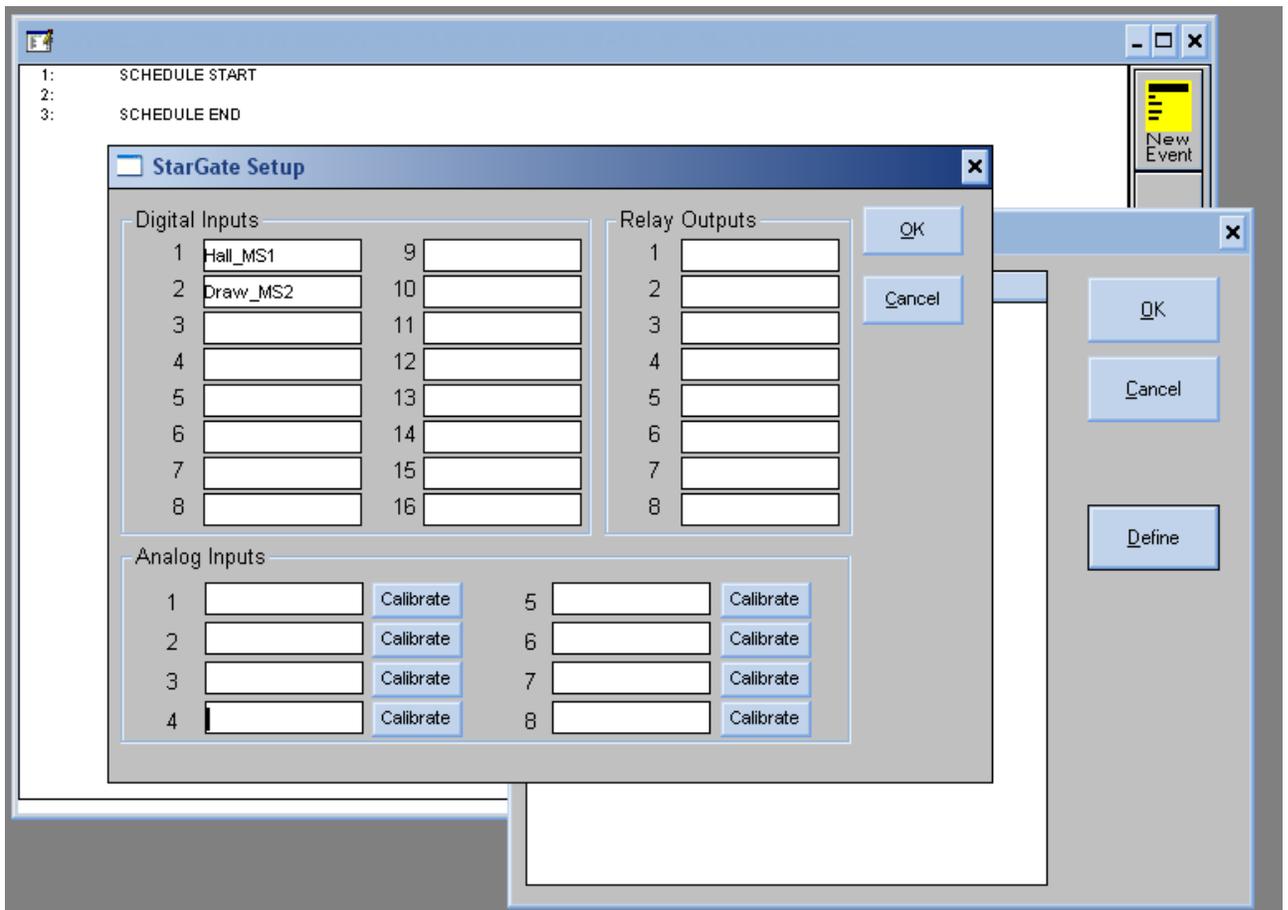


Рис.33

Сохраним выбор и можно начинать программирование.

В редакторе выбираем клавишу New Event, обозначаем первое событие, как some\_home, и к строке if с помощью клавиши Add и выбора из меню Digital Input выбираем событие – датчик движения hall\_MS1 перешел в состояние ON:

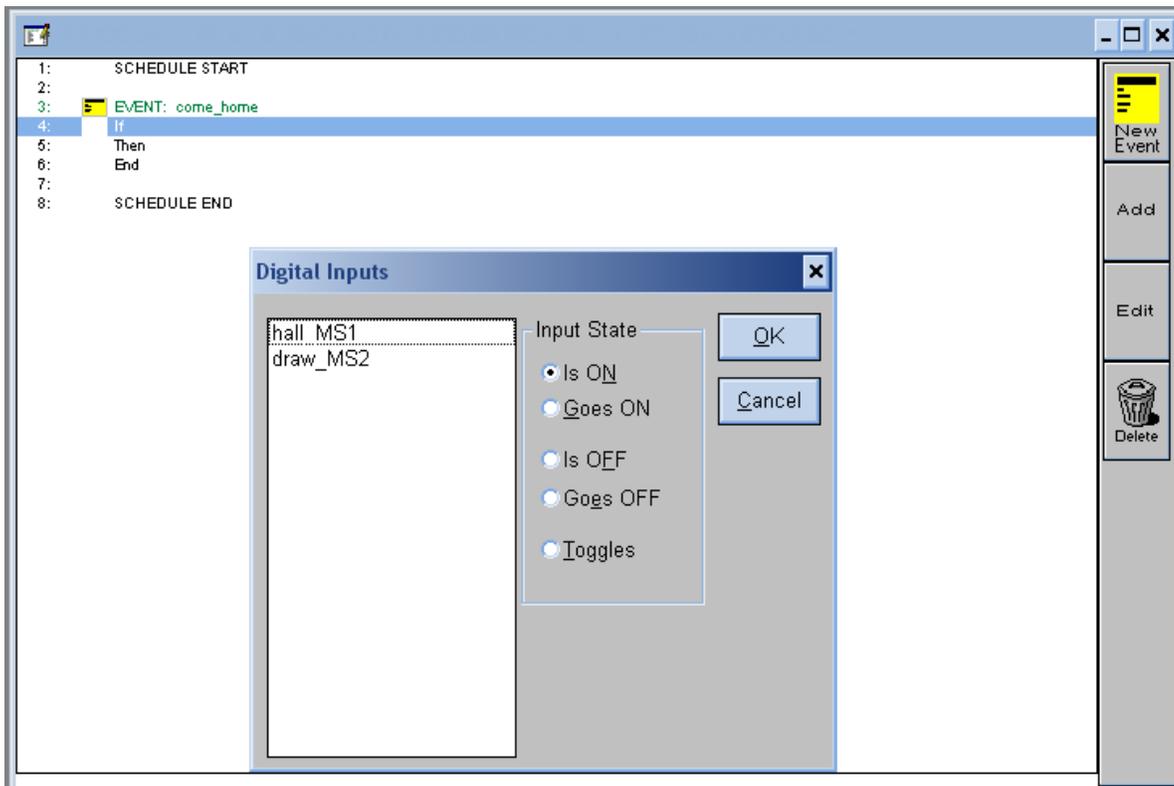


Рис.34

Продолжая этот процесс, с помощью клавиш New Event и Add пишем программу:

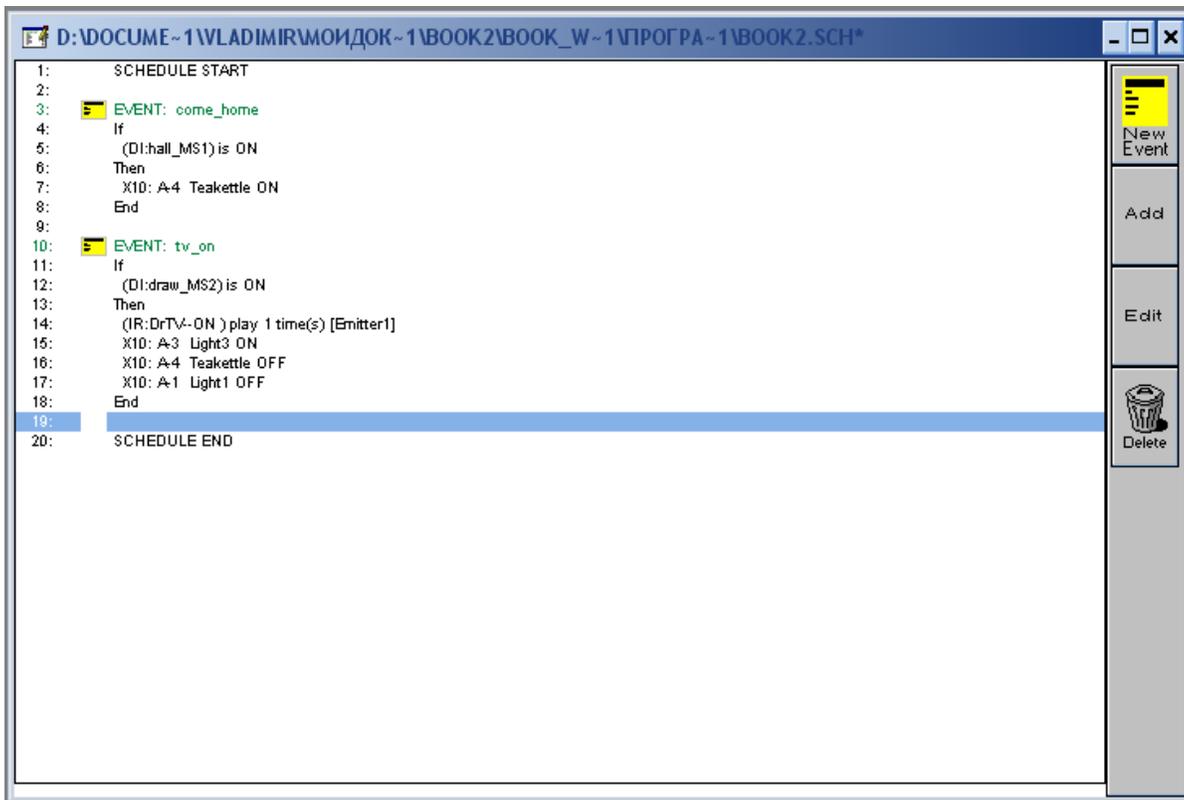


Рис.35

Обе системы после загрузки программы встретят меня после работы, зажгут свет в прихожей, вскипятят воду для кофе. А когда я перейду в гостиную, включат телевизор, чтобы я, усевшись в любимое кресло, посмотрел, что произошло в мире за день, пока я был на работе.

К выше сказанному следует добавить, что для управления телевизором обе системы требуют дополнительного устройства, которое можно назвать модулем считывания ИК команд. Они предназначены для запоминания ИК кодов с пультов управления, с тем, чтобы эти коды можно было впоследствии воспроизвести системным устройством. В системе Landmark это дополнительное устройство носит название IRIS. А StarGate работает с IR Xpander'ом. Последнее устройство не только прочитывает ИК команды с пультов и запоминает их, но может распознавать эти команды, оно же воспроизводит их.

Беглый обзор двух систем «Умный дом» завершен. Зачем он понадобился? Затем, чтобы можно было составить представление о том, как это выглядит в профессиональных разработках, и чтобы можно было сравнить сделанное нами с профессиональными разработками.

Хочу добавить, что не следует забывать - за видимой простотой процесса программирования, как в Landmark, так и в StarGate, скрывается система, которая рано или поздно проявит все свои свойства. При первом знакомстве может возникнуть иллюзия, что программирование по легкости схоже с игрой в кубики: сложил так, получил домик, переложил иначе – получаешь паровозик. Подобная иллюзия может стать причиной того, что домик, наподобие паровозика, запыхтит и поедет... однажды.

И последнее. Что следует «подвергать» автоматизации. Вот, как на это смотрят специалисты PHAST:

### **Что должно автоматизироваться?**

*В принципе, любое электрическое или механическое устройство, любая подсистема внутри или снаружи вашего дома может быть автоматизирована, хотя бы насколько-то.*

*Поскольку огромно количество устройств и подсистем, которые вы можете интегрировать с системой домашней автоматизации, важно выбрать систему достаточно гибкую, расширяемую, обновляемую, и по приемлемой цене. Это должна быть система, которая легко устанавливается, программируется и имеет программный интерфейс, не требующий от вас освоения новой техники программирования каждый раз, когда вы добавляете новое устройство в вашу систему.*

*Корпорация PHAST развивает современную автоматизацию дома, которая полностью отвечает этим требованиям.*

*Заметьте: Поскольку список характеристик, относящихся к разным системам автоматизации и компонентам, которые могут быть автоматизированы, велик, система автоматизации дома может стать весьма сложной. Для преодоления проблемы вам следует обращаться к системам, с которыми вы лучше всего знакомы. PHAST настоятельно рекомендует тщательное изучение всех незнакомых систем или устройств, прежде чем вы примете решение о внедрении их в систему Landmark.*

## **Постановка задачи**

Пойдем от противного, и в первую очередь решим, что мы не намерены делать?

Мы не намерены соревноваться с производителями оборудования, создателями систем автоматизации. Таким образом, мы не намерены осуществлять разработку самой современной системы, использовать промышленные протоколы работы или протоколы производителей оборудования.

Хотя идея использовать силовую сеть для построения системной сети, используя протокол X10, кажется весьма привлекательной, мы не будем этого делать по той причине, что эксперименты с силовой сетью опасны.

Приведенные выше примеры существующих решений дают возможность копировать их, но нужно ли это делать? Например, среда программирования, удобная и красивая, в первую очередь ориентирована на наискорейшую реализацию проекта. Нужно ли это нам? Не уверен. Если реализация коммерческого проекта – это цель работы, то наша цель – это процесс реализации. Нам спешить некуда.

С другой стороны, коммерческие проекты устремлены на поддержку всех существующих бытовых устройств, на включение в систему бытовых приборов, которые появятся в ближайшие годы, тогда как нам не обязательно поддерживать все. Мы хотим научиться делать это, но вовсе не намерены делать это.

Что же мы намерены сделать?

Создать свою систему, к которой можно, пусть с большими оговорками, применить термин «Умный дом» в той мере, в какой он применим к существующим системам.

Как правило, подобные системы ориентированы на управление, если они централизованные, от специализированного процессора, в который загружается предварительно подготовленная программа. Следует ли повторять это? Пока я в этом не уверен - у нас есть компьютер, который мы будем активно использовать на протяжении всей работы, так пусть компьютер и играет роль центрального процессора системы. Другие возможности обсудим по ходу дела.

Как мы объединим устройства в систему? Или как мы организуем системную сеть?

В общем случае сеть должна иметь определенную пропускную способность, некоторую емкость, определяющую наибольшее количество устройств в сети, и допускать необходимую протяженность. В нашем случае достаточно двух-трех устройств, включенных в сеть, и расположенных на рабочем столе. Этого достаточно для понимания того, как работает наша система, и как подойти к созданию подобной системы, а, значит, решает нашу задачу.

Одной из проблем при построении сети является проблема коллизий сети. Что это

значит? Если два или более устройств в сети одновременно проявляют активность, то в сети возникают проблемы аналогичные проблемам разговора в шумном обществе, где каждый говорит о своем. В итоге трудно понять, кто и о чем говорит. Есть простой способ избежать проблемы, оставив в сети только одно активное устройство. Так мы и поступим. Единственное активное устройство, оно же центральное управляющее устройство, избавит нас от необходимости вводить сложный протокол сетевой работы, или устанавливать арбитраж в сети.

В качестве сетевого интерфейса («дверки», за которой расположена сеть) можно выбрать достаточно широко применяемый, и достаточно удобный интерфейс RS485. Чем он удобен?

- Для работы сети достаточно двух проводов, все устройства подключаются к сети параллельно, не требуя дополнительных сетевых концентраторов или маршрутизаторов.
- Длина сети может достигать 1000 метров. Сеть мало подвержена влиянию наводок.
- Есть микросхемы, которые можно использовать для построения интерфейса.
- Программная часть протокола не определена, но многие используют стандарт RS232.

Таким образом, интерфейс перекрывает все наши нужды. Но есть проблема - компьютер не имеет порта, работающего с интерфейсом RS485. Однако есть конвертеры, преобразующие стандартный (пока еще) COM-порт компьютера, в интерфейс RS485.

Мы определились с центральным управляющим устройством. Компьютер, выполняя эту задачу, одновременно послужит и панелью управления. Мы определились с сетью. Пора определиться с составом нашей системы.

Какие устройства мы намерены разработать?

Полагаю, что вы уже заглянули на сайты производителей коммерческих систем, и убедились в огромном количестве производимых ими компонент. Выберем из них наиболее универсальные.

В первую очередь, как мне кажется, это релейный модуль. Релейный модуль можно применить для управления освещением (контакты реле будут включать и выключать свет). Его можно применить для управления бытовыми нагревательными приборами, например, электрическим чайником или обогревателем. Релейный модуль можно использовать в качестве коммутатора для подключения к музыкальному центру громкоговорителей, расположенных в разных помещениях. При желании с помощью релейного модуля можно управлять и громкостью. Очень удобный компонент системы!

Нам понадобятся какие-то средства управления, кроме компьютера. С этой целью мы создадим модуль для приема системных ИК (инфракрасных) команд.

И, наконец, разработаем модуль трансляции ИК команд. Практически вся бытовая аудио и видео техника управляется ИК пультами. Модуль будет играть роль такого пульта в системе.

## Хобби-электроникс 2. Умный дом.

Думаю, на первом этапе этого нам хватит, затем посмотрим, возможно, разработаем дополнительные модули.

Относительно среды программирования системы. Я предлагаю использовать Visual Basic (по причине того, что он у меня есть). В дальнейшем определимся.

Первую разработку первоначально проведем «в бумажном виде». Все, что нам потребуется – лист бумаги, конечно электронный, и документация на микропроцессор PIC16F628A. Эта документация есть в хорошем переводе на сайте [www.microchip.ru](http://www.microchip.ru). Существует и оригинальная документация. Некоторые детали, касающиеся описания микроконтроллера, будут, как мой конспект, включены в третью часть. Это не будут выдержки из русскоязычного варианта, а именно мой конспект оригинала. Пора за работу.

### Релейный модуль

Требования к модулю:

1. Модуль должен иметь реле, контакты которых позволят подключить нагрузку с параметрами: ~220В, 10А (мощности нагрузки ~ 2 кВт).
2. Модуль должен иметь сетевой интерфейс RS485 (системная сеть).
3. Модуль по отношению к системной сети должен быть пассивным, т.е. его основным режимом работы будет прослушивание сети и выполнение команд, полученных по сети, если они адресованы модулю.
4. Адрес модуля задается установкой переключателя в соответствующее положение.

Нарисуем структуру модуля.

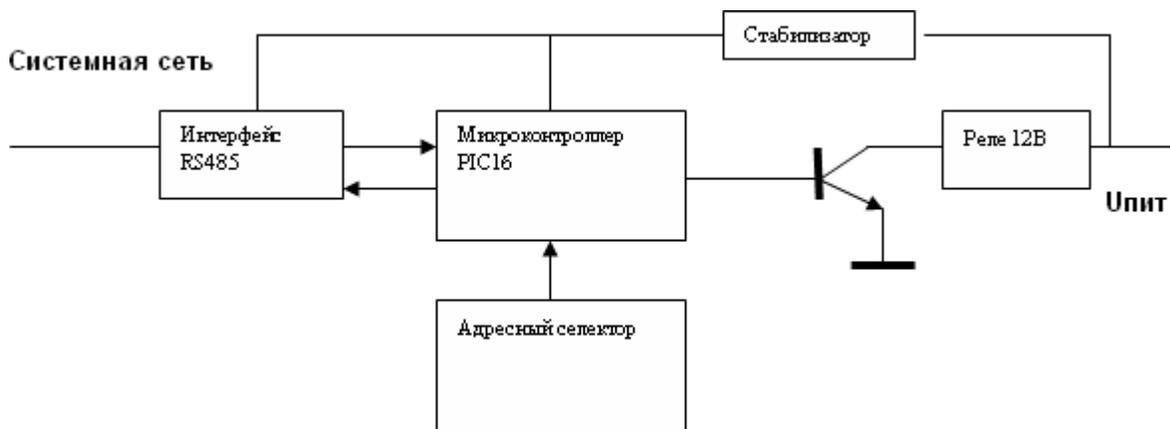


Рис.36

Интерфейс RS485, адресный селектор и микроконтроллер составят базовую часть модуля. Другие модули, которые предстоит разработать, будут иметь такую же базовую часть.

Теперь попробуем разобраться с некоторыми частностями. В первую очередь выберем организацию интерфейса RS485. Она проста. Это микросхема MAX1483CPA (можно

## Хобби-электроникс 2. Умный дом.

использовать MAX3082), предназначенная для построения интерфейса RS485 двухпроводной сети, поддерживающая до 256 устройств и работающая в полудуплексном режиме со скоростью до 115 kbps в диапазоне температур от 0 до 70°C.

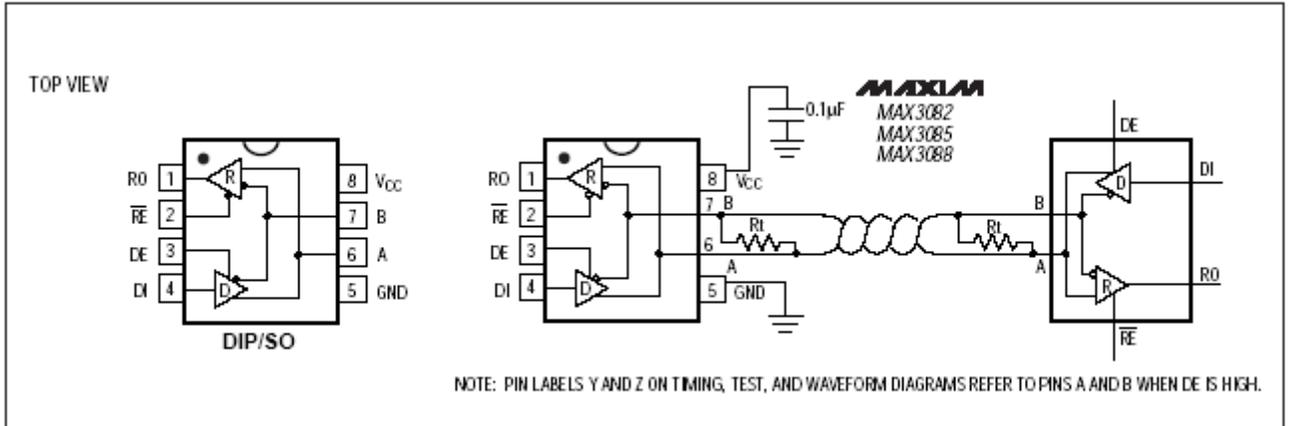


Figure 3. MAX3082/MAX3085/MAX3088 Pin Configuration and Typical Half-Duplex Operating Circuit

Рис.37

Микросхема имеет выходы, позволяющие переводить ее выход в третье состояние. Напряжение питания 5В, при этом потребляемый ток не превышает 1 мА.

Таблица состояний микросхемы:

### MAX3082/MAX3085/MAX3088

| TRANSMITTING    |    |    |          |        |
|-----------------|----|----|----------|--------|
| INPUTS          |    |    | OUTPUTS  |        |
| $\overline{RE}$ | DE | DI | B/Z      | A/Y    |
| X               | 1  | 1  | 0        | 1      |
| X               | 1  | 0  | 1        | 0      |
| 0               | 0  | X  | High-Z   | High-Z |
| 1               | 0  | X  | Shutdown |        |

| RECEIVING       |    |               |          |
|-----------------|----|---------------|----------|
| INPUTS          |    |               | OUTPUT   |
| $\overline{RE}$ | DE | A-B           | RO       |
| 0               | X  | $\geq -0.05V$ | 1        |
| 0               | X  | $\leq -0.2V$  | 0        |
| 0               | X  | Open/shorted  | 1        |
| 1               | 1  | X             | High-Z   |
| 1               | 0  | X             | Shutdown |

Рис.38

Поскольку контроллер PIC16F628A имеет встроенный USART, мы используем его для работы с интерфейсом RS485.

Адресный селектор можно организовать на микроконтроллере различным образом. Для начала используем наиболее очевидный способ - выводы порта, включенные на ввод

## Хобби-электроникс 2. Умный дом.

информации, соединены с контактами переключателя, который устанавливает их в состояние единица или ноль. По ходу дела мы обсудим другие способы задания адреса. Четыре бита дают нам возможность адресоваться к 16 устройствам, шесть битов расширяют эти возможности до 64 устройств.

Я думаю, что если использовать релейные модули по прямому назначению, то четырех-пяти устройств нам вполне хватило бы. Но! Мы можем использовать эти же модули для управления светом, правда, в режиме включено-выключено, т.е. без управления яркостью свечения. Замечу, на первый взгляд это не так интересно, как регулирование яркости, но для последнего режима выключатель светильника должен иметь (или очень желательно иметь) не только фазовый провод силовой сети, но и нулевой, что большинство реальных выключателей света не имеет. Вместе с тем, для большинства светильников режим «включено-выключено» вполне достаточен для работы. Я имею в виду, что для управления светом мы можем использовать те же релейные модули. А выключателей света в каждой квартире достаточно много. С другой стороны, применяя эти же модули для управления светом, мы можем изменить обращение к ним, оставив меньшее количество устройств в группе. Тем более что в качестве именно релейных модулей нам выгодно наполнить их максимальным количеством реле (4-8), а в качестве управляемого выключателя достаточно иметь одно или два реле. Примем для начала, что 4х адресных битов хватит для построения релейного модуля, а для других применений решим этот вопрос в тот момент, когда придет время определяться.

Протокол сетевой работы естественно использовать RS232 в смысле передачи байта информации со старт-стоповыми битами.



Рис.39<sup>1</sup>

Каждый бит имеет длительность 4 мкс (при данной скорости).

Для обращения к релейным модулям используем дополнительный адресный префикс «R». Адрес в сети будет выглядеть как R00-R15. Конечно, подобный трех байтовый адрес (каждый символ это байт) весьма расточителен, но наша цель научиться строить системы и наладивать их. А при отладке сетевых устройств удобно будет контролировать сеть с помощью терминала компьютера, т.е. получать печатные символы. Таким образом, наше расточительство оправдано удобствами наблюдения за сетью с помощью компьютера.

<sup>1</sup> Диаграмма взята из статьи «Применение интерфейса RS485» с сайта <http://measure.chat.ru>.

## Хобби-электроникс 2. Умный дом.

Практически, мы определились со всем необходимым для того, чтобы приступить к построению релейного модуля. Конструкцию модуля выберем в виде отдельного блока, имеющего гнездо RJ12 или клеммник для подключения системной сети и питающего напряжения, побитовый на четыре направления и два положения переключатель для задания адреса устройства, и клеммник (напряжение ~220В, ток 10А) для подключения контактов реле. Питающее напряжение будем подавать от централизованного блока питания по двум дополнительным проводам системной сети. Конечно, применение модуля в качестве устройства, управляющего светом в помещении, потребует изменения конструкции. Но вопрос с выключателями света мы отложим до второй части книги.

Можно приступить к работе над контроллером. Запишем

*Команды, которые будет принимать наш модуль:*

Включить – Rxx\$хN

Выключить – Rxx\$хF

Я добавлю команду передачи модулем состояния реле:

Передать статус – Rxx\$хS

При передаче статуса используем следующую символику:

Включено – Rxx#хN

Выключено – Rxx#хF

Здесь Rxx\$хN означает: R - релейный модуль, хх – два символа адреса от 00 до 15, \$ - символ команды (# - символ статуса), х – номер реле от 0 до 7, N – включить (F - выключить).

*Уточним, как мы будем использовать контроллер?*

Для ввода информации используем встроенный в контроллер USART.

Использование порта В:

RB0 – управление передатчиком (микросхема MAX1483).

RB1 – прием (приемник USART).

RB2 – передача (передатчик USART).

Для ввода заданного адреса используем четыре старших бита порта В (RB4-RB7).

Использование порта А:

Для подключения реле используем биты порта А (RA0-RA7), что предполагает использование восьми реле. При необходимости увеличить количество реле, можно использовать свободные (если они есть) биты порта В.

*Мы готовы приступить к написанию программы.*

Основные блоки программы:

- Инициализация (конфигурация контроллера).
- Ожидание и прием команды (ожидание активности в сети, чтение команды, проверка адреса, и, если адрес совпадает с адресом устройства, выполнение команды, возвращение к ожиданию команды).
- Выполнение команды.

Распишем их поподробнее.

### *Инициализация.*

В этом блоке программы мы должны задать конфигурацию контроллера, должны задать, если это будет использовано, переход в режим низкого энергопотребления (SLEEP), и условия выхода из этого режима. В блоке инициализации мы читаем адрес, задаваемый адресным переключателем.

### *Ожидание команды и прием команды.*

Этот блок программы является основным, поскольку в сети модуль играет пассивную роль, т.е. ожидает команды и выполняет команды, адресованные ему. При активизации сети прочитывается команда. Если префикс совпадает с префиксом модуля, а адрес с адресом устройства, команда выполняется, иначе модуль переходит в режим ожидания следующей сетевой команды. Адрес в слове команды сравнивается с ранее прочитанным (при инициализации) адресом модуля.

### *Выполнение команды запроса статуса.*

Этот блок программы отличается от предыдущего, начиная с момента получения символа «S» вместо «N» или «F». С этого момента программа должна определить состояние соответствующего вывода порта, переключить USART на передачу, и передать статус в формате Rxx#xN, если соответствующий вывод порта в состоянии «1» или Rxx#xF, если он в состоянии «0». После этого программа переходит к ожиданию следующей команды.

### *Блок инициализации контроллера.*

В первую очередь определим, нужен ли нам режим «SLEEP». Этот режим переводит контроллер в состояние с уменьшенным энергопотреблением. Режим очень удобен и важен в тех случаях, когда контроллер используется в условиях батарейного питания. Переход в режим «SLEEP» продлевает срок службы батарей или время работы без подзарядки аккумулятора. В нашем случае нет необходимости в поддержании этого режима, поскольку мы ориентированы на питание модуля от 12-вольтового блока питания.

Пока не забыл, есть часть конфигурирования, которую необходимо выполнить при программировании контроллера. Это относится к слову конфигурации по адресу 2007h. Здесь «h» после цифр – означает HEX, hexadecimal (шестнадцатеричное число). В слове конфигурации устанавливаются (или нет) биты защиты, выбирается режим работы тактового

## Хобби-электроникс 2. Умный дом.

генератора, и выбираются некоторые параметры, относящиеся к режиму питания контроллера.

*Вот первый вариант слова конфигурации для тактовой частоты 16-20 МГц:*

Бит 13 устанавливаем в «1» - выключаем защиту кода.

Бит 8 устанавливаем в «1» - выключаем защиту EEPROM.

Бит 7 устанавливаем в «0» - вывод RB4 работает как цифровой канал ввода-вывода.

Бит 6 устанавливаем в «0» - запрещаем сброс по снижению напряжения питания.

Бит 5 устанавливаем в «0» - вывод RB5 работает как цифровой канал ввода-вывода.

Бит 4 устанавливаем в «0» - режим HS генератора, кварц подключается к выводам RA6 и RA7.

Бит 3 устанавливаем в «0» - выключаем режим включения по таймеру.

Бит 2 устанавливаем в «0» - выключаем режим работы сторожевого таймера.

Бит 1 устанавливаем в «1» - режим HS генератора, кварц подключается к выводам RA6 и RA7.

Бит 0 устанавливаем в «0» - режим HS генератора, кварц подключается к выводам RA6 и RA7.

Слово конфигурации будет выглядеть так – 3F0Ah.

Для работы внутреннего тактового генератора следует использовать кварцевый резонатор с параллельным резонансом на 16 - 20 МГц.

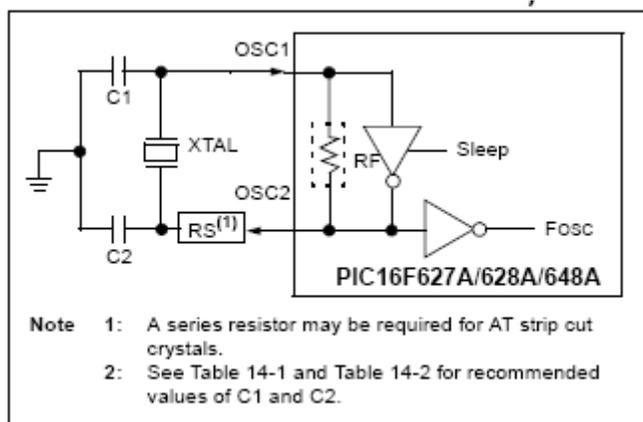


Рис. 40

Для частоты 16 МГц рекомендованные значения конденсаторов C1 и C2 – 10-22 пФ.

Давайте подумаем, а не использовать ли встроенный в контроллер генератор? Будем ли мы выполнять какие-либо операции, требующие стабильной тактовой частоты? На стабильность частоты внутреннего генератора в первую очередь влияют температура и напряжение питания. Будут ли они меняться значительным образом? Температура, если мы не планируем использовать модуль в уличном исполнении, будет меняться незначительно. Питающее напряжение, при условии, что мы после входного напряжения 12В поставим стабилизатор на 5В, будет изменяться еще меньше. В данный момент я больше склонен отказаться от кварцевого резонатора, и использовать внутренний тактовый генератор.

## Хобби-электроникс 2. Умный дом.

В этом случае есть два варианта на выбор. Для использования режима ER (3,6 МГц) к микросхеме подключается резистор (не более 40 кОм) к выводу RA7. Вывод RA6 может использоваться для ввода-вывода.

### *Второй вариант слова конфигурации.*

Биты 13-10 устанавливаем в «1» - выключаем защиту кода.

Бит 8 устанавливаем в «1» - выключаем защиту EEPROM.

Бит 7 устанавливаем в «0» - вывод RB4 работает как цифровой канал ввода-вывода.

Бит 6 устанавливаем в «0» - запрещаем сброс по снижению напряжения питания.

Бит 5 устанавливаем в «0» - вывод RB5 работает как цифровой канал ввода-вывода.

Бит 4 устанавливаем в «1» - режим ER генератора, резистор 40 кОм подключается к RA7.

Бит 3 устанавливаем в «0» - выключаем режим включения по таймеру.

Бит 2 устанавливаем в «0» - выключаем режим работы сторожевого таймера.

Бит 1 устанавливаем в «1» - режим ER генератора, резистор подключается к RA7.

Бит 0 устанавливаем в «0» - режим ER генератора, резистор подключается к RA7.

Слово конфигурации будет выглядеть так – 3F1Ah.

Еще более удобный вариант – использование внутреннего генератора без внешних элементов INTRC (4 МГц). Слово конфигурации в этом случае - 3F18h.

*Здесь я хотел бы сделать замечание, или признание – как вам угодно. Я, начиная с этого места, впервые буду использовать контроллер PIC16F628A. Я впервые буду пользоваться средой программирования контроллера MPLAB. Может возникнуть закономерный вопрос – а чему я могу научить кого-то, если сам впервые это делаю? Если кого-либо смущает этот аспект, то он может обратиться к многочисленной литературе по программированию контроллеров, пропустив дальнейшее, что и решит проблему. Но мне кажется, что, имея большой опыт работы с чем-то, основательно забываешь те трудности, с которыми столкнулся в начале работы. А если самому впервые начать работать, то все эти трудности будут налицо.*

В блоке инициализации контроллера необходимо в первую очередь установить конфигурацию портов А и В.

Нам потребуется установить биты порта А 0-6 на вывод, бит 7 на ввод (к выводу RA7 порта подключается резистор в режиме ER), или все выводы порта А установить на вывод для режима генератора INTRC.

Для этого в регистр TRISA необходимо записать «1» для входных выводов и «0» для выходных. Ниже приведен пример инициализации для порта А:

```
CLRF PORTA           ;Initialize PORTA by setting output data latches
                     ;(инициализация установкой защелок данных)
MOVLW 0x07           ;Turn comparators off and enable pins for I/O functions
MOVWF CMCON          ;(выключение компараторов для I/O функций)

BCF STATUS, RP1
BSF STATUS, RP0      ;Select Bank1 (выбор банка 1)
```

## Хобби-электроникс 2. Умный дом.

```
MOVLW 0x80      ;Value used to initialize data direction
MOVWF TRISA     ;Set RA<4:0> as outputs TRISA<5> always read as '1'.
                ;TRISA<7:6> depend on oscillator mode
                ; (в данном случае все на вывод, 7 ввод)
```

*Здесь 0x07 означает, что число 7 – шестнадцатеричное.*

Инициализация порта В связана с необходимостью использования USART. Необходимо установить RB0 на вывод и записать в него «0» (при передаче записывается «1»). Далее

RB1 устанавливается на ввод, а RB2 на вывод.  
RB3 установим на вывод (для управления передачей микросхемы MAX1483).  
RB4-RB7 устанавливаем на ввод для организации адресного селектора.

```
CLRF PORTB      ;Initialize PORTB by setting output data latches
                ;(инициализация установкой защелок данных)
MOVLW 0xF2      ;Value used to initialize data direction
MOVWF TRISB
```

Запишем необходимые данные в регистр управления приемника USART – RCSTA (адрес 18h):

Бит 7 – 1 модуль включен  
Бит 6 – 0 8-битный прием  
Бит 5 – не имеет значения  
Бит 4 – 1 прием разрешен  
Бит 3 – не имеет значения  
Бит 2 – только на чтение  
Бит 1 – только на чтение  
Бит 0 – только на чтение

Операции записи побитовые:

```
BSF RCSTA, SPEN
BCF RCSTA, RX9
BSF RCSTA, CREN
```

Аналогично запишем данные в регистр передатчика USART – TXSTA (адрес 98h):

Бит 7 – не имеет значения  
Бит 6 – 0 8-битная передача  
Бит 5 – 1 разрешение передачи  
Бит 4 – 0 асинхронный режим работы  
Бит 3 – не используется  
Бит 2 – 1 высокоскоростной режим  
Бит 1 – только чтение  
Бит 0 – проверку четности мы не используем

## Хобби-электроникс 2. Умный дом.

```
BCF TXSTA, TX9  
BCF TXSTA, SYNC  
BSF TXSTA, BRGH
```

(BSF TXSTA, TXEN – эта команда появится, когда мы начнем передачу статуса)

Кроме того, необходимо задать скорость работы в регистре SPBRG:

Для первого варианта тактового генератора: (SYNC=0, BRGH=1, 16 мГц)

```
MOVLW 0x67  
MOVWF SPBRG
```

Для второго варианта тактового генератора: (SYNC=0, BRGH=1, 3,6 мГц)

```
MOVLW 0x16  
MOVWF SPBRG
```

В результате блок инициализации (для второго варианта) будет выглядеть следующим образом:

```
list p=16f628a  
#include p16f628a.inc
```

### *Инициализация модуля*

```
BCF STATUS, RP1      ; Выбор банка 0  
BCF STATUS, RP0
```

*Если посмотреть раздел организации памяти контроллера, то видно, что часть регистров, которые мы используем, находится в банке 0, а часть в банке 1. Если забыть переключиться, то команды не будут выполнены, а среда программирования MPLAB напомнит, выводя в окне Output на странице MPLAB SIM напоминания при трансляции программы. Это были первые грабли, на которые я часто наступал.*

```
CLRF PORTA          ; Настройка порта А
```

```
MOVLW 0x07  
MOVWF CMCON
```

```
BCF STATUS, RP1      ; Выбор банка 1  
BSF STATUS, RP0
```

```
MOVLW 0x80  
MOVWF TRISA
```

## Хобби-электроникс 2. Умный дом.

MOVLW 0xF6 ; Настройка порта В  
MOVWF TRISB

BCF STATUS, RP1 ; Выбор банка 0  
BCF STATUS, RP0  
CLRF PORTB

;Настройка приемо-передатчика USART

BSF RCSTA, SPEN ; Настройка приемника  
BCF RCSTA, RX9  
BSF RCSTA, CREN

BCF STATUS, RP1 ; Выбор банка 1  
BSF STATUS, RP0

BCF TXSTA, TX9 ; Настройка передатчика  
BCF TXSTA, SYNC  
BSF TXSTA, BRGH

MOVLW 0x16  
MOVWF SPBRG

BCF STATUS, RP1 ; Выбор банка 0  
BCF STATUS, RP0

CLRWF ; Считывание собственного адреса  
ADDWF PORTB, 0 ; Прочитаем порт В  
ANDLW 0xF0 ; Нам не нужны младшие биты  
MOVWF 0x20 ; Сохраним адрес в регистре 20h  
SWAPF 0x20, 1 ; Нам не нужна работа с младшими битами  
CALL adrsim ; Преобразуем его в символьный вид

*Чтобы сравнить адрес, задаваемый переключателем, с адресом в слове команды, нужно преобразовать их к единому виду. Здесь несколько возможностей, я использую одну из них, вы можете поступить иначе.*

; прочитаем из EEPROM в 30h - состояние реле

*Совсем не обязательно хранить текущее состояние реле в энергонезависимой памяти. Но это может быть полезно, с одной стороны, и это дает возможность разобраться с механизмом записи в энергонезависимую память, с другой стороны. Можно, например, изменить метод задания адреса. Использовать вместо переключателя для установки адреса программную установку адреса – запись адреса в энергонезависимую память контроллера. Так, в частности, устроены промышленные модули. В этом случае перед первым использованием модуля его переводят в режим установки адреса и программно*

## Хобби-электроникс 2. Умный дом.

*устанавливают этот адрес. Механизм записи состояния реле и адреса одинаков.*

```
BSF STATUS, RP0      ; Выбор банка 1
BCF STATUS, RP1
MOVLW 0x00
MOVWF EEADR          ; Адрес считываемого регистра
BSF EECON1, RD       ; Чтение
MOVF EEDATA,W        ; w=EEDATA

BCF STATUS, RP1      ; Выбор банка 0
BCF STATUS, RP0
MOVWF 0x30
COMF 0x30,1          ; Будем хранить в EEPROM в инверсном виде
CLRW
ADDWF 0x30,0         ; А в регистре 30h в прямом
```

; перепишем 30h в порт

```
MOVWF PORTA
```

*Сохранение состояния реле в инверсном виде связано с тем, что в EEPROM по умолчанию записаны единицы. При первой инициализации прочтение состояния приведет к включению всех реле. Чтобы этого не было, мы будем инвертировать прочитанное.*

В строке 36 (CALL adrsim) мы вызываем подпрограмму, которая переводит адреса в символы:

*Это опять не обязательно. Но оставлять в блоке инициализации этот длинный «хвост» мне не захотелось. Перевод адреса в символьное представление, я думаю, можно сделать различными способами, я выбрал самый, с моей точки зрения, очевидный.*

Поскольку мы используем символьный обмен, я выпишу шестнадцатеричные коды некоторых символов ASCII:

«0» - 30h; «1» - 31h; «2» - 32h; «3» - 33h; «4» - 34h; «5» - 35h и т.д.  
«R» - 52h; «\$» - 24h; «#» - 23h; «N» - 4Eh; «F» - 46h; «S» - 53h

### ***Подпрограмма перевода адреса в символ ( храним по адресам 21h, 22h)***

```
adrsim:  CLRW          ; Если адрес 1 запишем символы «0» «1» (30h и 31h)
          ADDLW 0x30
          MOVWF 0x21
          CLRW
          ADDLW 0x31
          MOVWF 0x22
          MOVF 0x20, 0
```

## Хобби-электроникс 2. Умный дом.

```
BCF STATUS, Z
XORLW 0x1
BTFSC STATUS, Z
RETURN
```

```
CLRW ; Если адрес 2 запишем символы «0» «2» (30h и 32h)
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x32
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x2
BTFSC STATUS, Z
RETURN
```

```
CLRW ; Если адрес 3 запишем символы «0» «3»
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x33
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x3
BTFSC STATUS, Z
RETURN
```

```
CLRW ; Если адрес 4 запишем символы «0» «4»
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x34
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x4
BTFSS STATUS, Z
RETURN
```

```
CLRW ; Если адрес 5 запишем символы «0» «5»
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x35
MOVWF 0x22
MOVF 0x20, 0
```

## Хобби-электроникс 2. Умный дом.

```
BCF STATUS, Z
XORLW 0x5
BTFSS STATUS, Z
RETURN
```

```
CLRW                ; Если адрес 6 запишем символы «0» «6»
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x36
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x6
BTFSS STATUS, Z
RETURN
```

```
CLRW                ; Если адрес 7 запишем символы «0» «7»
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x37
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x7
BTFSS STATUS, Z
RETURN
```

```
CLRW                ; Если адрес 8 запишем символы «0» «8»
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x38
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x8
BTFSS STATUS, Z
RETURN
```

```
CLRW                ; Если адрес 9 запишем символы «0» «9»
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x39
MOVWF 0x22
MOVF 0x20, 0
```

## Хобби-электроникс 2. Умный дом.

```
BCF STATUS, Z
XORLW 0x9
BTFSS STATUS, Z
RETURN
```

```
CLRW ; Если адрес 10 (Ah) запишем символы «1» «0»
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x30
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xA
BTFSS STATUS, Z
RETURN
```

```
CLRW ; Если адрес 11 (Bh) запишем символы «1» «1»
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x31
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xB
BTFSS STATUS, Z
RETURN
```

```
CLRW ; Если адрес 12 (Ch) запишем символы «1» «2»
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x32
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xC
BTFSS STATUS, Z
RETURN
```

```
CLRW ; Если адрес 13 (Dh) запишем символы «1» «3»
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x33
MOVWF 0x22
MOVF 0x20, 0
```

## Хобби-электроникс 2. Умный дом.

```
BCF STATUS, Z
XORLW 0xD
BTFSS STATUS, Z
RETURN
```

```
CLRW                ; Если адрес 14 (Eh) запишем символы «1» «4»
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x34
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xE
BTFSS STATUS, Z
RETURN
```

```
CLRW                ; Если адрес 15 (Fh) запишем символы «1» «5»
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x35
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xF
BTFSS STATUS, Z
RETURN
```

### *Программный блок ожидания активности в сети*

Проверим бит RCIF в регистре PIR1.  
Если бит установлен, пропустим следующую команду.  
Вызовем подпрограмму обработки команды.  
Вернемся к началу.

Вот вся, собственно, программа.

```
start: BTFSS PIR1, RCIF      ; Ждем прихода первого символа команды
      GOTO start
      CALL cmnd             ; С приходом первого символа начинаем обработку
      GOTO start
```

*Следующие грабли располагаются в этом районе, хотя я наступил на них при обработке in1. Разумная, как мне казалось, конструкция:*

```
start: BTFSS PIR1, RCIF      ; Ждем прихода первого символа команды
```

## Хобби-электроникс 2. Умный дом.

### GOTO start

*работать в режиме отладки MPLAB не хотела. Я перепроверил многократно, тот ли флаг, та ли проверка. Все было правильно, но анимация программы безнадежно застревала, я решил, что программа не работает. Понять, в чем здесь дело помогло обращение к сайту MicroChip, где на форуме этот вопрос уже обсуждался. Причина столь не очевидного поведения программы очевидна (когда знаешь). Отладчик в цикле опроса ждет прохождения нескольких тысяч тактов внутреннего генератора, работающего на частоте в 4-20 мГц, пока заполнится регистр, заполнение которого происходит с частотой, примерно 10 кГц. Вдобавок настройки отладчика по умолчанию делают анимацию хорошо воспринимаемой (одно или два перемещения в секунду), но дожидаться нескольких тысяч таких перемещений лично у меня ума не хватило.*

Обработку команды я опять оформил в виде подпрограммы.

### **Подпрограмма обработки команды.**

; Обработка команды - проверка адреса, определение команды

```
cmnd:      BCF STATUS, Z
           MOVF RCREG, 0
           XORLW 0x52           ; Проверим наш ли модуль R (52h)
           BTFSS STATUS, Z     ; Если нет вернемся
           RETURN

in1:       BTFSS PIR1, RCIF    ; Ждем прихода первого символа адреса
           GOTO in1           ; Если совпадает, продолжим
           MOVF RCREG, 0
           BCF STATUS, Z
           XORWF 0x21, 0       ; Первый символ адреса в регистре 21h.
           BTFSS STATUS, Z
           RETURN

in2:       BTFSS PIR1, RCIF    ; Ждем прихода второго символа адреса
           GOTO in2           ; Если совпадает, продолжим
           MOVF RCREG, 0
           BCF STATUS, Z
           XORWF 0x22, 0       ; Второй символ адреса в регистре 22h.
           BTFSS STATUS, Z
           RETURN

in3:       BTFSS PIR1, RCIF    ; Ждем прихода символа
           GOTO in3           ; Если следом символ команды, продолжим
           MOVF RCREG, 0
           BCF STATUS, Z
```

## Хобби-электроникс 2. Умный дом.

```
XORLW 0x24          ; Символ команды «$» (24h)
BTFSC STATUS, Z
CALL swtch          ; Вызываем подпрограмму выполнения
RETURN
```

; Подпрограмма определения команды N (включить), F (выключить) или S (состояние)

```
swtch:  CLRW
in4:    BTFSS PIR1, RCIF      ; Ждем прихода символа
        GOTO in4

        MOVF RCREG, 0        ; Прочитаем номер реле
        MOVWF 0x23          ; Сохраним номер реле в регистре 23h
        MOVLW 0x30          ; Запишем регистр 30h в аккумулятор
        SUBWF 0x23, 0        ; Переведем символ в номер
        MOVWF 0x23          ; Сохраним номер реле в регистре 23h

in5:    BTFSS PIR1, RCIF      ; Ждем прихода символа
        GOTO in5
        MOVF RCREG, 0        ; Разберем N-включить, F-выключить, S-статус
        MOVWF 0x24          ; Сохраним команду в регистре 24h
        BCF STATUS, Z
        XORLW 0x4E          ; Включение N (4Eh)
        BTFSC STATUS, Z      ; Если нет, то пропустим
        CALL cmdset
        MOVF 0x24, 0        ; Перепишем из 24h в аккумулятор
        BCF STATUS, Z
        XORLW 0x46          ; Выключение F (46h)
        BTFSC STATUS, Z      ; Если нет, то пропустим
        CALL cmdreset
        MOVF 0x24, 0        ; Перепишем из 24h в аккумулятор
        BCF STATUS, Z
        XORLW 0x53          ; S (53h) запрос статуса
        BTFSC STATUS, Z      ; Если нет, то пропустим
        CALL stat
```

; Сохраним состояние всех реле в энергонезависимой памяти

; запишем регистр 30h состояния реле в EEPROM

```
CLRW
MOVLW 0x00          ; Запишем 0h в аккумулятор
MOVWF EEADR        ; Запишем адрес 0h в регистр адреса
BCF STATUS, RP1    ; Выбор банка 0
BCF STATUS, RP0
CLRW
ADDWF 0x30, 0      ; Запишем содержимое 30h в аккумулятор
COMF 0x30, 0       ; Инvertируем перед сохранением
BSF STATUS, RP0    ; Выбор банка 1
```

## Хобби-электроникс 2. Умный дом.

```
BCF STATUS, RP1  
MOVWF EEDATA
```

```
BSF STATUS, RP0  
BCF STATUS, RP1           ; Выбор банка 1  
BSF EECON1, WREN         ; Разрешить запись  
BCF INTCON, GIE         ; Запретить прерывания  
MOVLW 0x55  
MOVWF EECON2             ; Записать 55h  
MOVLW 0xAA  
MOVWF EECON2             ; Записать AAh  
BSF EECON1, WR           ; Установить флаг для начала запись  
BSF INTCON, GIE         ; Разрешить прерывания  
BCF EECON1, WREN         ; Запретить запись
```

*; Запись 55h и AAh относятся к обязательным при работе с EEPROM.*

```
chkwr: BCF STATUS, RP1           ; Выбор банка 0  
        BCF STATUS, RP0  
        BTFSF PIR1, EEIF      ; Проверка завершения записи  
        GOTO chkwr  
        CLRW  
        ADDWF 0x30, 0  
        MOVWF PORTA          ; Перепишем 30h в порт  
        BCF PIR1, EEIF       ; Сбросим флаг  
        RETURN
```

*В этом месте работы с отладчиком, получилось так, что вначале я проверил выполнение двух команд подряд. Все работало. Затем я вписал запись состояния реле в EEPROM с проверкой окончания записи. И, естественно, обнаружил, что вторая команда перестала выполняться. Но теперь я быстрее сообразил, что запись занимает время, в течение которого успевает пройти необработанная часть команды. Я не стал «мудрствовать лукаво», вставив между двумя нужными командами третью, ненужную. Хочу еще заметить, что изготовитель микросхем советует осуществить проверку записи в EEPROM, которую следует вставить после проверки завершения записи. Я этого не сделал. Возможно, я не прав. После создания прототипа я собираюсь переделать программу, удалив из слова команды символьное представление номера модуля и номера реле, вероятно, сделаю программное задание адреса. Тогда и добавлю проверку правильности записи в EEPROM. Если решу, что это все следует сделать.*

Три подпрограммы выполнения команд.

; Подпрограмма включение реле по номеру  
; Запишем в регистр 30h (установить соответствующий номеру реле бит)

```
cmdset: CLRW
```

## Хобби-электроникс 2. Умный дом.

```
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x0           ; реле 0
BTFSC STATUS, Z
BSF 0x30, 0
CLRW
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x1           ; реле 1
BTFSC STATUS, Z
BSF 0x30, 1
CLRW
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x2           ; реле 2
BTFSC STATUS, Z
BSF 0x30, 2
CLRW
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x3           ; реле 3
BTFSC STATUS, Z
BSF 0x30, 3
CLRW
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x4           ; реле 4
BTFSC STATUS, Z
BSF 0x30, 4
CLRW
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x5           ; реле 5
BTFSC STATUS, Z
BSF 0x30, 5
CLRW
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x6           ; реле 6
BTFSC STATUS, Z
BSF 0x30, 6
CLRW
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x7           ; реле 7
BTFSC STATUS, Z
BSF 0x30, 7
RETURN
```

## Хобби-электроникс 2. Умный дом.

; Подпрограмма выключения реле по номеру  
; Запишем в регистр 30h (сбросить соответствующий номеру реле бит)

```
cmdreset:      CLRW
               ADDWF 0x23,0
               BCF STATUS, Z
               XORLW 0x0           ; реле 0
               BTFSC STATUS, Z
               BCF 0x30, 0
               CLRW
               ADDWF 0x23,0
               BCF STATUS, Z
               XORLW 0x1           ; реле 1
               BTFSC STATUS, Z
               BCF 0x30, 1
               CLRW
               ADDWF 0x23,0
               BCF STATUS, Z
               XORLW 0x2           ; реле 2
               BTFSC STATUS, Z
               BCF 0x30, 2
               CLRW
               ADDWF 0x23,0
               BCF STATUS, Z
               XORLW 0x3           ; реле 3
               BTFSC STATUS, Z
               BCF 0x30, 3
               CLRW
               ADDWF 0x23,0
               BCF STATUS, Z
               XORLW 0x4           ; реле 4
               BTFSC STATUS, Z
               BCF 0x30, 4
               CLRW
               ADDWF 0x23,0
               BCF STATUS, Z
               XORLW 0x5           ; реле 5
               BTFSC STATUS, Z
               BCF 0x30, 5
               CLRW
               ADDWF 0x23,0
               BCF STATUS, Z
               XORLW 0x6           ; реле 6
               BTFSC STATUS, Z
               BCF 0x30, 6
               CLRW
```

## Хобби-электроникс 2. Умный дом.

```
ADDWF 0x23,0
BCF STATUS, Z
XORLW 0x7 ; реле 7
BTFSC STATUS, Z
BCF 0x30, 7
RETURN
```

; Подпрограмма передачи статуса реле

```
stat:      BCF STATUS, RP1      ; Выбор банка 0
           BCF STATUS, RP0
           CLRW              ; Проверим реле 0
           ADDWF 0x23, 0
           BCF STATUS, Z
           XORLW 0x0          ; реле 0
           BTFSC STATUS, Z
           CALL rel0
           CLRW              ; Проверим реле 1
           ADDWF 0x23, 0
           BCF STATUS, Z
           XORLW 0x1          ; реле 1
           BTFSC STATUS, Z
           CALL rel1
           CLRW              ; Проверим реле 2
           ADDWF 0x23, 0
           BCF STATUS, Z
           XORLW 0x2          ; реле 2
           BTFSC STATUS, Z
           CALL rel2
           CLRW              ; Проверим реле 3
           ADDWF 0x23, 0
           BCF STATUS, Z
           XORLW 0x3          ; реле 3
           BTFSC STATUS, Z
           CALL rel3
           CLRW              ; Проверим реле 4
           ADDWF 0x23, 0
           BCF STATUS, Z
           XORLW 0x4          ; реле 4
           BTFSC STATUS, Z
           CALL rel4
           CLRW              ; Проверим реле 5
           ADDWF 0x23, 0
           BCF STATUS, Z
           XORLW 0x5          ; реле 5
           BTFSC STATUS, Z
           CALL rel5
           CLRW              ; Проверим реле 6
```

## Хобби-электроникс 2. Умный дом.

```
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x6           ; реле 6
BTFSC STATUS, Z
CALL rel6
CLRW                ; Проверим реле 7
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x7           ; реле 7
BTFSC STATUS, Z
CALL rel7
```

; теперь это все отправим в передатчик

```
BSF PORTB, 0        ; Переключим драйвер RS485 на передачу
BCF STATUS, RP1     ; Выбор банка 1
BSF STATUS, RP0
BSF TXSTA, TXEN     ; Разрешаем передачу
BCF STATUS, RP1     ; Выбор банка 0
BCF STATUS, RP0
MOVLW 0x52
MOVWF TXREG         ; Отправим символ модуля
BCF STATUS, RP1     ; Выбор банка 1
BSF STATUS, RP0
outdat0: BTFSS TXSTA, TXIF ; Ждем отправки символа
GOTO outdat0
BCF STATUS, RP1     ; Выбор банка 0
BCF STATUS, RP0
CLRW
ADDWF 0x21,0
MOVWF TXREG         ; Отправим первый символ адреса
BCF STATUS, RP1     ; Выбор банка 1
BSF STATUS, RP0
outdat1: BTFSS TXSTA, TXIF ; Ждем отправки символа
GOTO outdat1
BCF STATUS, RP1     ; Выбор банка 0
BCF STATUS, RP0
CLRW
ADDWF 0x22,0
MOVWF TXREG         ; Отправим второй символ адреса
BCF STATUS, RP1     ; Выбор банка 1
BSF STATUS, RP0
outdat2: BTFSS TXSTA, TXIF ; Ждем отправки символа
GOTO outdat2
BCF STATUS, RP1     ; Выбор банка 0
BCF STATUS, RP0
MOVLW 0x23
MOVWF TXREG         ; Отправим символ статуса
```

## Хобби-электроникс 2. Умный дом.

```
outdat3:    BCF STATUS, RP1      ; Выбор банка 1
            BSF STATUS, RP0
            BTFSS TXSTA, TXIF ; Ждем отправки символа
            GOTO outdat3
            BCF STATUS, RP1      ; Выбор банка 0
            BCF STATUS, RP0
            CLRW
            ADDWF 0x23,0
            ADDLW 0x30
            MOVWF TXREG          ; Отправим символ номера реле
            BCF STATUS, RP1      ; Выбор банка 1
            BSF STATUS, RP0
outdat4:    BTFSS TXSTA, TXIF ; Ждем отправки символа
            GOTO outdat4
            BCF STATUS, RP1      ; Выбор банка 0
            BCF STATUS, RP0
            CLRW
            ADDWF 0x25,0
            MOVWF TXREG          ; Отправим символ состояния реле
            BCF STATUS, RP1      ; Выбор банка 1
            BSF STATUS, RP0
outdat5:    BTFSS TXSTA, RCIF ; Ждем отправки символа
            GOTO outdat5
            BCF TXSTA, TXEN      ; Запрещаем передачу
            BCF STATUS, RP1      ; Выбор банка 0
            BCF STATUS, RP0
            BCF PORTB, 0         ; Переключим драйвер RS485 на прием
            RETURN
```

; Подпрограмма обработки статуса реле

```
rel0:      BTFSC 0x30, 0
            CALL sostn
            BTFSS 0x30, 0
            CALL sostf
            RETURN
rel1:      BTFSC 0x30, 1
            CALL sostn
            BTFSS 0x30, 1
            CALL sostf
            RETURN
rel2:      BTFSC 0x30, 2
            CALL sostn
            BTFSS 0x30, 2
            CALL sostf
            RETURN
rel3:      BTFSC 0x30, 3
            CALL sostn
```

## Хобби-электроникс 2. Умный дом.

```
                BTFSS 0x30, 3
                CALL sostf
                RETURN
rel4:           BTFSC 0x30, 4
                CALL sostn
                BTFSS 0x30, 4
                CALL sostf
                RETURN
rel5:           BTFSC 0x30, 5
                CALL sostn
                BTFSS 0x30, 5
                CALL sostf
                RETURN
rel6:           BTFSC 0x30, 6
                CALL sostn
                BTFSS 0x30, 6
                CALL sostf
                RETURN
rel7:           BTFSC 0x30, 7
                CALL sostn
                BTFSS 0x30, 7
                CALL sostf
                RETURN
```

; Подпрограммы образования символов состояний

```
sostn:         MOVLW 0x4E           ; Состояние - включено
                MOVWF 0x25
                RETURN
sostf:         MOVLW 0x46           ; Состояние - выключено
                MOVWF 0x25
                RETURN
```

Если теперь соединить все представленные части программы, то программирование контроллера завершено.

Вот полная программа:

```
list    p=16f628a
#include p16f628a.inc

; Инициализация модуля

BCF STATUS, RP1      ; Выбор банка 0
BCF STATUS, RP0
```

## Хобби-электроникс 2. Умный дом.

```
CLRF PORTA          ;Настройка порта А

MOVLW 0x07
MOVWF CMCON

BCF STATUS, RP1    ; Выбор банка 1
BSF STATUS, RP0

MOVLW 0x80
MOVWF TRISA

MOVLW 0xF6        ;Настройка порта В
MOVWF TRISB

BCF STATUS, RP1    ; Выбор банка 0
BCF STATUS, RP0
CLRF PORTB
    ;Настройка приемо-передатчика USART

BSF RCSTA, SPEN    ; Настройка приемника
BCF RCSTA, RX9
BSF RCSTA, CREN

BCF STATUS, RP1    ; Выбор банка 1
BSF STATUS, RP0

BCF TXSTA, TX9     ; Настройка передатчика
BCF TXSTA, SYNC
BSF TXSTA, BRGH

MOVLW 0x16
MOVWF SPBRG

BCF STATUS, RP1    ; Выбор банка 0
BCF STATUS, RP0

CLRW               ; Считывание собственного адреса
ADDWF PORTB, 0     ; Прочитаем порт В
ANDLW 0xF0         ; Нам не нужны младшие биты
MOVWF 0x20         ; Сохраним адрес в 20h
SWAPF 0x20, 1     ; Нам не нужна работа с младшими битами
CALL adrsim       ; Преобразуем его в символьный вид

    ; прочитаем из EEPROM 30h - состояние реле

BSF STATUS, RP0    ; Выбор банка 1
BCF STATUS, RP1
MOVLW 0x00
```

## Хобби-электроникс 2. Умный дом.

```
MOVWF EEADR      ; Адрес считываемого регистра
BSF EECON1, RD   ; Чтение
MOVF EEDATA,W    ; w=EEDATA
```

```
BCF STATUS, RP1  ; Выбор банка 0
BCF STATUS, RP0
MOVWF 0x30
COMF 0x30,1      ; Будем хранить в EEPROM в инверсном виде
CLRWF
ADDWF 0x30,0     ; A в регистре 30h в прямом
```

; перепишем 30h в порт

```
MOVWF PORTA
```

; Начало работы, ожидание команды, отработка первой команды

```
CLRWF            ; Очистим аккумулятор, ждем команды по USART
```

```
start: BTFSS PIR1, RCIF ; Ждем прихода первого символа команды
        GOTO start
        BCF STATUS, RP1 ; Выбор банка 0
        BCF STATUS, RP0
        CALL cmnd       ; С приходом первого символа начинаем обработку
        GOTO start
        NOP
```

; Обработка команды - проверка адреса, определение команды

```
cmnd: BCF STATUS, Z
        MOVF RCREG, 0
        XORLW 0x52     ; Проверим наш ли модуль R (52h)
        BTFSS STATUS, Z ; Если нет, вернемся
        RETURN
```

```
in1: BTFSS PIR1, RCIF ; Ждем прихода первого символа адреса
        GOTO in1      ; Если совпадает, продолжим
        MOVF RCREG, 0
        BCF STATUS, Z
        XORWF 0x21, 0 ; Первый символ адреса
        BTFSS STATUS, Z
        RETURN
```

```
in2: BTFSS PIR1, RCIF ; Ждем прихода второго символа адреса
        GOTO in2      ; Если совпадает, продолжим
        MOVF RCREG, 0
        BCF STATUS, Z
        XORWF 0x22, 0 ; Второй символ адреса
        BTFSS STATUS, Z
```

## Хобби-электроникс 2. Умный дом.

RETURN

```
in3:  BTFSS PIR1, RCIF      ; Ждем прихода символа
      GOTO in3             ; Если за адресом следует символ команды, продолжим
      MOVF RCREG, 0
      BCF STATUS, Z
      XORLW 0x24           ; Выполнение $(24h)
      BTFSC STATUS, Z
      CALL swtch          ; Вызываем подпрограмму выполнения
      RETURN
```

;Подпрограмма перевода адреса в символы, их храним в 21h, 22h

```
adrsim:  CLRW              ; Если адрес 1 запишем символы 01 (30h и 31h)
         ADDLW 0x30
         MOVWF 0x21
         CLRW
         ADDLW 0x31
         MOVWF 0x22
         MOVF 0x20, 0
         BCF STATUS, Z
         XORLW 0x1
         BTFSC STATUS, Z
         RETURN
```

```
         CLRW              ; Если адрес 2 запишем символы 02 (30h и 32h)
         ADDLW 0x30
         MOVWF 0x21
         CLRW
         ADDLW 0x32
         MOVWF 0x22
         MOVF 0x20, 0
         BCF STATUS, Z
         XORLW 0x2
         BTFSC STATUS, Z
         RETURN
```

```
         CLRW              ; Если адрес 3 запишем символы 03
         ADDLW 0x30
         MOVWF 0x21
         CLRW
         ADDLW 0x33
         MOVWF 0x22
         MOVF 0x20, 0
         BCF STATUS, Z
         XORLW 0x3
         BTFSC STATUS, Z
         RETURN
```

```
CLRW ; Если адрес 4 запишем символы 04
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x34
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x4
BTSS STATUS, Z
RETURN
```

```
CLRW ; Если адрес 5 запишем символы 05
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x35
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x5
BTSS STATUS, Z
RETURN
```

```
CLRW ; Если адрес 6 запишем символы 06
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x36
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x6
BTSS STATUS, Z
RETURN
```

```
CLRW ; Если адрес 7 запишем символы 07
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x37
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x7
BTSS STATUS, Z
RETURN
```

## Хобби-электроникс 2. Умный дом.

```
CLRW           ; Если адрес 8 запишем символы 08
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x38
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x8
BTSS STATUS, Z
RETURN
```

```
CLRW           ; Если адрес 9 запишем символы 09
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x39
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x9
BTSS STATUS, Z
RETURN
```

```
CLRW           ; Если адрес 10 (A) запишем символы 10
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x30
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xA
BTSS STATUS, Z
RETURN
```

```
CLRW           ; Если адрес 11 запишем символы 11
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x31
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xB
BTSS STATUS, Z
RETURN
```

```
CLRW ; Если адрес 12 запишем символы 12
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x32
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xC
BTSS STATUS, Z
RETURN
```

```
CLRW ; Если адрес 13 запишем символы 13
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x33
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xD
BTSS STATUS, Z
RETURN
```

```
CLRW ; Если адрес 14 запишем символы 14
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x34
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xE
BTSS STATUS, Z
RETURN
```

```
CLRW ; Если адрес 15 запишем символы 15
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x35
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xF
BTSS STATUS, Z
RETURN
```

## Хобби-электроникс 2. Умный дом.

; Подпрограмма включение реле по номеру

```
cmdset:      CLRW                ; Запишем в 30h (установить бит)
              ADDWF 0x23, 0
              BCF STATUS, Z
              XORLW 0x0          ; реле 0
              BTFSC STATUS, Z
              BSF 0x30, 0
              CLRW
              ADDWF 0x23, 0
              BCF STATUS, Z
              XORLW 0x1          ; реле 1
              BTFSC STATUS, Z
              BSF 0x30, 1
              CLRW
              ADDWF 0x23, 0
              BCF STATUS, Z
              XORLW 0x2          ; реле 2
              BTFSC STATUS, Z
              BSF 0x30, 2
              CLRW
              ADDWF 0x23, 0
              BCF STATUS, Z
              XORLW 0x3          ; реле 3
              BTFSC STATUS, Z
              BSF 0x30, 3
              CLRW
              ADDWF 0x23, 0
              BCF STATUS, Z
              XORLW 0x4          ; реле 4
              BTFSC STATUS, Z
              BSF 0x30, 4
              CLRW
              ADDWF 0x23, 0
              BCF STATUS, Z
              XORLW 0x5          ; реле 5
              BTFSC STATUS, Z
              BSF 0x30, 5
              CLRW
              ADDWF 0x23, 0
              BCF STATUS, Z
              XORLW 0x6          ; реле 6
              BTFSC STATUS, Z
              BSF 0x30, 6
              CLRW
              ADDWF 0x23, 0
              BCF STATUS, Z
```

## Хобби-электроникс 2. Умный дом.

```
XORLW 0x7           ; реле 7
BTFSC STATUS, Z
BSF 0x30, 7
RETURN
```

; Подпрограмма выключения реле по номеру

```
cmdreset:  CLRW ; запишем в 30h (сбросить бит)
            ADDWF 0x23,0
            BCF STATUS, Z
            XORLW 0x0           ; реле 0
            BTFSC STATUS, Z
            BCF 0x30, 0
            CLRW
            ADDWF 0x23,0
            BCF STATUS, Z
            XORLW 0x1           ; реле 1
            BTFSC STATUS, Z
            BCF 0x30, 1
            CLRW
            ADDWF 0x23,0
            BCF STATUS, Z
            XORLW 0x2           ; реле 2
            BTFSC STATUS, Z
            BCF 0x30, 2
            CLRW
            ADDWF 0x23,0
            BCF STATUS, Z
            XORLW 0x3           ; реле 3
            BTFSC STATUS, Z
            BCF 0x30, 3
            CLRW
            ADDWF 0x23,0
            BCF STATUS, Z
            XORLW 0x4           ; реле 4
            BTFSC STATUS, Z
            BCF 0x30, 4
            CLRW
            ADDWF 0x23,0
            BCF STATUS, Z
            XORLW 0x5           ; реле 5
            BTFSC STATUS, Z
            BCF 0x30, 5
            CLRW
            ADDWF 0x23,0
            BCF STATUS, Z
            XORLW 0x6           ; реле 6
            BTFSC STATUS, Z
```

## Хобби-электроникс 2. Умный дом.

```
BCF 0x30, 6
CLRWF
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x7           ; реле 7
BTFSC STATUS, Z
BCF 0x30, 7
RETURN
```

### *Подпрограмма определения команды N (включить), F (выключить) или S (состояние)*

```
swtch: CLRWF
in4:   BTFSS PIR1, RCIF           ; Ждем прихода символа
      GOTO in4

      MOVF RCREG, 0              ; Прочитаем номер реле
      MOVWF 0x23                 ; Сохраним номер реле в 23h
      MOVLW 0x30                 ; Запишем 30h в аккумулятор
      SUBWF 0x23, 0             ; Переведем символ в номер
      MOVWF 0x23                 ; Сохраним номер реле в 23h

in5:   BTFSS PIR1, RCIF           ; Ждем прихода символа
      GOTO in5
      MOVF RCREG, 0              ; Прочитаем команду N-включить F-выключить
      MOVWF 0x24                 ; Сохраним команду в 24h
      BCF STATUS, Z
      XORLW 0x4E                 ; Включение N (4Eh)
      BTFSC STATUS, Z           ; Если не включить, то пропустить
      CALL cmdset
      MOVF 0x24, 0              ; Перепишем из 24h в аккумулятор
      BCF STATUS, Z
      XORLW 0x46                 ; Выключение F (46h)
      BTFSC STATUS, Z           ; Если не выключить, то пропустить
      CALL cmdreset
      MOVF 0x24, 0              ; Перепишем из 24h в аккумулятор
      BCF STATUS, Z
      XORLW 0x53                 ; S (53h) запрос статуса
      BTFSC STATUS, Z           ; Если не запрос статуса, то пропустить
      CALL stat
```

; Сохраним состояние всех реле в энергонезависимой памяти

```
CLRWF           ; Запишем 30h - состояние реле в EEPROM
BSF STATUS, RP0 ; Выбор банка 1
BCF STATUS, RP1
MOVLW 0x00      ; Запишем 0h в аккумулятор
MOVWF EEADR     ; Запишем адрес 0h в регистр адреса
BCF STATUS, RP1 ; Выбор банка 0
```

## Хобби-электроникс 2. Умный дом.

```
BCF STATUS, RP0
CLRWF
ADDWF 0x30,0           ; Запишем содержимое 30h в аккумулятор
COMF 0x30,0           ; Инvertируем перед сохранением
BSF STATUS, RP0       ; Выбор банка 1
BCF STATUS, RP1
MOVWF EEDATA
BSF EECON1, WREN      ; Разрешить запись
BCF INTCON, GIE       ; Запретить прерывания
MOVLW 0x55
MOVWF EECON2          ; Записать 55h
MOVLW 0xAA
MOVWF EECON2          ; Записать AAh
BSF EECON1, WR        ; Установить флаг для начала запись
BSF INTCON, GIE       ; Разрешить прерывания
BCF EECON1, WREN      ; Запретить запись

BCF STATUS, RP1       ; Выбор банка 0
BCF STATUS, RP0

chkwr: BTFSS PIR1, EEIF ; Проверка завершения записи
GOTO chkwr
CLRWF
ADDWF 0x30, 0
MOVWF PORTA           ; Перепишем 30h в порт
BCF PIR1, EEIF        ; Сбросим флаг
RETURN

; Подпрограмма передачи статуса реле

stat:  BCF STATUS, RP1   ; Выбор банка 0
BCF STATUS, RP0
CLRWF           ; Проверим реле 0
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x0       ; Реле 0
BTFSC STATUS, Z
CALL rel0
CLRWF           ; Проверим реле 1
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x1       ; Реле 1
BTFSC STATUS, Z
CALL rel1
CLRWF           ; Проверим реле 2
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x2       ; реле 2
```

## Хобби-электроникс 2. Умный дом.

```
BTFSK STATUS, Z
CALL rel2
CLRW                ; Проверим реле 3
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x3           ; реле 3
BTFSK STATUS, Z
CALL rel3
CLRW                ; Проверим реле 4
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x4           ; реле 4
BTFSK STATUS, Z
CALL rel4
CLRW                ; Проверим реле 5
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x5           ; реле 5
BTFSK STATUS, Z
CALL rel5
CLRW                ; Проверим реле 6
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x6           ; реле 6
BTFSK STATUS, Z
CALL rel6
CLRW                ; Проверим реле 7
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x7           ; реле 7
BTFSK STATUS, Z
CALL rel7
```

; теперь это все отправим в передатчик

```
BSF PORTB, 0        ; Переключим драйвер RS485 на передачу
BCF STATUS, RP1     ; Выбор банка 1
BSF STATUS, RP0
BSF TXSTA, TXEN     ; Разрешаем передачу
BCF STATUS, RP1     ; Выбор банка 0
BCF STATUS, RP0
MOVLW 0x52
MOVWF TXREG         ; Отправим символ модуля
BCF STATUS, RP1     ; Выбор банка 1
BSF STATUS, RP0
outdat0: BTFSK TXSTA, TXIF ; Ждем отправки символа
GOTO outdat0
BCF STATUS, RP1     ; Выбор банка 0
```

## Хобби-электроникс 2. Умный дом.

```
BCF STATUS, RP0
CLRW
ADDWF 0x21,0
MOVWF TXREG           ; Отправим первый символ адреса
BCF STATUS, RP1       ; Выбор банка 1
BSF STATUS, RP0
outdat1: BTFSS TXSTA, TXIF ; Ждем отправки символа
GOTO outdat1
BCF STATUS, RP1       ; Выбор банка 0
BCF STATUS, RP0
CLRW
ADDWF 0x22,0
MOVWF TXREG           ; Отправим второй символ адреса
BCF STATUS, RP1       ; Выбор банка 1
BSF STATUS, RP0
outdat2: BTFSS TXSTA, TXIF ; Ждем отправки символа
GOTO outdat2
BCF STATUS, RP1       ; Выбор банка 0
BCF STATUS, RP0
MOVLW 0x23
MOVWF TXREG           ; Отправим символ статуса
BCF STATUS, RP1       ; Выбор банка 1
BSF STATUS, RP0
outdat3: BTFSS TXSTA, TXIF ; Ждем отправки символа
GOTO outdat3
BCF STATUS, RP1       ; Выбор банка 0
BCF STATUS, RP0
CLRW
ADDWF 0x23,0
ADDLW 0x30
MOVWF TXREG           ; Отправим символ номера реле
BCF STATUS, RP1       ; Выбор банка 1
BSF STATUS, RP0
outdat4: BTFSS TXSTA, TXIF ; Ждем отправки символа
GOTO outdat4
BCF STATUS, RP1       ; Выбор банка 0
BCF STATUS, RP0
CLRW
ADDWF 0x25,0
MOVWF TXREG           ; Отправим символ состояния реле
BCF STATUS, RP1       ; Выбор банка 1
BSF STATUS, RP0
outdat5: BTFSS TXSTA, RCIF ; Ждем отправки символа
GOTO outdat5
BCF TXSTA, TXEN       ; Запрещаем передачу
BCF STATUS, RP1       ; Выбор банка 0
BCF STATUS, RP0
BCF PORTB, 0          ; Переключим драйвер RS485 на прием
```

## Хобби-электроникс 2. Умный дом.

RETURN

### *Подпрограмма обработки статуса реле*

```
rel0:  BTFSC 0x30, 0
        CALL sostn
        BTFSS 0x30, 0
        CALL sostf
        RETURN
rel1:  BTFSC 0x30, 1
        CALL sostn
        BTFSS 0x30, 1
        CALL sostf
        RETURN
rel2:  BTFSC 0x30, 2
        CALL sostn
        BTFSS 0x30, 2
        CALL sostf
        RETURN
rel3:  BTFSC 0x30, 3
        CALL sostn
        BTFSS 0x30, 3
        CALL sostf
        RETURN
rel4:  BTFSC 0x30, 4
        CALL sostn
        BTFSS 0x30, 4
        CALL sostf
        RETURN
rel5:  BTFSC 0x30, 5
        CALL sostn
        BTFSS 0x30, 5
        CALL sostf
        RETURN
rel6:  BTFSC 0x30, 6
        CALL sostn
        BTFSS 0x30, 6
        CALL sostf
        RETURN
rel7:  BTFSC 0x30, 7
        CALL sostn
        BTFSS 0x30, 7
        CALL sostf
        RETURN
```

; Подпрограммы образования символов состояний

sostn: MOVLW 0x4E ; Состояние - включено

Хобби-электроникс 2. Умный дом.

```
    MOVWF 0x25
    RETURN
sostf: MOVLW 0x46          ; Состояние - выключено
    MOVWF 0x25
    RETURN
```

END

Мы написали программу для контроллера. Постараемся ее проверить и отладить в программе MPLAB.

## Основы работы в среде MPLAB

После загрузки программы появляется рабочее окно.

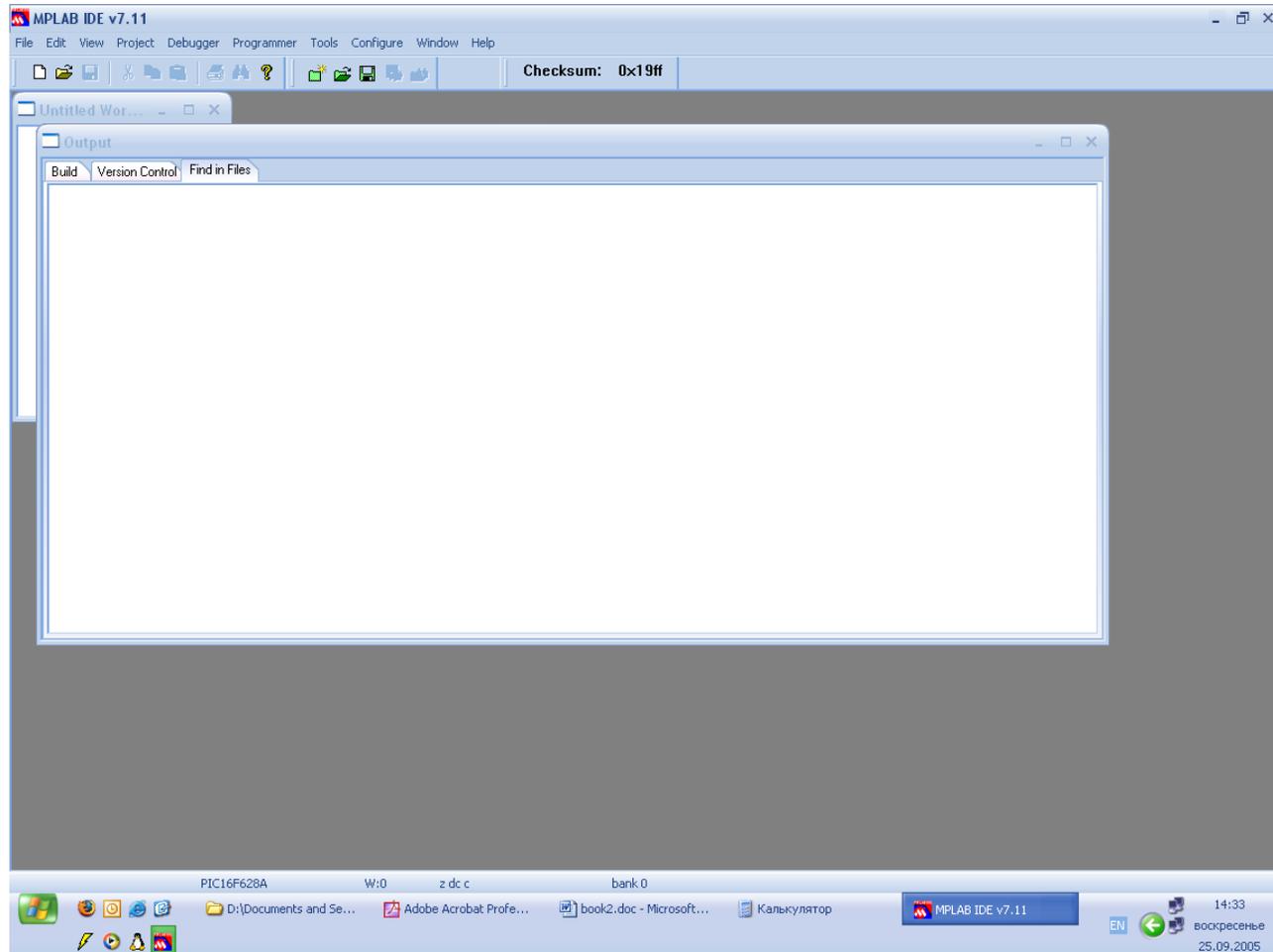


Рис.41

Вид программы обычен для Windows, и, думаю, не требует особых пояснений. Первое, что мы сделаем, создадим новый проект, в основном меню Project-New. Задаем название проекту relay, в папке, например, Relay, которую я советую открыть в основном разделе диска в папке MPLAB. Неоднократно я сталкивался с проблемой, которая не всегда очевидна. Многие программы, да это и удобно, предлагают хранить проект в папке «Мои документы». Проблема не возникает, если вы пользуетесь англоязычной версией Windows или русскоязычной версией программы. Но многие специализированные англоязычные программы начинают вытворять чудеса, если вы работаете в русскоязычной версии операционной системы. Впервые я столкнулся с этим, когда одна из сред программирования при компиляции программы стала выдавать ошибку в строке -1. Что она имела в виду под строкой с отрицательным номером, я не знаю. Но отыскать ошибку в правильно написанной программе оказалось не так просто. Ошибка крылась в том, что программа, предлагая через операционную систему сохранить проект в папке «Мои документы», эту папку распознать не могла.

## Хобби-электроникс 2. Умный дом.

После создания нового проекта появляется окно навигатор проекта. Теперь выберем микросхему контроллера в разделе основного меню Configure-Select Device. Выбираем PIC16F628A.

Выглядит это, примерно, так:

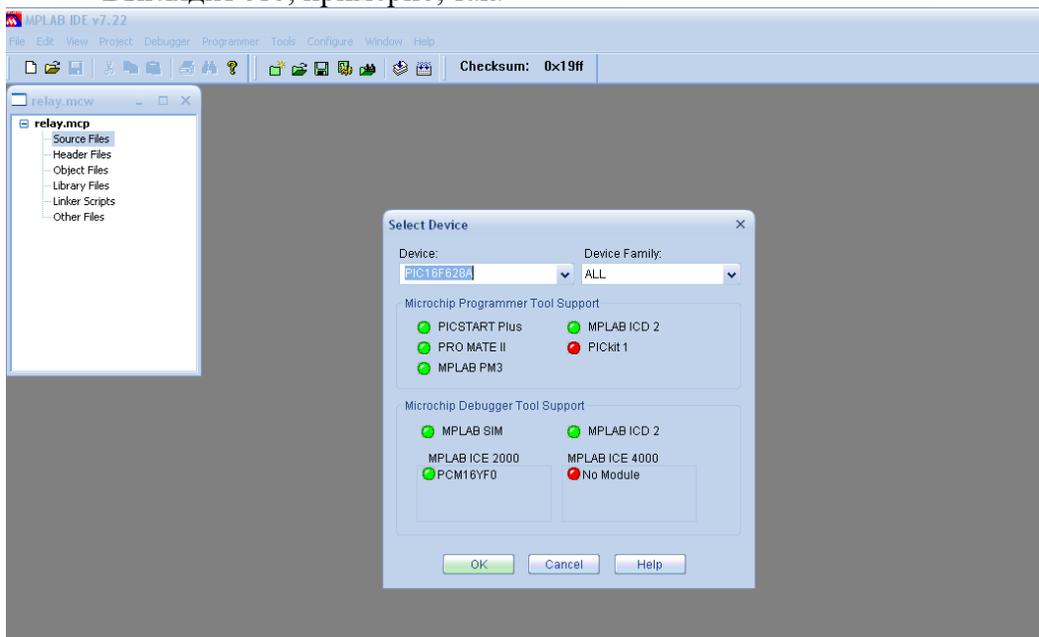


Рис.42

Завершив выбор, следует создать файл основной программы. Выбираем File-New. Появляется окно редактора. Сохраним файл под именем relay.asm (File-Save As).

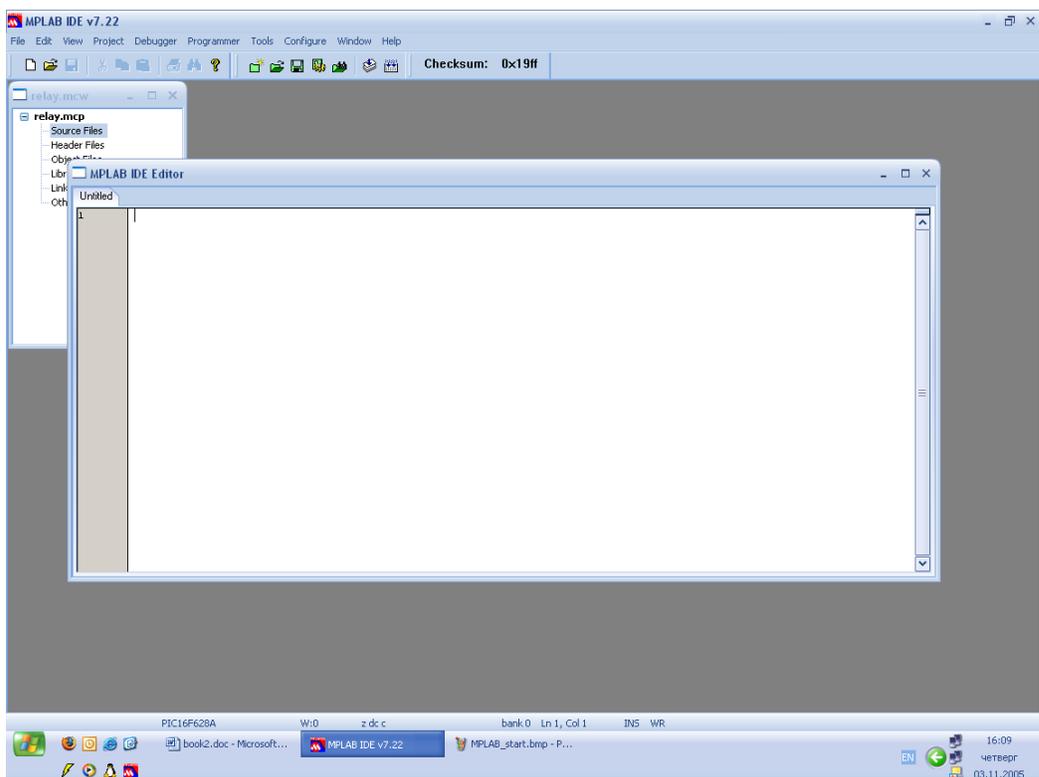


Рис.43

## Хобби-электроникс 2. Умный дом.

Далее добавим этот файл в проект. Для этого щелкнем правой кнопкой мыши по разделу Source Files в окне навигатора. В открывшемся меню выбираем Add Files, и указываем свой файл.

Для начала перенесем в окно редактора (прямым копированием из текста), небольшой фрагмент программы: инициализация, плюс сама программа, плюс две подпрограммы CALL adrsim и CALL cmnd. Последние в усеченном виде. Адрес будем использовать в данный момент «01». Не забудем поставить END в конце программы.

```
adrsim:    CLRW                ; Если адрес 1 запишем символы «0» «1» (30h и 31h)
           ADDLW 0x30
           MOVWF 0x21
           CLRW
           ADDLW 0x31
           MOVWF 0x22
           MOVF 0x20, 0
           BCF STATUS, Z
           XORLW 0x1
           BTFSC STATUS, Z
           RETURN

cmnd:     BCF STATUS, Z
           MOVF RCREG, 0
           XORLW 0x52        ; Проверим наш ли модуль R (52h)
           BTFSS STATUS, Z  ; Если нет, вернемся
           RETURN

in1:     BTFSS PIR1, RCIF   ; Ждем прихода первого символа адреса
           GOTO in1        ; Если совпадает, продолжим
           MOVF RCREG, 0
           BCF STATUS, Z
           XORWF 0x21, 0    ; Первый символ адреса, запомненный в регистре 21h.
           BTFSS STATUS, Z
           RETURN

in2:     BTFSS PIR1, RCIF   ; Ждем прихода второго символа адреса
           GOTO in2        ; Если совпадает, продолжим
           MOVF RCREG, 0
           BCF STATUS, Z
           XORWF 0x22, 0    ; Второй символ адреса, запомненный в регистре 22h.
           BTFSS STATUS, Z
           RETURN
```

Для отладки откроем окно наблюдения View-Watch, в котором выберем регистры STATUS, WREG, PIR1, EEDATA, RCREG, 20h, 21h, 22h, 30h. Необходимые регистры открываются кнопкой с обозначением стрелки вниз, правее названия регистра, которое, в свою очередь, рядом с кнопочкой ADD SFR. Ее следует нажать после выбора регистра.

## Хобби-электроникс 2. Умный дом.

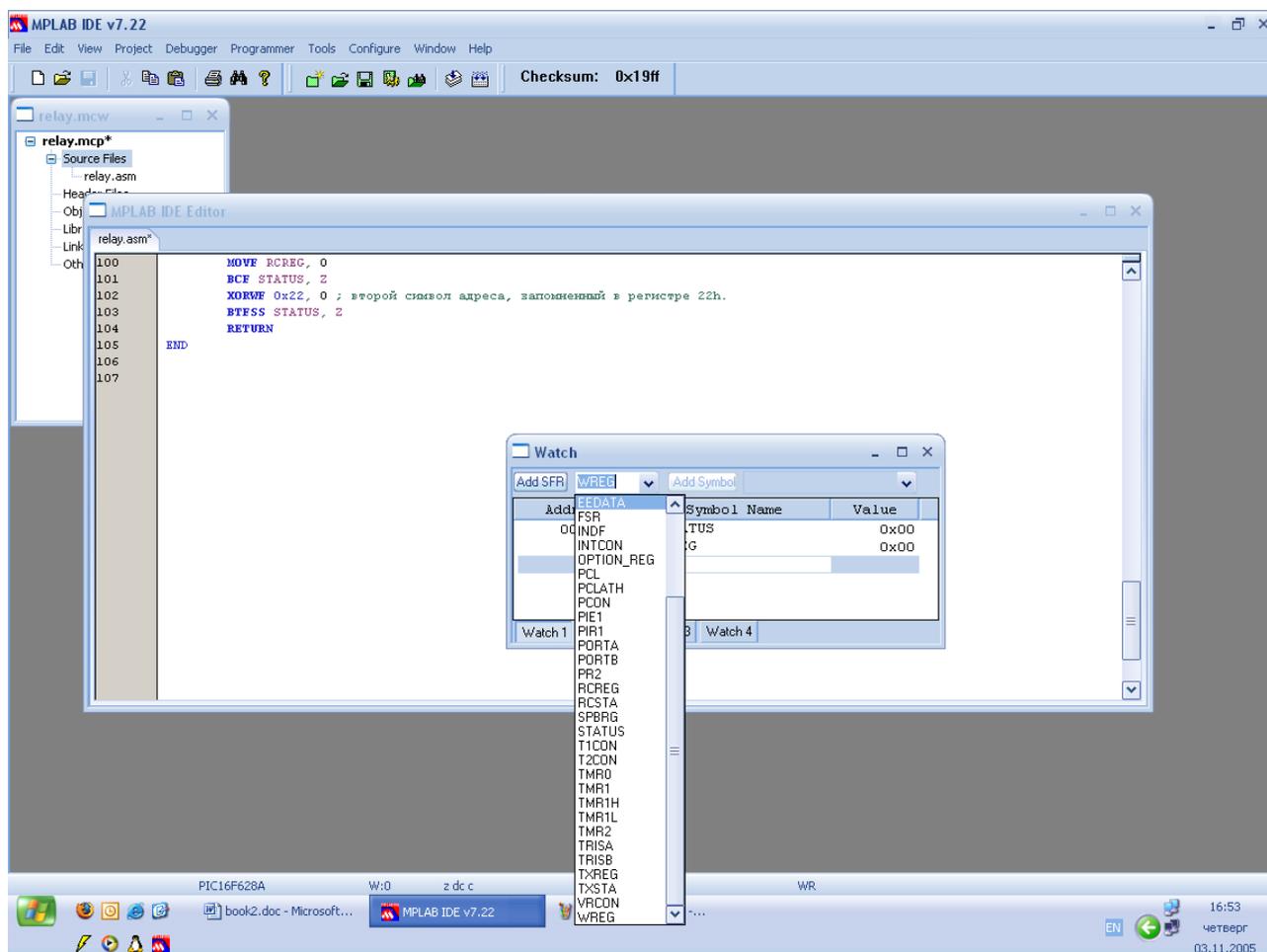


Рис.44

Регистры без имени (20h, 21h и т.д.) мы добавляем, просто вводя адрес в колонку Address на новой строке.

Добавим и EEPROM через выбор View - EEPROM. Установим в качестве симулятора программный (Debugger-Select Tool-MPLAB SIM). Создадим два файла input.txt и output.txt (File-New). В файле input.txt, который открывается в окне редактора, запишем слово команды "R01\$" в виде строки в кавычках. Сохраним этот файл (File-Save). Теперь сделаем установки отладчика (Debugger-Settings...): частоту процессора зададим 4 мГц, на вкладке Uart1 IO установим опцию Enable Uart1 IO, укажем входной файл input.txt (browse) и выходной output.txt. Подтвердим замену последнего файла, и установим опцию Rewind Input. Изменим значение на вкладке Animation/Realtime на 1 мс. Все это будет выглядеть, как на рисунке 24. Нажмем «Применить» и «ОК». Включим в наши окна View-Output. Создадим новый сценарий, который позволит нам ввести адрес, имитируя переключатель. Для этого выберем в основном меню Debugger-Stimulus Controller-New Scenario. Выберем нужный нам RB4 в окне Pin/SFR. В окне Action выберем Set High. Нажмем кнопку со стрелкой вправо рядом с этим в колонке Fire. Сохраним сценарий под именем relay и свернем его (не покинем, а свернем!). Теперь откроем окно (Configure-Configuration Bits), где установим биты, записываемые по адресу 2007h. Слово должно получиться 3F1Ah. Закроем это окно.

## Хобби-электроникс 2. Умный дом.

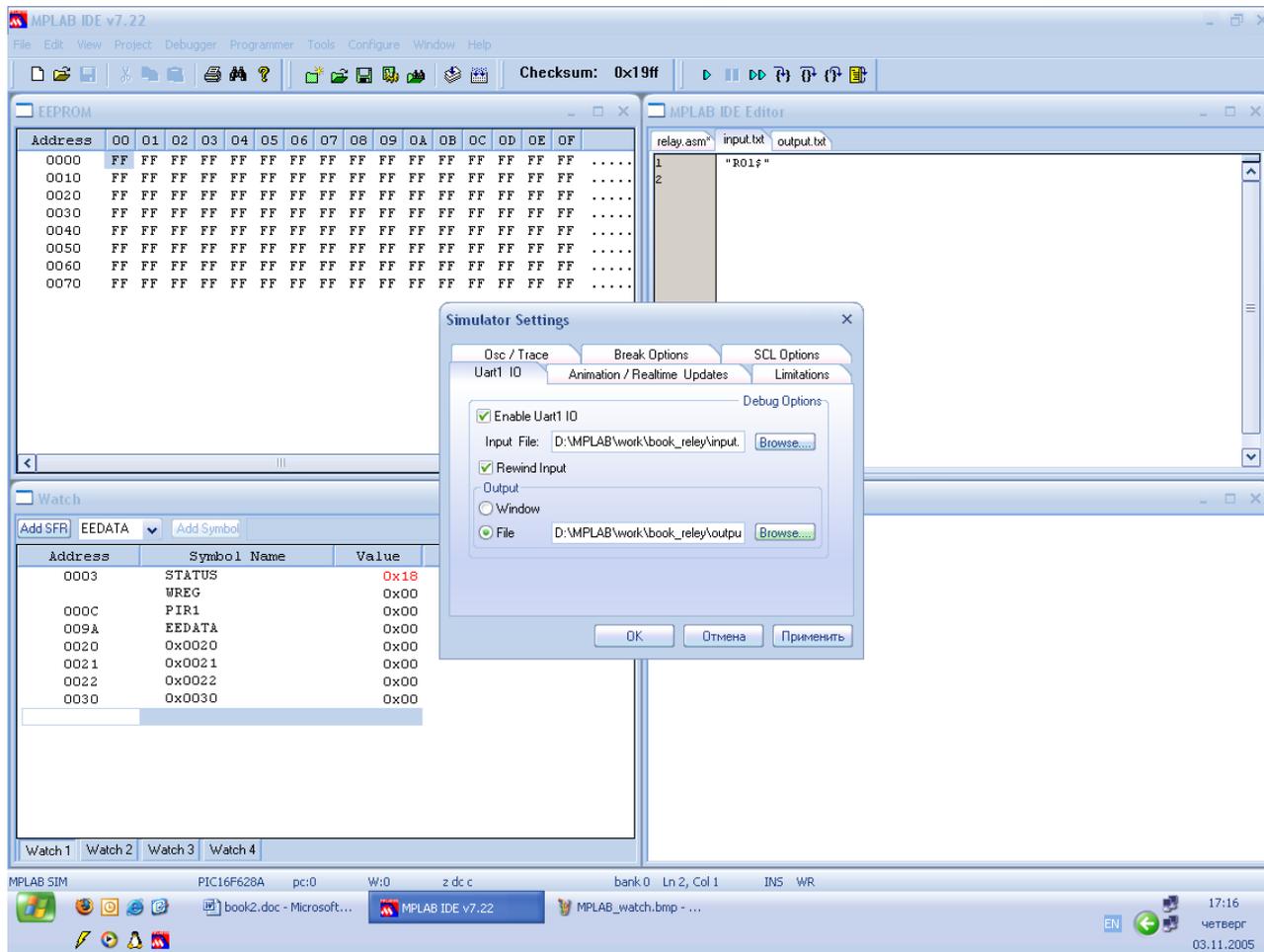


Рис.45

В завершение упорядочим окна (Windows-Tile Horizontally), сохраним все (File-Save All), и добавим в проект файл todo.txt, который предварительно создадим, а в менеджере проекта поместим его в раздел Other Files. В файле todo.txt будем вести план работы. Теперь откомпилируем проект Project-Build All. Сохраним и вид проекта File –Save Workspace. Мы готовы к отладке программы.

*После выхода из программы MPLAB, и подтверждения сохранения вида проекта, новой загрузки программы и открытия проекта (Project-Open) мне приходится обнулять адрес 0h в EEPROM, нажимать Fire в Scenario, и вводить адреса 20h, 21h и т.д. в окне Watch, которые программа пишет Not Found. Я не уверен, что дернул за все веревочки, но...*

Если вы правильно перенесли программу, то, нажав на инструментальной панели кнопку обозначенную ►► вы увидите анимацию, а в регистре RCREG (приемный регистр USART) появятся шестнадцатеричные коды символов из строки файла input.txt. Можно вписать в ячейку по адресу 0h EEPROM значение 1h, и увидеть, как она переписывается в регистр 30h.

*Есть еще несколько полезных возможностей. Одна из них проверить работу с некоторого места. Для этого остановим анимацию (кнопочкой |||),*

## Хобби-электроникс 2. Умный дом.

*установим курсор к нужной строке и нажмем правую клавишу мышки. Выберем в выпадающем меню Set PC at Cursor. Теперь, нажимая значок Step Into{↵}, мы можем отследить все изменения.*

В окончательном виде я работаю в среде, которая выглядит так:

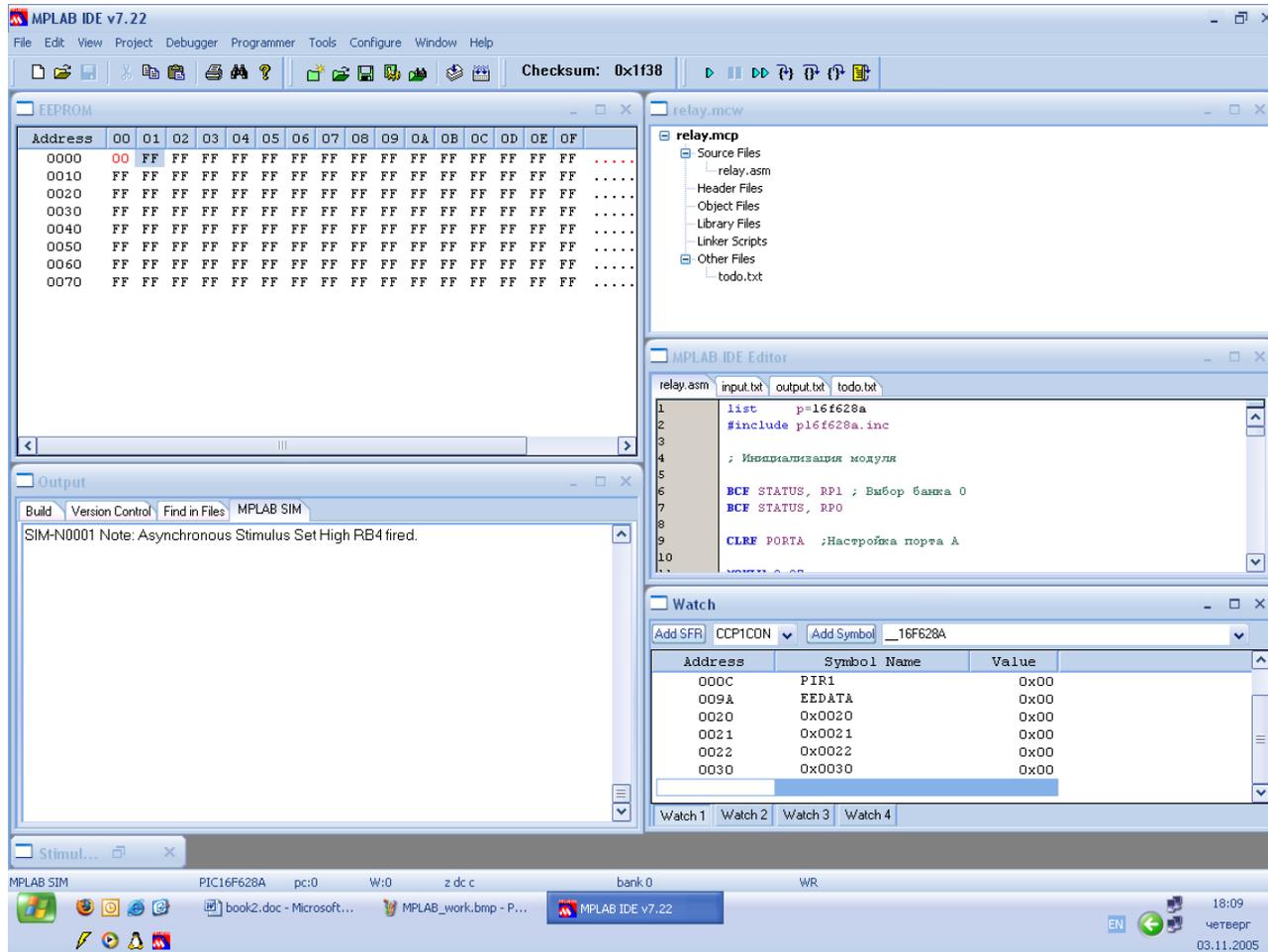


Рис.46

Кроме написания программы на ассемблере, среда MPLAB поддерживает написание программ на языке «С». Существуют компиляторы разных производителей. Я использую демонстрационную версию кросс-компилятора Hi-Tech. Посмотрим, не будет ли проще написать предыдущую программу на языке «С»?

## ***Релейный модуль, версия программы на языке «С»***

Выбор языка программирования происходит при задании в Project-Select Language Toolsuite:

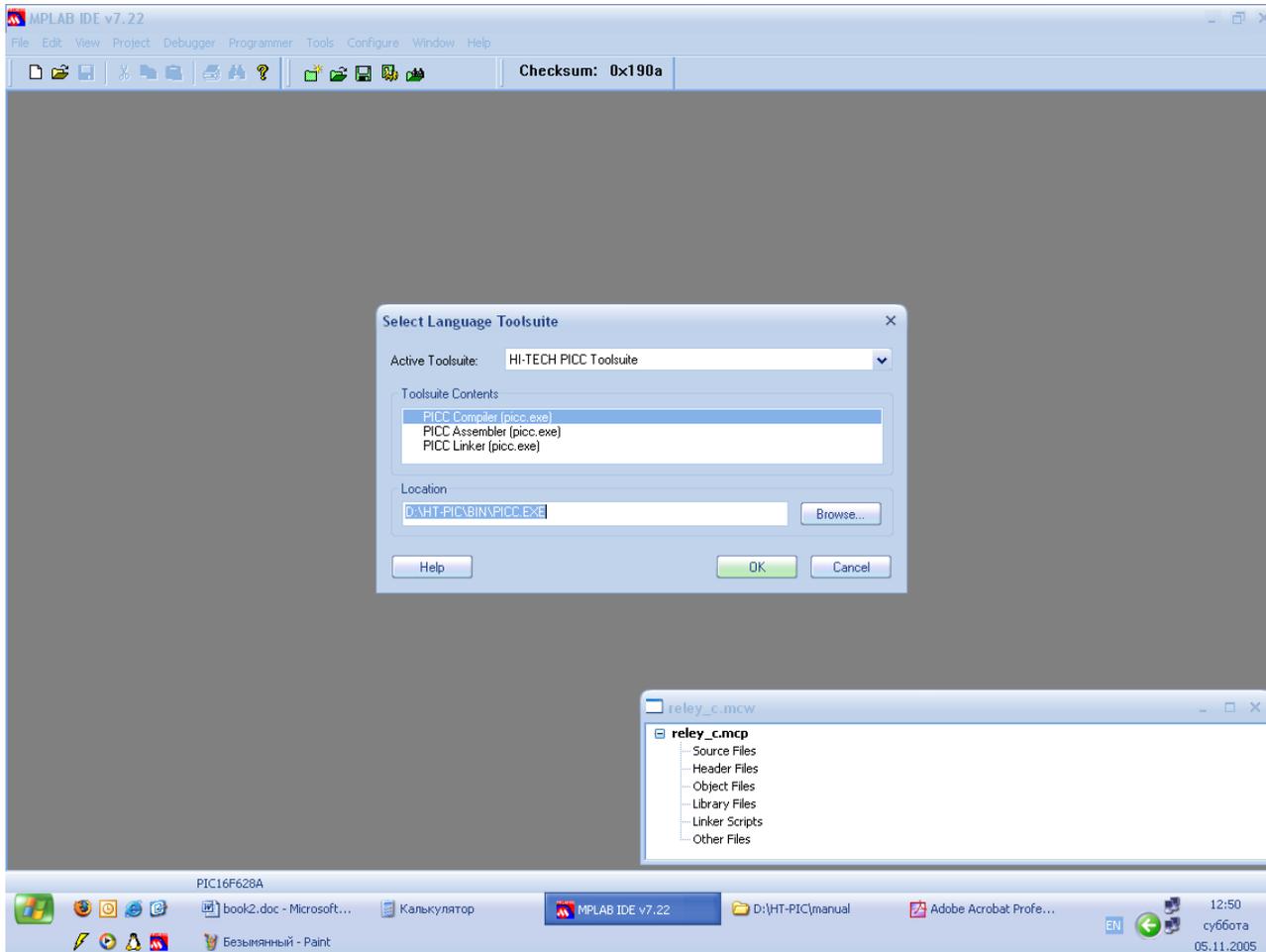


Рис.47

Создадим файл заголовка и основной файл. Добавим в файлы заголовка нужный нам контроллер. Остальная часть работы мало, чем отличается от работы на ассемблере. В программе я не сохраняю состояние реле в EEPROM. В окончательном виде файл заголовка:

```
#define MODULNAMESIM 'R'  
#define CMDSIM '$'  
  
#define bitset(var,bitno) ((var) |= 1 << (bitno))  
#define bitclr(var,bitno) ((var) &= ~(1 << (bitno)))  
  
void putch(unsigned char);  
unsigned char getch(void);  
int init_comms();  
int sim_num_adr();  
int cmd();  
84
```

## Хобби-электроникс 2. Умный дом.

```
int rel_on(int num);
int rel_off(int num);
int rel_stat(int num);
```

Основной файл:

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "reley_c.h"

unsigned char input;           // Считываем содержимое приемного регистра
unsigned char MOD_SIM1;       // Первый символ адреса модуля
unsigned char MOD_SIM2;       // Второй символ адреса модуля
unsigned char REL_SIM;        // Символ реле
unsigned char COMMAND;        // Символ команды
int MOD_ADDR;                 // Заданный адрес модуля, как число
int sim1_num = 0;
int sim2_num = 0;
int sim_end_num = 0;
int MOD_NUM;                  // Полученный адрес модуля, как число
int REL_NUM;                  // Номер реле
int RELSTAT = 0;              // Статус реле (позиционно): 1 - вкл, 0 - выкл.

/* Получение байта */

unsigned char getch()
{
    while(!RCIF)               // Устанавливается, когда регистр не пуст
        continue;
    return RCREG;
}

/* вывод одного байта */

void putch(unsigned char byte)
{
    PORTB = 1;                 // Переключим драйвер RS485 на передачу
    TXEN = 1;                  // Разрешаем передачу
    while(!TXIF)               // Устанавливается, когда регистр пуст
        continue;
    TXREG = byte;
}

/* Преобразуем символьный адрес в число*/

int sim_num_adr()
{
```

## Хобби-электроникс 2. Умный дом.

```
    sim_end_num = 0;
    sim1_num = getch();           // Чтение первого символа номера
    MOD_SIM1 = sim1_num;         // Сохраним первый символ
    sim2_num = getch();           // Чтение второго символа номера
    MOD_SIM2 = sim2_num;         // Сохраним второй символ
    sim1_num = sim1_num - 0x30;    // От первого символа к числу
    sim2_num = sim2_num - 0x30;    // От второго символа к числу
    sim_end_num = sim1_num*0x0A + sim2_num; // Объединим в одно число
    return sim_end_num;
}

/* Получение и выполнение команды */

int cmd()
{
    input = getch();
    REL_NUM = input;             // Номер реле в символьном виде
    REL_SIM = REL_NUM;
    REL_NUM = REL_NUM - 0x30;    // Номер реле в числовом виде

    switch (COMMAND = getch())  // Прочитаем команду
    {
        case 'N': rel_on(REL_NUM); // Если команда включить
        break;
        case 'F': rel_off(REL_NUM); // Если команда выключить
        break;
        case 'S': rel_stat(REL_NUM); // Если команда передать состояние
        break;
    }
}

/* Выполнение команды включения заданного реле */

int rel_on(int num)
{
    bitset (RELSTAT, REL_NUM); // В переменной RELSTAT побитовое состояние реле
    PORTA = RELSTAT;           // Установив бит, переписываем в порт А
}

/* Выполнение команды выключения заданного реле */

int rel_off(int num)
{
    bitclr (RELSTAT, REL_NUM);
    PORTA = RELSTAT;           // Сбросив бит, переписываем в порт А
}
```

## Хобби-электроникс 2. Умный дом.

/\* Выполнение команды передачи состояния заданного реле \*/

```
int rel_stat(int num)
{
    putchar('R');           // Отправляем символ R
    putchar(MOD_SIM1);      // Первый символ номера модуля
    putchar(MOD_SIM2);      // Второй символ номера модуля
    putchar('#');           // Символ статуса
    putchar(REL_SIM);       // Символ номера реле
    if ((RELSTAT>>REL_NUM)&0x01) putchar('N'); // Проверяем, установлен ли бит?
    if (!(RELSTAT>>REL_NUM)&0x01) putchar('F'); // Проверяем, сброшен ли бит?
    putchar(0x0A); // Только для вывода в файл!!!!!!
}

```

```
int init_comms() // Инициализация модуля
{
    PORTA = 0x0;           // Настройка портов А и В
    CMCON = 0x7;
    TRISA = 0x80;
    TRISB = 0xF6;

    RCSTA = 0x90;         // Настройка приемника
    TXSTA = 0x4;          // Настройка передатчика
    SPBRG = 0x16;         // Настройка режима приема-передачи

    INTCON=0;             // Запретить прерывания
    PORTB = 0;            // Выключим передатчик драйвера RS485
}

```

```
void main(void)
{
    /* Инициализация модуля */
    init_comms();

    /* Прочитаем и преобразуем номер модуля */

    MOD_ADDR = PORTB;     // Номер модуля в старших битах
    MOD_ADDR=MOD_ADDR>>4; // Сдвинем на четыре бита

    /* Начинаем работать */
start: input = getch();
    while(input != MODULNAMESIM) input = getch(); // Ждем, если не наш модуль
    MOD_NUM = sim_num_adr(); // Чтение из сети (файла)
    if (MOD_NUM == MOD_ADDR) // Если наш адрес модуля

```

## Хобби-электроникс 2. Умный дом.

```
{
    input = getch();
    if (input == CMDSIM) cmd();           // Если символ команды
}
goto start;
}
```

Это не шедевр программы на языке «С», но, вроде бы, работает. В качестве входного файла команд я использовал input.txt (примечания, которые я сделал, только для книжного варианта, в файле их делать не следует) такого вида:

|           |   |
|-----------|---|
| "noperat" | Проверим, не будет ли мешать посторонняя команда              |
| "L03\$3N" | Проверим, не отвечает ли модуль на обращение к другим модулям |
| "R11\$2N" | Проверим, не отвечает ли модуль на чужие адреса               |
| "R03\$2N" | Включим второе реле третьего релейного модуля                 |
| "R01\$2S" | Проверим, не отвечает ли модуль на чужие адреса               |
| "R03\$2S" | Запросим состояние второго реле третьего релейного модуля     |
| "R15\$1N" | Проверим, не отвечает ли модуль на чужие адреса               |
| "R03\$1N" | Включим первое реле третьего релейного модуля                 |
| "R03\$1S" | Запросим состояние первого реле третьего релейного модуля     |
| "R03#1F"  | Проверим, не отвечает ли модуль на передачу состояния         |
| "R03\$1F" | Выключим первое реле третьего релейного модуля                |
| "R03\$1S" | Запросим состояние первого реле третьего релейного модуля     |
| "R03\$2S" | Запросим состояние второго реле третьего релейного модуля     |

Получаем выходной файл output.txt:

```
R03#2N
R03#1N
R03#1F
R03#2N
```

Теперь, пока вы опробуете работу в среде MPLAB, я, пожалуй, прерву работу над книгой. Поеду в магазин «Чип и Дип», что на Земляном валу, возле Курского вокзала, куплю все необходимое для сборки конвертора RS232-RS485, программатора и создания прототипа. Затем соберу прототип на макетной плате, а когда закончу и проверю, поделюсь впечатлениями.

Но не обязательно сразу!

**Конвертер RS232-RS485 для COM-порта, к которому будут подключаться модули**

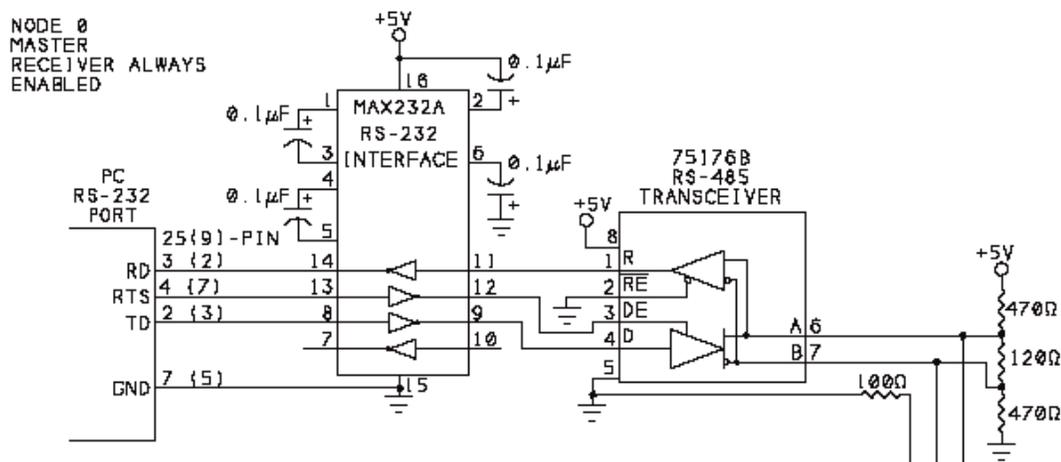


Рис.48

В реальную схему я добавил стабилизатор на 5 вольт, а сам конвертер подключил к 12-вольтовому блоку питания. Напряжение 12В затем подается вместе с сигнальными проводами к макетной плате модулей. В качестве RS485 tranceiver'a я использую микросхему MAX1483. Поэтому я исключил резисторы 470 Ом и их подключение к +5В и общему проводу, оставив только резистор 120 Ом (согласно рекомендации изготовителя микросхемы MAX1483).

Спецификация.

| № | Обозначение | Изделие        | Кол-во | Цена (руб.) | Примечания |
|---|-------------|----------------|--------|-------------|------------|
| 1 | DD1         | MAX232ACPE     | 1      | 80          |            |
| 2 | DD2         | MAX1483        | 1      | 96          |            |
| 3 | R1, R3      | 470 Ом 0.25 Вт | 2      | 1           |            |
| 4 | R2          | 120 Ом 0.25 Вт | 1      | 1           |            |
| 5 | C1-C4       | 0.1 мкФ        | 4      | 5           |            |
| 6 | DD3         | LM78L05        | 1      | 68          |            |
| 7 | C5          | 47 мкФ         | 1      | 20          |            |
| 8 | DB9         | Разъем гнездо  | 1      | 10          |            |

Ориентировочная стоимость изделий 281 руб., плата 100 руб., коробка 100 руб. Всего 481 руб.

Как может выглядеть плата.

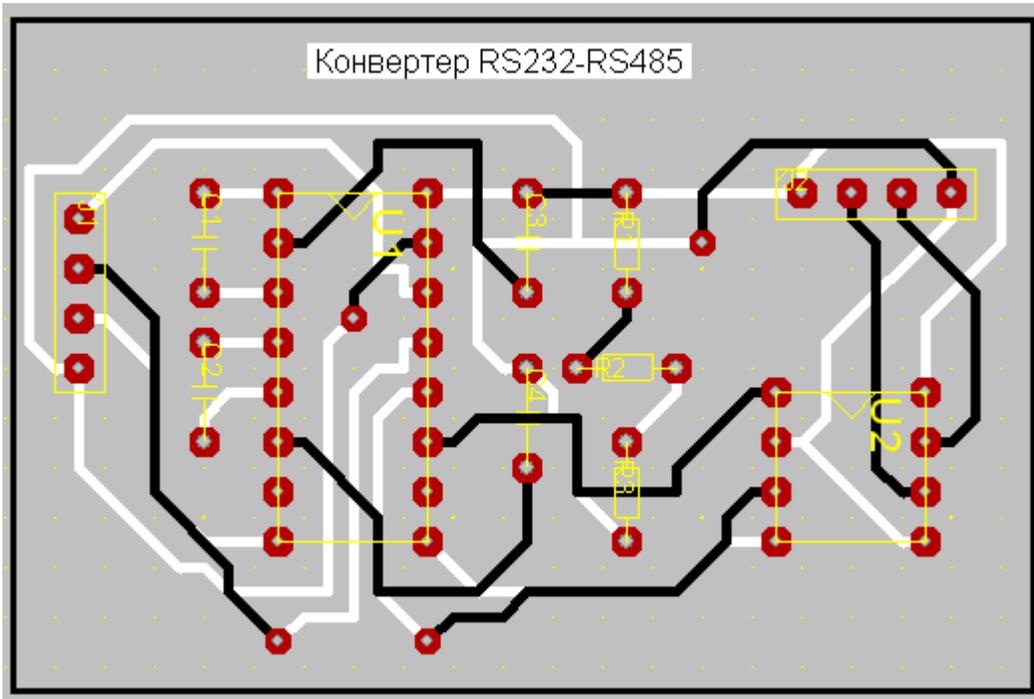


Рис.49

Здесь разъем J1 (клеммник) соединен через DB9 с COM-портом компьютера, а J2 (клеммник) предназначен для подключения питающего напряжения и линии RS485.

*Программатор для программирования микроконтроллеров (совместно с PonyProg)*

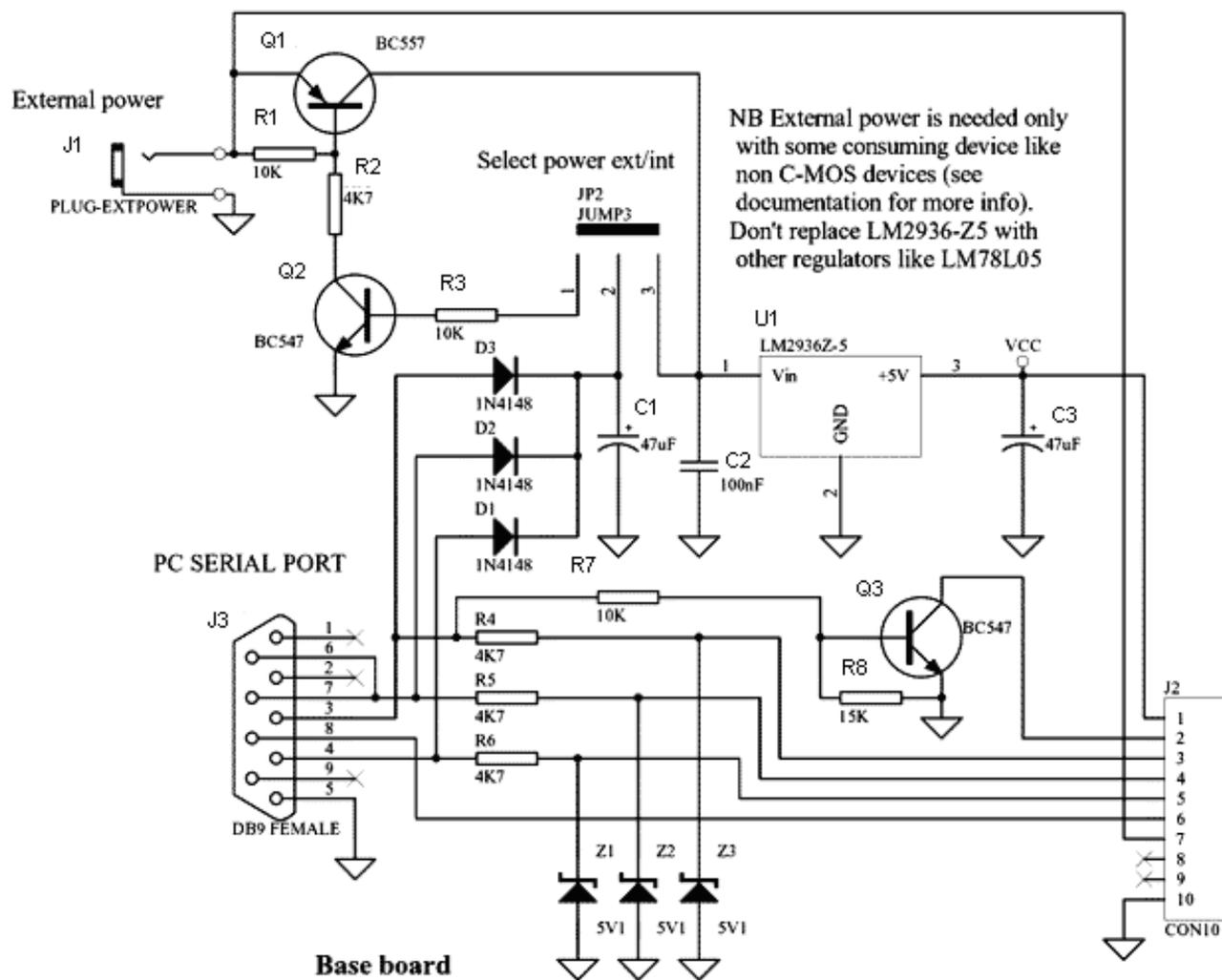


Рис.50

Плата программатора.

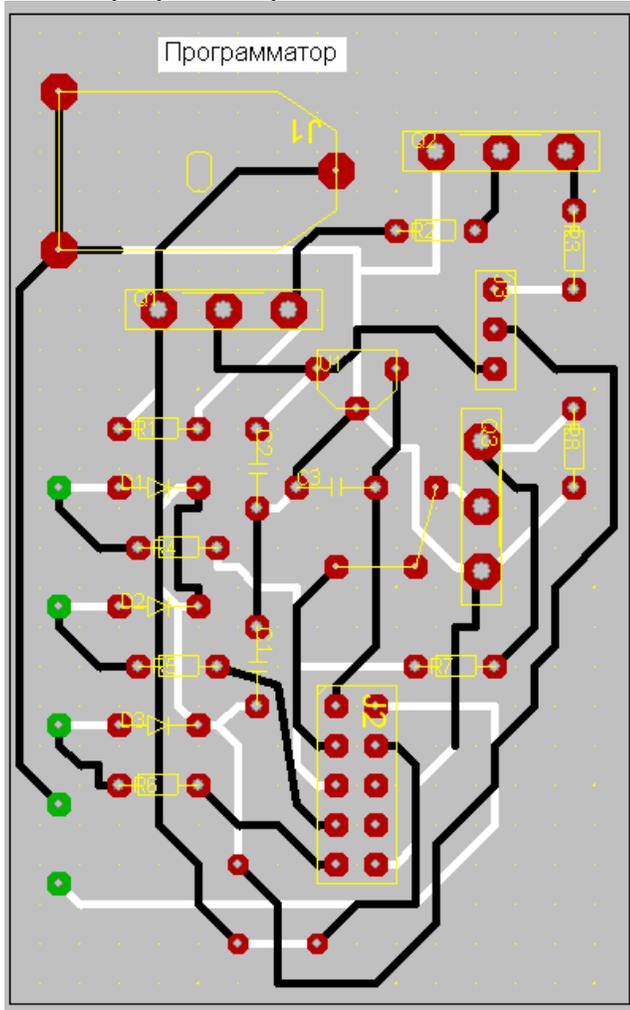
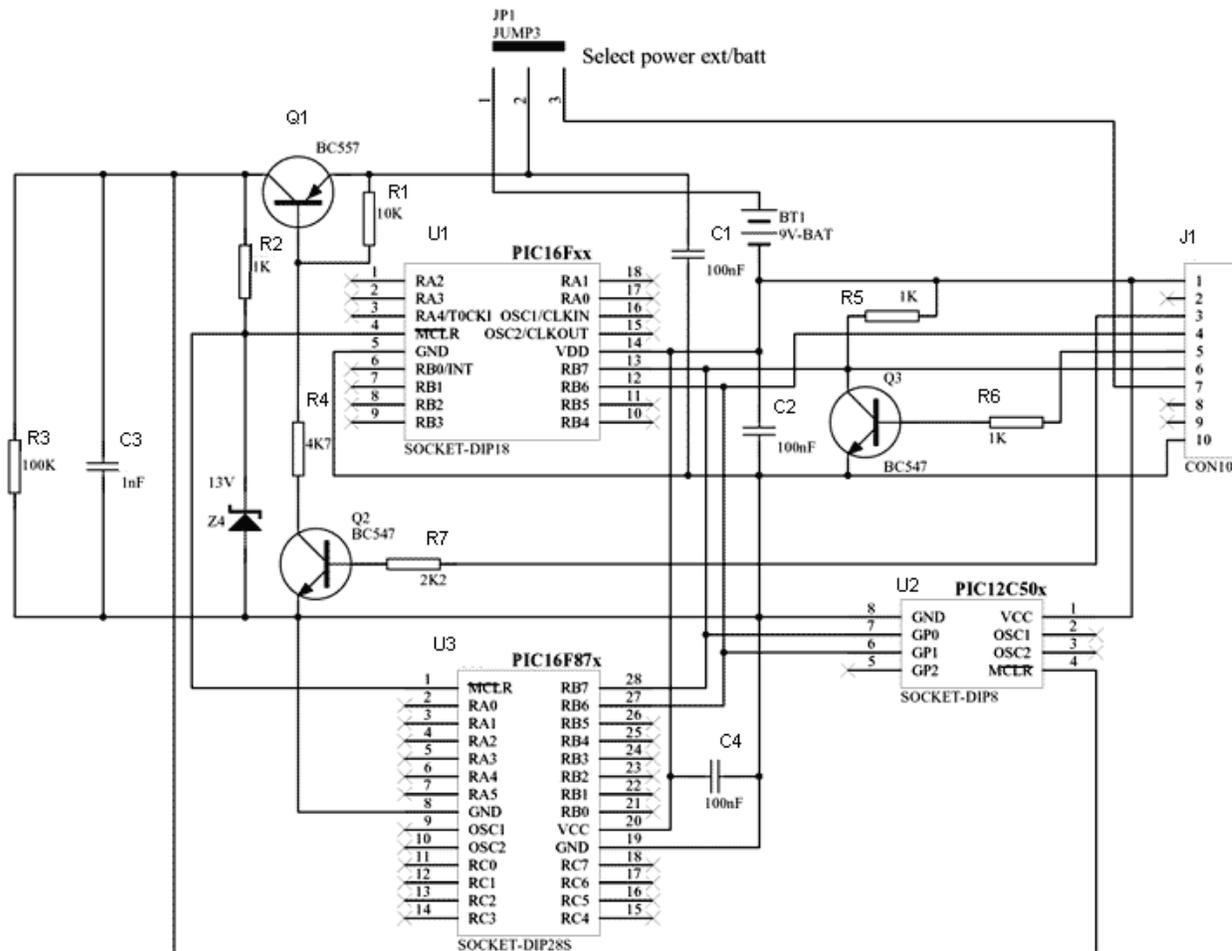


Рис.51

Спецификация

| №  | Обозначение | Изделие          | Кол-во | Цена (руб.) | Примечания |
|----|-------------|------------------|--------|-------------|------------|
| 1  | U1          | LM2936-Z5        | 1      | 68          |            |
| 2  | Q2, Q3      | КТ315Д           | 2      | 5           |            |
| 3  | Q1          | КТ361Д           | 1      | 5           |            |
| 4  | D1-D3       | КД522            | 3      | 3           |            |
| 5  | Z1-Z3       | КС147            | 3      | 5           |            |
| 6  | R2, R4-R6   | 4.7 кОм 0.25 Вт  | 4      | 1           |            |
| 7  | R8          | 15 кОм 0.25 Вт   | 1      | 1           |            |
| 8  | R1, R3, R7  | 10 кОм 0.25 Вт   | 2      | 1           |            |
| 9  | C1, C3      | 47 мкФ 16 В      | 2      | 20          |            |
| 10 | C2          | 0.1 мкФ          | 1      | 10          |            |
| 11 | J2 con10    | Разъем, пара 10к | 2      | 10          |            |
| 12 | DB9         | Разъем, гнездо   | 1      | 10          |            |
| 13 | J1          | Разъем power     | 1      | 10          |            |

Ориентировочная стоимость элементов 147 руб., плата 100 руб. Всего 247 руб.



PIC adapter

Рис.52

Вид платы.

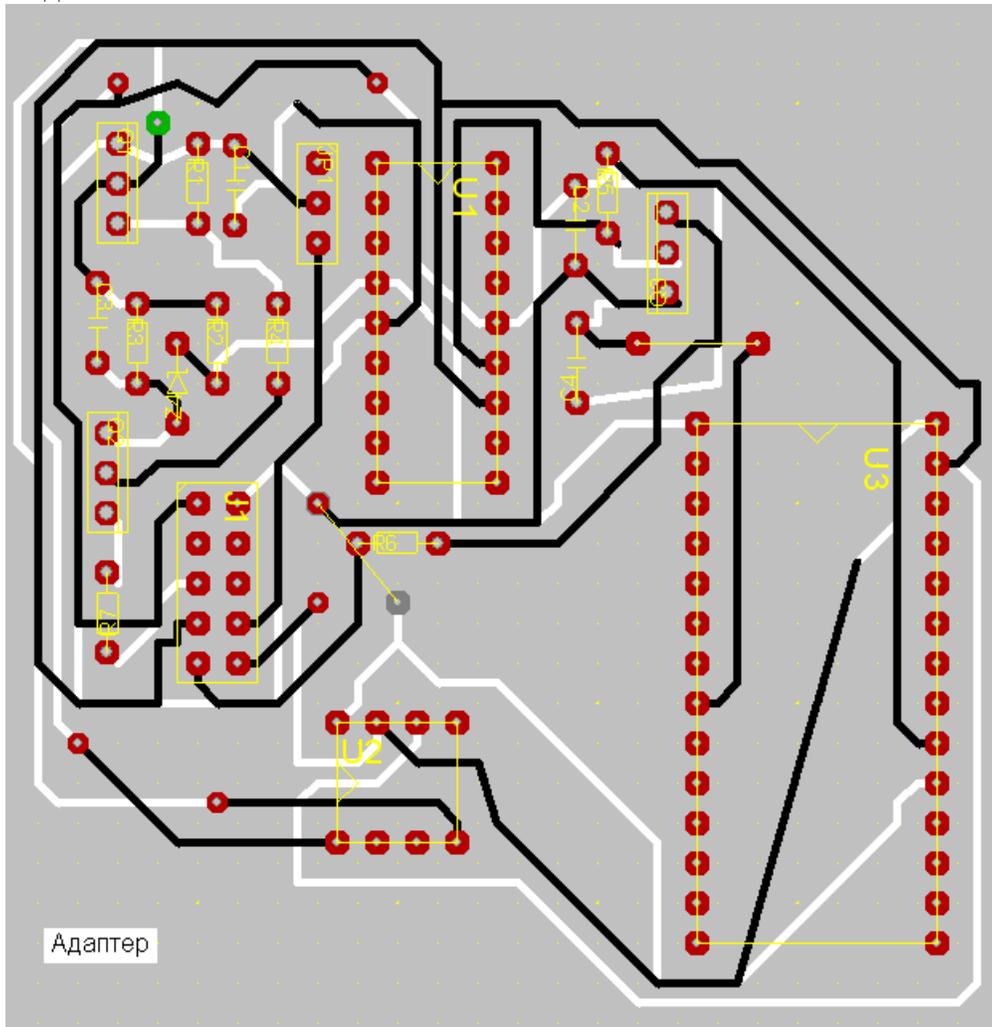


Рис.53

Спецификация

| № | Обозначение | Изделие         | Кол-во | Цена (руб.) | Примечания |
|---|-------------|-----------------|--------|-------------|------------|
| 1 | Q2-Q3       | КТ315Г          | 2      | 5           |            |
|   | Q1          | КТ361Г          |        | 5           |            |
|   | Z4          | КС213           | 1      | 3           |            |
|   | R1          | 10 кОм 0.25 Вт  | 1      | 1           |            |
|   | R2, R5, R6  | 1 кОм 0.25 Вт   | 3      | 1           |            |
|   | R3          | 100 кОм 0.25 Вт | 1      | 1           |            |
|   | R4          | 4.7 кОм 0.25 Вт | 1      | 1           |            |
|   | R7          | 2.2 кОм 0.25 Вт | 1      | 1           |            |
|   | C1, C2, C4  | 0.1 мкФ         | 3      | 5           |            |
|   | C3          | 1000            | 1      | 3           |            |
|   | U1          | DIP18 socket    | 1      | 21          |            |
|   | U2          | DIP8 socket     | 1      | 10          |            |
|   | U3          | DIP 28S         | 1      | 50          |            |
|   | J1          | CON10, пара     | 2      | 10          |            |

Ориентировочная стоимость элементов 117 руб., плата 100 руб. Всего 217 руб.

Позже я несколько упростил программатор.

Нарисуем эскиз нашего модуля.

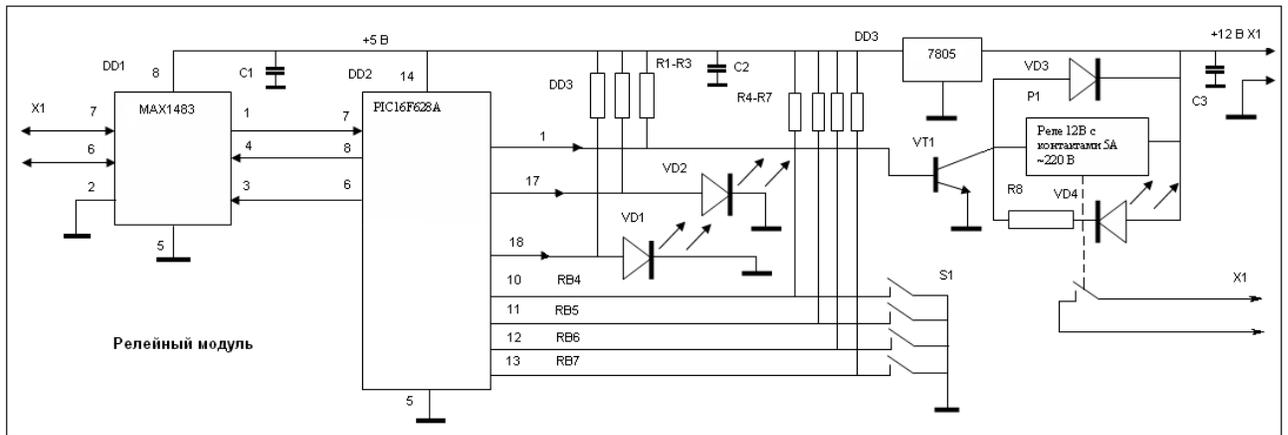


Рис.54

Эскиз получился не самый красивый, но его достаточно для работы. В первую очередь, для макетирования, я думаю использовать клеммники, а не разъемы. Удобнее при отладке. Реле я пока не буду устанавливать. Вместо реле используем индикаторы (светодиоды АЛ307).

Вид платы:

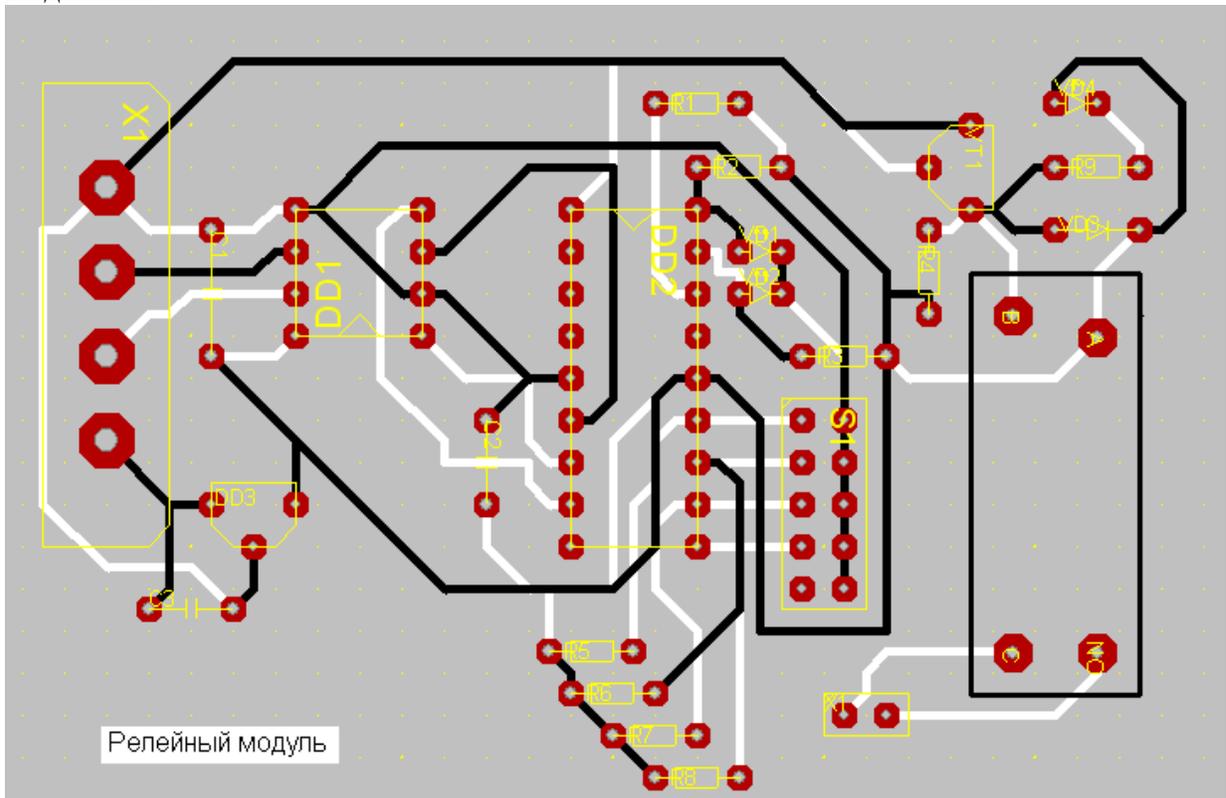


Рис.55

Спецификация.

| №  | Обозначение | Изделие            | Кол-во | Цена (руб.) | Примечания             |
|----|-------------|--------------------|--------|-------------|------------------------|
| 1  | DD1         | MAX1483            | 1      | 96          |                        |
| 2  | DD2         | PIC16F628A         | 1      | 100         | Установить на панельку |
| 3  | DD3         | LM2936-Z5          | 1      | 68          |                        |
| 4  | VT1         | КТ503Г             | 1      | 5           |                        |
| 5  | VD1, VD2    | АЛ301              | 2      | 3           |                        |
| 6  | VD3         | КД522              | 1      | 3           |                        |
| 7  | VD4         | АЛ301              | 1      | 3           |                        |
| 8  | P1          | 12В, 5А/~220В      | 1      | 170         |                        |
| 10 | R1 – R3     | 1 кОм 0.25 Вт      | 3      | 1           |                        |
| 11 | R4-R7       | 10 кОм 0.25 Вт     | 4      | 1           |                        |
| 12 | R8          | 2.2 кОм 0.25 Вт    | 1      | 1           |                        |
| 13 | C1, C2      | 0.1 мкФ            | 2      | 5           |                        |
| 14 | C3          | 100 мкФ 16 В       | 1      | 20          |                        |
| 15 | X1          | Кл. 6 конт.        | 1      | 10          |                        |
| 16 | S1          | Пер. 2 пол. 4 нап. | 1      | 50          |                        |
| 17 | SOC         | DIP18              | 1      | 21          |                        |

Ориентировочная стоимость элементов 558 руб., стоимость платы 100 руб. Всего ~658 руб.

### ***Вернулся я из магазина – грабли «железные» и мягкие***

Пришло время, как и обещал, поделиться впечатлениями.  
Делюсь.

Во-первых: дай-то Бог продержаться сети магазинов «Чип и Дип»! На месте нашего достославного государства в лице его чиновников я не только заботился бы о собственном «посадочном месте», но холил и лелеял магазины технического творчества, да и любого другого творчества. Иначе им не выжить. Но это, к слову.

Подсчитав свои финансовые возможности, я отказался от некоторых запланированных покупок. Я решил упростить программатор, поскольку не собираюсь в настоящий момент работать с множеством микросхем, а только с контроллером PIC16F628A. Я убрал из схемы программатора внешнее питание, из схемы адаптера к программатору убрал все панельки, кроме 18-ножечной, и использовал батарейку «Крона» в качестве внутреннего источника питания (для высоковольтного режима программирования).

На макетной плате я тоже установил панельку под микросхему, с тем, чтобы проверить все схемы на одной макетной плате.

Схема программатора получилась следующая:

Упрощенная схема программатора для контроллеров серии PIC16Fxx

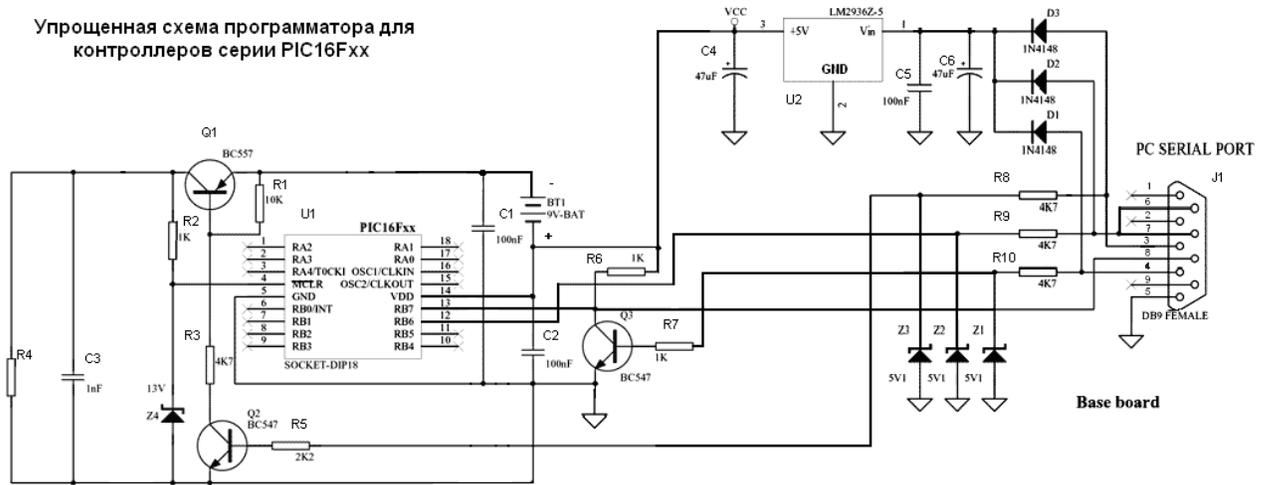


Рис.56

К схеме конвертера я добавил, как говорил выше, стабилизатор на 5 вольт (в спецификации он есть, на схеме отсутствует). Кабель от разъема DB9 конвертера RS232-RS485 я распаял на плате конвертера. После исправления нескольких монтажных ошибок (и эти «грабли» имели место) первым делом я решил проверить программатор, поскольку не в полной мере был уверен, что правильно разобрался со схемой.

С программой MPLAB, что было бы идеально, программатор, естественно, не работает. Использую программу PonyProg2000, схему программатора к которой собрал. Приходится скачивать последнюю версию<sup>2</sup> программы, поскольку в предыдущей нет контроллера PIC16F628A.

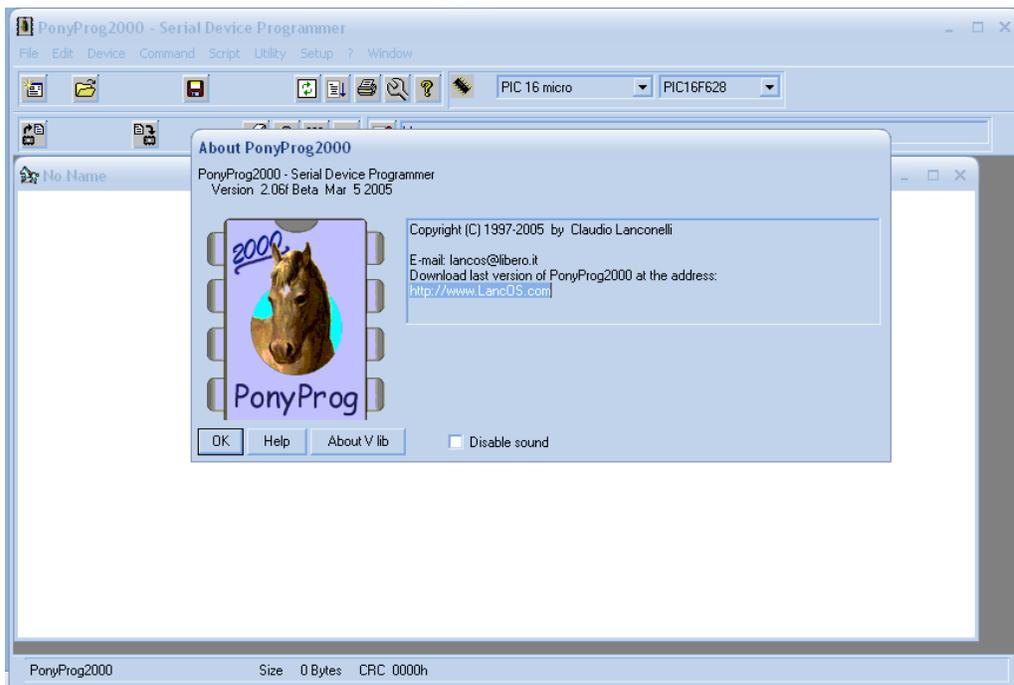


Рис.57

<sup>2</sup> <http://www.LancOS.com>

## Хобби-электроникс 2. Умный дом.

Первое, что я делаю после запуска программы, выбираю устройство в меню Device-Pic 16 micro, или в окошках на панели. После выбора устройства я проверяю работу порта в меню Setup (Interface Setup... используя COM2-порт), задаю номер порта и SI Prog I/O. После нажатия клавиши Probe, я получаю сообщение - Test Ok. Затем провожу калибровку (меню Setup-Calibration), и опять получаю одобрение.

На этом везение заканчивается.

Вставляю микросхему в панельку, и первое, на что отваживаюсь – нажимаю клавишу чтения Read Device на инструментальной панели. Итог:



Программа не распознает микросхемы. Обращаюсь на сайт, с которого «срисовал» программу. В разделе типовых вопросов и ответов нахожу похожую ситуацию и рекомендацию воспользоваться клавишей Ignore. Что и осуществляю.

Процесс проходит успешно, о чем сообщает программа, но в буфере сплошные единицы, а у меня сплошные сомнения – прочитал ли я хоть что-то? Я отключаю разъем от COM-порта, получаю сообщение об успешном чтении, но в буфере одни нули, и тест порта не проходит.

Появляется надежда, что я что-то читаю.

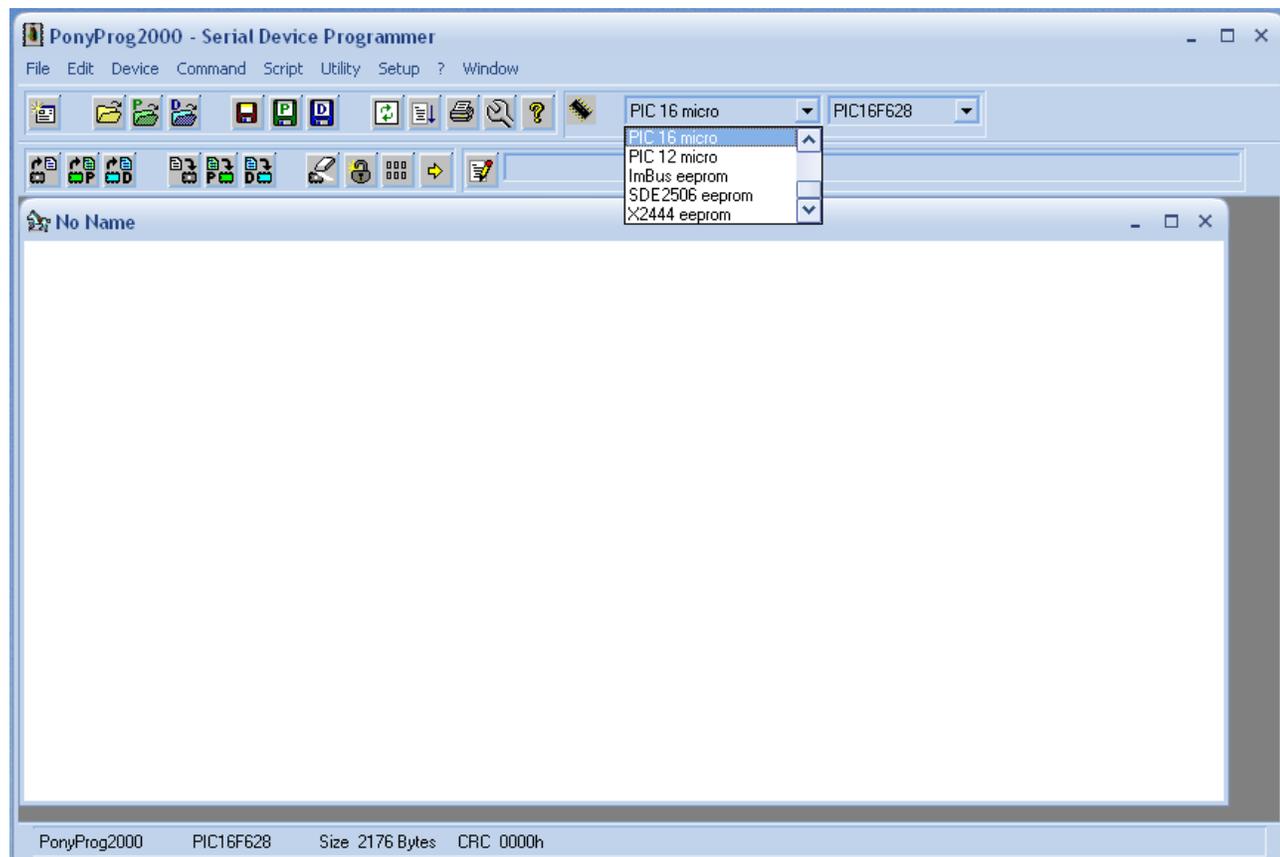
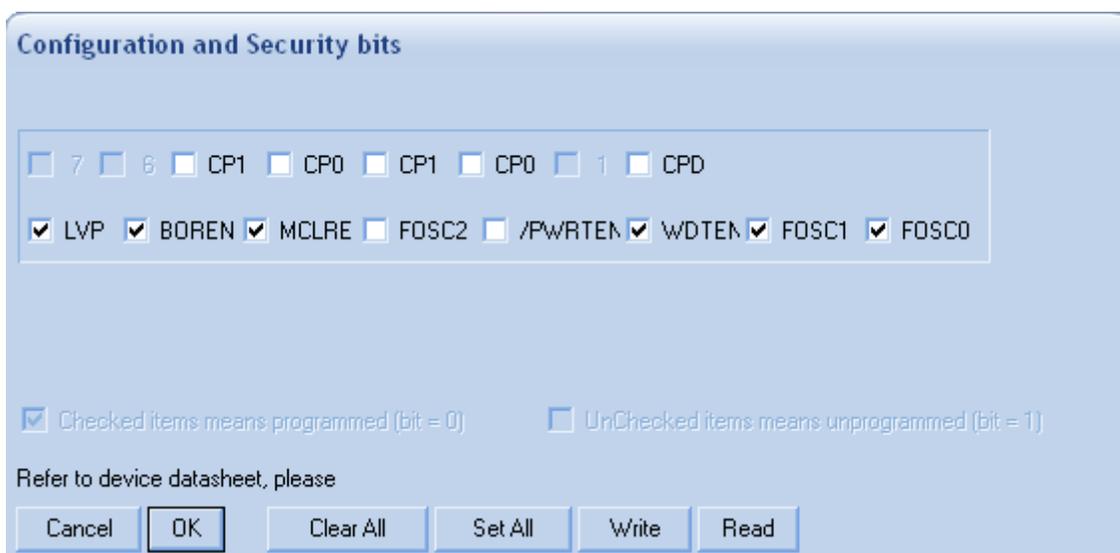


Рис.58

## Хобби-электроникс 2. Умный дом.

Вторая волна сомнений касается вопроса о работе микросхемы, если я не записал биты конфигурации, определяющие режим работы тактового генератора контроллера. Поэтому я решаю сделать первую запись – записать биты конфигурации, выбрав режим работы тактового генератора INTRC (сэкономил, отказавшись от покупки кварца). В меню Command-Security and Configuration Bits... устанавливаю флажки.



*Сознаюсь, что, не разглядев, как устанавливаются биты, я сделал все наоборот – поставил галочки там, где должны быть нули. Я полагал, что, отметив биты, я установлю их в единицу. В результате, я закрыл все программирование. Пришлось разобраться с этим, и исправить свою ошибку.*

Биты конфигурации записались, я сбрасываю их и читаю. Появляется надежда, что вопрос с программированием решен.

Но первые «мягкие» грабли, на которые я наступаю – программа ведет себя не так, как я от нее ожидаю. Я загружаю в программатор файл, созданный по версии, написанной на языке «С». В итоге, я могу записать в контроллер все нули, или все единицы. Но стоит мне попытаться изменить в буфере хотя бы один байт, используя возможность редактирования в меню Edit-Edit Buffer enabled, чтобы записать что-то другое, как контроллер полностью обнуляется. С этой проблемой я долго не могу разобраться по причине не понимания, вина ли это программы, или я что-то делаю не так. Я работаю с программой впервые, и, судя по ошибке с установкой бит конфигурации, могу сделать множество ошибок. Конечно, все попытки обойти проблему успехом не увенчались.

Устав от проб и ошибок, я решаю изменить подход – если я не смогу запрограммировать контроллер, то сама микросхема, которую я так берегу, мне будет не нужна. Либо я смогу ее запрограммировать, либо выброшу сразу.

Я решаю сделать попытку запрограммировать ее как микросхему другого типа. Удачной оказывается попытка программирования PIC16X84. И я решаю, что именно так я буду ее программировать, а биты конфигурации запишу под эгидой PIC16F628. Это

## Хобби-электроникс 2. Умный дом.

неудобно, но не покупать же достаточно дорогой программатор!

Итак, я могу записать в микросхему что-то кроме нулей и единиц.

Я загружаю, используя на инструментальной панели клавишу Read Program Memory (FLASH), HEX-файл откомпилированный программой, написанной на языке «С».

Вторые «мягкие» грабли, на которые я наступаю, выглядят так - вся программа не занимает и двух строк в буфере. Я в это не верю. Поскольку файл имеет расширение HEX, я пытаюсь просмотреть его HEX-редактором и не вижу ничего, кроме символьной записи. Тогда я открываю файл блокнотом и вижу:

```
:10000000830100308A0004282030840038300D201D
:100010008301392B04068001840A0406031D0A2883
:020020000034AA
:1004E20083018C1E712A1A0808008301B700831247
:1004F20003130C1E782A370899000800F401F5014D
:100502000310F30CF20C031C8D2A7008F407710817
:100512000318710AF5070310F00DF10D7208730448
:1005220003190034812A8301850107309F00831655
:100532008501FE30860090308312980006308316C3
:100542009800683099008312061083169B011C14D0 и т. д.
```

Признаться, не ожидал. Пришлось углубиться в руководства и понять, что этот файл предназначен для загрузки в программатор, что формат файла имеет вид:

```
:VVAAAATTNNN... NNCC
```

Где VV – количество байт в строке файла, AAAA – адрес записи данных, TT – указатель типа строки (00 – данные, 01 – конец файла, 02 – адрес сегмента, 04 – линейный адрес), NNN...NN – собственно данные, а CC – контрольная сумма строки.

И данных в файле, конечно, больше, чем я получаю в буфере. В чем причина? После того, как я понял, что после первых строк идет безусловный переход по адресу, в конце программной области, а первый адрес после перехода лежит вне буфера, становится понятнее, почему я не получаю в буфере ничего интересного.

Первой мысль - исправить адреса в файле HEX. Но это потребует изменения контрольных сумм, которые придется либо рассчитывать на калькуляторе, либо писать программу пересчета. Сущность же проблемы, если я правильно понимаю, либо в том, что файл подготовлен под загрузку в программатор, работающий с MPLAB, либо в том, что срок действия демонстрационной версии компилятора языка «С» закончился.

В размышлениях – купить компилятор (каким-то образом), либо найти знакомых, работающих с микроконтроллерами, и «напроситься в гости», чтобы поработать некоторое время на чужом компьютере (меня эти походы изрядно притомили), я решаю поступить иначе.

## Хобби-электроникс 2. Умный дом.

Делаю последнюю попытку – компилирую программу в MPLAB для микросхемы PIC16F627A, которая имеет объем памяти программы вдвое меньше, чем моя PIC16F628A.

Теперь, загружая HEX-файл в программу PonyProg2000, я вижу больше пары строк в буфере. Записываю этот файл (для микросхемы PIC16X84), используя клавишу Write Program Memory (FLASH). Записываю биты конфигурации (для микросхемы PIC16F628). Проверяю в меню Command-Verify All, и получаю подтверждение правильности записи, затем переношу микросхему на макетную плату, где собран интерфейс и установлены светодиоды.

С этого момента опять начинаются «железные» грабли. Ничего у меня, естественно, не работает.

Первое, что следует сделать, как я считаю, это определиться, что у меня работает, а что нет. Уверен ли я, что программатор у меня работает? Нет. Уверен ли я, что программа написана правильно? Нет. И т.д. Я понимаю, что «нет», это самое определенное, что я получил к настоящему моменту.

Тогда я решаю последовательно удалять эти «нет». В Интернете нахожу пример программирования на языке «С» - простая программа, которая заставляет светодиод мигать.

```
#include      "pic16f62xa.h"

#define bitset(var,bitno) ((var) |= 1 << (bitno))
#define bitclr(var,bitno) ((var) &= ~(1 << (bitno)))

main() {
    unsigned int k;
    CMCON = 0x07;           //компараторы выключены
    TRISA = 0b11111110;    //RA0 выход
repeat:
    for (k=0; k<45000; k++); //""пустой" цикл для временной задержки
    bitset(PORTA, 0);       //выставить на RA0 высокий уровень
    for (k=0; k<45000; k++);
    bitclr(PORTA, 0);      //выставить на RA0 низкий уровень
    goto repeat;          //повторить ещё раз
}
```

Я компилирую ее, и программирую микросхему. После установки на макетную плату и включения питающего напряжения я получаю первое «да». Светодиод мигает. По меньшей мере, программатор работает.

Теперь я хочу проверить работу конвертера RS232-RS485. Удобнее всего было бы использовать осциллограф. Но мой осциллограф в отлучке, и у меня нет ничего кроме мультиметра с набором обычных функций измерения напряжения, тока, сопротивления.

Кое-что, я все-таки проверить могу. Для этого мне потребуется программа работы с COM-портом. Совсем вылетело из головы, что терминал Windows предназначен для работы с

## Хобби-электроникс 2. Умный дом.

телефоном. Можно написать что-то на любом из языков программирования, что работало бы с COM-портом. Но это займет время (позже это придется сделать). Нахожу в Интернете программу под названием RS232Pro, которая позволит мне 25 дней поработать с ней. Есть, правда, предложение зарегистрировать программу, но сайт, где предлагается ее зарегистрировать, отсутствует. Думаю, двадцать пять дней – это срок, когда я либо получу положительные результаты, либо откажусь от всего.

Проверяю, включив мультиметр на измерение постоянного напряжения на пределе 20В, входные напряжения от COM-порта. Некоторое время уходит на то, чтобы понять, что нужно нажимать кнопку RTS OFF в программе RS232Pro, но, наконец, я делаю и этот решительный шаг. Затем отправляю в порт последовательность символов 12345, и вижу, что у меня меняется напряжение на выводе TXD. Меньше, но меняется напряжение на выводе 3 микросхемы MAX1483 (см. Рис.48). Хотелось бы проверить линию RS485, тем более что я не стал добавлять резисторы 470 Ом, как это нарисовано на оригинальной схеме конвертера, а ограничился резистором 120 Ом в линии, как рекомендовано изготовителем микросхемы MAX1483.

Для этой цели я пытаюсь передать простой текстовый файл с помощью программы RS232Pro, встав мультиметром на линию (в режиме измерения постоянного напряжения на пределе 20В). Напряжение заметно меняется, что убеждает меня в том, что конвертер, может быть, и не совсем правильно, но работает. Что-то дышит и на соответствующем вводе контроллера.

Порт и модуль я настраиваю на скорость 9600, 8 бит данных, 1 стоповый бит без бита четности. Первым делом я упрощаю программу релейного модуля, оставив только один символ R на прием, который сразу зажигает светодиод. Модуль не работает, но обращение к документации на микросхему контроллера PIC16F628A заставляет меня задуматься в правильности выбора скорости. Я проверяю, меняю значение регистра SPBRG, и разные скорости (соответственно, каждый раз перепрограммируя контроллер), пока не зажигается светодиод. Скорость оказывается равной 2400 (Я даже не уверен, что ее не следует снизить до 1200). Модуль начинает реагировать на команды, передаваемые с компьютера.

В данный момент мне кажется, что дальше все образуется само собой, и модуль заработает. Но получается не совсем так. Мне даже удается получить ответ от модуля на запрос о состоянии реле. Ответ немного странный, но, хотя бы какой-то, он есть. Теперь беда другая. Одно выполнение команды, один ответ на запрос о состоянии реле, и модуль «виснет». Он не реагирует больше на команды до момента включения после сброса питания.

Я человек терпеливый, но и мое терпение истощается по мере продвижения к выполнению поставленной задачи. Главная беда - я не уверен, что программа RS232Pro, с которой я работаю, вполне достаточна для работы. По этой причине отправляюсь к знакомым, чтобы написать программку для работы с модулем. Думаю, я так надоел им, что они дарят мне старенькую версию Visual Basic (не совсем полную), которую кто-то из них получил на презентации во время одной из конференций. Мне важно, что версия работает.

Итак, я создаю заготовку для среды программирования (о чем речь пойдет в конце первой части), куда добавляю три кнопки – одну для отправки команд, другую для получения команд, последняя для того, чтобы полученные команды отобразить.

## Хобби-электроникс 2. Умный дом.

Используя оба варианта работы с COM-портом, я постепенно начинаю понимать, что после получения команды запроса состояния реле, модуль начинает передавать ответ, но одновременно читает, все, что есть на линии RS485. Прочитав же обращение к релейному модулю, прочитав свой номер модуля, он пытается реагировать на эти события. Т.е. при передаче модулем чего-либо в сеть нужно бы дать время на передачу, это, во-первых, и отключить на время передачи приемник, это, во-вторых.

Не с первого раза получилась и запись состояния реле в EEPROM - скоро сказка сказывается, да не скоро дело делается.

Вот такие впечатления, которыми я обещал поделиться, остались у меня после посещения магазина, и попытки претворить в жизнь компьютерную версию разработанного модуля.

В результате программа релейного модуля приобрела следующий вид:

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "reley_c.h"

unsigned char input;           // Для считывания приемного регистра
unsigned char MOD_SIM1;       // первый символ адреса модуля
unsigned char MOD_SIM2;       // второй символ адреса модуля
unsigned char REL_SIM;        // Символ реле
unsigned char command_reciev [6]; // Массив для полученной команды

int MOD_ADDR;                 // Заданный адрес модуля, как число
int sim_end_num = 0;          // Полный символьный номер модуля
int MOD_NUM;                  // Полученный адрес модуля, как число
int REL_NUM;                  // Номер реле
unsigned char RELSTAT = 0;     // Статус реле (позиционно): 1 - вкл, 0 - выкл.
int i;

/* получение байта */
unsigned char getch()
{
    while(!RCIF)              /* устанавливается, когда регистр не пуст */
        continue;
    return RCREG;
}

/* вывод одного байта */
void putch(unsigned char byte)
{
    while(!TXIF)              /* устанавливается, когда регистр пуст */
        continue;
    TXREG = byte;
}
```

## Хобби-электроникс 2. Умный дом.

```
}

/* Преобразуем символьный адрес в число*/

int sim_num_adr()
{
    sim_end_num = 0;
    MOD_SIM1 = command_reciev [1];    // первый символ номера
    MOD_SIM2 = command_reciev [2];    // второй символ номера
    MOD_SIM1 = MOD_SIM1 - 0x30;        // в виде числа
    MOD_SIM2 = MOD_SIM2 - 0x30;
    sim_end_num = MOD_SIM1*0x0A + MOD_SIM2;
    return sim_end_num;
}

/* Получение и выполнение команды */

int cmd()
{
    REL_SIM = command_reciev [4];
    REL_NUM = REL_SIM - 0x30;

    switch (command_reciev [5]) {
        case 'N': rel_on(REL_NUM);
        break;
        case 'F': rel_off(REL_NUM);
        break;
        case 'S': rel_stat(REL_NUM);
        break;
    }
}

/* Выполнение команды включения заданного реле */

int rel_on(int num)
{
    bitset (RELSTAT, REL_NUM);
    PORTA = RELSTAT;
    EEADR =0x0;           // Запишем состояние реле в EEPROM
    EEDATA = ~RELSTAT;
    WREN = 1;
    GIE = 0;
    EECON2 = 0x55;
    EECON2 = 0xAA;
    WR =1;
    while (WR);
    GIE = 1;
    WREN = 0;
}
```

## Хобби-электроникс 2. Умный дом.

```
}

/* Выполнение команды выключения заданного реле */

int rel_off(int num)
{
    bitclr (RELSTAT, REL_NUM);
    PORTA = RELSTAT;
    EEADR =0x0;           // Запишем состояние реле в EEPROM
    EEDATA = ~RELSTAT;
    WREN = 1;
    GIE = 0;
    EECON2 = 0x55;
    EECON2 = 0xAA;
    WR =1;
    while (WR);
    GIE = 1;
    WREN = 0;
}

/* Выполнение команды передачи состояния заданного реле */

int rel_stat(int num)
{
    command_reciev[0] = 'R';
    command_reciev[1] = MOD_SIM1+0x30;
    command_reciev[2] = MOD_SIM2+0x30;
    command_reciev[3] = '#';
    command_reciev[4] = REL_SIM;
    if ((RELSTAT>>REL_NUM)&0x01) command_reciev[5] = 'N';
    if (!(RELSTAT>>REL_NUM)&0x01) command_reciev[5] = 'F';

    CREN =0;           // Запрещаем прием
    RB0 = 1;           // Переключим драйвер RS485 на передачу
    TXEN = 1;         // Разрешаем передачу
    for (i=0; i<6; ++i)    putchar(command_reciev[i]);
    for (i=0;i<1000;i++); // Задержка для вывода
    for (i=0; i<6; ++i) command_reciev [i] = ' ';
    RB0 = 0;           // Выключаем драйвер RS485 на передачу
    TXEN = 0;         // Запрещаем передачу
    CREN =1;           // Разрешаем прием
}

int init_comms()      // Инициализация модуля
{
    PORTA = 0x0;       // Настройка портов А и В
    CMCON = 0x7;
    TRISA = 0x0;
}
```

## Хобби-электроникс 2. Умный дом.

```
TRISB = 0xFE;

RCSTA = 0b10010000; // Настройка приемника
TXSTA = 0b00000110; // Настройка передатчика
SPBRG = 0x68; // Настройка режима приема-передачи
RB0 = 0; // Выключаем драйвер RS485 на передачу

/* Прочитаем состояние реле из EEPROM*/
EEADR = 0x0;
RD = 1;
RELSTAT = ~EEDATA;
PORTA = RELSTAT;
}

void main(void)
{
    // Инициализация модуля
    init_comms();
    for (i=0; i<6; ++i) command_reciev [i] = ' ';
    command_reciev [0] = 'R';
    // Прочитаем и преобразуем номер модуля
    MOD_ADDR = PORTB; // Номер модуля в старших битах
    MOD_ADDR=MOD_ADDR>>4; // Сдвинем на четыре бита

    // Начинаем работать
start: CREN =1;
    input = getch();
    switch (input)
    {
        case 'R': // Если обращение к релейному модулю
            for (i=1; i<6; ++i) // Запишем команду в массив
            {
                input = getch();
                command_reciev [i] = input;
            }

            MOD_NUM = sim_num_adr(); // Чтение из сети

            if (MOD_NUM != MOD_ADDR) break; // Если не наш адрес
            else
                if (command_reciev [3] = '$') cmd(); // Если команда
            default: goto start;
    }
}
```

## Хобби-электроникс 2. Умный дом.

```
goto start;  
}
```

*Правку в программу на ассемблере у меня не хватило сил внести. Если кому-либо это нужно, то исправить ее можно по тексту программы на языке «С».*

Вот HEX-файл для релейного модуля:

```
:10000000830100308A0004282030840038300D201D  
:100010008301392B04068001840A0406031D0A2883  
:020020000034AA  
:1004E20083018C1E712A1A0808008301B700831247  
:1004F20003130C1E782A370899000800F401F5014D  
:100502000310F30CF20C031C8D2A7008F407710817  
:100512000318710AF5070310F00DF10D7208730448  
:1005220003190034812A8301850107309F00831655  
:100532008501FE30860090308312980006308316C3  
:100542009800683099008312061083169B011C14D0  
:100552001A098312A200850008008301AD01AE01D1  
:100562003008A0003108A100D030A007A1070A304E  
:10057200F200F3012008F000F1017F222108740744  
:10058200AD0075080318750AAE00F1002D08F000E1  
:1005920008000130F000831203132908F100F10A68  
:1005A200D42A0310F00DF10BD22A7008A2042208FB  
:1005B200850083169B018312220983169A001C155B  
:1005C2008B1355309D00AA309D009C149C18E72A7D  
:1005D2008B171C11831208000130F00083120313E1  
:1005E2002908F100F10AF72A0310F00DF10BF52AA0  
:1005F2007009A2052208850083169B018312220935  
:1006020083169A001C158B1355309D00AA309D004D  
:100612009C149C180A2B8B171C118312080083014F  
:100622003308A300D030F000FF30F1002308700738  
:10063200A90071080318710AAA002E2B2908B50017  
:100642002A08B600CA2A2908B5002A08B600ED2AE7  
:100652002908B5002A08B6008C2B3408463A03193B  
:10066200242B083A03191F2B1D3A031D0800292BBE  
:100672009422AB01AC01462B2B082F3E840083133E  
:1006820020308000AB0A0319AC0A2C08803AF00033  
:1006920080307002063003192B02031C3D2B5230AE  
:1006A200AF000608A500A6010430F000260DA60C36  
:1006B200A50CF00B572B852BAB01AB0AAC012C0818  
:1006C200803AF00080307002063003192B020318C2  
:1006D200762B7122A4002B082F3E8400831324085A  
:1006E2008000AB0A0319AC0A602BAE227008A70087  
:1006F2007108A8002606031D802B25082706031D66  
:10070200852B2430B200102318167122A400523A0D  
:1007120003195D2B852B52308301AF002008303E38  
:10072200B0002108303EB1002330B2002308B300EC
```

## Хобби-электроникс 2. Умный дом.

:100732002208F0002908F100F10AA12B0310F00CA5  
:10074200F10B9F2B7008F000701CA92B4E30B400E7  
:100752002208F0002908F100F10AB12B0310F00C75  
:10076200F10BAF2B7008F0007018B92B4630B400B3  
:1007720018120614831698168312AB01AC012C08CA  
:10078200803AF00080307002063003192B02031801  
:10079200D42B2B082F3E8400831300087622AB0A49  
:1007A2000319AC0AC02BAB01AC012C08803AF00053  
:1007B20083307002E83003192B020318E42BAB0AD2  
:1007C2000319AC0AD62BAB01AC012C08803AF0001D  
:1007D20080307002063003192B020318FA2B2B0803  
:1007E2002F3E8400831320308000AB0A0319AC0A29  
:0E07F200E62B061083169812831218160800C4  
:00000001FF

Когда я написал, что отправляюсь в магазин, и поделюсь впечатлениями, я и сам не ожидал, что впечатлений будет так много. Конечно, отсутствие опыта, не совсем подходящие средства разработки, разница между тем, что нарисовано на бумаге, и тем, что будет создано на основе этого рисунка – все так, но... Я не ожидал, что столько «граблей» окажется на пути реализации достаточно простой конструкции. Бывает.

Однако продолжим создание запланированных модулей.

### ***Модуль приема инфракрасных кодов***

Первый вопрос - зачем нам нужно что-либо, имеющее отношение к инфракрасным кодам?

Домашняя аппаратура – телевизоры, музыкальные центры и т.п. – управляется с помощью пультов дистанционного управления излучающих команды в инфракрасном диапазоне. Чтобы управлять аппаратурой программно, нам потребуется излучатель инфракрасных кодов (ИК кодов) - модуль, который по команде компьютера будет излучать необходимые ИК команды. Для работы этого модуля нам потребуется еще и считыватель инфракрасных кодов.

Кроме того, пора подумать об устройствах управления в системе. Я уже говорил о сенсорных панелях и универсальных пультах, запоминающих команды с пультов управления бытовой техникой, когда приводил примеры коммерческих систем. Но хотелось бы иметь нечто более дешевое.

Одним из устройств управления, как мы решили, станет компьютер. Можно подумать о создании устройства управления с использованием клавиатуры: нажатие клавиши отправляет в системную сеть команду управления.

Но и у сенсорной панели, и у клавишного модуля есть небольшой недостаток. Их удобно держать на стене или на столе, но неудобно держать на кресле, где мы проводим достаточно много времени. Вопрос о клавишном модуле управления отложим до второй части книги. И рассмотрим возможность управления с помощью старого пульта от

## Хобби-электроникс 2. Умный дом.

видеомагнитофона или телевизора, которые давно отправились бы на свалку, если не завалились бы на полке.

Конечно, если ваши финансовые возможности позволяют, то вы можете применить в этом качестве недорогой универсальный пульт, запоминающий инфракрасные коды.

Но в обоих случаях нам нужен модуль приема инфракрасных кодов.

Что собой представляют инфракрасные коды, излучаемые пультами управления?

Не вдаваясь в теоретические тонкости, можно сказать так. Когда на пульте управления, положим, телевизором нажимается клавиша, то установленный в пульте светодиод (инфракрасного диапазона) начинает мигать. При этом он воспроизводит последовательность вспышек с некоторой частотой (от 20 кГц до 400 кГц) и пауз, которые в совокупности и есть код управления. Каждая клавиша имеет свой набор вспышек и пауз. Клавиши разных пультов излучают разные коды управления, частота (несущая частота) вспышек может быть разной. В качестве примера приведу структуру кодов управления фирмы Sony:

### Technical Info

Code length (длина кода): 12 bits (15 bits для некоторых функций видеокамеры)  
Carrier (несущая): 40kHz

Header (Заголовок):

```

+-----+
|         |
+         +---+..
      4T   T
```

1 кодируется:

```

+-----+
|         |
..+       +---+..
      2T   T
```

0 кодируется:

```

+---+
|     |
..+   +---+..
      T   T
```

T = 550us приблизительно

Пауза между данными: 25ms

data: hhhhxxxxxxxxxyyyyyy  
          ^                  ^  
          MSB                LSB

xxxxxx command (команда)  
yyyyyy address (номер аппарата)

Таким образом, сначала идет заголовок длиной приблизительно в 2,75 мс (инфракрасный свет мигает с частотой 40 кГц). Затем пауза в 0,55 мс. Затем следует код, в качестве которого, для простоты, рассмотрим последовательность 1001. Что будет соответствовать вспышке в 1,1 мс, паузе в 0,55 мс, вспышке в 0,55 мс, паузе в 0,55 мс,

## Хобби-электроникс 2. Умный дом.

вспышке в 0,55 мс, паузе в 0,55 мс, вспышке в 1,1 мс, паузе в 0,55 мс, паузе в 25 мс. Уф!

Как будут выглядеть коды других производителей? Скорее всего, иначе. Есть несколько рекомендаций по применению ИК кодов, а производители вольны придерживаться их или нет. Кодирование логической единицы может производиться переходом (фронтом) от паузы к вспышке, логического нуля обратным переходом. Заголовок может отсутствовать. И т.д., и т.п.

Многие универсальные пульты управления, запоминающие ИК коды, запоминают длительности посылок и пауз, ждут длинной паузы между посылками, а затем сохраняют запомненные значения в формате собственного времени, которое зависит от тактовой частоты синхрогенератора. Если при воспроизведении кода они могут менять несущую частоту, то записывают служебную информацию, определяющую эту несущую частоту.

Итак. Какие требования мы будем предъявлять к нашему модулю приемника инфракрасных кодов? Учтем, что основная задача модуля – принимать коды, которые мы назовем системными.

В качестве системных кодов, удобно взять числа от 1 до 255 в формате выше разобранной структуры ИК кода. На данном этапе примем этот вариант. Повторим еще раз, как будет выглядеть системный ИК код. Вспышка с несущей частотой 37 кГц длительностью 2,2 мс (заголовок). Байт команды от 1 до 255 с соответствующими вспышками и паузами. Пауза 25 мс между командами.

Требования к модулю:

1. Модуль должен принимать системные ИК команды с несущей частотой 37 кГц.
2. Модуль должен отправлять номер команды по запросу центрального управляющего устройства (компьютера) по интерфейсу RS485.

Нарисуем структуру модуля:

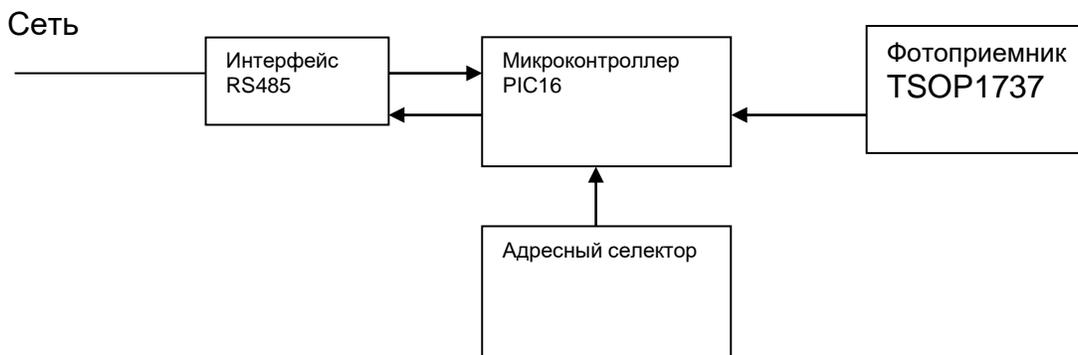


Рис.59

Для того чтобы приступить к работе над контроллером, определим команды, которые будет принимать наш модуль, задав префикс модуля большой латинской буквой «С».

Для обращения к модулю, принята ли новая ИК команда -  $S_{xx} \$0S$  (аналогично команде запроса статуса релейного модуля).

Ответ модуля –  $S_{xxxkkk}$ , если команда пришла и  $S_{xx\#ff}$ , если новая ИК команда не приходила.

Здесь  $xx$ , как и ранее, двухбайтовый адрес модуля.  $kkk$  – три байта символов номера команды от 001 до 255.  $ff$  подтверждает отсутствие изменений за время опроса.

Несколько слов о комплектующих элементах. Разумнее всего, с моей точки зрения, использовать фотоприемник TSOP1737 (если вы используете пульт с несущей частотой 37 кГц, иначе следует выбрать другой фотоприемник из этой серии). В отсутствие команд его выход в высоком состоянии. При наличии ИК пульсаций (с частотой 37 кГц), он переходит в низкое состояние.

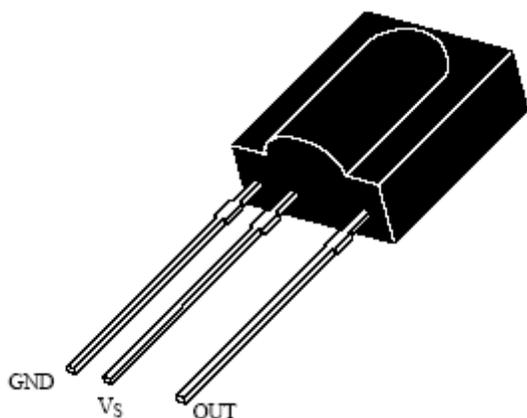


Рис.60

Так выглядит фотоприемник.

### Application Circuit

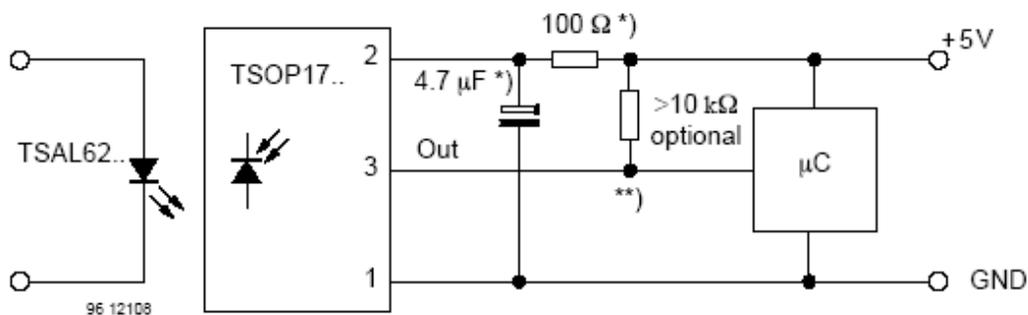


Рис.61

Это рекомендации по подключению. Конденсатор 4,7 мкФ желательно располагать непосредственно около фотоприемника, если между модулем и приемником несколько метров провода.

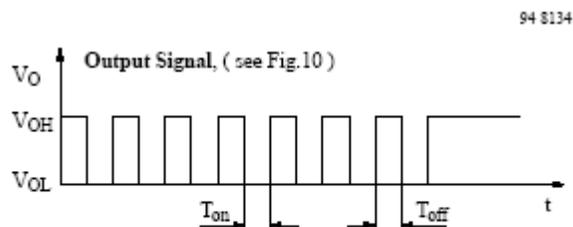


Figure 8. Output Function

Рис.62

Здесь время  $T_{on}$  соответствует послыке ИК импульсов,  $T_{off}$  – паузе.

Можно начинать писать программу.

Как и прежде, для установки заданного адреса используем четыре последних бита порта В (RB4-RB7). Для работы с сетью используем встроенный в микроконтроллер USART. А фотоприемник подключим к выводу RA6. Добавим еще светодиод для индикации наличия сигнала, который подключим к выводу RA0. Поскольку остальные выходы портов нас не интересует, отметим только, что в отличие от предыдущей конфигурации порта А, вывод RA6 следует установить на ввод (в предыдущем варианте порт работал на вывод).

Основные блоки программы:

- Инициализация модуля.
- Ожидание активности приемника USART.
- Ожидание активности от фотоприемника.
- Обработка сетевой команды от компьютера.
- Обработка ИК команды от фотоприемника.

В отличие от предыдущей программы, где мы могли спокойно ждать активности в сети и ни о чем не заботиться, в этой программе нам приходится ожидать два события – активности в сети, и активности фотоприемника. Будем считать, что главное ожидание – это ожидание активности фотоприемника. И попробуем представить, что будет происходить при работе с реальной системой.

Каждое слово команды, состоящее из 6 байт, занимает (байт – 44 мкс) 0,264 мс. Положим, мы добавим 0,5 мс между командами.

Основное занятие компьютера – это постоянный опрос модулей, поскольку программа (основная программа) должна выявить события (изменение состояния модулей) и выполнить указанные в программе действия, соответствующие этим событиям.

В этом плане наш модуль, который опрашивает два источника – сеть и фотоприемник – должен отслеживать изменения, как в сети, так и состояния фотоприемника. Пока модуль «слушает» сеть, он не «слышит» фотоприемник, и наоборот.

## Хобби-электроникс 2. Умный дом.

Вероятно, разумно было бы использовать механизм прерывания, хотя бы для работы по прерыванию с фотоприемником. Основанием стал бы тот факт, что в сети постоянно циркулируют запросы состояния, на которые модуль постоянно «отвлекается». При этом легко пропустить изменение состояния фотоприемника.

Но для начала попробуем оценить, насколько это соответствует действительности. Мы определили длительность заголовка ИК команды в 2,75 мс. Основное назначение заголовка – дать «понять» фотоприемнику, что далее последует команда.

Таким образом, изменение состояния фотоприемника, с целью «привлечь внимание» контроллера модуля, будет длиться около трех миллисекунд. За это время успеет пройти около 10 (длительность слова команды в сети около 0,3 мс) слов команды, что дает основания предполагать, что наш модуль успеет «оценить» ситуацию, и перейдет к приему ИК команды. В отсутствие прерываний, он примет команду полностью, если это системная команда, прежде чем вернется к прослушиванию сети.

С другой стороны, скорость поступления ИК команд едва ли будет превышать 0,3 с (интервал между двумя нажатиями на одну и ту же клавишу). Т.е. за время между поступлениями команд у компьютера будет возможность послать около 1000 запросов. В итоге, если мы не напишем программу, которая будет нуждаться в 2000 запросов, а пользователь не будет играть на клавишах управления, как на рояле, особых проблем возникнуть не должно?

Мы не собираемся строить очень большую сеть. Сеть с большим количеством устройств. Но, даже если бы мы этого захотели, мы могли бы увеличить скорость работы в сети с 9600 б/с до 115000 б/с. Это изменение при прежних рассуждениях увеличило бы количество возможных запросов до 12000. Реальное время между командами я бы тоже оценил в 3-30 секунд. А это позволило бы увеличить количество возможных сетевых запросов до 120000. Если это не поможет, что ж, переделаем все полностью!

Итак, предварительная оценка, я не исключаю, что ошибочная, позволяет нам считать возможной работу модуля без использования прерывания. На том и порешим.

Писать программу на ассемблере или на языке «С»? Как вам удобнее. Я, пожалуй, пока не перестанет работать демонстрационная версия компилятора «С», использую эту возможность.

### *Блок инициализации модуля*

Не мудрствуя лукаво, поправим и используем в этом блоке код предыдущего модуля.

```
int    init_comms()           // Инициализация модуля
{
PORTA = 0x0;                  // Настройка портов А и В
CMCON = 0x7;
TRISA = 0xC0;                // Здесь только RA6 на ввод.
TRISB = 0xF6;
```

## Хобби-электроникс 2. Умный дом.

```
RCSTA = 0x90;           // настройка приемника
TXSTA = 0x4;           // настройка передатчика
SPBRG = 0x16;         // настройка режима приема-передачи

INTCON=0;             // запретить прерывания
PORTB = 0;           // Выключим передатчик драйвера RS485
}
```

### *Блок обработки команды фотоприемника*

Пока RA6 в высоком состоянии («1»), ничего не надо делать. Проверим состояние USART. Если обращение идет не к нашему модулю, то делать ничего не надо, вернемся к определению состояния RA6.

Я, как мне кажется, весьма рассудительно подошел к этому блоку программы. Но сомнения остаются. Я не уверен будет ли вообще работать эта часть программы, переключаясь с прослушивания от одного источника к другому.

По этой причине я, пожалуй, выделю блок и проверю его до написания остальной части программы. Не забудем установить нужное состояние конфигурации (3F1Ah) по адресу 2007h!!!

Мне понадобится файл заголовка:

```
#define MODULNAMESIM 'C'
#define CMDSIM '$'
```

```
void putch(unsigned char);
unsigned char getch(void);
int  init_comms();
int sim_num_adr();
int cmd();
```

И файл программы:

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "ir_rec.h"
```

```
unsigned char input;           // Считываем содержимое приемного регистра
unsigned char MOD_SIM1;       // Первый символ адреса модуля
unsigned char MOD_SIM2;       // Второй символ адреса модуля
unsigned char COMMAND;        // Символ команды
int MOD_ADDR;                 // Заданный адрес модуля, как число
int sim1_num = 0;
int sim2_num = 0;
int sim_end_num = 0;
```

## Хобби-электроникс 2. Умный дом.

```
int MOD_NUM;           // Полученный адрес модуля, как число
int PHOTO;
int PHOTOCOME = 0;
int FLAG = 0;         // Временная переменная

unsigned char getch()
{
    while(!RCIF);     /* устанавливается, когда регистр не пуст */
    continue;
    return RCREG;
}

/* вывод одного байта */
void putch(unsigned char byte)
{
    PORTB = 1;        // Переключим драйвер RS485 на передачу
    TXEN = 1;        // Разрешаем передачу
    while(!TXIF)     /* устанавливается, когда регистр пуст */
        continue;
    TXREG = byte;
}

int  init_comms()     // Инициализация модуля
{
    PORTA = 0xC0;     // Настройка портов А и В
    CMCON = 0x7;
    TRISA = 0xC0;
    TRISB = 0xF6;

    RCSTA = 0x90;    // Настройка приемника
    TXSTA = 0x4;     // Настройка передатчика
    SPBRG = 0x16;    // Настройка режима приема-передачи

    INTCON=0;        // Запретить прерывания
    PORTB = 0;       // Выключим передатчик драйвера RS485
}

/* Получение и выполнение команды */
int cmd()
{
    input = getch();
}

/* Преобразуем символьный адрес в число*/
int sim_num_adr()
```

## Хобби-электроникс 2. Умный дом.

```
{
    sim_end_num = 0;
    sim1_num = getch();           // Чтение первого символа номера
    MOD_SIM1 = sim1_num;         // Сохраним первый символ
    sim2_num = getch();           // Чтение второго символа номера
    MOD_SIM2 = sim2_num;         // Сохраним второй символ
    sim1_num = sim1_num - 0x30;
    sim2_num = sim2_num - 0x30;
    sim_end_num = sim1_num*0x0A + sim2_num;
    return sim_end_num;
}

/* Начнем работать */

void main(void)
{
    init_comms();                // Инициализация модуля
                                // Прочитаем и преобразуем номер модуля
    MOD_ADDR = PORTB;            // Номер модуля в старших битах
    MOD_ADDR=MOD_ADDR>>4;       // Сдвинем на четыре бита

start: PHOTO = PORTA;           // Начинаем работать
    if (PHOTO&0x40) PHOTOCOME = 0; // Нет ИК-сигнала
    if (!(PHOTO&0x40)) PHOTOCOME = 1; // Появился ИК-сигнал
while (PHOTOCOME == 1)
{
    /* Обрабатываем ИК-команду */
    FLAG = 1 ;
break;
}
    /* Нет ИК-сигнала, проверим сеть */
input = getch();
while(input == MODULNAMESIM)
{
    FLAG = 0 ;
    break;
}

    goto start;
}
```

В среде программирования MPLAB это будет выглядеть так:

## Хобби-электроникс 2. Умный дом.

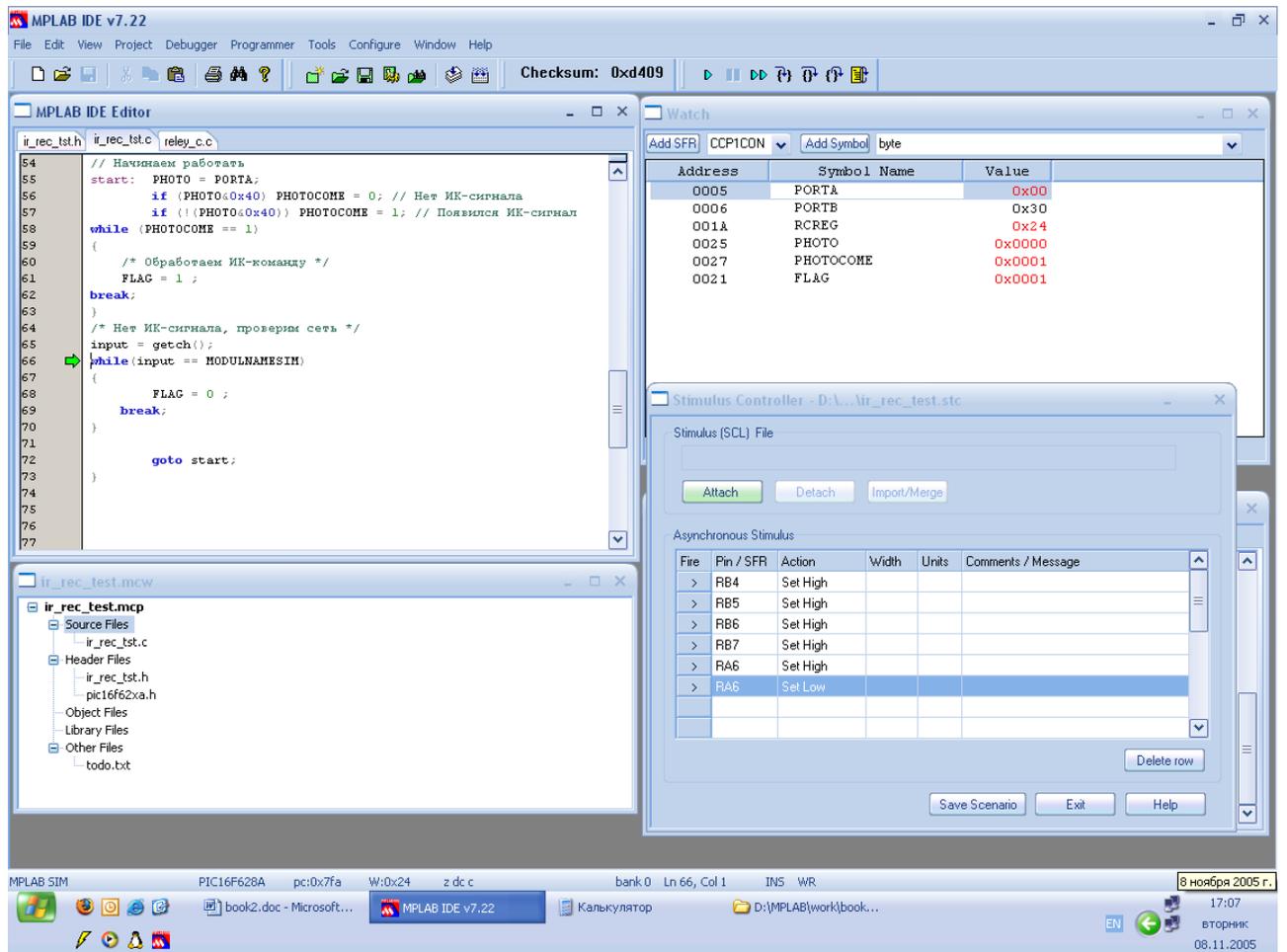


Рис.63

В файл input.txt запишем:

```
"R03$0N"  
"C03$0S"
```

Перед началом проверки запустим RB4 “Set High” и RB5 “Set High”, и RA6 “Set High” на Stimulus Controller (я изменил адрес с 01 на 03, а фотоприемник на вводе RA6 переведем в пассивное состояние).

Переменная FLAG покажет нам, попадаем ли мы в нужное место программы. Впоследствии мы вместо нее будем вызывать подпрограммы обработки команды cmd() и ИК команды ir\_cmd().

Кнопочками RA6 “Set High” и RA6 “Set Low” мы будем имитировать изменение состояния фотоприемника.

Вроде бы работает. Хотя и не сразу, признаюсь. Забыл задать слово конфигурации по адресу 2007h, еще что-то забыл. Но это не важно.

Перейдем к обработке ИК команды (обработку команд сети, я думаю, возьмем из

## Хобби-электроникс 2. Умный дом.

предыдущей программы, подправим, попробуем).

После получения заголовка и паузы в 0,55 мс мы можем ожидать прихода всего двух сочетаний импульс-пауза. Ноль – это импульс-пауза одинаковой длительности 0,55 мс. Единица импульс-пауза, где импульс 1,1 мс, а пауза 0,55 мс.

Таким образом, нам нужны два интервала времени 1,1 мс и 0,55 мс. Для получения интервалов можно воспользоваться таймерами, а можно отсчитать их в пустом цикле. Я воспользуюсь второй возможностью.

Все команды, кроме переходов, выполняются за один машинный цикл (200 нс при частоте 20 МГц). Положим, что при частоте 4 МГц машинный цикл 1 мкс. Тогда нам нужно досчитать до 1100 для промежутка в 1,1 мс, и до 550 для короткого промежутка.

Сравнивая эти промежутки времени с приходящими импульсами, мы можем определить, единицу или ноль мы получаем в ИК команде. Более того, мы можем установить только один счетчик до 550 (для верности возьмем 560) для вычисления байта команды. Попробуем.

Все было бы хорошо, если бы не было плохо. При первой проверке я успевал нажать кнопку RA6, чтобы имитировать приход ИК команды. Теперь не успеваю. Слишком все быстро. Попробуем поискать, не может ли MPLAB помочь с симуляцией, как это было в случае с USART.

И действительно, как любил повторять один из моих знакомых – «Грамотная программа!». В разделе Debugger-SCL Generator находим New Workbook. Нажимаем и получаем окошко, в котором можно настроить имитатор работы фотоприемника.

Пройдя по граблям (вместо того, чтобы почитать описание), я начинаю понимать, что в первой колонке мне нужно задать текущее время, выделяя события. Для этого в первую очередь следует, нажав на кнопку со стрелкой правее надписи Time Units, выбрать единицы времени мкс (us). Событиями в данном случае будет состояние вывода RA6 порта A, которые мы зададим, нажав на большую клавишу Click here to Add Signals. Из выпадающего меню выбираем RA6 и получаем еще одну колонку с этим именем. Теперь в колонку Time (dec) вписываем десятичные значения текущего времени (мкс), а в колонку RA6 состояния входа, соединенного с фотоприемником. Не следует забывать, что в отсутствие активности фотоприемник на выходе имеет высокий уровень (логическая «1»). Первый ИК-сигнал, с которым я хочу работать, выглядит так: 10000000.

Если вернуться к разговору об инфракрасных командах, и выбранному решению по системным командам, то сигнал, поступающий с фотоприемника на вход RA6 должен выглядеть следующим образом.

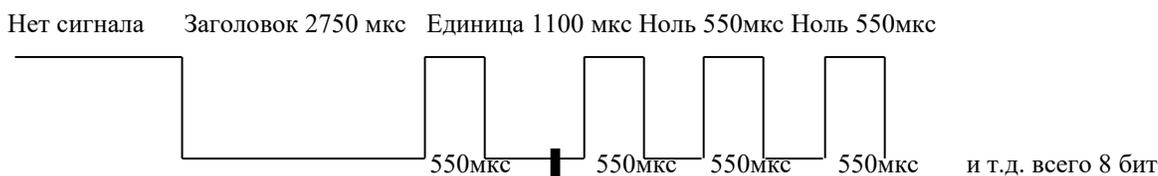


Рис.64

### Проверка

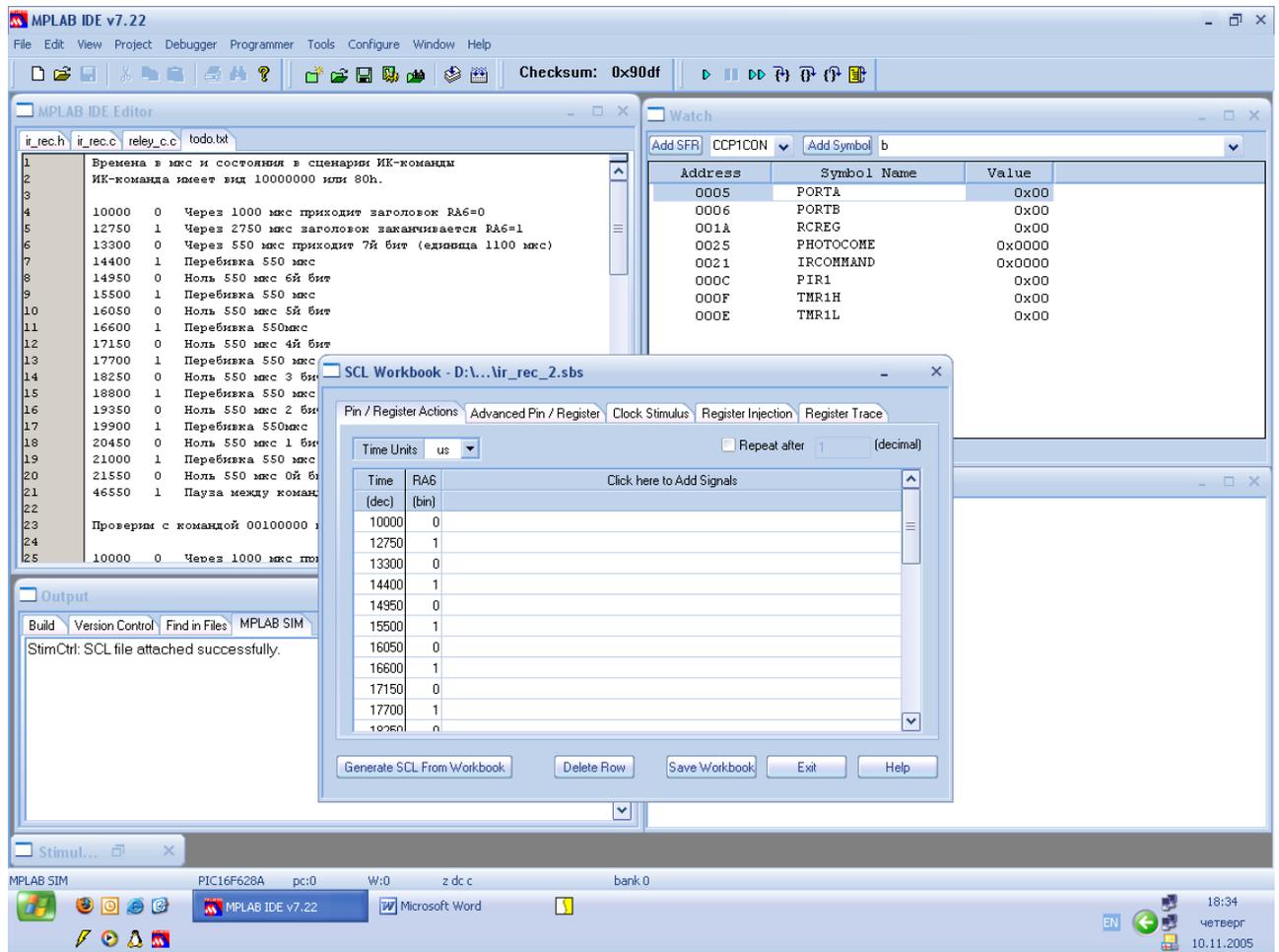


Рис.65

Значения, которые следует проставить, запишем в таблицу.

**Таблица №1**

*Времена в мкс и RA6 в сценарии ИК команды. ИК команда имеет вид 10000000 или 80h.*

| Time  | RA6 | Мои пометки (в книге)                            |
|-------|-----|--|
| 10000 | 0   | Через 1000 мкс приходит заголовок RA6=0          |
| 12750 | 1   | Через 2750 мкс заголовок заканчивается RA6=1     |
| 13300 | 0   | Через 550 мкс приходит 7й бит (единица 1100 мкс) |
| 14400 | 1   | Перебивка 550 мкс                                |
| 14950 | 0   | Ноль 550 мкс 6й бит                              |
| 15500 | 1   | Перебивка 550 мкс                                |
| 16050 | 0   | Ноль 550 мкс 5й бит                              |
| 16600 | 1   | Перебивка 550мкс                                 |
| 17150 | 0   | Ноль 550 мкс 4й бит                              |
| 17700 | 1   | Перебивка 550 мкс                                |
| 18250 | 0   | Ноль 550 мкс 3 бит                               |
| 18800 | 1   | Перебивка 550 мкс                                |
| 19350 | 0   | Ноль 550 мкс 2 бит                               |

## Хобби-электроникс 2. Умный дом.

|       |   |                                 |
|-------|---|---------------------------------|
| 19900 | 1 | Перебивка 550мкс                |
| 20450 | 0 | Ноль 550 мкс 1 бит              |
| 21000 | 1 | Перебивка 550 мкс               |
| 21550 | 0 | Ноль 550 мкс 0й бит             |
| 22100 | 1 | Перебивка 550 мкс               |
| 47100 | 1 | Пауза между командами 25000 мкс |

Теперь нужно сохранить Workbook, и, нажав на клавишу Generate SCL From Workbook, сохранить .scl файл. Для использования этой эмуляции ИК команды в Simulus Controller нажать клавишу Attach, указав сохраненный прежде .scl файл. После сохранения Workspace все необходимое будет появляться после загрузки проекта.

Спасибо создателям программы MPLAB, которые позаботились о возможности работы с портами ввода-вывода в разных их применениях!

Теперь я собираюсь проделать следующее. Когда активизируется RA6, я перехожу к функции обработки ИК команды:

1. Пропускаем заголовок.
2. Пропускаем перебивку в 550 мкс.
3. Когда происходит активизация RA6 приходящим битом, мы запустим «ожидалку» на 560 мкс (немного длиннее импульса в 550 мкс), после которой проверим состояние ввода RA6. Если это 0, то бит «1», если 1, то бит «0». Как это обозначено на рис.64 словом «**Проверка**».
4. Если единица, то еще раз включим «ожидалку», если ноль, то не будем этого делать.
5. Запишем бит в переменную IRCOMMAND, сдвинем влево на одну позицию.
6. Все это проделаем с 8 битами ИК команды.

Поскольку я человек ленивый, я «ожидалку» изобразил самым простым образом:

```
for (i = 0; i<560; ++i);    // Ждем-с
```

Было ли это наказанием за лень, или я не справился с настройками сценария, но, промучившись несколько часов, я выяснил, что во время выполнения цикла for сценарий подхватывает циклы команды и успевают выполниться весь, прежде чем осуществляется выход из цикла ожидания. Я пробовал в качестве единиц измерения Time Units и циклы (сус) и мкс (us) – не помогло.

Конечно, хорошо бы выяснить, в чем проблема, но единожды наказанный за лень, я решил из двух зол выбрать ближнее, организовав «ожидалку» на таймере, как это и положено. Пока я, чертыхаясь, пытался оживить программу с циклом for, я заметил, что цикл while не мешает работе сценария.

В конечном счете, эта часть программы (программа в состоянии проверки этой части) получилась следующей:

```
#include <pic16f62xa.h>
#include <stdio.h>
```

## Хобби-электроникс 2. Умный дом.

```
#include "ir_rec.h"

unsigned char input;           // считываем содержимое приемного регистра
int PHOTOCOME = 0;           // Флаг активности фотоприемника
int MOD_ADDR;                // Заданный адрес модуля, как число
int IRCOMMAND = 0;

unsigned char getch()
{
    while(!RCIF)             /* устанавливается, когда регистр не пуст */
        continue;
    return RCREG;
}

/* вывод одного байта */
void putch(unsigned char byte)
{
    PORTB = 1;               // Переключим драйвер RS485 на передачу
    TXEN = 1;                // Разрешаем передачу
    while(!TXIF)            /* устанавливается, когда регистр пуст */
        continue;
    TXREG = byte;
}

int init_comms()             // Инициализация модуля
{
    PORTA = 0xC0;            // Настройка портов А и В
    CMCON = 0x7;
    TRISA = 0xC0;
    TRISB = 0xF6;

    RCSTA = 0x90;           // Настройка приемника
    TXSTA = 0x4;            // Настройка передатчика
    SPBRG = 0x16;           // Настройка режима приема-передачи

    INTCON=0;               // Запретить прерывания
    PORTB = 0;              // Выключим передатчик драйвера RS485

    PIE1 = 0;               // Настройка таймера 1, запрет прерывания
    T1CON = 0;              // Выбор внутреннего генератора, бит 1 в ноль
}

int ir_cmd ()
{
    int b = 0;
}
```

## Хобби-электроникс 2. Умный дом.

```
while ((PORTA&0x40)== 0) continue;    // Ждем окончания заголовка
for (b=0; b<8;++b)                    // Обработаем наши импульсы
{
    while ((PORTA&0x40) != 0) continue; // Дождемся импульса
    time ();                            // Включаем таймер 1
    while (!TMR1IF);                    // Дождемся такта
    T1CON = 0x0;                        // Выключаем таймер
    TMR1IF = 0x0;                       // Сбросим флаг
    if ((PORTA&0x40) == 0)              // Если низкий уровень, то "1"
    {
        IRCOMMAND = IRCOMMAND +1;      // Запишем это
        time ();                        // Включаем таймер 1
        while (!TMR1IF);                // Дождемся такта
        T1CON = 0x0;                    // Выключаем таймер
        TMR1IF = 0x0;                   // Сбросим флаг
    } else                               // Высокий уровень, значит "0"
    {
        IRCOMMAND = IRCOMMAND +0;      // Запишем это
    }
    IRCOMMAND = IRCOMMAND<<1;          // Сместимся влево на бит
}

// Уехали мы в смещениях на бит далеко, поэтому
IRCOMMAND = IRCOMMAND>>1;             // Сместимся вправо на бит
PHOTOCOME = 0;

}

int cmd()
{
    input = getch();

//    switch (COMMAND = getch()) {
//        case 'N': rel_on(REL_NUM);
//        break;
//        case 'F': rel_off(REL_NUM);
//        break;
//        case 'S': rel_stat(REL_NUM);
//        break;
//    }
}

void time()
{
    TMR1H = 0xFD;                       // Установка числа циклов до переполнения
    TMR1L = 0xA7;                       // FFFF минус 600 (258h)
    T1CON = 0x1;                        // Включение таймера
}
}
```

## Хобби-электроникс 2. Умный дом.

```
/* Начнем работать */

void main(void)
{

    init_comms();                // Инициализация модуля
                                // Прочитаем и преобразуем номер модуля
    MOD_ADDR = PORTB;           // Номер модуля в старших битах
    MOD_ADDR=MOD_ADDR>>4;      // Сдвинем на четыре бита
// Начинаем работать
start:  if (PORTA&0x40) PHOTOCOME = 0;        // Нет ИК-сигнала
        if (!(PORTA&0x40)) PHOTOCOME = 1;    // Появился ИК-сигнал
while (PHOTOCOME == 1)
{
    /* Обрабатываем ИК команду */
    ir_cmd ();
break;
}
    /* Нет ИК-сигнала, проверим сеть */
input = getch();
while(input == MODULNAMESIM)
{
    cmd ();
    break;
}

    goto start;
}
```

Все это получилось не сразу, но, вроде бы, заработало. Теперь осталось проверить с другими вариантами ИК кода. Например

Проверим с командой 00100000 или 20h.

|         |  |
|---------|--|
| 10000 0 | Через 1000 мкс приходит заголовок RA6=0      |
| 12750 1 | Через 2750 мкс заголовок заканчивается RA6=1 |
| 13300 0 | Через 550мкс приходит 7й бит                 |
| 13850 1 | Перебивка 550 мкс                            |
| 14400 0 | Ноль 550 мкс 6й бит                          |
| 14950 1 | Перебивка 550 мкс                            |
| 16050 0 | Единица 1100 мкс 5й бит                      |
| 17150 1 | Перебивка 550 мкс                            |
| 17700 0 | Ноль 550 мкс 4й бит                          |
| 18250 1 | Перебивка 550 мкс                            |
| 18800 0 | Ноль 550мкс 3 бит                            |
| 19350 1 | Перебивка 550 мкс                            |

## Хобби-электроникс 2. Умный дом.

19900 0 Ноль 550 мкс 2 бит  
 20450 1 Перебивка 550 мкс  
 21000 0 Ноль 550 мкс 1 бит  
 21550 1 Перебивка 550 мкс  
 22100 0 Ноль 550 мкс 0й бит  
 22650 1 Перебивка 550 мкс  
 47650 1 Пауза между командами 25000 мкс

Проверим с командой 10000011 или 83h

10000 0 Через 1000 мкс приходит заголовок RA6=0  
 12750 1 Через 2750 мкс заголовок заканчивается RA6=1  
 13300 0 Через 550 мкс приходит 7й бит (единица 1100 мкс)  
 14400 1 Перебивка 550 мкс  
 14950 0 Ноль 550 мкс 6й бит  
 15500 1 Перебивка 550 мкс  
 16050 0 Ноль 550 мкс 5й бит  
 16600 1 Перебивка 550мкс  
 17150 0 Ноль 550 мкс 4й бит  
 17700 1 Перебивка 550 мкс  
 18250 0 Ноль 550 мкс 3 бит  
 18800 1 Перебивка 550 мкс  
 19350 0 Ноль 550 мкс 2 бит  
 19900 1 Перебивка 550мкс  
 20450 0 Единица 1100 мкс мкс 1 бит  
 21550 1 Перебивка 550 мкс  
 22100 0 Единица 1100 мкс мкс 0й бит  
 23200 1 Перебивка 550 мкс  
 48200 1 Пауза между командами 25000 мкс

И можно двигаться дальше. Предстоит добавить обработку запросов по системной сети, преобразование позиционного кода ИК команды в символьный вид. Функцию обработки запросов по системной сети, подправив, я возьму из версии программы для предыдущего модуля. Можно подправить и сценарий имитации работы фотоприемника. Можно добавить задание адреса модуля (пусть он пока остается «03»), и подключение к порту А фотоприемника RA6=1.

Для этого в Workbook SCL Generator'е клавишей Click here to Add Signals добавим еще три столбца RB4, RB5 и PORTA. И в конце (как в таблице №1) добавим:

**Таблица №2** *Добавления к сценарию*

| Time  | RA6 | RB4 | RB5 | PORTA | Мои пометки (в книге)                         |
|-------|-----|-----|-----|-------|---|
| 10000 | 0   |     |     |       | Через 1000 мкс приходит заголовок RA6=0       |
|       |     |     |     |       | И т.д.  |
| 47100 | 1   |     |     |       | Пауза между командами 25000 мкс               |
| 5     |     | 1   | 1   | 40    | Начальная инициализация адреса, фотоприемника |

## Хобби-электроникс 2. Умный дом.

Теперь при запуске программы будет задан адрес модуля, фотоприемник установит RA6 в «1».

В итоге получилась, не слишком аккуратная, следующая программа:

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "ir_rec.h"

unsigned char input;           // Считываем содержимое приемного регистра
unsigned char MOD_SIM1;       // Первый символ адреса модуля
unsigned char MOD_SIM2;       // Второй символ адреса модуля
unsigned char IRSIM1;
unsigned char IRSIM2;
unsigned char IRSIM3;
int PHOTOCOME = 0;           // Флаг активности фотоприемника
int MOD_ADDR;                // Заданный адрес модуля, как число
int IRCOMMAND = 0;
int sim1_num = 0;
int sim2_num = 0;
int sim_end_num = 0;
int MOD_NUM;                 // Полученный адрес модуля, как число

unsigned char getch()
{
    while(!RCIF)             /* устанавливается, когда регистр не пуст */
        continue;
    return RCREG;
}

/* вывод одного байта */
void putch(unsigned char byte)
{
    PORTB = 1;               // Переключим драйвер RS485 на передачу
    TXEN = 1;                // Разрешаем передачу
    while(!TXIF)             /* устанавливается, когда регистр пуст */
        continue;
    TXREG = byte;
}

int init_comms()             // Инициализация модуля
{
    PORTA = 0xC0;            // Настройка портов А и В
    CMCON = 0x7;
    TRISA = 0xC0;
    TRISB = 0xF6;
}
```

## Хобби-электроникс 2. Умный дом.

```
RCSTA = 0x90;           // Настройка приемника
TXSTA = 0x4;           // Настройка передатчика
SPBRG = 0x16;          // Настройка режима приема-передачи

INTCON=0;              // Запретить прерывания
PORTB = 0;             // Выключим передатчик драйвера RS485

PIE1 = 0;              // Настройка таймера 1, запрет прерывания
T1CON = 0;             // Выбор внутреннего генератора, бит 1 в ноль

    /* Определим номер модуля */
MOD_ADDR = PORTB;      // Номер модуля в старших битах
MOD_ADDR=MOD_ADDR>>4;  // Сдвинем на четыре бита
}

    /* Преобразуем символьный адрес в число*/
int sim_num_adr()
{
    sim_end_num = 0;
    sim1_num = getch(); // Чтение первого символа номера
    MOD_SIM1 = sim1_num; // Сохраним первый символ
    sim2_num = getch(); // Чтение второго символа номера
    MOD_SIM2 = sim2_num; // Сохраним второй символ
    sim1_num = sim1_num - 0x30;
    sim2_num = sim2_num - 0x30;
    sim_end_num = sim1_num*0x0A + sim2_num;
    return sim_end_num;
}

int ir_cmd ()
{
    int b = 0;

    while ((PORTA&0x40)== 0) continue; // Ждем окончания заголовка
    for (b=0; b<8;++b)                 //Обработаем наши импульсы
    {
        while ((PORTA&0x40) != 0) continue; // Дождемся импульса
        time ();                             // Включаем таймер 1
        while (!TMR1IF);                     // Дождемся такта
        T1CON = 0x0;                          // Выключаем таймер
        TMR1IF = 0x0; /                       / Сбросим флаг
        if ((PORTA&0x40) == 0)                // Если низкий уровень, то "1"
        {
            IRCOMMAND = IRCOMMAND +1;        // Запишем это
            time ();                          // Включаем таймер 1
            while (!TMR1IF);                 // Дождемся такта
            T1CON = 0x0;                      // Выключаем таймер
        }
    }
}
```

## Хобби-электроникс 2. Умный дом.

```
        TMR1IF = 0x0;           // Сбросим флаг
    } else                       // Высокий уровень, значит "0"
    {
        IRCOMMAND = IRCOMMAND + 0;    // Запишем это
    }
    IRCOMMAND = IRCOMMAND << 1;    // Сместимся влево на бит
}

// Уехали мы в смещениях на бит далеко, поэтому
IRCOMMAND = IRCOMMAND >> 1;    // Сместимся вправо на бит
PHOTOCOME = 0;
}

int cmd()
{
    MOD_NUM = sim_num_adr();    // Чтение из сети (файла)
    if (MOD_NUM == MOD_ADDR)    // Если наш адрес модуля
    {
        input = getch();
        if (input == '$')        // Если символ команды
        {
            while ((input = getch()) != 'S');    // Ждем
            ir_stat();
        }
    }
}

void time()
{
    TMR1H = 0xFD;                // Установка числа циклов до переполнения
    TMR1L = 0xA7;                // FFFF минус 600 (258h)
    T1CON = 0x1;                // Включение таймера
}

int ir_stat()
{
    int ircom;
    int ircom1;
    int ircom2;
    int ircom3;

    ircom = IRCOMMAND;    // Преобразуем команду в символы
    ircom1 = ircom / 0x64;
    IRSIM1 = ircom1 + 0x30;
    ircom2 = (ircom - ircom1 * 0x64) / 0xA;
    IRSIM2 = ircom2 + 0x30;
    ircom3 = (ircom - ircom1 * 0x64 - ircom2 * 0xA);
}
```

## Хобби-электроникс 2. Умный дом.

```
    IRSIM3 = ircom3 + 0x30;

    putchar('C');
    putchar(MOD_SIM1);
    putchar(MOD_SIM2);
    if (IRCOMMAND == 0)    // Команда не менялась
    {
        putchar('#');
        putchar('f');
        putchar('f');
        putchar(0x0A);    // Только для вывода в файл!!!!!!
    } else    // За время между двумя запросами пришла ИК команда
    {
        putchar(IRSIM1);
        putchar(IRSIM2);
        putchar(IRSIM3);
        putchar(0x0A);    // Только для вывода в файл!!!!!!
    }
    IRCOMMAND = 0;    // Мы передали ИК команду, она нам больше не нужна
}

/* Начнем работать */

void main(void)
{
    init_comms();    // Инициализация модуля
                    // Прочитаем и преобразуем номер модуля
    MOD_ADDR = PORTB;    // Номер модуля в старших битах
    MOD_ADDR=MOD_ADDR>>4;    // Сдвинем на четыре бита
    // Начинаем работать
start:  if (PORTA&0x40) PHOTOCOME = 0;    // Нет ИК-сигнала
        if (!(PORTA&0x40)) PHOTOCOME = 1;    // Появился ИК-сигнал
    while (PHOTOCOME == 1)
    {
        /* Обработаем ИК команду */
        ir_cmd ();
    break;
    }
    /* Нет ИК-сигнала, проверим сеть */
    input = getch();
    while(input == MODULNAMESIM)    // Если в сети обращение к модулю
    {
        cmd ();    // Обработаем сетевую команду
        break;
    }
    goto start;
}
```

## Хобби-электроникс 2. Умный дом.

В настоящий момент меня уже одолевают сомнения. Какого рода?

Пока активность в сети не мешает приему ИК команд. Не будет ли она мешать тогда, когда появится функция обработки запросов? Можно попытаться рассчитать это, но расчеты могут оказаться достаточно сложными, это раз. И не будет ли мешать прием ИК команд системным запросам? Не стану считать, попробуем, посмотрим.

В последнем случае, если не забуду, в основной программе системы при отсылке запроса о состоянии модуля ИК-приемника, сделаю паузу, и, если ответа нет, повторю запрос.

Есть и еще одно сомнение – не будет ли в реальных условиях неправильно считываться ИК команда? Здесь тоже можно что-нибудь придумать. Например, можно системную команду повторять трижды, а в модуль добавить три считывания, сравнивая их и принимая в качестве команды ту, что совпадает дважды. На данном этапе я предпочту отложить решение проблем до момента их появления, хотя, как мне кажется, самое время обо всем позаботиться.

Работа выше приведенной программы происходит при input.txt такого вида:

```
"R03$0N"  
"R01$0N"  
"C03$0S"
```

и опция Rewind Input на странице Uart1 IO установлена. В результате имитация сетевых команд постоянно «крутится», принимая три команды, последняя из которых запрашивает состояние модуля фотоприемника.

Результат работы программы выводится, как и раньше, в файл output.txt

```
C03#ff          // Команда не менялась  
C03131         // Пришла системная команда 131 (десятичная) или 10000011 двоичная  
C03#ff          // Команда не менялась после последнего запроса  
C03#ff          // Команда не менялась  
C03#ff          // Команда не менялась
```

*По дороге я наступил на очередные «грабли». Как они выглядят? Внеся исправления в программу, я обнаружил, что она не желает работать. Во всяком случае, работать так, как мне хотелось бы. Пытаясь понять, в чем дело, я много раз просматривал программу, пока не обнаружил, что один из операторов имеет тот же цвет, что и комментарий. Как это получилось? Я много использую однострочных комментариев (тоже следствие лени), и при удалении промежутков между операторами программы, если в конце строчного комментария не нажат ввод, следующий оператор воспринимается программой как продолжение комментария. Вот такие грабли!*

## Хобби-электроникс 2. Умный дом.

Чтобы частично развеять сомнения, я хочу переделать программу для тестирования. Что я имею в виду?

1. Проверить, правильно ли будет работать программа, если после первой ИК команды придет вторая через 50 мс.
2. Проверить, много ли запросов о состоянии модуля пропускается, если это происходит.

Для этого я в папке, где лежат все проекты, создам еще одну папку, которую обозначу с добавкой `_test`. Скопирую в нее все содержимое папки модуля приемника ИК команд. Переименую все файлы, добавив `_test` (исключая `input` и `output` файлы). Затем открою этот проект в программе MPLAB, настрою, добавлю в сценарий второй ИК код, добавлю в программу счетчики и попробую ответить на те два вопроса, которые задаю себе сейчас.

*Переделки:*

В раздел переменных добавим (я использую глобальные переменные, чтобы их легче было отслеживать, поскольку локальные переменные до обращения к функциям не видны):

```
int COUNTER = 0;
```

А в основной программе этот счетчик после вызова команды обработки сетевых команд:

```
/* Нет ИК-сигнала, проверим сеть */
input = getch();
while(input == MODULNAMESIM)    // Если в сети обращение к модулю
{
    cmd ();                      // Обработаем сетевую команду
    ++COUNTER;
    break;
}

goto start;
```

Результат эксперимента – за время прохождения двух ИК команд (83h и 20h) счетчик досчитал до 5.

Файл `input.txt`, который «крутится» без перерывов выглядит так:

```
"R03$0N"
"R01$0N"
"C03$0S"    Обращение к модулю приемника ИК команд
"R03$0N"
"R01$0N"
"C03$0S"    Обращение к модулю приемника ИК команд
```

Файл `output.txt` получился следующим:

```
C03#ff      Команда не обновлялась
```

## Хобби-электроникс 2. Умный дом.

|        |                        |
|--------|------------------------|
| C03131 | Команда 131 (83h)      |
| C03#ff | Команда не обновлялась |
| C03#ff | Команда не обновлялась |
| C03032 | Команда 32 (20h)       |

Таким образом:

1. ИК команды не потерялись за непрерывно следующими сетевыми командами.
2. ИК команды прочитались правильно.
3. Между приходами ИК команд запрос статуса правильно отображал отсутствие обновления.

Посмотрим, как это все выглядит во времени.

Первая ИК команда приходит через 10 мс и заканчивается через 24 мс, вторая приходит через 60 мс и заканчивается через 73 мс после начала работы. Сетевые команды начинают поступать сразу после начала работы. Каждая сетевая команда занимает, примерно, (каждая команда имеет 60 бит, проходящих со скоростью ~9600 бит/сек) 6,3 мс. До начала ИК команды при такой скорости должна пройти одна сетевая команда. За время считывания первой ИК команды  $((10 \text{ мс} - 6 \text{ мс}) + 24 \text{ мс}) = 28 \text{ мс}$  пройдет 4 команды, после завершения первой ИК команды и до конца второй пройдет  $73 \text{ мс} - 24 \text{ мс} = 49 \text{ мс}$ , что соответствует прохождению 7-8 команд. Таким образом, за все время пройдет 12-13 команд. Т.е. наш input файл прочитается дважды или немного более. Что дает 4-5 обращений к модулю приемника ИК команд.

Наш output файл фиксирует 5 обращений, счетчик фиксирует 5 обращений.

Похоже, что все успевают поработать без видимых потерь. Для большей уверенности посмотрим, как распределяются сетевые обращения:

1. За время от начала работы до завершения считывания ИК команды проходит 5 системных запросов, среди которых один к ИК-приемнику.
2. В выходном файле этому соответствует один ответ об отсутствии обновления команд.
3. Следующая команда во входном файле это запрос к модулю.
4. В выходном файле на втором месте стоит ответ о приходе команды 131 (к этому моменту считывание первой команды завершено).
5. До завершения считывания второй команды проходит семь системных команд, среди которых два обращения к модулю приемника.
6. В выходном файле два ответа – команда не обновлялась (предыдущая уже передана).
7. Следующая сетевая команда – запрос о состоянии модуля приемника, который следует сразу за завершением считывания ИК команды, ответ – команда 32 в выходном файле.

Если я ничего не напутал, то даже при короткой основной программе (программе центрального управляющего устройства), когда запросы о состоянии модуля приемника идут достаточно часто, ответ успевает доходить до адресата. Если учесть, что в тестовом варианте две команды проходят с интервалом в 36 мс, что в 10-30 раз быстрее реально отправляемых команд (реально интервал между двумя нажатиями на одну и ту же клавишу 0,3 - 1 сек), то

## Хобби-электроникс 2. Умный дом.

запас, я думаю, есть.

Конечно, мои подсчеты не страдают излишней продуманностью и точностью, но я на время решил успокоиться. А вы?

## Хобби-электроникс 2. Умный дом.

### Схема модуля системного фотоприемника.

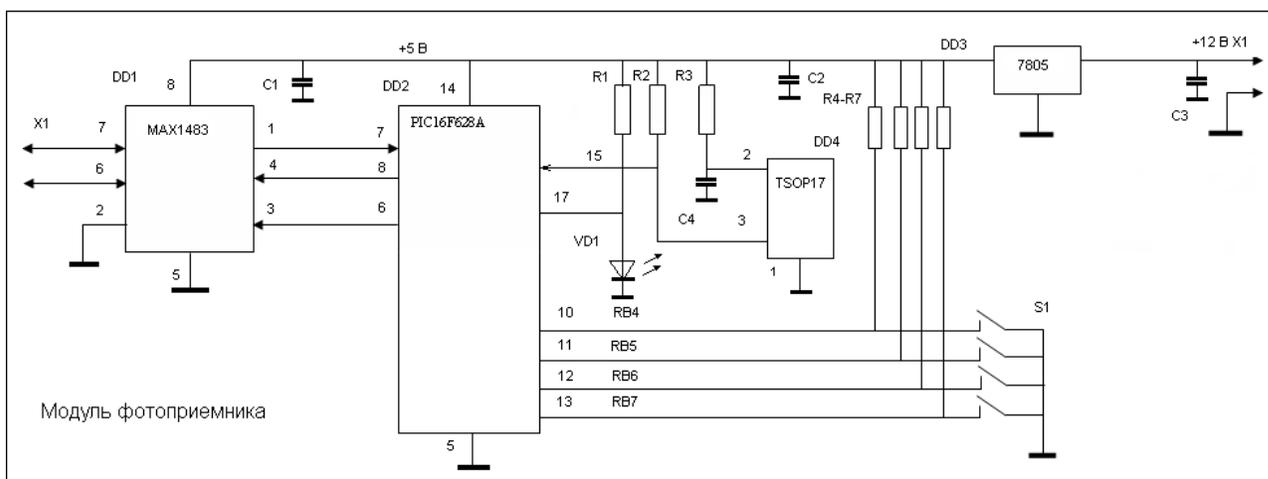


Рис.66

### Спецификация. Модуль фотоприемника.

| №  | Обозначение | Изделие         | Кол-во | Цена (руб.) | Примечания             |
|----|-------------|-----------------|--------|-------------|------------------------|
| 1  | DD1         | MAX1483         | 1      | 96          |                        |
| 2  | DD2         | PIC16F628A      | 1      | 100         | Установить на панельку |
| 3  | DD3         | LM2936-Z5       | 1      | 68          |                        |
| 4  | DD4         | TSOP17 (16)     | 1      | 10          |                        |
| 5  | VD1         | АЛ307           | 1      | 3           |                        |
| 7  | R1          | 1 кОм 0.25      | 1      | 1           |                        |
| 8  | R2          | 12 кОм 0.25 Вт  | 1      | 1           |                        |
| 9  | R3          | 100 Ом 0.25 Вт  | 1      | 1           |                        |
| 10 | R4-R7       | 10 кОм 0.25 Вт  | 4      | 4           |                        |
| 11 | C1, C2      | 0.1 мкФ         | 2      | 6           |                        |
| 12 | C3          | 100,0 мкФ 16В   | 1      | 5           |                        |
| 13 | C4          | 4,7 мкФ 5В      | 1      | 3           |                        |
| 14 | X1          | Клеммник 4 к.   | 1      | 3           |                        |
| 15 | S1          | Перекл 2пол, 4н | 1      | 5           |                        |
| 16 | Socket      | DIP18           | 1      | 21          |                        |

Ориентировочная стоимость элементов - 84 руб. (Макет с учетом предыдущих закупок).

Вид платы.

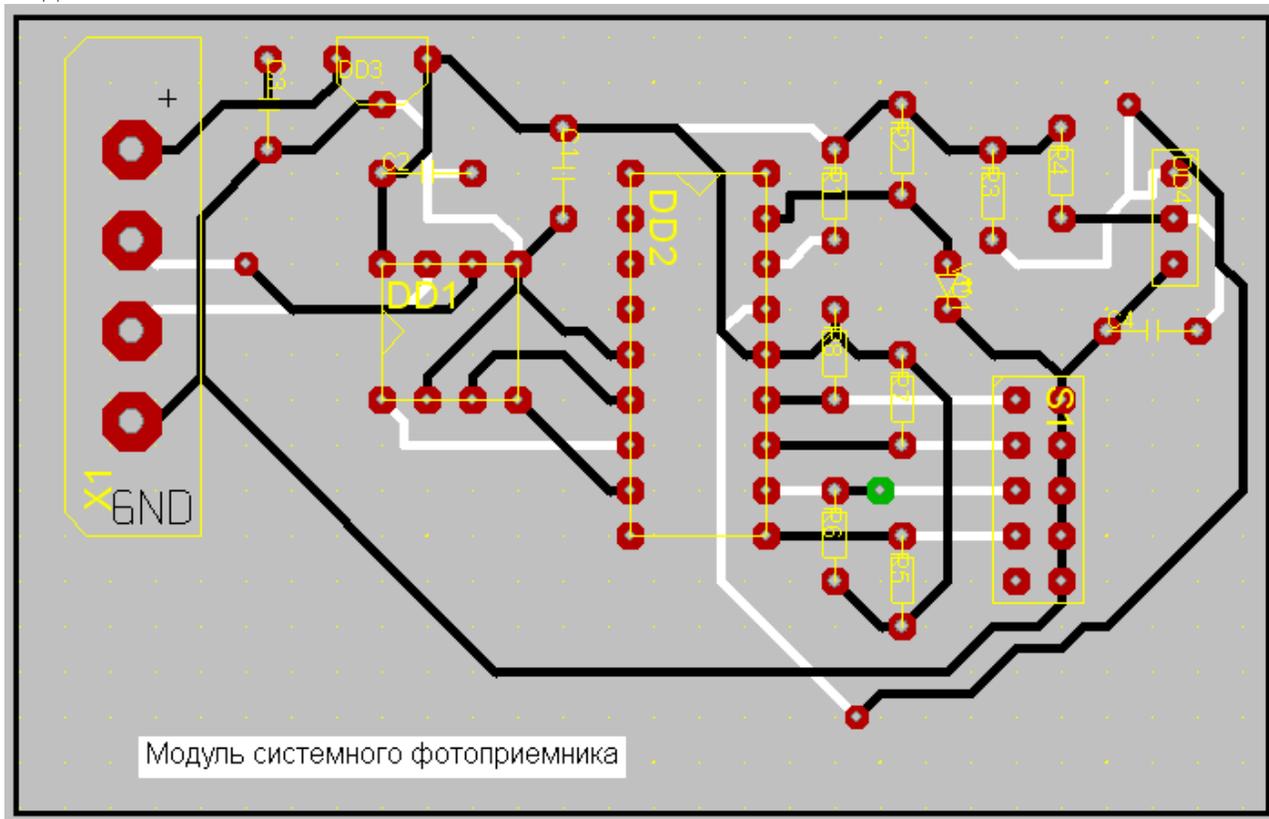


Рис.67

### ***Борьба с собственной гениальностью***

Прототип я делаю на той же макетной плате, на которой собирал релейный модуль. По этой причине я включаю фотоприемник на вход RB3. Дополнительно использую вывод RA0, к которому уже подключен светодиод, для индикации приема ИК команд. Когда устанавливается флаг прихода ИК команды, светодиод включается. Флаг снимается, светодиод выключается.

```

if (!(RB3&0x01))
{
    PHOTOCOME = 1;
    RA0 = 0x01;
}
else
{
    PHOTOCOME = 0;
    RA0 = 0x00;
}
    
```

Вот и первые «грабли»!

Приоритет приема ИК команд хорошо бы закрепить однозначно. Например, так:

## Хобби-электроникс 2. Умный дом.

```
{
    while((!RCIF)&(RB3 == 1))
        /* устанавливается, когда регистр не пуст */
        continue;
    return RCREG;
}
```

Цветом я выделил добавленный фрагмент. У прототипа фотоприемник подключен к выводу RB3. Смысл добавленного фрагмента в том, что на сетевые запросы ответ будет в том случае, если модуль не занят приемом ИК команд, т.е. RB3 = 1.

Далее. Неплохо было бы защитить модуль от помех по RB3. При включении может «проскочить» короткий нулевой импульс (на чем я и споткнулся), и модуль перейдет в режим приема ИК команды, которой нет. Поправил я это так:

```
start: if (RB3&0x01)      // Нет ИК-сигнала
    {
        PHOTOCOME = 0;
        RA0 = 0x00;
        break;
    }
    if (!(RB3&0x01))      // Появился ИК-сигнал
    {
        for (k=0; k<30; ++k); // Поставим задержку
        if (!(RB3&0x01)) // ИК сигнал не пропал?
        {
            PHOTOCOME = 1;
            RA0 = 0x01;
        }
    } else
    {
        PHOTOCOME = 0;
        RA0 = 0x00;
        break;
    }
}
```

Я добавляю задержку, а затем еще раз проверяю наличие активности фотоприемника. Короткий случайный импульс проскочит за время задержки, а длинный импульс команды приведет механизм считывания в действие. Помогло. Если в первый раз после включения питания (сразу или через несколько секунд) модуль «подвисал» на приеме ИК команды, то после исправления перестал.

Хорошо, модуль получает команду. Возможно, распознает ее. Он даже передает ее по запросу через RS485. А что получится, если команда не единичная, т.е. в том случае, когда идет поток одинаковых команд? Не знаю.

Что будет, если команды не системные, а чужие? Не знаю.

## Хобби-электроникс 2. Умный дом.

Вот, пока не узнаю, и говорить не о чем!

Первое, что приходит в голову – запустить еще один таймер сразу после завершения считывания ИК команды. И пока он не завершит отсчет времени больший, чем занимают 2-3 команды, не принимать ИК коды. Попутно я пытаюсь воспроизвести код аналогичный тому, что использовал в отладчике MPLAB, воспроизвести с устройства, работающего под управлением программы WinLIRC.

Проходит два дня. Результаты не впечатляют. Коды прочитываются не слишком уверенно. Я увеличиваю интервал сканирования с 600 до 700 мкс, но совсем не удовлетворен стабильностью работы модуля. В отличие от релейного, модуль приема системных ИК команд мне совсем не нравится. Он, вроде бы работает, но...

В конечном счете, я меняю излучатель кодов. Использую оригинал – старый пульт от видеомэгнофона Sony. Меняю второй таймер на обычный цикл в команде считывания кодов (выделено красным цветом), попутно решаю внести исправления в сдвиг на один бит. У меня получился лишний сдвиг, который я после получения ИК команды убирал обратным сдвигом. После замены переменной IRCOMMAND с int-типа на unsigned char я стал терять старший бит.

Замена всех этих «шатаний из стороны в сторону» оказалась простой (изменение выделено синим цветом). Если я не ошибаюсь, то этого достаточно.

```
int ir_cmd ()
{
    int b = 0;

    while (RB3 == 0)    continue;           // Ждем окончания заголовка
    for (b=0; b<8; b++) //Обработаем наши импульсы
    {
        while (RB3 != 0)
        continue;           // Дождемся импульса
            time ();           // Включаем таймер 1
            while (!TMR1IF);   // Дождемся сброса таймера
            T1CON = 0x0;       // Выключаем таймер
            TMR1IF = 0x0;      // Сбросим флаг
            if (RB3 == 0)      // Если низкий уровень, то "1"
            {
                IRCOMMAND = IRCOMMAND +1; // Запишем это
                time ();           // Включаем таймер 1
                while (!TMR1IF);   // Дождемся сброса таймера
                T1CON = 0x0;       // Выключаем таймер
                TMR1IF = 0x0;      // Сбросим флаг
            } else             // Высокий уровень, значит "0"
                IRCOMMAND = IRCOMMAND; // Запишем это
    }
    if (b<7) IRCOMMAND = IRCOMMAND<<1; // Сместимся влево на бит
}
```

## Хобби-электроникс 2. Умный дом.

```
PHOTOCOME = 0x0;
RA0 = 0x0;
RA1=0x1;
for (m=0; m<3; ++m) // Перестанем принимать ИК коды
    for (l=0; l<10000; ++l);
RA1 = 0x0;
}
```

Здесь на выводе порта A RA1 я использую еще один светодиод, чтобы видеть окончание команды.

Результатом перемен стало то, что модуль хорошо распознает команды от пульта видеоманитфона. Чувствительность высокая, распознавание стабильное.

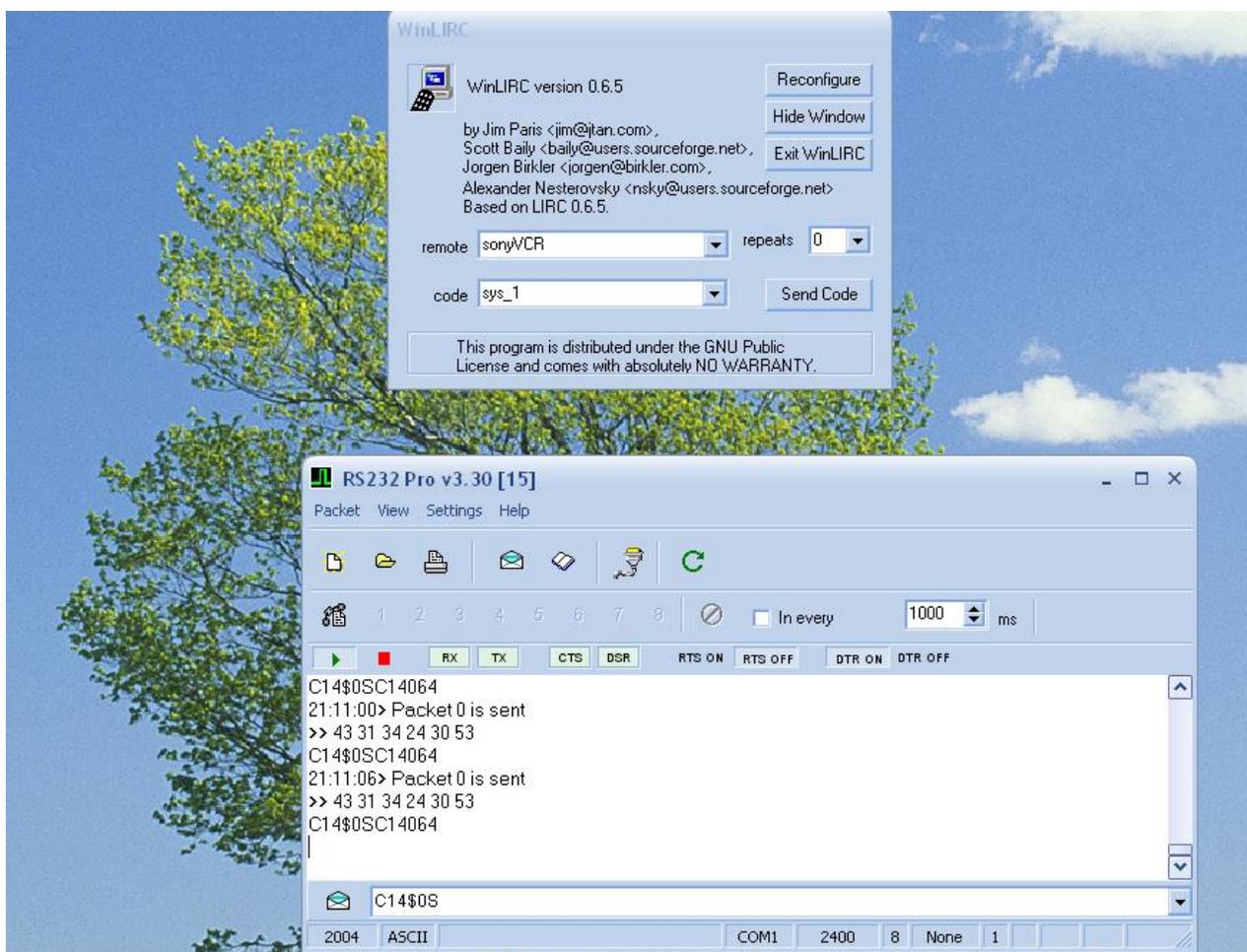


Рис.68

Не удовлетворяют меня только два момента:

1. Во время паузы между приемами ИК команд иногда запрос по сети «подвешивает» модуль в нераспознанном месте. Помогло сокращение времени этой паузы.
2. Коды с пульта плохо согласуются с моими ожиданиями.

## Хобби-электроникс 2. Умный дом.

Вот сравнение кодов, прочитанных с помощью WinLIRC в режиме распознавания, и кодов, которые транслирует модуль в ответ на запрос по сети RS485:

Номер канала 1:

Модуль возвращает номер команды 001 – 00000001 (C14001 – в ответ на запрос статуса).

7F2h - 1111110010 (инверсия 00000001101), без последних трех бит 0000001.

Номер канала 2:

Прочитано модулем 129 - 10000001

3F2h - 0111110010 (инверсия 10000001101), без последних трех бит 10000001.

Номер канала 3:

Прочитано модулем 65 - 10000001

5F2h - 1011110010 (инверсия 01000001101), без последних трех бит 10000001

Номер канала 6:

Прочитано модулем 161 - 10100001

2F2h - 0101110010 (инверсия 10100001101), без последних трех бит 10100001.

Номер канала 9:

Прочитано модулем 017 - 10001

772h - 1110110010 (инверсия 00010001101), без последних трех бит 00010001.

power:

Прочитано модулем 169 - 10101001

2B2h - 1010110010 (инверсия 0101001101), без последних трех бит 0101001.

Команда включения канала 1 в записи WinLIRC - 7F2h, но я определяю единицу и ноль, как это описано в технической информации о кодах Sony, которой располагаю, а в WinLIRC сделано наоборот. Не суть важно, можно инвертировать код. Кроме того, я использую 8и битовый код, а оригинальный код 11и битовый. Тоже не страшно, отбросим три последних бита. Смущающим меня обстоятельством служит то, что команды каналов 1 и 2 не совпадают, как я ожидал бы после превращений с инверсией и отбрасыванием. Аналогичная история с командой power. Пока я не понимаю, как это происходит. Хотя положительная сторона в этом есть – больше различных команд.

*Шутка калькулятора, который отбрасывает первые нули!!*

Последнее, что я делаю, измученный борьбой с «собственной гениальностью» - проверяю работу модуля в заготовке под среду программирования. Модуль работает. Я использую два кода – номер включения канала 1 для тестирования команды, а канала 2 для включения лампы. Оба кода стабильно работают.

Думаю, пока следует остановиться на достигнутом. Основная задача – получить модуль считывания системных команд, которые стабильно распознавались бы системой, достигнута. При этом, как это и планировалось, используется старенький пульт от видеоманитона. А то, что что-то осталось не понятно, что ж... будем надеяться, что это позже прояснится.

*Самым непонятным для меня оказалось то, что попытка использовать третий светодиод для индикации включенного модуля – я несколько раз вытаскивал микросхему из панельки без отключения питающего напряжения –*

## Хобби-электроникс 2. Умный дом.

*эта простая попытка успехом не увенчалась. Я попробовал разные варианты включения. Ничего не получилось! Мистика!?*

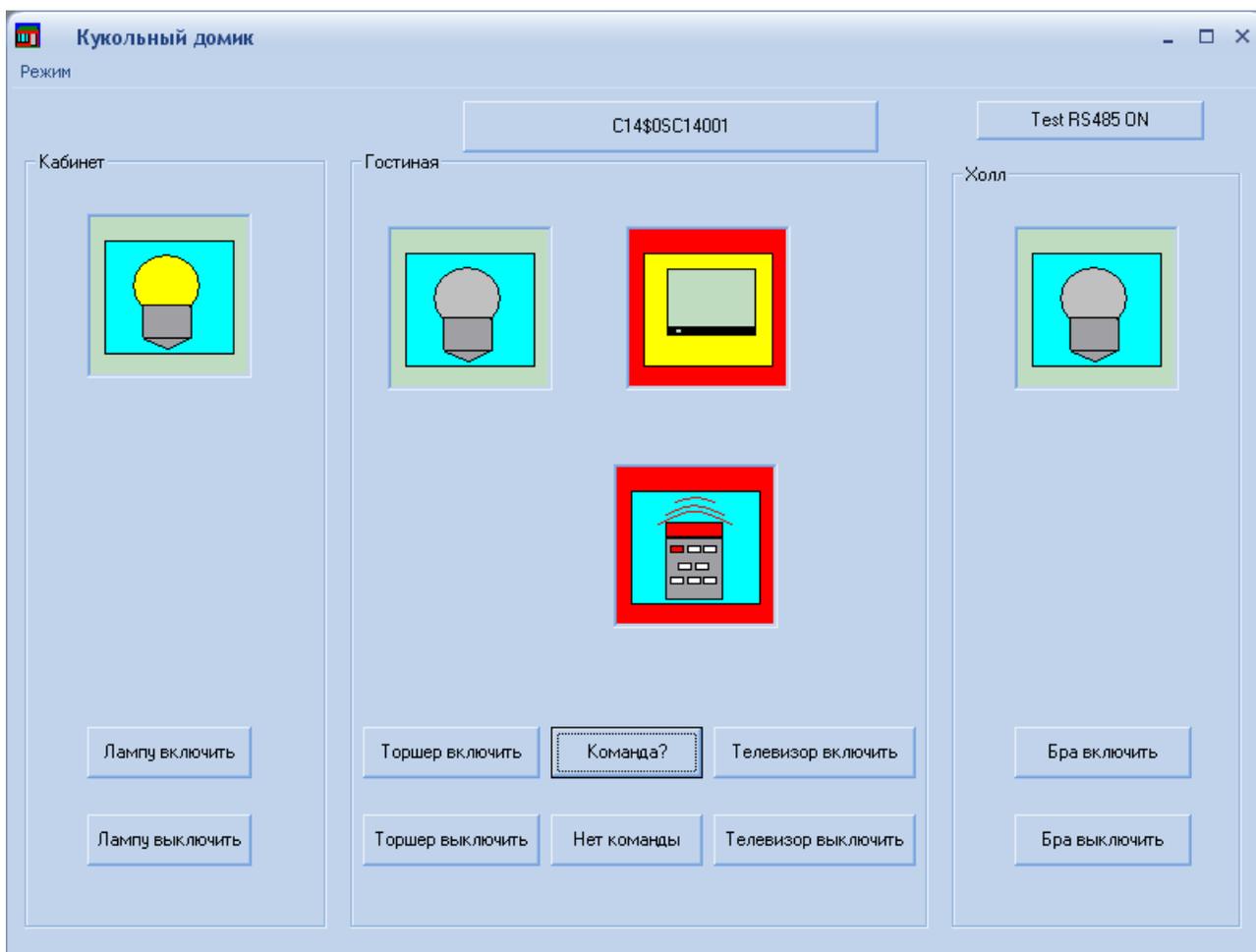


Рис.69

В данный момент программа модуля приема системных кодов для кукольного домика получилась такой.

**Файл *ir\_rec.h*:**

```
#define MODULNAMESIM 'C'
```

```
unsigned char getch(void);  
int init_comms();  
int ir_cmd ();  
int cmd ();  
void time();  
void time_cmd();  
int ir_stat();
```

*Файл ir\_rec.c:*

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "ir_rec_01.h"

unsigned char input;           // Считываем содержимое приемного регистра
unsigned char MOD_SIM1;       // первый символ адреса модуля
unsigned char MOD_SIM2;       // второй символ адреса модуля
unsigned char IRSIM1;
unsigned char IRSIM2;
unsigned char IRSIM3;
unsigned char command_reciev [6]; // Массив для полученной команды
int PHOTOCOME = 0;           // Флаг активности фотоприемника
int MOD_ADDR;                // Заданный адрес модуля, как число
unsigned char IRCOMMAND = 0;
int sim1_num = 0;
int sim2_num = 0;
int sim_end_num = 0;
int MOD_NUM;                 // Полученный адрес модуля, как число
int i;
int k;
int l;
int m;

unsigned char getch()
{
    while((!RCIF)&(RB3 == 1))
        /* устанавливается, когда регистр не пуст */
        continue;
    return RCREG;
}

/* вывод одного байта */
void putch(unsigned char byte)
{
    while(!TXIF)
        /* устанавливается, когда регистр пуст */
        continue;
    TXREG = byte;
}

int init_comms()             // Инициализация модуля
{
    PORTA = 0x00;
    CMCON = 0x7;            // Настройка портов А и В
    TRISA = 0x00;
}
```

## Хобби-электроникс 2. Умный дом.

```
TRISB = 0xFE;

PCON = 0xFF;           // Тактовая частота 4 МГц

RCSTA = 0b10010000;   // настройка приемника
TXSTA = 0b00000110;   // настройка передатчика
SPBRG = 0x68;         // настройка режима приема-передачи 2400, N,8,1

INTCON = 0x0;         // запретить прерывания
RB0 = 0x0;            // Выключим передатчик драйвера RS485

PIE1 = 0x0;           // Настройка таймера 1, запрет прерывания
T1CON = 0x0;          // Выбор внутреннего генератора, бит 1 в ноль
TMR1H = 0x00;         // Обнулим таймер
TMR1L = 0x00;

IRCOMMAND = 0x0;

/* Определим номер модуля */
MOD_ADDR = PORTB;     // Номер модуля в старших битах
MOD_ADDR=MOD_ADDR>>4; // Сдвинем на четыре бита
}

/* Преобразуем символьный адрес в число*/
int sim_num_adr()
{
    sim_end_num = 0x0;
    MOD_SIM1 = command_reciev [1]; // первый символ номера
    MOD_SIM2 = command_reciev [2]; // второй символ номера
    MOD_SIM1 = MOD_SIM1 - 0x30;
    MOD_SIM2 = MOD_SIM2 - 0x30;
    sim_end_num = MOD_SIM1*0x0A + MOD_SIM2;
    return sim_end_num;
}

int ir_cmd ()
{
    int b = 0;

    while (RB3 == 0) continue; // Ждем окончания заголовка
    for (b=0; b<8; b++)        //Обработаем наши импульсы
    {
        while (RB3 != 0)
        continue;             // Дождемся импульса
        time ();              // Включаем таймер 1
        while (!TMR1IF);     // Дождемся сброса таймера
        T1CON = 0x0;         // Выключаем таймер
    }
}
```

## Хобби-электроникс 2. Умный дом.

```
TMR1IF = 0x0;           // Сбросим флаг
if (RB3 == 0)           // Если низкий уровень, то "1"
{
    IRCOMMAND = IRCOMMAND +1;    // Запишем это
    time ();                    // Включаем таймер 1
    while (!TMR1IF);           // Дождемся сброса
    T1CON = 0x0;              // Выключаем таймер
    TMR1IF = 0x0;             // Сбросим флаг
} else                    // Высокий уровень, значит "0"
    IRCOMMAND = IRCOMMAND;    // Запишем это

if (b<7) IRCOMMAND = IRCOMMAND<<1; // Сместимся влево на бит
}
PHOTOCOME = 0x0;
RA0 = 0x0;
RA1=0x1;
for (m=0; m<3; ++m)        // Перестанем принимать ИК
    for (l=0; l<10000; ++l);
RA1 = 0x0;
}

int cmd()
{
    if (command_reciev [5] = 'S') ir_stat();
}

void time() // Таймер 1 для чтения ИК
{
    TMR1H = 0xFD;           // Установка числа циклов до переполнения
    TMR1L = 0x43;           // FFFF минус 700 (2BCh)
    T1CON = 0x1;           // Включение таймера
}

int ir_stat()
{
    int ircom;
    int ircom1;
    int ircom2;
    int ircom3;

    command_reciev[0] = 'C';
    command_reciev[1] = MOD_SIM1+0x30;
    command_reciev[2] = MOD_SIM2+0x30;

    ircom = IRCOMMAND;      // Преобразуем команду в символы
    ircom1 = ircom/0x64;
```

## Хобби-электроникс 2. Умный дом.

```
    IRSIM1 = ircom1 + 0x30;
    ircom2 = (ircom - ircom1*0x64)/0xA;
    IRSIM2 = ircom2 + 0x30;
    ircom3 = (ircom - ircom1*0x64 - ircom2*0xA);
    IRSIM3 = ircom3 + 0x30;

    if (IRCOMMAND == 0)
    {
        command_reciev[3] = '#';           // Команда не менялась
        command_reciev[4] = 'f';
        command_reciev[5] = 'f';

        CREN = 0x0;                       // Запрещаем прием
        RB0 = 0x1;                         // Переключим драйвер RS485 на передачу
        TXEN = 0x1;                       // разрешаем передачу
        for (i=0; i<6; ++i)    putchar(command_reciev[i]);
        for (i=0; i<1000; i++);           // Задержка для вывода
        for (i=0; i<6; ++i) command_reciev [i] = ' ';
        RB0 = 0x0;                         // Выключаем драйвер RS485 на передачу
        TXEN = 0x0;                       // запрещаем передачу
        CREN = 0x1;                       // Разрешаем прием

    } else                                // За время между двумя запросами пришла ИК команда
    {
        command_reciev[3] = IRSIM1;
        command_reciev[4] = IRSIM2;
        command_reciev[5] = IRSIM3;

        CREN = 0x0;                       // Запрещаем прием
        RB0 = 0x1;                         // Переключим драйвер RS485 на передачу
        TXEN = 0x1;                       // разрешаем передачу
        for (i=0; i<6; ++i)    putchar(command_reciev[i]);
        for (i=0; i<1000; i++);           // Задержка для вывода
        for (i=0; i<6; ++i) command_reciev [i] = ' ';
        RB0 = 0x0;                         // Выключаем драйвер RS485 на передачу
        TXEN = 0x0;                       // запрещаем передачу
        CREN = 0x1;                       // Разрешаем прием
    }
    IRCOMMAND = 0x0;                      // Мы передали команду, она больше не нужна
}

/* Начнем работать */

void main(void)
{
    init_comms();                        // Инициализация модуля
```

## Хобби-электроникс 2. Умный дом.

```
for (i=0; i<6; ++i) command_reciev [i] = ' ';
command_reciev [0] = 'C';

MOD_ADDR = PORTB; // Прочитаем и преобразуем номер модуля
MOD_ADDR=MOD_ADDR>>4; // Номер модуля в старших битах
RA2 = 0x1; // Сдвинем на четыре бита
// Чтобы видно, что модуль включен
// Так и не получилось

// Начинаем работать
start: if (RB3) // Нет ИК-сигнала
{
    PHOTOCOME = 0x0;
    RA0 = 0x00;
}
if (!RB3) // Появился ИК-сигнал
{
    for (k=0; k<30; ++k); // Поставим задержку
    if (!RB3) // ИК-сигнал не пропал?
    {
        PHOTOCOME = 0x1;
        RA0 = 0x01;
    }
} else
{
    PHOTOCOME = 0x0;
    RA0 = 0x00;
}

while (PHOTOCOME == 1)
{
    /* Обрабатываем ИК-команду */
    ir_cmd ();
break;
}

/* Нет ИК-сигнала, проверим сеть */

CREN =0x1;
input = getch();
switch (input)
{
case 'C': // Если обращение к фото модулю
for (i=1; i<6; ++i) // Запишем команду в массив
{
    input = getch();
    command_reciev [i] = input;
}

MOD_NUM = sim_num_adr(); // Чтение из сети
```

```
if (MOD_NUM != MOD_ADDR) break;    // Если не наш адрес
else
    if (command_reciev [3] = '$') cmd();        // Если команда
        default: goto start;
    }
goto start;
}
```

Пойдем дальше.

## **Модуль излучения инфракрасных кодов**

Модуль должен получать команду по сети и излучать ИК код. ИК коды в «собственном формате» системы могут храниться в EEPROM (можно использовать внешнюю энергонезависимую память). Особенно актуально, с дополнительной памятью, если мы хотим модуль излучения инфракрасных кодов превратить в универсальный пульт управления, запоминая коды других пультов. Для этой цели можно добавить к модулю фотоприемник, и подпрограмму считывания-запоминания. Но вернемся к конкретной задаче.

Здесь придется потрудиться над представлением этого «собственного формата».

EEPROM хранит 128 байт. Если мы будем использовать модуль излучателя ИК кодов для одного устройства, то попробуем прикинуть, сколько управляющих команд нужно для него в системе.

Возьмем, например, DVD-проигрыватель. Какие команды нужны пользователю, чтобы смотреть фильмы?

Проиграть, Стоп, Предыдущий фрагмент, Следующий фрагмент, Громче, Тише, Выключить звук, команды курсора Влево, Вправо, Вверх, Вниз, Ввод, Меню.

Для ровного счета возьмем 15 команд. Таким образом, на одну команду мы можем потратить 8 байт. Много это или мало? Ответ на вопрос следовало бы искать в следующем разделе - «Модуль считывания инфракрасных кодов».

В данный момент я решаю, что нет смысла хранить коды в модуле воспроизведения ИК команд, будем хранить их на компьютере. Примем формат команды следующего вида:

| Устр-во | Команда | Повторов | Коэф-т | Частота | Импульс | Пауза и т.д. | EOF    |
|---------|---------|----------|--------|---------|---------|--------------|--------|
| ? байт  | ? байт  | байт     | байт   | 2 байта | 2 байта | 2 байта      | Байт 0 |

При этом в отличие от предыдущих модулей, модуль излучения должен получать по сети не только команду, но и данные, следующие за командой. Если мы определим префикс модуля с помощью латинской буквы «I», то команда может выглядеть следующим образом:

Ixx\$nP и байты данных.

где n – количество повторов. Я думаю, что повторов от 0 до 9 должно хватить.

## Хобби-электроникс 2. Умный дом.

Байты данных начнутся с байта коэффициента и закончатся нулевым байтом. Мы не знаем, сколько именно будет передано байт. Для определенности возьмем 50 байт. Значит, все данные будут переданы приблизительно через 50 мс. После приема данных модуль может начать воспроизведение ИК кода. Таким образом, задержка между нажатием на клавишу управления и отправкой управляющего ИК кода будет не более 0,1-0,2 секунд. Думаю, это не слишком много.

Осталось два момента, которые мне сейчас не ясны. Первый – хватит ли места в регистрах контроллера для хранения команды? В первых двух банках памяти 176 байт отведено для регистров общего назначения. Полагаю, этого хватит, но хранение в двух банках...

Второй момент, смущающий меня, относится к некоторой вероятности того, что передаваемые данные, а это байты, совпадут с видом команды. Не думаю, что это очень вероятно, но эту неприятность мы можем отслеживать на этапе записи ИК кода. Если, паче чаяния, это произойдет, воспользуемся коэффициентом пересчета, преобразовав данные к другому виду.

Теперь мы готовы начать разработку нового модуля.

За основу возьмем предыдущий модуль, у которого используем вывод RA6 на выход. К этому выводу порта мы будем подключать светодиод ИК диапазона (излучатель, или ИК эмиттер).

Как обычно, я создаю папку в рабочей папке MPLAB, которую называю ir\_trans, и в которую копирую все файлы предыдущего модуля. Теперь я открываю в программе MPLAB предыдущий проект, но из этой папки. Сохраняю его под новым именем, как ir\_trans, сохраняю под новым именем файл заголовков и программы, меняю их в менеджере проекта, включая файл todo.txt, настраиваю установки Debugger'a. Осталось поменять файлы сценария, удалить из папки все старые файлы, и можно начинать работу.

Вся эта процедура не обязательна. Можно открыть новый проект, но при этом не следует забывать, что все настройки нового проекта следует сделать заново. Задать рабочую тактовую частоту контроллера, задать скорость выполнения анимации. Иначе можно наступить на те грабли, которые подстерегали в самом начале работы.

В отличие от предыдущих модулей данный модуль при настройке должен прочитывать из файла input.txt последовательность данных. Вернемся к рекомендациям изготовителя микроконтроллера PIC16F628A, и после строки команды в файле input.txt впишем данные в виде шестнадцатеричных чисел, записанных в столбик, и завершаемых командой 0Ah. Но сначала подготовим их. Возьмем код из раздела «Модуль считывания инфракрасных кодов».

pulse 2455 Заголовок

space 532

pulse 1291 Единица

space 506

## Хобби-электроникс 2. Умный дом.

pulse 664 Ноль  
space 533  
pulse 1266 Единица  
space 533  
pulse 665 Ноль  
space 591  
pulse 1211 Единица  
space 533  
pulse 685 Ноль  
space 515  
pulse 693 Ноль  
space 526  
pulse 1271 Единица  
space 506  
pulse 1265 Единица  
space 533  
pulse 666 Ноль  
space 557  
pulse 1241 Единица  
space 533  
pulse 664

input.txt

"R03\$0N"

"I03\$5P"

|    |  |
|----|--|
|    | Команда обращения к модулю трансляции ИК команды           |
| 01 | Количество повторов  |
| 01 | Коэффициент  |
| 25 | Частота несущей 37 кГц                                     |
| 97 | Младший байт импульса заголовка (шестнадцатеричного числа) |
| 09 | Старший байт импульса заголовка                            |
| 14 | Младший байт паузы   |
| 02 | Старший байт паузы   |
| E3 | Младший байт импульса единицы (1251 мкс – среднее)         |
| 04 | Старший байт импульса единицы                              |
| 4A | Младший байт паузы единицы (586 мкс – среднее)             |
| 02 | Старший байт паузы единицы                                 |
| 4A | Младший байт импульса нуля (586 мкс – среднее)             |
| 02 | Старший байт импульса нуля                                 |
| 4A | Младший байт паузы нуля (586 мкс – среднее)                |
| 02 | Старший байт паузы нуля                                    |
| E3 | Младший байт импульса единицы (1251 мкс – среднее)         |
| 04 | Старший байт импульса единицы                              |
| 4A | Младший байт паузы единицы (586 мкс – среднее)             |
| 02 | Старший байт паузы единицы                                 |
| 4A | Младший байт импульса нуля (586 мкс – среднее)             |
| 02 | Старший байт импульса нуля                                 |

## Хобби-электроникс 2. Умный дом.

|    |  |
|----|--|
| 4A | Младший байт паузы нуля (586 мкс – среднее)        |
| 02 | Старший байт паузы нуля                            |
| E3 | Младший байт импульса единицы (1251 мкс – среднее) |
| 04 | Старший байт импульса единицы                      |
| 4A | Младший байт паузы единицы (586 мкс – среднее)     |
| 02 | Старший байт паузы единицы                         |
| 4A | Младший байт импульса нуля (586 мкс – среднее)     |
| 02 | Старший байт импульса нуля                         |
| 4A | Младший байт паузы нуля (586 мкс – среднее)        |
| 02 | Старший байт паузы нуля                            |
| 4A | Младший байт импульса нуля (586 мкс – среднее)     |
| 02 | Старший байт импульса нуля                         |
| 4A | Младший байт паузы нуля (586 мкс – среднее)        |
| 02 | Старший байт паузы нуля                            |
| E3 | Младший байт импульса единицы (1251 мкс – среднее) |
| 04 | Старший байт импульса единицы                      |
| 4A | Младший байт паузы единицы (586 мкс – среднее)     |
| 02 | Старший байт паузы единицы                         |
| E3 | Младший байт импульса единицы (1251 мкс – среднее) |
| 04 | Старший байт импульса единицы                      |
| 4A | Младший байт паузы единицы (586 мкс – среднее)     |
| 02 | Старший байт паузы единицы                         |
| 4A | Младший байт импульса нуля (586 мкс – среднее)     |
| 02 | Старший байт импульса нуля                         |
| 4A | Младший байт паузы нуля (586 мкс – среднее)        |
| 02 | Старший байт паузы нуля                            |
| E3 | Младший байт импульса единицы (1251 мкс – среднее) |
| 04 | Старший байт импульса единицы                      |
| 4A | Младший байт паузы единицы (586 мкс – среднее)     |
| 02 | Старший байт паузы единицы                         |
| 4A | Младший байт импульса нуля (586 мкс – среднее)     |
| 02 | Старший байт импульса нуля                         |
| 00 | Завершающий нулевой байт                           |

Конечно, файл не должен содержать моих пометок. Я усреднил значения, использовал одинаковые усредненные числа. Прав ли я? Это можно будет проверить только при воспроизведении. На данном этапе это не играет роли. Позже, возможно, мы напишем программу, которая будет упаковывать считанные данные. Посмотрим.

Теперь нам нужно позаботиться о том, где мы будем хранить полученные данные. Для этой цели используем массив беззнаковых символов.

```
unsigned char ir_cmd [53];
```

Я задал массив длиной в 53 байта. Вообще-то мы не знаем, какой длины получится массив, но можно вернуться к этому позже, когда узнаем.

Основной переделке подвергается функция прочитывания команды. За последним

## Хобби-электроникс 2. Умный дом.

символом команды 'P' следуют данные. Количество повторов мы получаем перед командой. Но мы оставили под это байт данных. Поэтому пока будем игнорировать количество повторов в команде.

Ожидаем 'P', начинаем заполнение массива.

Основная часть программы:

```
// Начинаем работать
/* Ждем прихода команды и данных */
start: while(input != MODULNAMESIM) input = getch(); // Ждем обращения к модулю

        cmd ();                                // Обработаем сетевую команду
        goto start;
```

Обработка сетевой команды:

```
int cmd()
{
    MOD_NUM = sim_num_adr();                // Чтение из сети (файла)
    if (MOD_NUM == MOD_ADDR)                // Если наш адрес модуля
    {
        while (input != 'P')    input = getch();    // Ждем
            if (input == 'P')    // Если символ завершения команды
            {
                // Принимаем данные
                while (i<54)
                {
                    input = getch();
                    ir_cmd[i] = input;
                    ++i;
                }
            }
    }
}
```

Перед запуском программы я не забываю установить в высокое состояние биты, имитирующие переключатель адреса модуля (RB4, RB5 в окне Stimulus Controller).

*Первые грабли, на которые я наступаю – программа не работает. После первого символа в регистре приемника USART появляется 02. Это не номер модуля, дальше ничего не получается. Не пытаюсь разобраться, просто добавляю в файл input.txt еще пару команд до команды обращения к модулю.*

```
"R02$1N"
"R01$1N"
"I03$5P"
```

## Хобби-электроникс 2. Умный дом.

Теперь программа работает, массив заполняется. Добавив его в наблюдение, можно посмотреть, как он заполняется, и есть ли печатные символы в наших данных.

После заполнения массива мы готовы передать ИК команду.

У нас три временных интервала

pulse 2455 Заголовок  
space 532  
pulse 1291 Единица

Несущая частота 37 кГц период ~27 мс. Через время ~13 мс будем менять состояние вывода RA6 порта А, если у нас в массиве записан импульс.

Можно, как в программе предыдущего модуля, использовать встроенные таймеры контроллера. Но для начала попробуем использовать простые циклы.

```
void ir_trns()
{
    i = 0;
    c = 3; // В первых трех байтах служебная информация
    while (ir_cmd[i] != 0) // Наш байт окончания команды
    {
        // Импульс
        b = ir_cmd[c+1]; // Сначала прочитаем старший байт
        b = b << 8; // Сместимся на один байт
        b = b + ir_cmd[c]; //Сложим старший и младший байт, получая длительность импульса
        c = c + 2; // Сместимся на два байта
        while (b != 0)
        {
            for (d=0;d<14;++d) RA6 = 1; // Теперь с частотой в 37 кГц будем
            for (d=0;d<14;++d) RA6 = 0; // создавать несущую частоту
            --b;
        }
        // Пауза
        b = ir_cmd[c+1];
        b = b << 8;
        b = b + ir_cmd[c]; // В этой переменной мы храним длительность паузы
        c = c + 2;
        while (b != 0)
        {
            for (d=0;d<28;++d) RA6 = 0; // Передаем паузу
            --b;
        }
    }
}
```

## Хобби-электроникс 2. Умный дом.

В глобальные переменные я добавил несколько индексных переменных. Их можно добавить локально, но глобальные переменные удобнее наблюдать.

```
int i = 0;  
int b = 0;  
int c = 0;  
int d = 0;
```

Пытаясь передать массив, я обнаружил, что по умолчанию он не обнуляется, поэтому в раздел инициализации модуля я добавляю обнуление массива. Проблема, которая возникает без этого обнуления в том, что мы записываем байт, а работаем с двух байтовыми числами. Нулевой байт записывается в конец массива, но второй байт не нулевой, тогда как условие `while (b != 0)`, где `b` – целое.

```
for (i=0; i<61; ++i) ir_cmd[i] = 0; // Обнуление массива.
```

Теперь, вроде бы все работает, но мне хочется посмотреть, что у меня транслируется излучателем, соединенным с выводом RA6. Программа позволяет воспользоваться программным логическим анализатором. Для этого я выделяю всю часть `void ir_trns()`, правой клавишей мышки вызываю выпадающее меню, в котором выбираю раздел `Add Filter-in Trace`. Теперь, во `View` основного меню добавляю к рабочему окну `Simulator Trace` и жду, когда заполненный массив начнет транслироваться.

Я ожидаю, что мне удастся увидеть весь передаваемый через RA6 код. Запускаю `Simulator Logic Analyzer`, в котором с помощью клавиши `Channels` вхожу в меню выбора отслеживаемых выводов. Затем выбираю RA6, и добавляю вывод в наблюдение – `Add`.

Вопреки желаемому, потратив достаточно много времени, получаю очередное напоминание о «граблях». Удастся зафиксировать только фрагменты. Пытаюсь добавить управление еще одним выводом порта А – RA0, устанавливая его в единицу, перед формированием импульса, и сбрасывая в ноль при паузе. Единственное, что получилось, это ряд фрагментов. Приблизительно таких:

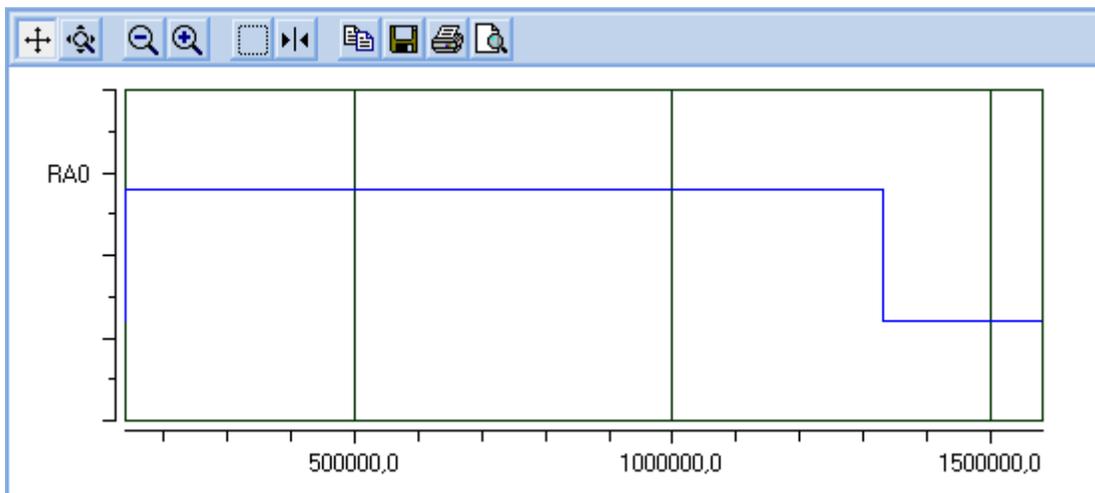


Рис.70

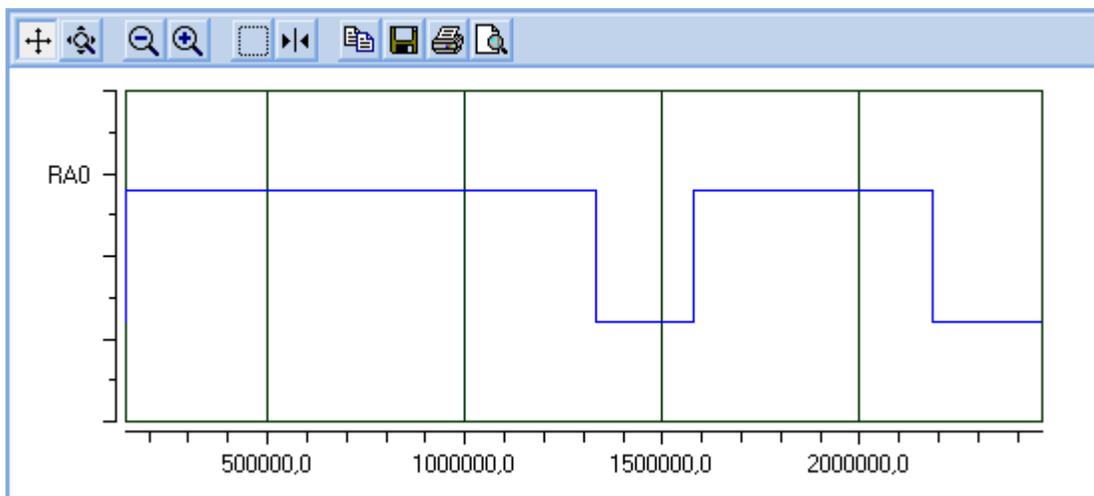


Рис.71

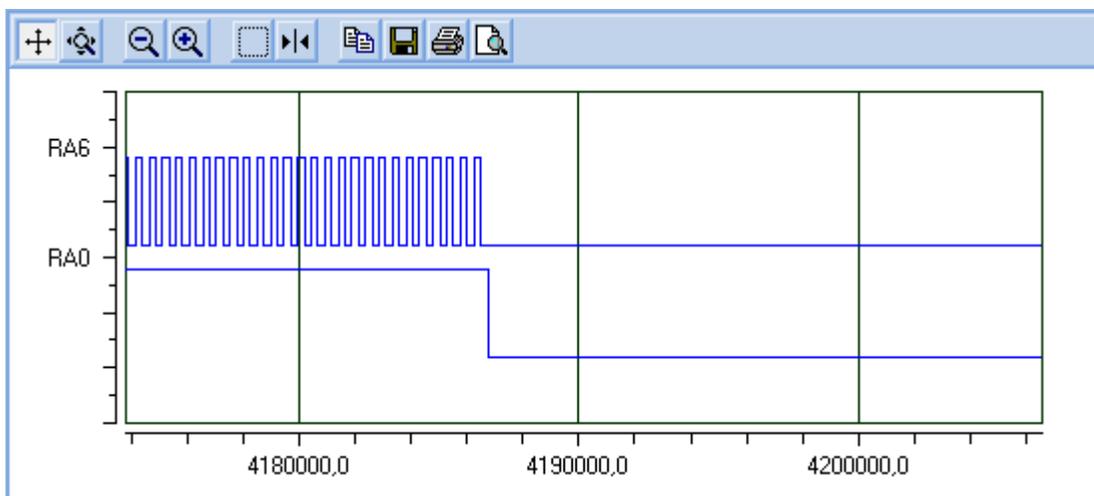


Рис.72

Первый из фрагментов (рис.70) относится к моменту завершения передачи заголовка. Второй (рис.71) – следом за заголовком передается единица. На последнем фрагменте видно, что передача несущей сменяется паузой. Не густо.

Оставлю программу модуля, как есть, удалив управление выводом RA0, и проверю работу модуля, когда соберу его. У меня нет уверенности, что частота несущей 37 кГц (или близка к ней), что длительности импульсов и пауз не требуют дополнительной калибровки. Пока я не могу сделать большего, оставляю все, как есть.

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "ir_trans.h"
```

```
unsigned char input;           // Считываем содержимое приемного регистра
unsigned char MOD_SIM1;       // Первый символ адреса модуля
unsigned char MOD_SIM2;       // Второй символ адреса модуля
int MOD_ADDR;                 // Заданный адрес модуля, как число
```

## Хобби-электроникс 2. Умный дом.

```
int MOD_ADDR;           // Заданный адрес модуля, как число
int sim1_num = 0;
int sim2_num = 0;
int sim_end_num = 0;
int MOD_NUM;           // Полученный адрес модуля, как число
int i = 0;
int b = 0;
int c = 0;
int d = 0;
unsigned char ir_cmd [60];

unsigned char getch()
{
    while(!RCIF)        /* устанавливается, когда регистр не пуст */
        continue;
    return RCREG;
}

/* вывод одного байта */
void putch(unsigned char byte)
{
    PORTB = 1;          // Переключим драйвер RS485 на передачу
    TXEN = 1;          // Разрешаем передачу
    while(!TXIF)       /* устанавливается, когда регистр пуст */
        continue;
    TXREG = byte;
}

int init_comms()       // Инициализация модуля
{
    PORTA = 0x0;        // Настройка портов А и В
    CMCON = 0x7;
    TRISA = 0x80;
    TRISB = 0xF6;

    RCSTA = 0x90;      // Настройка приемника
    TXSTA = 0x4;       // Настройка передатчика
    SPBRG = 0x16;      // Настройка режима приема-передачи

    INTCON=0;          // Запретить прерывания
    PORTB = 0;         // Выключим передатчик драйвера RS485

    /* Определим номер модуля */
    MOD_ADDR = PORTB;  // Номер модуля в старших битах
    MOD_ADDR=MOD_ADDR>>4; // Сдвинем на четыре бита
    for (i=0; i<61; ++i) ir_cmd[i] = 0;
```

## Хобби-электроникс 2. Умный дом.

```
}

/* Преобразуем символьный адрес в число*/
int sim_num_adr()
{
    sim_end_num = 0;
    sim1_num = getch();    // Чтение первого символа номера
    MOD_SIM1 = sim1_num;   // Сохраним первый символ
    sim2_num = getch();    // Чтение второго символа номера
    MOD_SIM2 = sim2_num;   // Сохраним второй символ
    sim1_num = sim1_num - 0x30;
    sim2_num = sim2_num - 0x30;
    sim_end_num = sim1_num*0x0A + sim2_num;
    return sim_end_num;
}

int cmd()
{
    MOD_NUM = sim_num_adr();    // Чтение из сети (файла)
    if (MOD_NUM == MOD_ADDR)    // Если наш адрес модуля
    {
        while (input != 'P')    input = getch();    // Ждем
            if (input == 'P')    // Если символ завершения команды
            {
                i=0;
                // Принимаем данные
                while (i<54)
                {
                    input = getch();
                    ir_cmd[i] = input;
                    ++i;
                }
            }
    }
}

void ir_trns()
{
    i = 0;
    c = 3;
    while (ir_cmd[i] != 0)    // Наш байт окончания команды
    {
        // Импульс
        b = ir_cmd[c+1];
        b = b<<8;
        b = b + ir_cmd[c];
    }
}
```

## Хобби-электроникс 2. Умный дом.

```
c = c + 2;
RA0 = 1;
while (b != 0)
{
    for (d=0;d<14;++d) RA6 = 1;
    for (d=0;d<14;++d) RA6 = 0;
    --b;
}
// Пауза
b = ir_cmd[c+1];
b = b<<8;
b = b + ir_cmd[c];
c = c + 2;
RA0 = 0;
while (b != 0)
{
    for (d=0;d<28;++d) RA6 = 0;
    --b;
}
}

/* Начнем работать */

void main(void)
{
    init_comms();           // Инициализация модуля
    // Прочитаем и преобразуем номер модуля
    MOD_ADDR = PORTB;      // Номер модуля в старших битах
    MOD_ADDR=MOD_ADDR>>4;  // Сдвинем на четыре бита

    // Начинаем работать
    /* Ждем прихода команды и данных */
start: while(input != MODULNAMESIM) input = getch(); // Ждем обращения к модулю

    cmd ();                // Обработаем сетевую команду
    ir_trns();              // Отправим ИК команду на излучатель
    goto start;
}

}
```

Все у меня получается разумно и красиво, логично и грамотно. Ай, да я!

Смущает одно. Когда я проверяю времена, соответствующие картинкам анализатора, то получаю совсем не то, что хотелось бы. В моем приборном арсенале нет записывающего осциллографа. Представив, как я пытаюсь поймать и измерить последовательность импульсов ИК команды обычным осциллографом (если вам не приходилось, попробуйте), я решаю вернуться к MPLAB. Верю я тем, кто создал эту замечательную программу! И,

## Хобби-электроникс 2. Умный дом.

начинаю понимать, что совсем «Не – Ай, да я!». Где-то я ошибаюсь.

Пишем простую программку:

```
#include <pic16f62ха.h>
#include <stdio.h>

int d = 0;
int c = 0;

int  init_comms()      // Инициализация модуля
{
PORTA = 0x0;          // Настройка портов А и В
CMCON = 0x7;
TRISA = 0x80;
TRISB = 0xF6;

RCSTA = 0x90;        // Настройка приемника
TXSTA = 0x4;         // Настройка передатчика
SPBRG = 0x16;        // Настройка режима приема-передачи

INTCON=0;            // Запретить прерывания
PORTB = 0;           // Выключим передатчик драйвера RS485
}

void main(void)
{
    init_comms();    // Здесь поставим точку останова
                    // Начинаем работать

start:              for (d=0;d<2;++d) RA6 = 1;
                    for (c=0;c<2;++c) RA6 = 0;
                    goto start;
}
}
```

Настроим, установив тактовую частоту контроллера 4 МГц, все, что настраиваем обычно. Выделим с помощью Add Filter-in Trace строки выделенные цветом, и в пошаговом режиме сделаем четыре шага по программе. Посмотрим, что нам рисует логический анализатор.

## Хобби-электроникс 2. Умный дом.

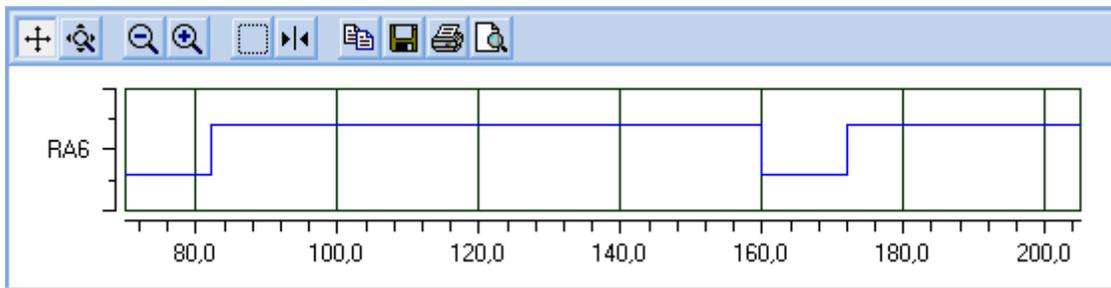


Рис.73

Время, определяемое в циклах команд контроллера, для состояния «0» около 75 циклов, или 75 мкс. Но я устанавливаю, как я полагал, вывод RA6 в ноль на один цикл. В состоянии «единица», что по моим предположениям тоже должно соответствовать 1 мкс, я «зависаю» на 78 мкс. Проверяю маркером:

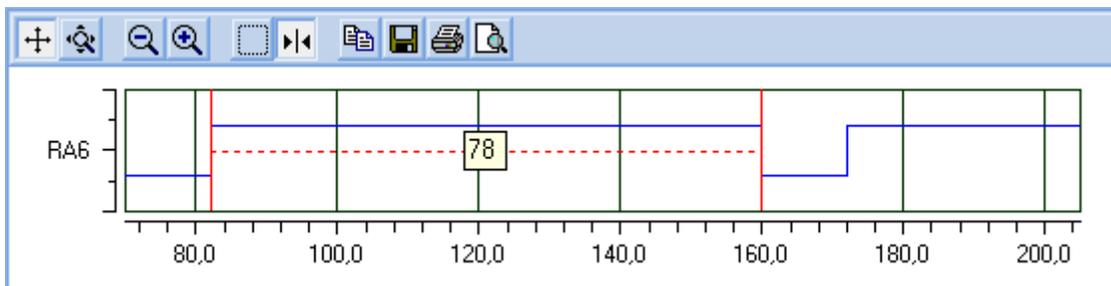


Рис.74

Так и есть.

Не нравится мне это, но пока я не понимаю, в чем дело?

Включив в Simulator Trace режим просмотра в инженерных единицах, я получаю следующую картинку за один проход моих циклов for (Рис.75). Это не полная картина трассировки, строки следуют до номера 45, что соответствует моменту времени в 115 мкс от начала процесса. Логический анализатор, соответственно, рисует:

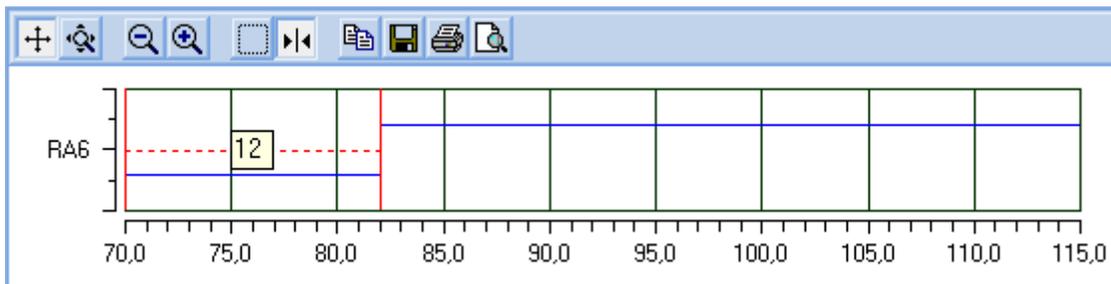


Рис.75

И я медленно, но начинаю понимать...

Это в своем правильном логическом мире, описанном языком «С», я за одну команду успеваю перейти от «0» к «1» и обратно. В реальном мире контроллера это занимает ровно столько времени, сколько требует команд. На 82й микросекунде текущего времени я устанавливаю RA6 в единицу, что трассировщик отображает в 12й строке командой BSF 0x5,

## Хобби-электроникс 2. Умный дом.

0x6:

The screenshot displays the MPLAB IDE v7.22 interface. The main window shows assembly code with columns for Line, Addr, Op, Label, Instruction, SA, SD, DA, DD, and Time. The code includes instructions like CLRF, MOVF, XORLW, MOVWF, MOVLW, SUBWF, and GOTO. Below the code is a trace window showing the execution of the assembly code. The status bar at the bottom indicates the target device is PIC16F628A and the current address is 0x7de.

| Line | Addr | Op   | Label | Instruction    | SA   | SD | DA   | DD | Time       |
|------|------|------|-------|----------------|------|----|------|----|------------|
| 0    | 07DE | 01A2 |       | CLRF 0x22      | ---- | -- | ---- | -- | 70.000e-6  |
| 1    | 07DF | 01A3 |       | CLRF 0x23      | ---- | -- | ---- | -- | 71.000e-6  |
| 2    | 07E0 | 0823 |       | MOVF 0x23, W   | 0023 | 00 | 0023 | 00 | 72.000e-6  |
| 3    | 07E1 | 3A80 |       | XORLW 0x80     | W    | -- | W    | 00 | 73.000e-6  |
| 4    | 07E2 | 00F0 |       | MOVWF 0x70     | ---- | -- | 0070 | 80 | 74.000e-6  |
| 5    | 07E3 | 3080 |       | MOVLW 0x80     | W    | -- | W    | 80 | 75.000e-6  |
| 6    | 07E4 | 0270 |       | SUBWF 0x70, W  | 0070 | 80 | 0070 | 00 | 76.000e-6  |
| 7    | 07E5 | 3002 |       | MOVLW 0x2      | W    | -- | W    | 02 | 77.000e-6  |
| 8    | 07E6 | 1903 |       | BTFSC 0x3, 0x2 | 0003 | 1F | ---- | -- | 78.000e-6  |
| 9    | 07E7 | 0222 |       | SUBWF 0x22, W  | 0022 | 00 | 0022 | FE | 79.000e-6  |
| 10   | 07E8 | 1803 |       | BTFSC 0x3, 0   | 0003 | 18 | ---- | -- | 80.000e-6  |
| 11   | 07E9 | 0000 |       | NOP            | 0000 | 00 | ---- | -- | 81.000e-6  |
| 12   | 07EA | 1705 |       | BSF 0x5, 0x6   | 0005 | 00 | ---- | -- | 82.000e-6  |
| 13   | 07EB | 0AA2 |       | INCF 0x22, F   | 0022 | 00 | 0022 | 01 | 83.000e-6  |
| 14   | 07EC | 1903 |       | BTFSC 0x3, 0x2 | 0003 | 18 | ---- | -- | 84.000e-6  |
| 15   | 07ED | 0000 |       | NOP            | 0000 | 00 | ---- | -- | 85.000e-6  |
| 16   | 07EE | 2FE0 |       | GOTO 0x7e0     | ---- | -- | ---- | -- | 86.000e-6  |
| 17   | 07EF | 0000 |       | NOP            | ---- | -- | ---- | -- | 87.000e-6  |
| 18   | 07E0 | 0823 |       | MOVF 0x23, W   | 0023 | 00 | 0023 | 00 | 88.000e-6  |
| 19   | 07E1 | 3A80 |       | XORLW 0x80     | W    | -- | W    | 00 | 89.000e-6  |
| 20   | 07E2 | 00F0 |       | MOVWF 0x70     | ---- | -- | 0070 | 80 | 90.000e-6  |
| 21   | 07E3 | 3080 |       | MOVLW 0x80     | W    | -- | W    | 80 | 91.000e-6  |
| 22   | 07E4 | 0270 |       | SUBWF 0x70, W  | 0070 | 80 | 0070 | 00 | 92.000e-6  |
| 23   | 07E5 | 3002 |       | MOVLW 0x2      | W    | -- | W    | 02 | 93.000e-6  |
| 24   | 07E6 | 1903 |       | BTFSC 0x3, 0x2 | 0003 | 1F | ---- | -- | 94.000e-6  |
| 25   | 07E7 | 0222 |       | SUBWF 0x22, W  | 0022 | 01 | 0022 | FF | 95.000e-6  |
| 26   | 07E8 | 1803 |       | BTFSC 0x3, 0   | 0003 | 18 | ---- | -- | 96.000e-6  |
| 27   | 07E9 | 0000 |       | NOP            | 0000 | 00 | ---- | -- | 97.000e-6  |
| 28   | 07EA | 1705 |       | BSF 0x5, 0x6   | 0005 | 40 | ---- | -- | 98.000e-6  |
| 29   | 07EB | 0AA2 |       | INCF 0x22, F   | 0022 | 01 | 0022 | 02 | 99.000e-6  |
| 30   | 07EC | 1903 |       | BTFSC 0x3, 0x2 | 0003 | 18 | ---- | -- | 100.000e-6 |
| 31   | 07ED | 0000 |       | NOP            | 0000 | 00 | ---- | -- | 101.000e-6 |
| 32   | 07EE | 2FE0 |       | GOTO 0x7e0     | ---- | -- | ---- | -- | 102.000e-6 |
| 33   | 07EF | 0000 |       | NOP            | ---- | -- | ---- | -- | 103.000e-6 |
| 34   | 07E0 | 0823 |       | MOVF 0x23, W   | 0023 | 00 | 0023 | 00 | 104.000e-6 |
| 35   | 07E1 | 3A80 |       | XORLW 0x80     | W    | -- | W    | 00 | 105.000e-6 |
| 36   | 07E2 | 00F0 |       | MOVWF 0x70     | ---- | -- | 0070 | 80 | 106.000e-6 |

Рис.76

*Спасибо программистам, создавшим программу MPLAB – сидеть бы мне за осциллографом, чертыхаясь, долго и безуспешно!*

*Спасибо тем, кто учил меня писать программы в машинных кодах, когда я начал заниматься микропроцессорной техникой; тем, кто терпеливо пояснял мне, как правильно отсчитывать «branch» в командах!*

Посыпав голову пеплом, как и полагается, попробуем изменить ситуацию к лучшему. В первую очередь постараемся добиться того, чтобы циклы были одинаковы. Я, ведь, хочу получить меандр частотой 37 кГц.

Экран в данный момент выглядит так (я добавил внешний цикл for):

## Хобби-электроникс 2. Умный дом.

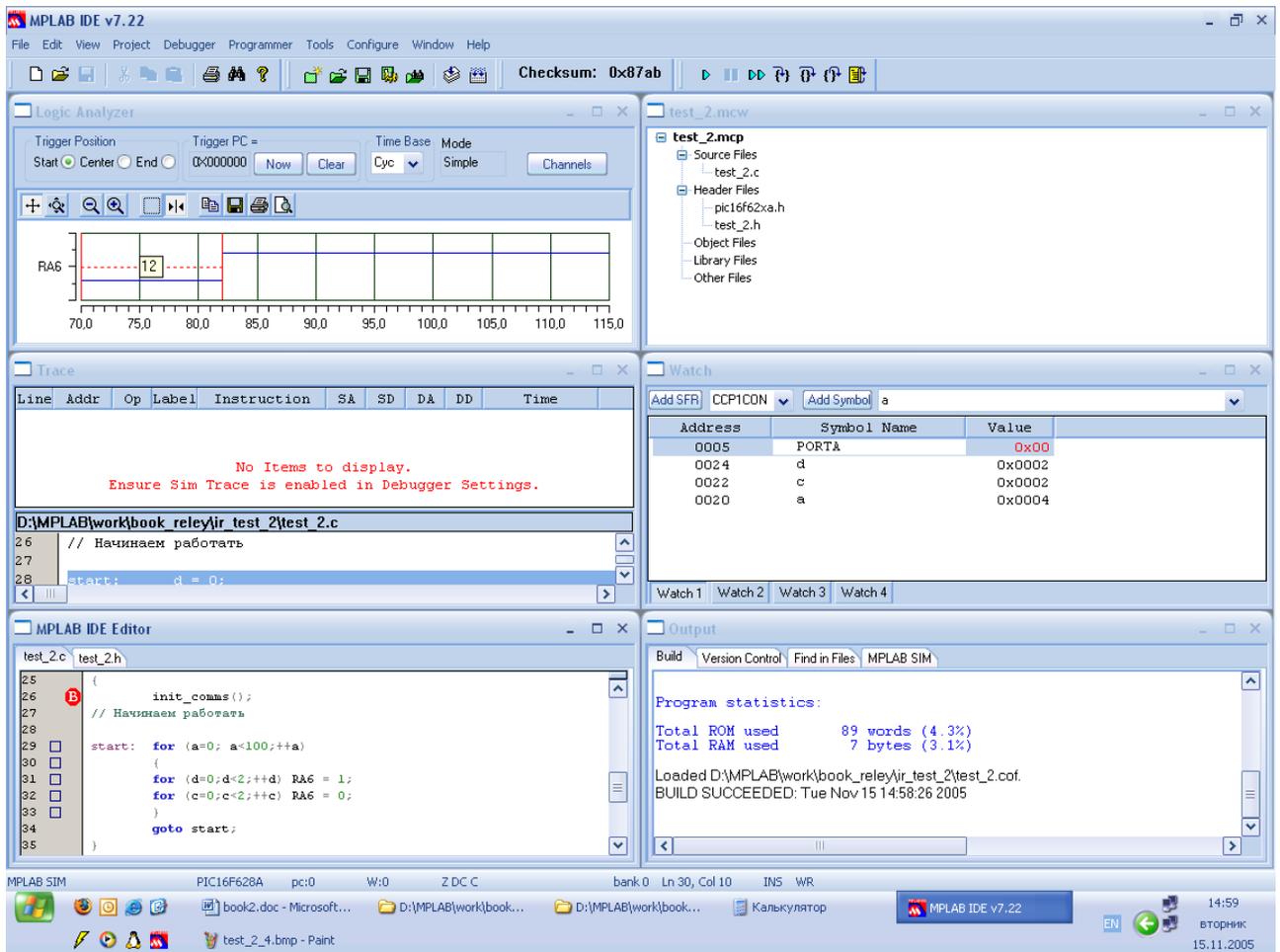


Рис.77

Изменив часть программы, и манипулируя числами, я несколько меняю положение дел:

```

start: a = 0;
        while (a<100)
        {
            for (d=0;d<3;++d) RA6 = 1;
            for (c=0;c<2;++c) RA6 = 0;
            ++a;
        }
        goto start;
    
```

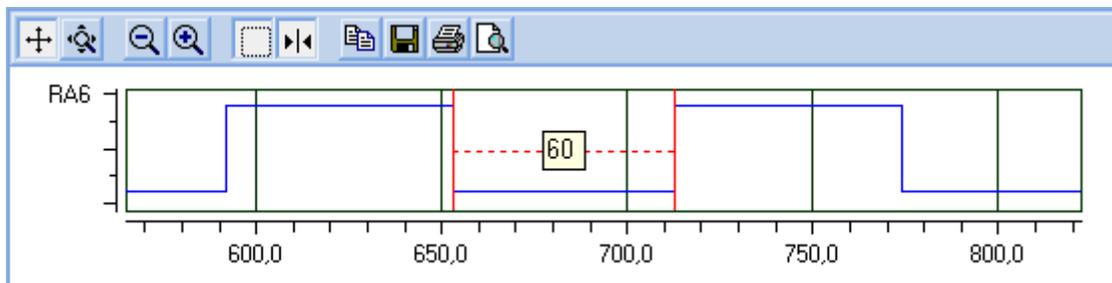


Рис.78

## Хобби-электроникс 2. Умный дом.

Меандр я получаю, но частота такой несущей около 8 кГц, тогда как мне нужна частота раз в пять выше. Пока я вижу несколько решений – повысить в пять раз тактовую частоту контроллера, сделать внешний генератор на частоту 37 кГц, или попробовать использовать внутренние резервы контроллера.

Попытка использовать таймер несколько улучшило ситуацию, но не в полной мере. Последний вариант, не затрагивающий коренных переделок всей программы или модуля, выглядит следующим образом:

```
start: a = 0;
for (a=0; a<100;++a)
    {
        RA6 = 1;
        RA6 = 1;

        RA6 = 0;
        RA6 = 0;
    }
goto start;
```

При этом сигнал получается:

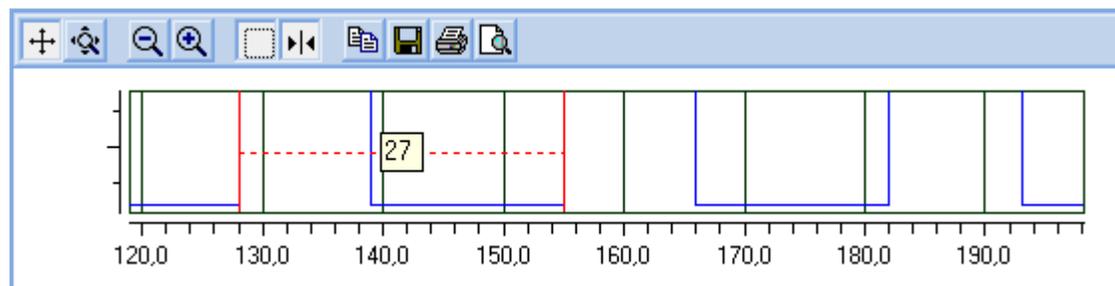


Рис.79

Период сигнала в 27 мкс соответствует частоте несущей 37 кГц. Это не самое изящное решение. В первую очередь теперь следует отказаться от изменения несущей частоты ИК команды «на лету». Модуль будет работать с фиксированной несущей. Пока это не самое главное. Посмотрим, можно ли поправить программу модуля?

*Не забыть бы, проверить модуль приема ИК кодов. Нет ли и там подобной*

## Хобби-электроникс 2. Умный дом.

*ошибки!*

С несущей частотой, вроде бы, положение несколько улучшилось, но времена посылок раз в 10 превышают реальные. Первое, что мне приходит в голову, это уменьшить в 10 раз времена в файле input.txt. В этом есть и привлекательная сторона – теперь массив, в котором я храню эти времена, становится не двух байтовым, а однобайтовым.

Но послылки все еще слишком длинные. Укорачиваем их еще вдвое, разделив значение времени уже в программе на два. И последнее, в плане «заметания мусора под ковер» - деление в файле input.txt всех значений на 1,38.

Мне не нравится, то, что я делаю. Но в итоге я получаю следующее:

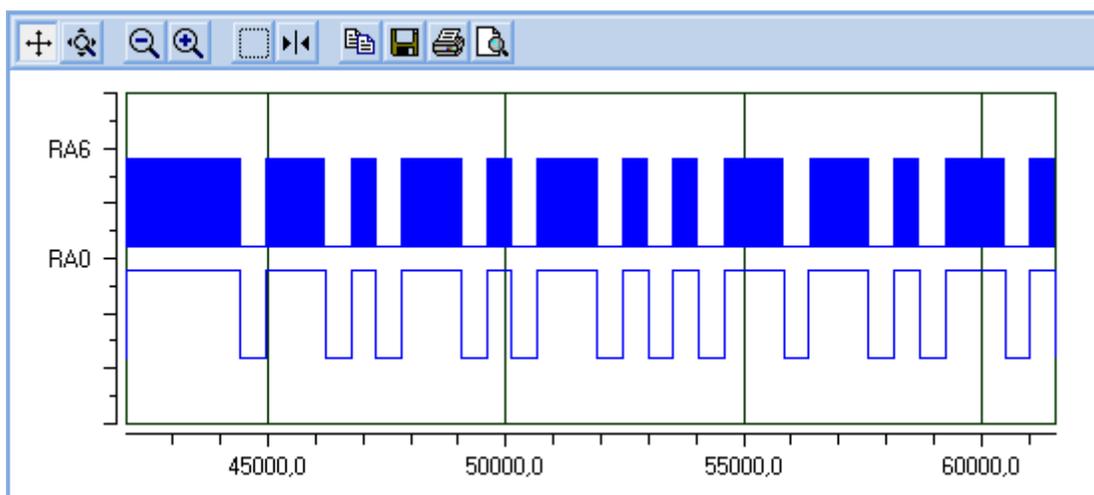


Рис.80

Это даже больше, чем я ожидал. Вернее, я ожидал этого в начале, но, не справившись с логическим анализатором, отказался от попыток получить эту картинку. Здесь несущая, если ее увеличить, выделив затемненный участок сигнала RA6 с помощью мышки и клавиши «□» (пятая слева), выглядит так:

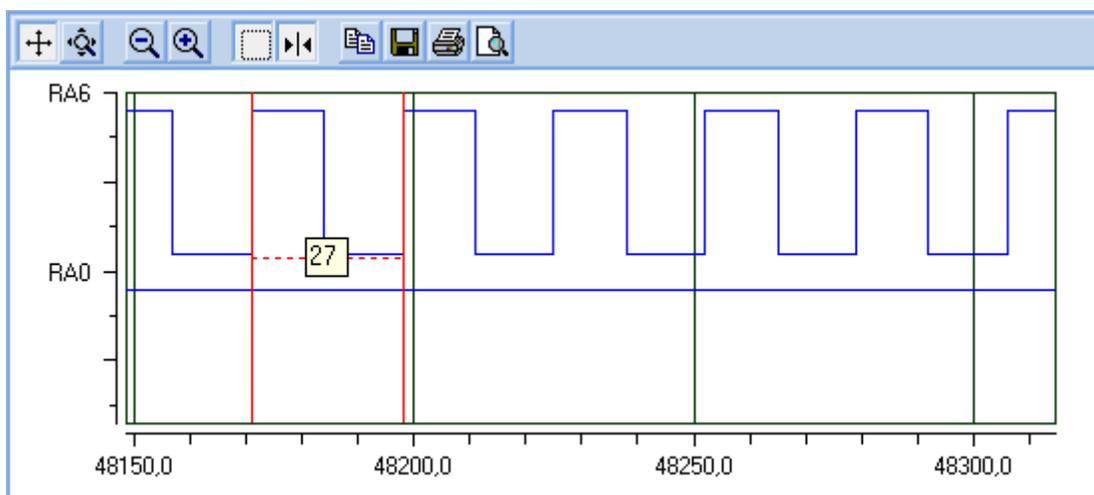


Рис.81

## Хобби-электроникс 2. Умный дом.

Вполне правдоподобно. А времена соответственно:

Заголовок

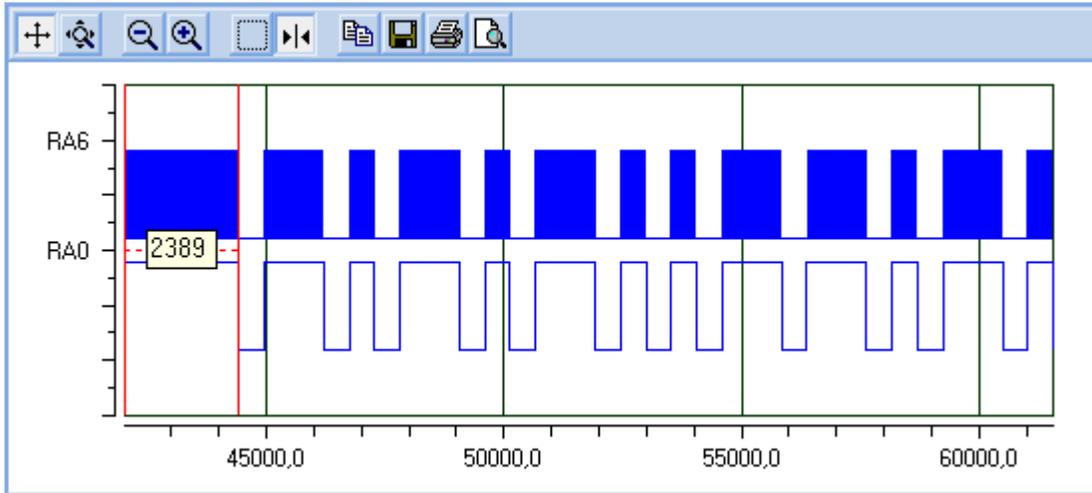


Рис.82

Посылка единицы

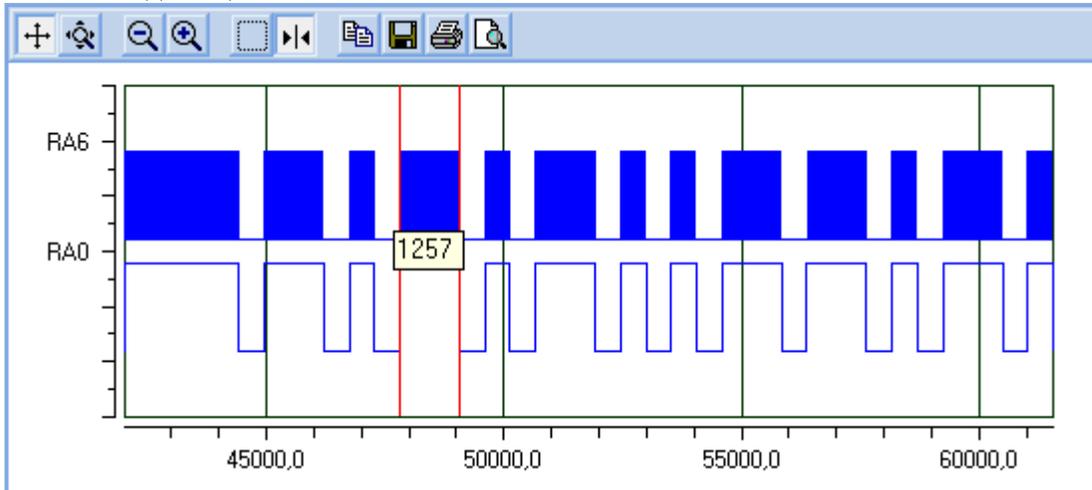


Рис.83

Пауза

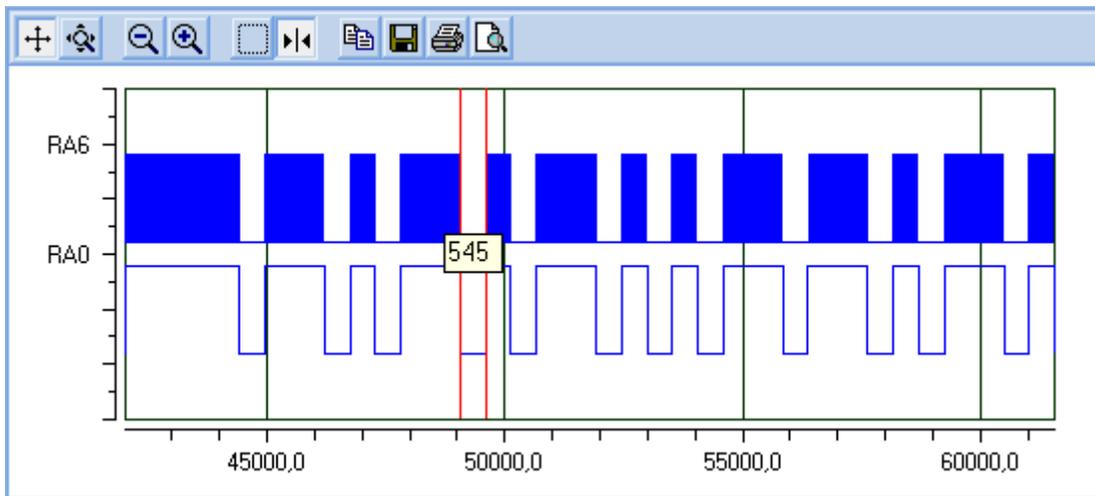


Рис.84

Времена близки к оригиналу. Последние грабли, на которые я умудряюсь наступить – в функции обработки массива

```
void ir_trns()
{
    i = 0;
    c = 3;
    while (ir_cmd[i] != 0)    // Наш байт окончания команды
```

забываю, заменить i на c. Вот так

```
void ir_trns()
{
    i = 0;
    c = 3;
    while (ir_cmd[c] != 0)    // Наш байт окончания команды
```

Без этого, закончив передачу ИК команды, программа меняет байт, из которого я начинаю позже вычитать, и, вычитая из нуля, я получаю опять ненулевой байт, который при проверке циклом while вместо завершения работы функции начинает «пороть чушь». Для отправки я распаковываю массив не с самого начала, а с третьего байта, поскольку в первых трех (от 0 до 2) у меня служебная информация (поэтому c=3), а в этом месте забыл поменять.

Итоговая программа выглядит так:

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "ir_trans.h"

unsigned char input;    // Считываем содержимое приемного регистра
unsigned char MOD_SIM1;    // Первый символ адреса модуля
unsigned char MOD_SIM2;    // Второй символ адреса модуля
int MOD_ADDR;    // Заданный адрес модуля, как число
int MOD_ADDR;    // Заданный адрес модуля, как число
```

## Хобби-электроникс 2. Умный дом.

```
int sim1_num = 0;
int sim2_num = 0;
int sim_end_num = 0;
int MOD_NUM;           // Полученный адрес модуля, как число
int i = 0;
int c = 0;
unsigned char ir_cmd [60];
unsigned char b = 0;

unsigned char getch()
{
    while(!RCIF)       /* устанавливается, когда регистр не пуст */
        continue;
    return RCREG;
}

/* вывод одного байта */
void putch (unsigned char byte)
{
    PORTB = 1;         // Переключим драйвер RS485 на передачу
    TXEN = 1;         // разрешаем передачу
    while(!TXIF)      /* устанавливается, когда регистр пуст */
        continue;
    TXREG = byte;
}

int  init_comms()     // Инициализация модуля
{
    PORTA = 0x0;       // Настройка портов А и В
    CMCON = 0x7;
    TRISA = 0x80;
    TRISB = 0xF6;

    RCSTA = 0x90;     // Настройка приемника
    TXSTA = 0x4;      // Настройка передатчика
    SPBRG = 0x16;     // Настройка режима приема-передачи

    INTCON=0;         // Запретить прерывания
    PORTB = 0;        // Выключим передатчик драйвера RS485

    /* Определим номер модуля */
    MOD_ADDR = PORTB; // Номер модуля в старших битах
    MOD_ADDR=MOD_ADDR>>4; // Сдвинем на четыре бита
    for (i=0; i<61; ++i) ir_cmd[i] = 0;
}
```

## Хобби-электроникс 2. Умный дом.

```
    /* Преобразуем символьный адрес в число */
int sim_num_adr()
{
    sim_end_num = 0;
    sim1_num = getch();      // Чтение первого символа номера
    MOD_SIM1 = sim1_num;    // Сохраним первый символ
    sim2_num = getch();      // Чтение второго символа номера
    MOD_SIM2 = sim2_num;    // Сохраним второй символ
    sim1_num = sim1_num - 0x30;
    sim2_num = sim2_num - 0x30;
    sim_end_num = sim1_num*0x0A + sim2_num;
    return sim_end_num;
}

int cmd()
{
    MOD_NUM = sim_num_adr();    // чтение из сети (файла)
    if (MOD_NUM == MOD_ADDR)    // Если наш адрес модуля
    {
        while (input != 'P')    input = getch();    // Ждем
            if (input == 'P')    // Если символ завершения команды
            {
                i=0;
                input = getch();
                // Принимаем данные
                while (input != 0x0)
                {
                    ir_cmd[i] = input/0x2;
                    input = getch();
                    ++i;
                }
            }
    }
}

void ir_trns()
{
    i = 0;
    c = 3;
    while (ir_cmd[c] != 0)    // Наш байт окончания команды
    {
        // Импульс
        b = ir_cmd[c];
        ++c;
    }
}
```

## Хобби-электроникс 2. Умный дом.

```
while (b != 0)
{
    RA6 = 1;
    RA6 = 1;

    RA6 = 0;
    --b;
}
// Пауза
b = ir_cmd[c];
++c;

while (b != 0)
{
    RA6 = 0;
    RA6 = 0;

    RA6 = 0;
    RA6 = 0;
```

## Хобби-электроникс 2. Умный дом.

```
        RA6 = 0;
        --b;
    }
}

/* Начнем работать */

void main(void)
{
    init_comms();           // Инициализация модуля
    // Прочитаем и преобразуем номер модуля
    MOD_ADDR = PORTB;      // Номер модуля в старших битах
    MOD_ADDR=MOD_ADDR>>4; // Сдвинем на четыре бита

    // Начинаем работать
    /* Ждем прихода команды и данных */
    start: while(input != MODULNAMESIM) input = getch(); // Ждем обращения к модулю

        cmd ();           // Обработаем сетевую команду
        ir_trns();        // Отправим ИК команду на излучатель
        goto start;
}
}
```

Если ее запустить, набраться терпения и дождаться результата, то результат будет выглядеть следующим образом (отслеживание RA0 я убрал, оно было полезно для определения времен):

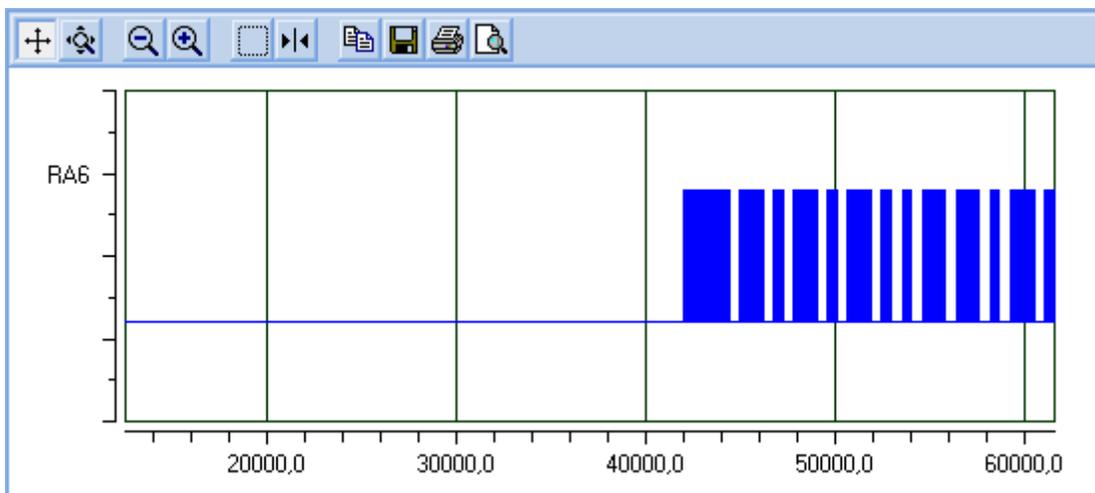


Рис.85

## Хобби-электроникс 2. Умный дом.

Именно такой сигнал мы могли бы увидеть на записывающем осциллографе, если бы воспроизвели ИК команду с пульта устройства перед фотоприемником (без встроенной обработки сигнала). Картинка не была бы столь четкой, вероятно, но была бы очень похожа. Видно, как сигнал начинается с заголовка (pulse), затем следует короткая пауза (space), затем идет единичная посылка несущей со своей паузой, и т.д. По этой картинке можно прочитать сигнал – 10101001101.

Вернувшись к началу этого раздела, мы найдем эту команду.

В подобных случаях «утаптывания» проблем, почему я и говорю, что мне не нравится, то, что я делаю, вместе с водой можно выплеснуть и ребенка. Проверить работу модуля с этой командой можно несколькими способами (собрав модуль). Можно прочитать команду с пульта и с модуля считывающим устройством. И сравнить результаты. Лучше отправить команду на устройство, которое этой командой управляется. Работает устройство, значит, работает и команда.

Однако прежде, чем проверять работу собранного модуля, есть смысл проверить его работу в MPLAB с другими командами. У меня есть несколько считанных команд, проверкой работы которых, я и хочу заняться. Для этого я запишу их в новый input.txt файл, и попробую понять, что из этого получается.

Вот как выглядит команда PLAY одного из устройств фирмы Sharp (в представлении, приведенном к виду, который получился бы с помощью считывания в WinLIRC).

Исходный вид:

```
4000 d101 4000 d101 4000 c700 4000 c700
4000 c700 4000 c700 4000 d101 4000 c700
4000 c700 4000 c700 4000 d101 4000 c700
4000 c700 4000 d101 4000 c700 4000 6f2b
```

После первого преобразования:

```
0040 Импульс
01d1 Пауза
0040 Импульс
01d1 Пауза
0040 Импульс и т.д.
00c7
0040
00c7
0040
00c7
0040
00c7
0040
01d1
```

## Хобби-электроникс 2. Умный дом.

0040  
00c7  
0040  
00c7  
0040  
00c7  
0040  
01d1  
0040  
00c7  
0040  
00c7  
0040  
01d1  
0040  
00c7  
0040  
2b6f

Времена 64 (0040), 465 (01d1), 199 (00c7), 11119 (2b6f)  
Первый импульс – 256 мкс. Что предполагает к-т 4.  
Тогда времена 256 мкс, 1860 мкс, 796 мкс, 44476 мкс  
И, наконец, в том виде, с которым мы начали работать (в шестнадцатеричном представлении):

|            |    |   |
|------------|----|---|
| pulse 256  | 12 | Я разделил на 13,8 и перевел в HEX формат |
| space 1860 | 87 |   |
| pulse 256  | 12 |   |
| space 1860 | 87 |   |
| pulse 256  | 12 |   |
| space 796  | 3A |   |
| pulse 256  | 12 |   |
| space 796  | 3A |   |
| pulse 256  | 12 |   |
| space 796  | 3A |   |
| pulse 256  | 12 |   |
| space 796  | 3A |   |
| pulse 256  | 12 |   |
| space 1860 | 87 |   |
| pulse 256  | 12 |   |
| space 796  | 3A |   |
| pulse 256  | 12 |   |
| space 796  | 3A |   |
| pulse 256  | 12 |   |
| space 796  | 3A |   |
| pulse 256  | 12 |   |
| space 1860 | 87 |   |
| pulse 256  | 12 |   |

## Хобби-электроникс 2. Умный дом.

space 796            3A  
pulse 256           12  
space 796            3A  
pulse 256           12  
space 1860           87  
pulse 256           12  
space 796            3A  
pulse 256           12  
space 44476          C96

Осталось заменить данные в файле input.txt. Посмотрим, что получится.

Общий вид:

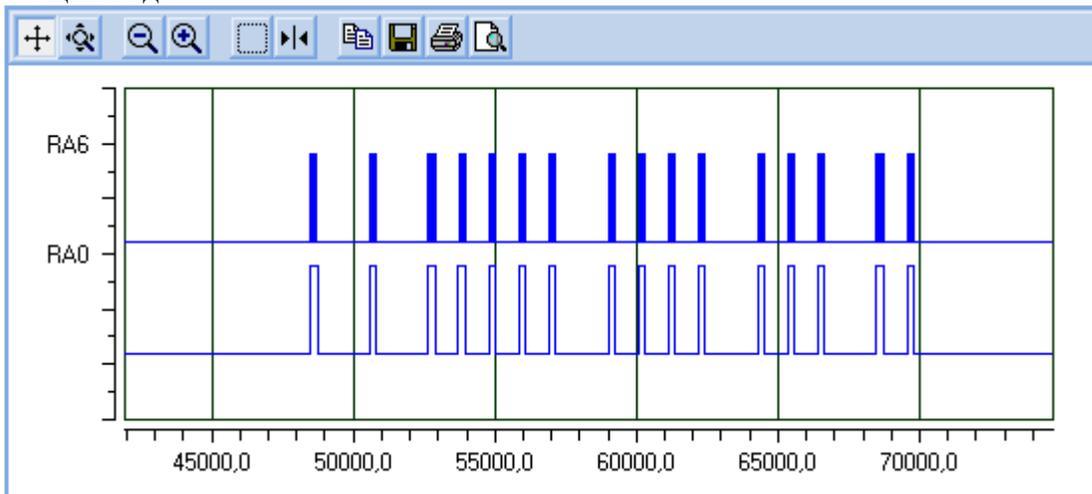


Рис.86

Первая пауза:

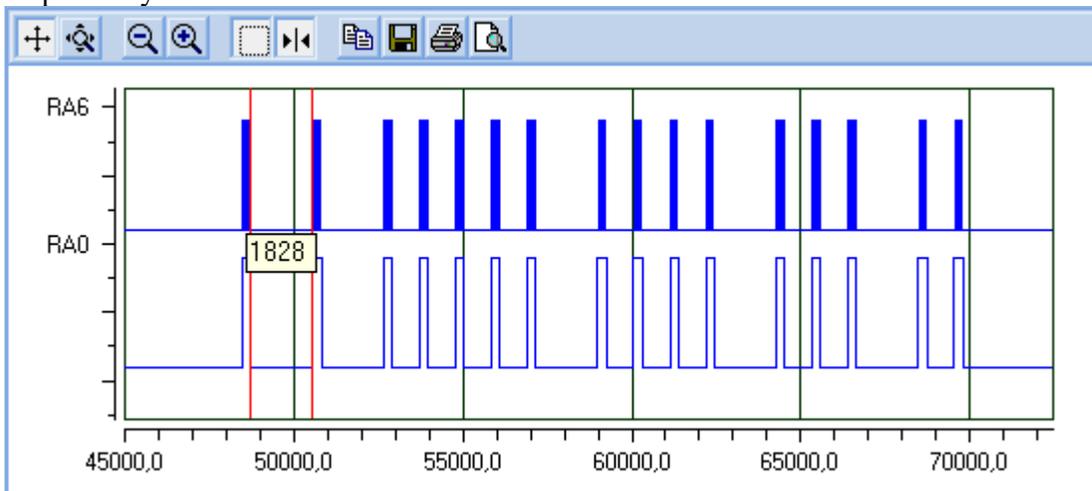


Рис.87

Вторая пауза:

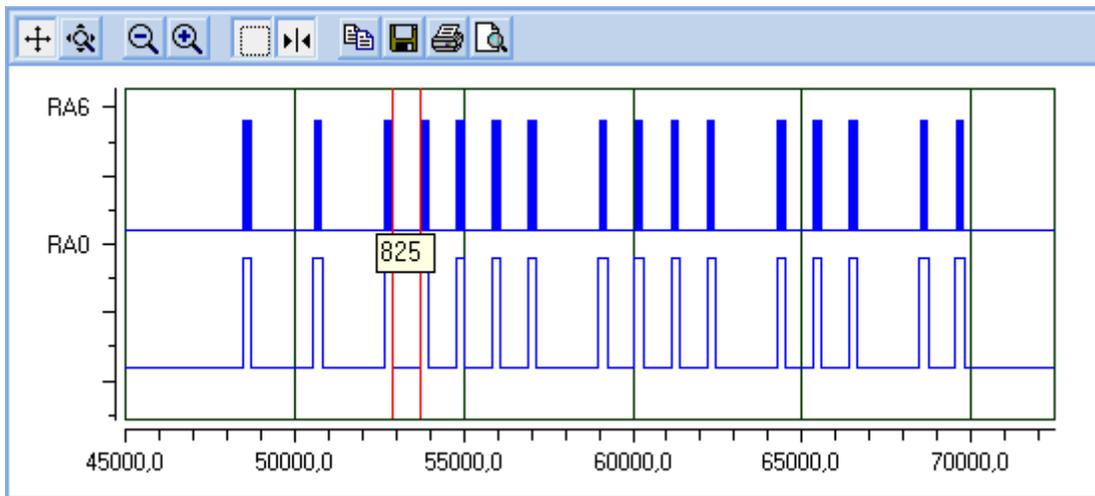


Рис.88

Импульс:

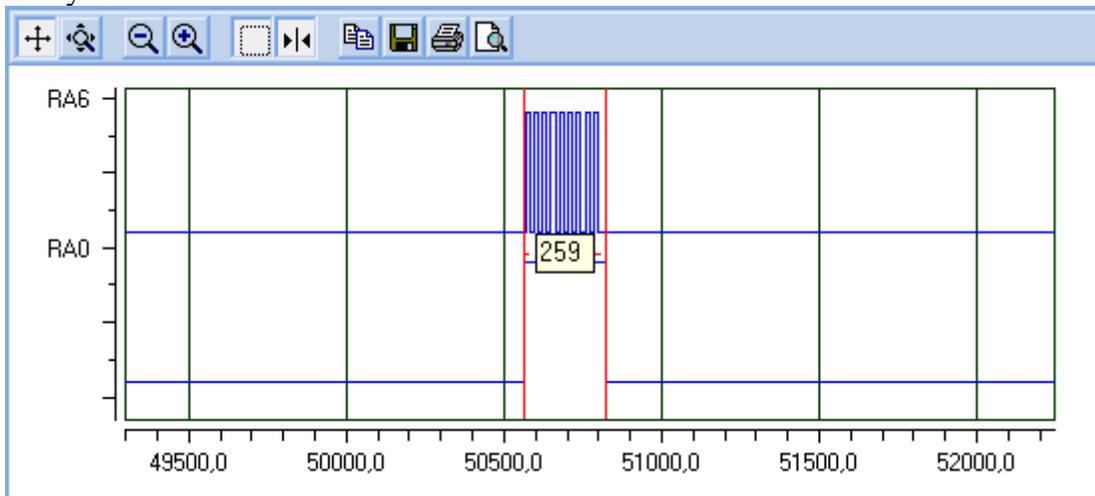


Рис.89

Несущая:

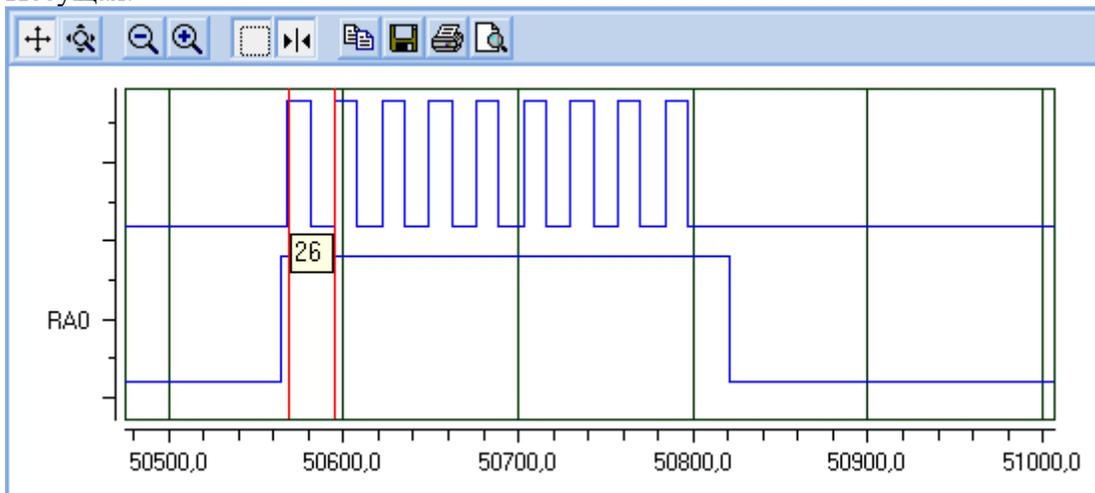


Рис.90

## Хобби-электроникс 2. Умный дом.

Для сравнения, вот так выглядят информационные импульсы в программе, из базы данных которой взяты ИК коды Sharp.

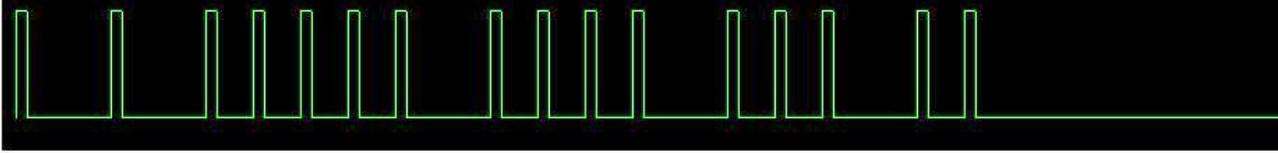


Рис.91

Сравнивая этот вид с видом импульсов, представленных, как RA0 на рис.88 можно убедиться в их схожести. А времена (256 мкс против 259, 1860 мкс против 1828, 796 мкс против 825) не столь разительно отличны. Несущая, правда, уехала к 38,5 кГц, но это может быть и не совсем точно измеренное значение (маркер несколько съехал с фронта импульса).

Приступая к разработке, мы вполне разумно определились в том, что хотим от данного модуля. В процессе разработки, столкнувшись с трудностями, мы не менее разумно отказались от части первоначальных запросов. Написав первую версию кода программы, выглядевшую вполне «рассудительно и логично», мы отказались от нее. Предполагая хранить значения текущей длительности интервала в двух байтах, мы для ускорения работы программы приняли однобайтовый вариант хранения.

С точки зрения обучения – это яркий пример того, *как не надо делать*. С практической точки зрения это пример того, как приходится иногда поступать, чтобы справиться с проблемой хотя бы как-то. Чтобы обойти проблему хотя бы на время.

В нашем случае, пытаясь овладеть приемами работы с программой MPLAB, и одновременно создать некоторую любительскую систему, мы в праве принять несколько решений:

1. Оставить все, как есть. Мы не можем менять частоту несущей ИК команды «на лету», но пока не убедились, что это нужно, можем не тревожиться. Как выглядит программа, изящно или нет, после загрузки ее в контроллер едва ли будет кому-либо интересно. Все что требуется, так это, чтобы модуль исправно работал.
2. Можно собрать модуль, опробовать. Если работает, то забыть о существовании проблем.
3. И, наконец, третье решение – в конце работы пересмотреть все, что сделано, и создать модернизированные версии тех же модулей. К концу работы появится больше опыта, чаще будет приходить в голову мысль о том, что знать бы раньше, можно было бы сделать и лучше!

Лично я склоняюсь к тому, что, когда появятся эти мысли, тогда и займемся модернизацией. А пока, рисуем схему, отправляемся в «Чип и Дип», собираем модуль.

Напомню формат, который мы «перелопатили», и действия, которые мы осуществляем, вопреки первоначальным замыслам.

Формат записи в файл ИК команды:

|         |          |        |         |              |        |
|---------|----------|--------|---------|--------------|--------|
| Устр-во | Повторов | Коэф-т | Импульс | Пауза и т.д. | EOF    |
| байт    | байт     | байт   | байт    | байт         | Байт 0 |

Импульс и пауза – это времена в мкс, деленные на 13,8 после прочитывания в программе WinLIRC в виде шестнадцатеричных чисел. Затем, в программе, они будут еще раз разделены на 2.

*В программе в настоящий момент не используется служебная информация, относящаяся к устройству, количеству повторов и коэффициенту. Позже, если решим модернизировать модуль, мы сделаем несколько выходов, которые будут выбираться по записи «Устройство», добавим обработку количества повторов, и, возможно, используем коэффициент 13 или 14 для деления*

## Хобби-электроникс 2. Умный дом.

*времени, полученного при подготовке кода, самой программой.*

Схема модуля.

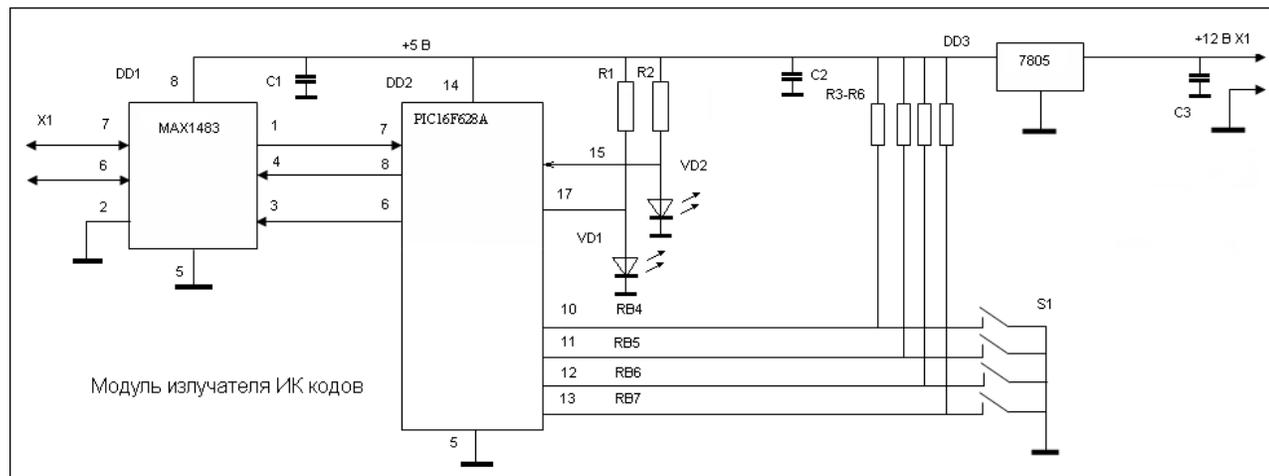


Рис.92

### Спецификация.

| №  | Обозначение | Изделие        | Кол-во | Цена (руб.) | Примечания |
|----|-------------|----------------|--------|-------------|------------|
| 1  | DD1         | MAX1483        | 1      | 96          |            |
| 2  | DD2         | PIC16F628A     | 1      | 100         |            |
| 3  | DD3         | LM2936-Z5      | 1      | 68          |            |
| 4  | VD1         | АЛ307          | 1      | 3           |            |
| 5  | VD2         | АЛ144А         | 1      | 20          |            |
| 6  | C1, C2      | 0.1 мкФ        | 2      | 2           |            |
| 7  | C3          | 100.0 мкФ 16В  | 1      | 5           |            |
| 9  | R1, R2      | 1 кОм 0.25 Вт  | 2      | 1           |            |
| 10 | R3-R6       | 10 кОм 0.25 Вт | 4      | 1           |            |

Ориентировочная стоимость элементов 297 руб., стоимость платы 100 руб. Всего 397 руб.

Вид платы.

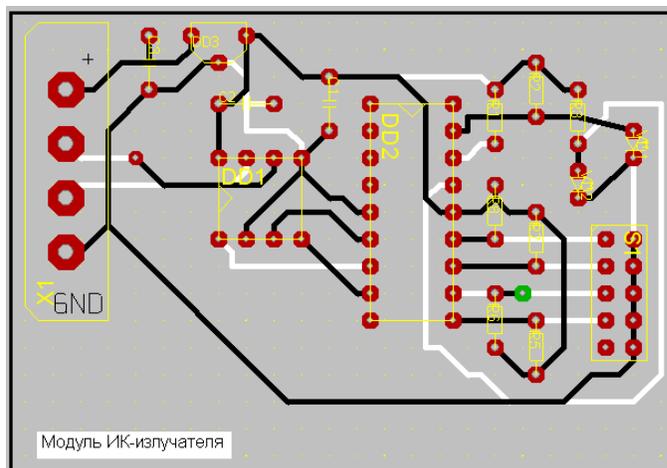


Рис.93

## Модуль считывания инфракрасных кодов

Используем программу WinLIRC<sup>3</sup>, доступную для свободного скачивания, с соответствующим модулем в качестве считывающего устройства для предварительной подготовки кодов. Излучатель WinLIRC используем для генерации системных ИК кодов.

Схема фотосчитывателя и излучателя для работы с программой WinLIRC.

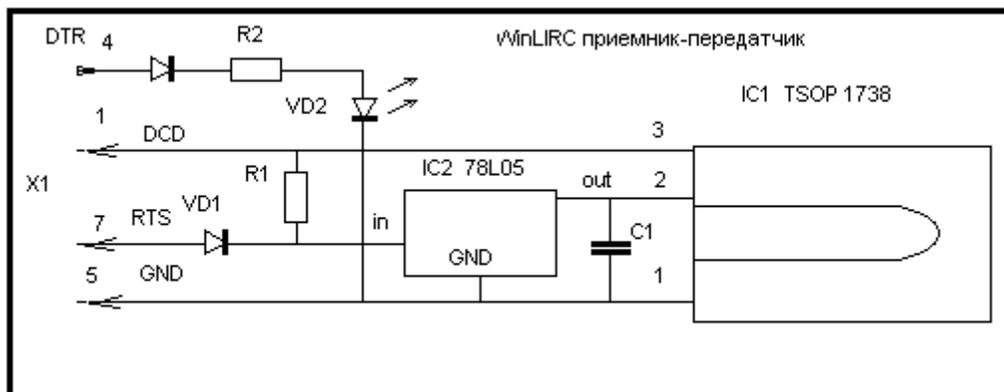


Рис.94

Спецификация.

| № | Обозначение | Изделие         | Кол-во | Цена (руб.) | Примечания |
|---|-------------|-----------------|--------|-------------|------------|
| 1 | IC1         | TSOP 1738       | 1      | 40          |            |
| 2 | IC2         | 78L05           | 1      | 30          |            |
| 3 | VD1, VD3    | 1N4148          | 2      | 5           |            |
| 4 | VD2         | АЛ144А          | 1      | 20          |            |
| 5 | R1          | 4.7 кОм 0.25 Вт | 1      | 1           |            |
| 6 | R2          | 2 кОм 0.25 Вт   | 1      | 1           |            |
| 7 | C1          | 4.7 мкФ 16 В    | 1      | 20          |            |
| 8 | X1          | DB9 гнездо      | 1      | 10          |            |

Вид платы.

<sup>3</sup> Программа доступна на сайте: <http://winlirc.sourceforge.net>

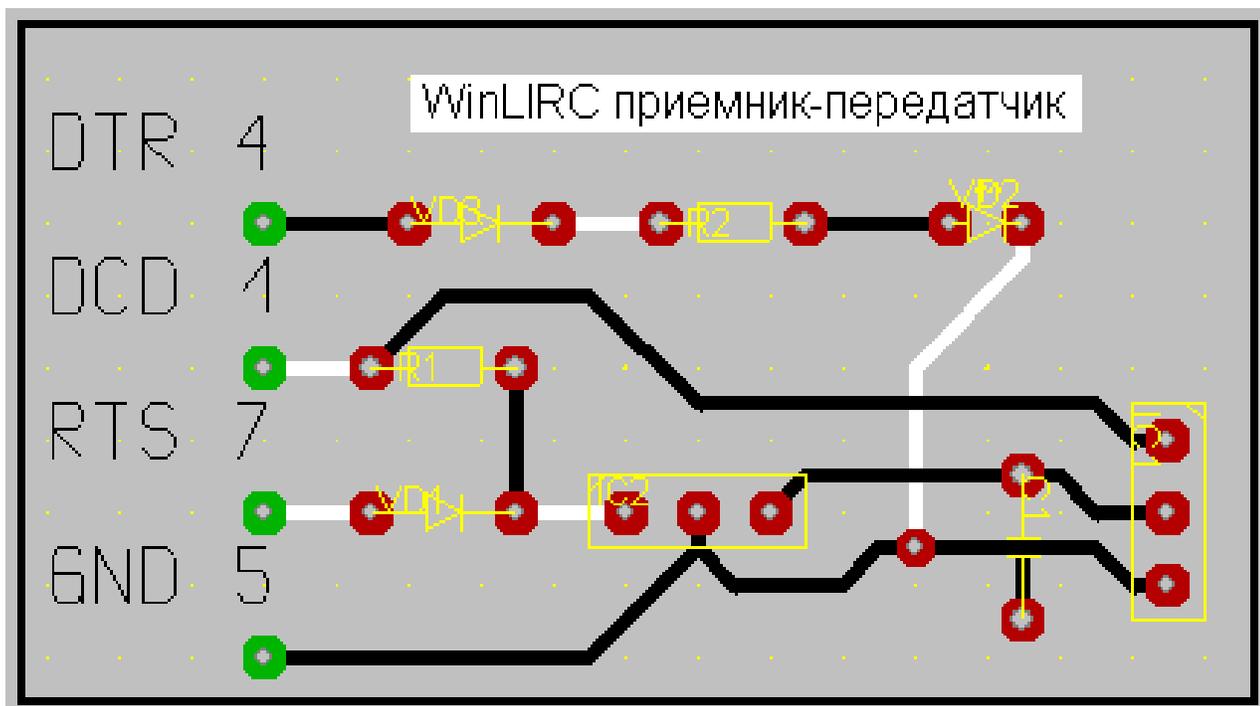


Рис.95

Программа WinLIRC работает в двух режимах при прочитывании ИК кодов. В первом режиме она определяет соответствие кодов стандарту, и создает текстовый файл, в котором записаны параметры кода. Во втором режиме она непосредственно выводит во встроенное окно времена посылок и пауз ИК команды.

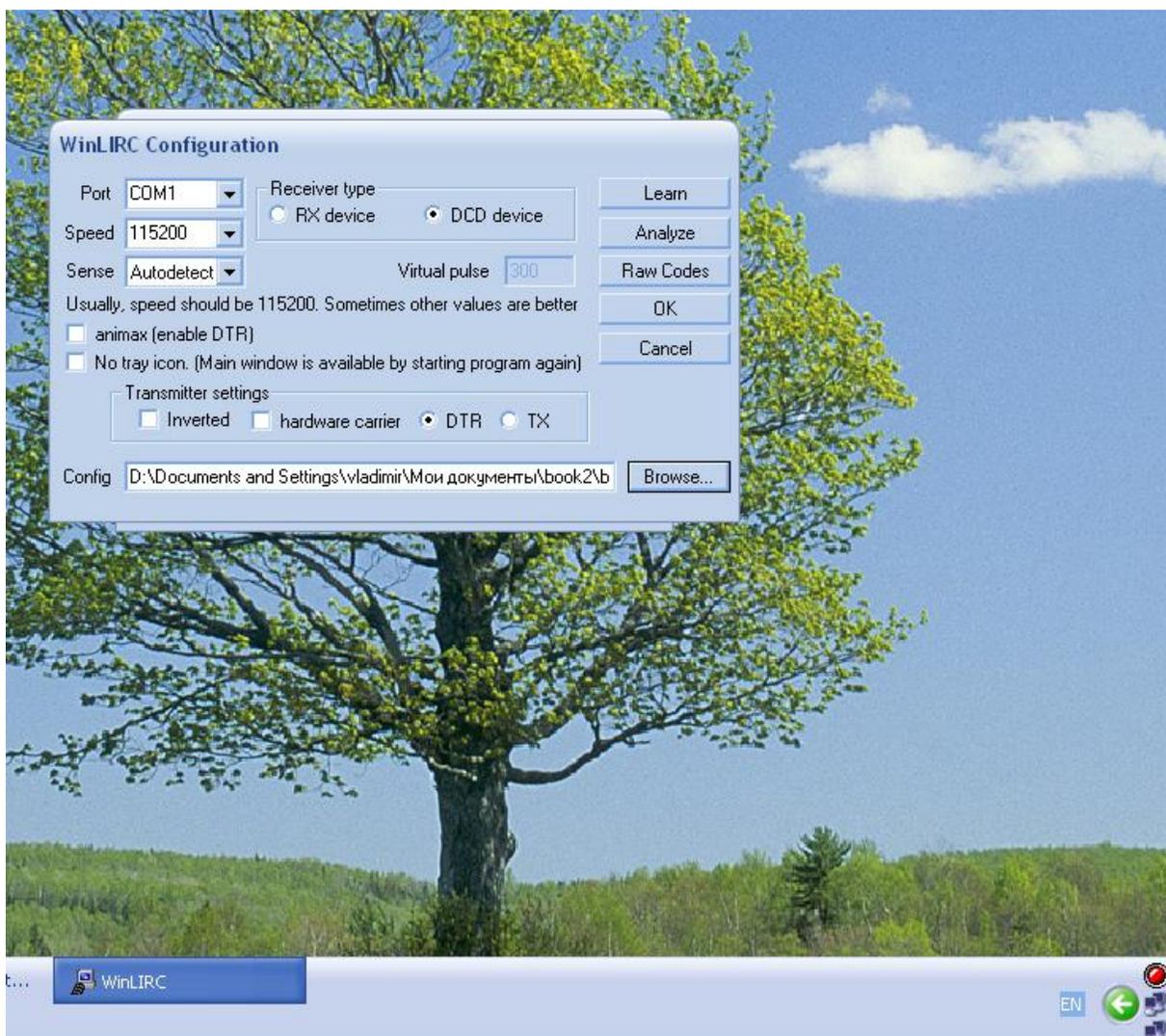


Рис.96

Появление в правом углу панели значка  означает, что программа работает. Клавишей Browse можно задать файл конфигурации. И, нажав клавишу Raw Codes, прочитать ИК команду.

Думаю, что для считывания команд, мы используем второй метод, поскольку существует много кодов, не читаемых первым способом. Файл, записанный с помощью программы WinLIRC, выглядит следующим пугающим образом:

Коды с пульта VCR Sony

Прямое считывание, клавиша power:

Outputting raw mode2 data.

space 12856572

pulse 2455

## Хобби-электроникс 2. Умный дом.

space 532  
pulse 1291  
space 506  
pulse 664  
space 533  
pulse 1266  
space 533  
pulse 665  
space 591  
pulse 1211  
space 533  
pulse 685  
space 515  
pulse 693  
space 526  
pulse 1271  
space 506  
pulse 1265  
space 533  
pulse 666  
space 557  
pulse 1241  
space 533  
pulse 664  
space 24527  
pulse 2463  
space 579  
pulse 1219  
space 534  
pulse 664  
space 534  
pulse 1266  
space 534  
pulse 666  
space 531  
pulse 1266  
space 534  
pulse 663  
space 536  
pulse 662  
space 567

Реально эта запись может получиться еще длиннее, если пульт при нажатии клавиши постоянно воспроизводит одну команду до тех пор, пока клавиша не будет отпущена.

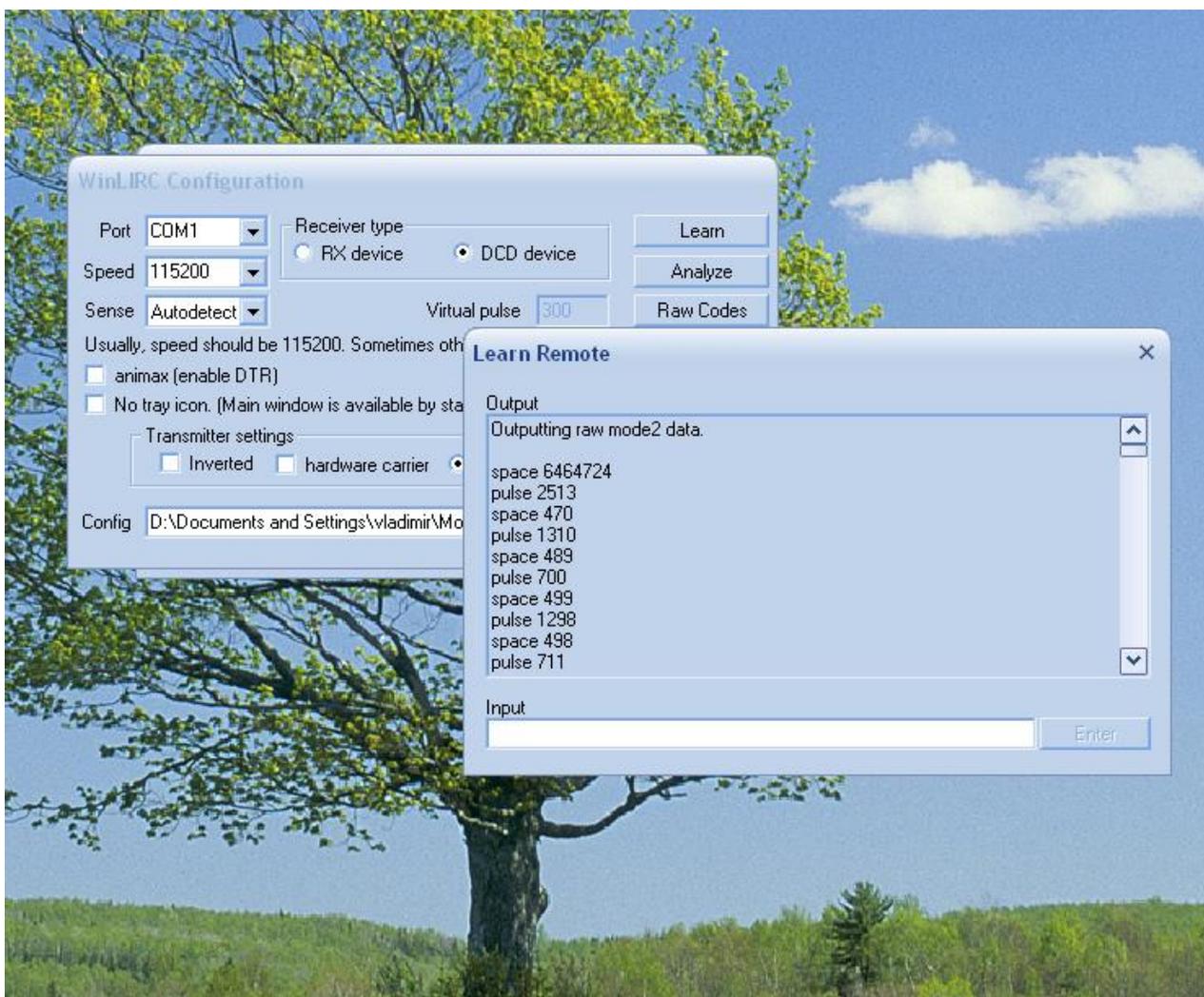


Рис.97

Собственно команда, интересующая нас, содержится в следующей записи:

Outputting raw mode2 data.

pulse 2455  
space 532  
pulse 1291  
space 506  
pulse 664  
space 533  
pulse 1266  
space 533  
pulse 665  
space 591  
pulse 1211  
space 533  
pulse 685  
space 515  
179

## Хобби-электроникс 2. Умный дом.

pulse 693  
space 526  
pulse 1271  
space 506  
pulse 1265  
space 533  
pulse 666  
space 557  
pulse 1241  
space 533  
pulse 664  
space 24527

Здесь **pulse** – вспышка излучателя с частотой, положим, 36 кГц, а **space** – пауза между вспышками.

Раскрасим эту запись в соответствии с тем, что говорилось о кодах Sony выше, выделяя заголовок, единицы и нули:

pulse 2455 Заголовок  
space 532  
pulse 1291 Единица  
space 506  
pulse 664 Ноль  
space 533  
pulse 1266 Единица  
space 533  
pulse 665 Ноль  
space 591  
pulse 1211 Единица  
space 533  
pulse 685 Ноль  
space 515  
pulse 693 Ноль  
space 526  
pulse 1271 Единица  
space 506  
pulse 1265 Единица  
space 533  
pulse 666 Ноль  
space 557  
pulse 1241 Единица  
space 533  
pulse 664  
space 24527 Пауза между командами

Числа здесь – времена в микросекундах. Код, если записать информационное представление, выглядит как 10101001101 в двоичном виде или 54D в шестнадцатеричном.

## Хобби-электроникс 2. Умный дом.

Конечно, хорошо бы хранить команду в информационном представлении, в виде двух байт. В этом случае можно было бы хранить до 64 команд в EEPROM.

Но не все ИК коды таковы. Некоторые коды имеют до 48 бит, т.е. в информационном представлении требуют 6 байт. Есть команды, состоящие из двух последовательно воспроизводимых команд, и еще более длинных.

Можно придумать систему шифрования кодов (с целью их сжатия), но мне кажется, что лучше отказаться от идеи хранения кодов в модуле передатчика ИК команд, хранить их в файле данных, который компьютер будет прочитывать при работе основной программы. При необходимости трансляции кода, он будет передаваться по системной линии, запоминаться модулем излучения ИК команд, и воспроизводиться им. В этом случае в модуле нам предстоит запомнить только одну команду, что снимает ограничения на общее количество команд и сложность команд в плане длинных кодов.

Итак, мы будем с помощью программы WinLIRC прочитывать ИК коды с пультов устройств, обрабатывать, записывать в текстовый файл (или файлы), и воспроизводить их по мере необходимости, передавая соответствующему модулю. В этом случае формат команды можно оставить таким, каким мы использовали при работе с модулем трансляции. В нескольких байтах будет храниться служебная информация. Далее будет следовать последовательность импульсов и пауз, каждый из которых будет занимать байт. Заканчиваться эта последовательность будет байтом с нулями.

Формат записи:

| Устр-во | Повторов | Коэф-т | Импульс | Пауза и т.д. | EOF    |
|---------|----------|--------|---------|--------------|--------|
| ? байт  | байт     | байт   | байт    | байт         | Байт 0 |

На этом мы можем завершить первую версию «Умного домика» в части исполняющих модулей. И пришло время создания основной программы для управляющего компьютера.

## ***Первая версия основной программы***

В начале работы еще раз перечислим модули, которые мы разработали для системы.

1. Релейный модуль.
2. Модуль приема системных ИК команд (от управляющего устройства).
3. Модуль трансляции ИК команд для управления бытовой аппаратурой.

Модулей немного. Но на их основе можно создать вполне интересную версию системы - «Умный кукольный домик».

Чтобы не усложнять задачу по созданию управляющей программы, в качестве среды разработки этой программы используем Visual Basic, либо любую доступную и удобную для вас среду разработки. Я выбрал Visual Basic только по той причине, что у меня есть возможность поработать с ним. Вдобавок, этот вариант упрощает работу с СОМ портом.

Перечислим команды модулей, которые могут потребоваться при написании основной программы:

### ***Релейный модуль***

Возможно, не будем запрашивать статус реле, используя только команды включения и выключения. Используем три модуля с адресами 01, 02, 03, которые будут располагаться в трех комнатах и включать настольную лампу в кабинете, торшер в гостиной и бра в холле.

Включить – Rxx\$хN  
Выключить – Rxx\$хF

Где хх – это 01, 02, 03; х после символа команды «\$» - это «1» (будем использовать только по одному реле в модуле).

Передать статус – Rxx\$хS  
При передаче статуса используются следующие символы  
Включено – Rxx#хN  
Выключено – Rxx#хF

### ***Модуль приема системных ИК команд***

Используем один модуль, который расположим в гостиной.  
Запрос статуса – Схх\$0S (аналогично команде запроса статуса релейного модуля).  
Ответ модуля – Сххkkk, если команда пришла  
Схх#ff, если ИК команда не приходила.

### ***Модуль трансляции ИК кодов***

Будем предполагать, что мы по команде с системного пульта будем включать и выключать телевизор. Т.е. нам потребуется отправлять команду power для телевизора, которую предварительно прочитаем с помощью WinLIRC и обработаем, записав в файл, для

## Хобби-электроникс 2. Умный дом.

которого выберем расширение .irc.

Какой я представляю себе в настоящий момент работающую программу?

При запуске программы отображается пользовательский интерфейс, который поможет проверить работу модулей и запустить основную программу.

Помещений, где будут установлены модули, три – кабинет, гостиная, холл.

В Visual Basic, как и в других средах программирования, есть удобное средство создания пользовательского интерфейса программы – форма (Form). Этим мы и воспользуемся.

Заменяем название формы (свойство Caption) на «Кукольный умный домик». В редакторе изображений создадим иконку, которую вставим, используя свойство Icon.



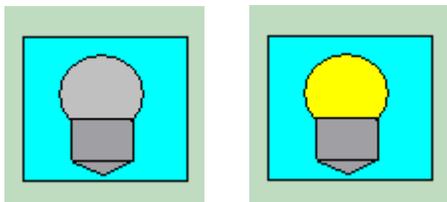
Теперь откроем три Frame, которые назовем соответственно Кабинет, Гостиная, Холл. Получим такую картинку:



Рис.98

В редакторе изображений нарисуем изображения включенной и выключенной ламп.

## Хобби-электроникс 2. Умный дом.



Их мы используем впоследствии для отображения состояния ламп (или реле). Вставим картинки в каждое из ранее нарисованных помещений на форме (добавим PictureBox с вкладки General), добавим клавиши, которые будут управлять этими лампами (с помощью реле). Картинки наложим одна поверх другой. Получим следующий вид формы (последние две картинки не совмещены):

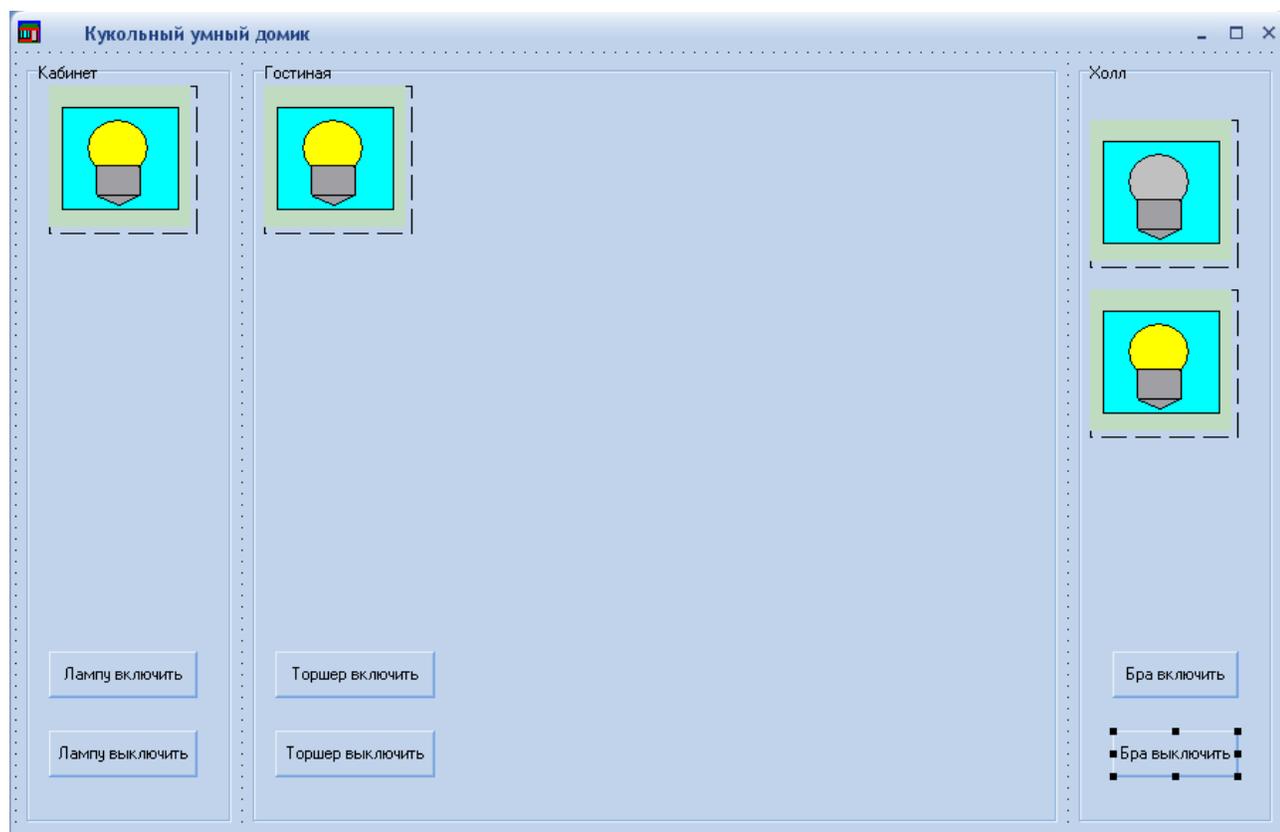


Рис.99

Теперь при запуске программы нажатием клавиши «Лампу включить» мы будем получать следующий результат. Напомню только, что картинки с изображением включенной лампы мы наградим свойством `Visible False`, а для ввода кода нужно дважды щелкнуть мышкой по клавише на форме, например «Лампу включить», и вписать небольшой текст:

```
Private Sub Reley01_on_Click()  
Form1.Lamp_off.Visible = False  
Form1.Lamp_on.Visible = True  
End Sub
```

Красным цветом я выделил то, что создает Visual Basic при щелчке по клавише на форме.

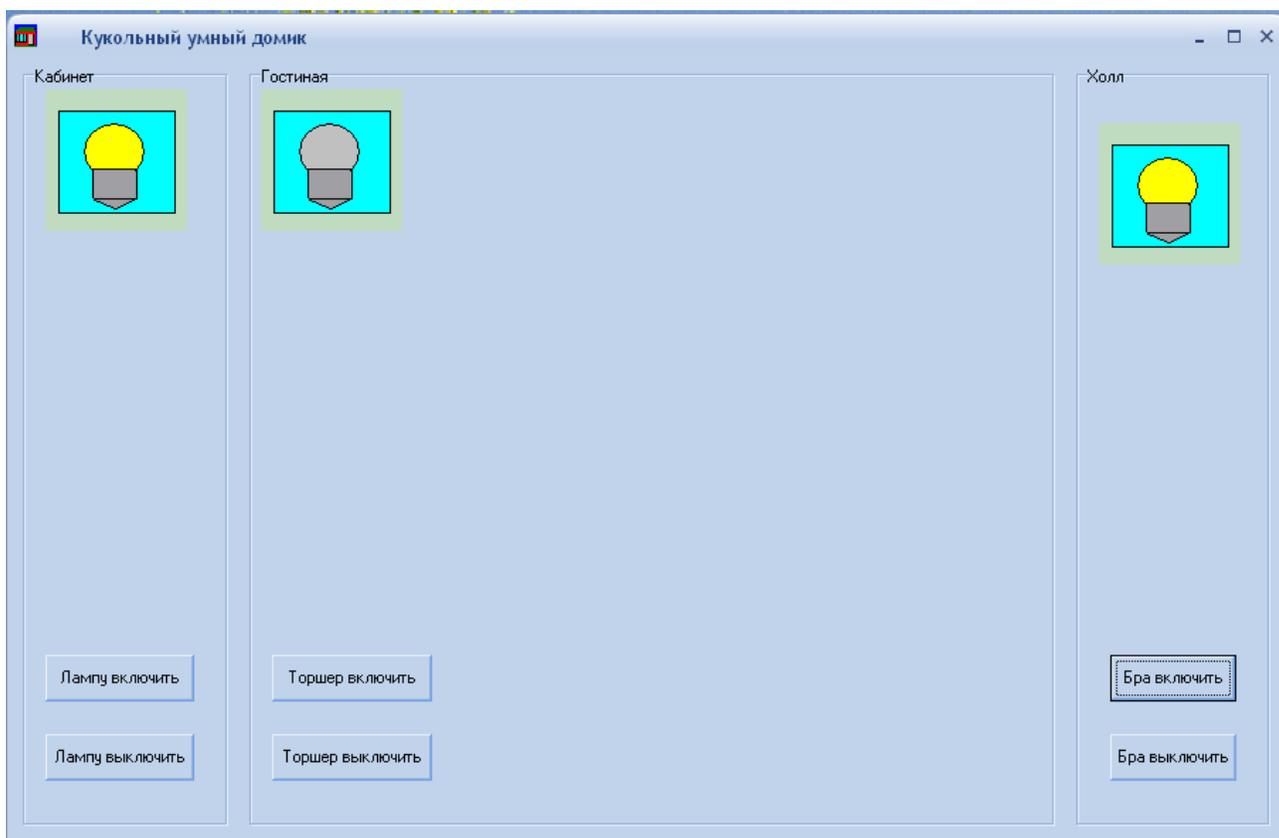


Рис.100

Расположим в гостиной пульт управления, с которого будем принимать команды (он сыграет роль модуля фотоприемника), и телевизор, которым мы будем управлять с помощью модуля излучателя ИК команд. Картинки рисуем в графическом редакторе и накладываем друг на друга (конечно, соответствующие, как с лампами). Добавляем код, аналогичный вышеупомянутому, и получаем работающую заготовку, которую в первую очередь можно использовать для тестирования модулей. Клавиши включения и выключения ламп будут посылать команды модулям (проверять состояние реле?), и менять картинки в соответствии с состоянием – «Включено» или «Выключено». Аналогично мы поступим и с остальными модулями. А пока можно проверить отображение будущих идей.

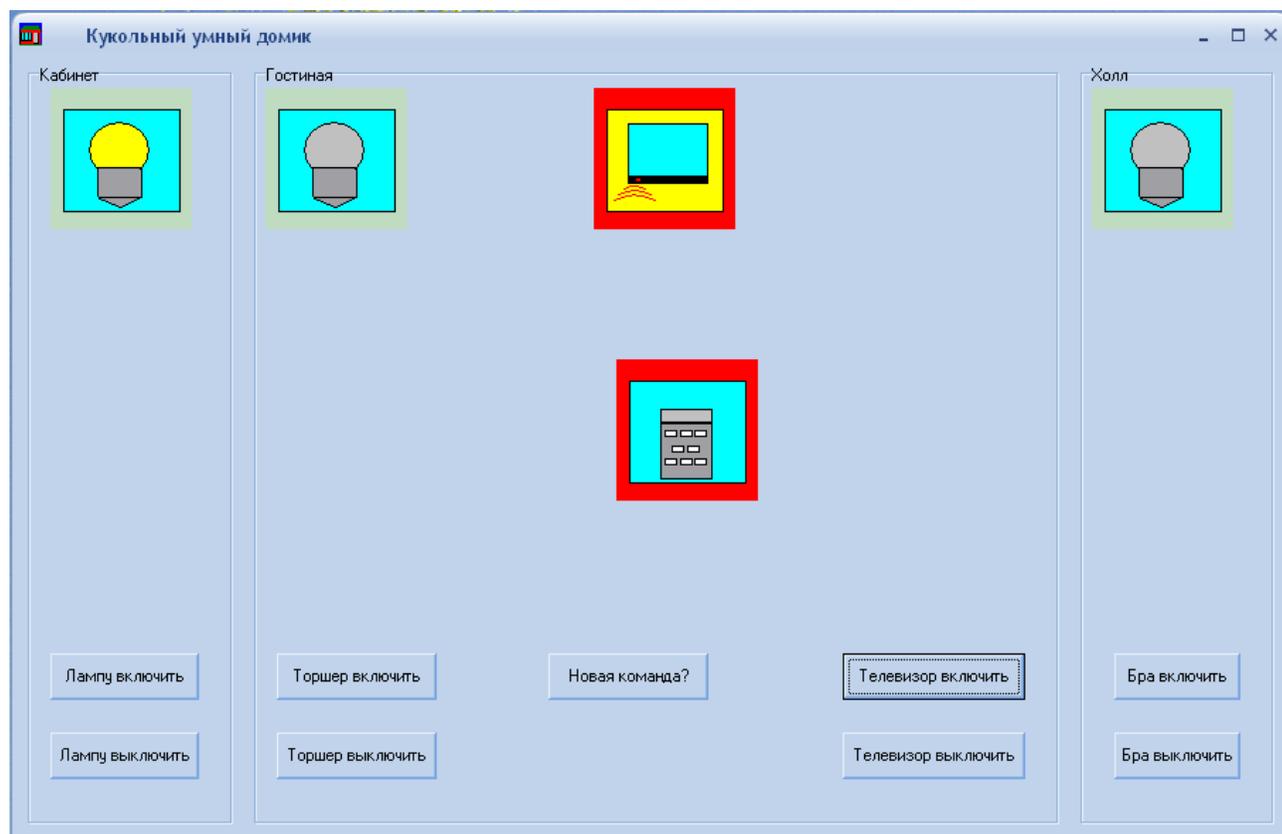


Рис.101

Поскольку мы планируем использовать два режима работы – тестовый и основной – я думаю, есть смысл сделать главное меню с двумя режимами работы. Для этой цели на форме щелкнем правой клавишей, и в открывшемся меню выберем «Редактор Меню». С его помощью создадим меню с основным пунктом «Режим» и двумя подпунктами «Проверка» и «Работа».

При выборе режима «Проверка» (щелкаем по этому разделу меню) мы сделаем видимыми все клавиши:

**Private Sub mnuTest\_Click()**

```
Form1.Reley01_off.Enabled = True
Form1.Reley01_off.Visible = True
Form1.Reley02_off.Enabled = True
Form1.Reley02_off.Visible = True
Form1.Reley03_off.Enabled = True
Form1.Reley03_off.Visible = True
Form1.Reley01_on.Enabled = True
Form1.Reley01_on.Visible = True
Form1.Reley02_on.Enabled = True
Form1.Reley02_on.Visible = True
Form1.Reley03_on.Enabled = True
Form1.Reley03_on.Visible = True
Form1.Photo01_on.Enabled = True
```

## Хобби-электроникс 2. Умный дом.

```
Form1.Photo01_on.Visible = True  
Form1.IRemitter01_off.Visible = True  
Form1.IRemitter01_off.Enabled = True  
Form1.IRemitter01_on.Visible = True  
Form1.IRemitter01_on.Enabled = True  
Form1.Photo01_off.Enabled = True  
Form1.Photo01_off.Visible = True
```

End Sub

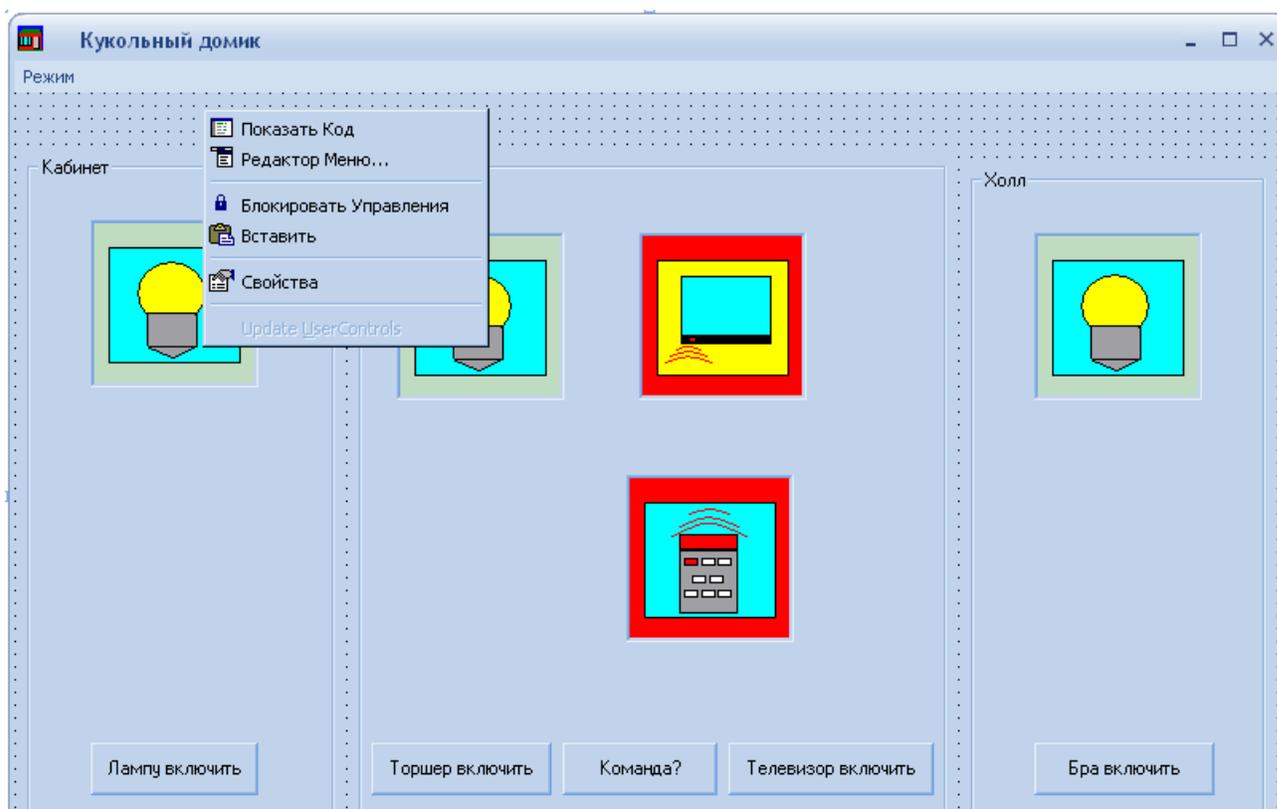


Рис.102

При переходе в режим «Работа» мы их уберем «с глаз долой»:

**Private Sub muWork\_Click()**

```
Form1.Reley01_off.Enabled = False  
Form1.Reley01_off.Visible = False  
Form1.Reley02_off.Enabled = False  
Form1.Reley02_off.Visible = False  
Form1.Reley03_off.Enabled = False  
Form1.Reley03_off.Visible = False  
Form1.Reley01_on.Enabled = False  
Form1.Reley01_on.Visible = False  
Form1.Reley02_on.Enabled = False
```

## Хобби-электроникс 2. Умный дом.

```
Form1.Reley02_on.Visible = False
Form1.Reley03_on.Enabled = False
Form1.Reley03_on.Visible = False
Form1.Photo01_on.Enabled = False
Form1.Photo01_on.Visible = False
Form1.IRemitter01_off.Visible = False
Form1.IRemitter01_off.Enabled = False
Form1.IRemitter01_on.Visible = False
Form1.IRemitter01_on.Enabled = False
Form1.Photo01_off.Enabled = False
Form1.Photo01_off.Visible = False
```

End Sub

Теперь работающая программа будет выглядеть в двух режимах следующим образом:

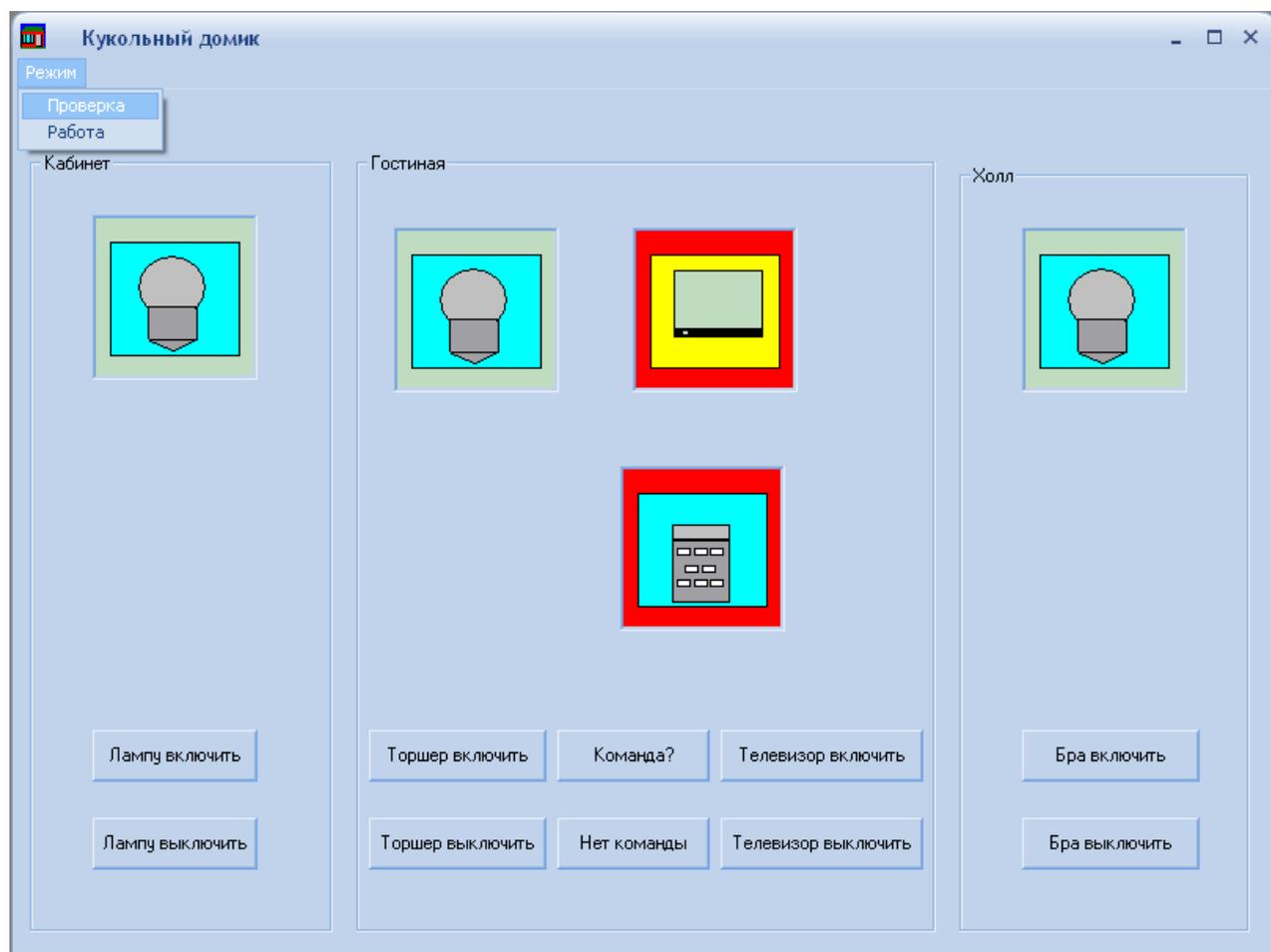


Рис.103

В программе я использовал обозначения Reley01\_on (off), Photo01\_off (on) и IRemitter01\_on (off) для клавиш управления «Лампу включить» и т.д. Клавиша «Команда?» - запрос фотоприемнику. Обозначения содержат адрес модулей Reley01, Reley02, Photo01 и т.д.

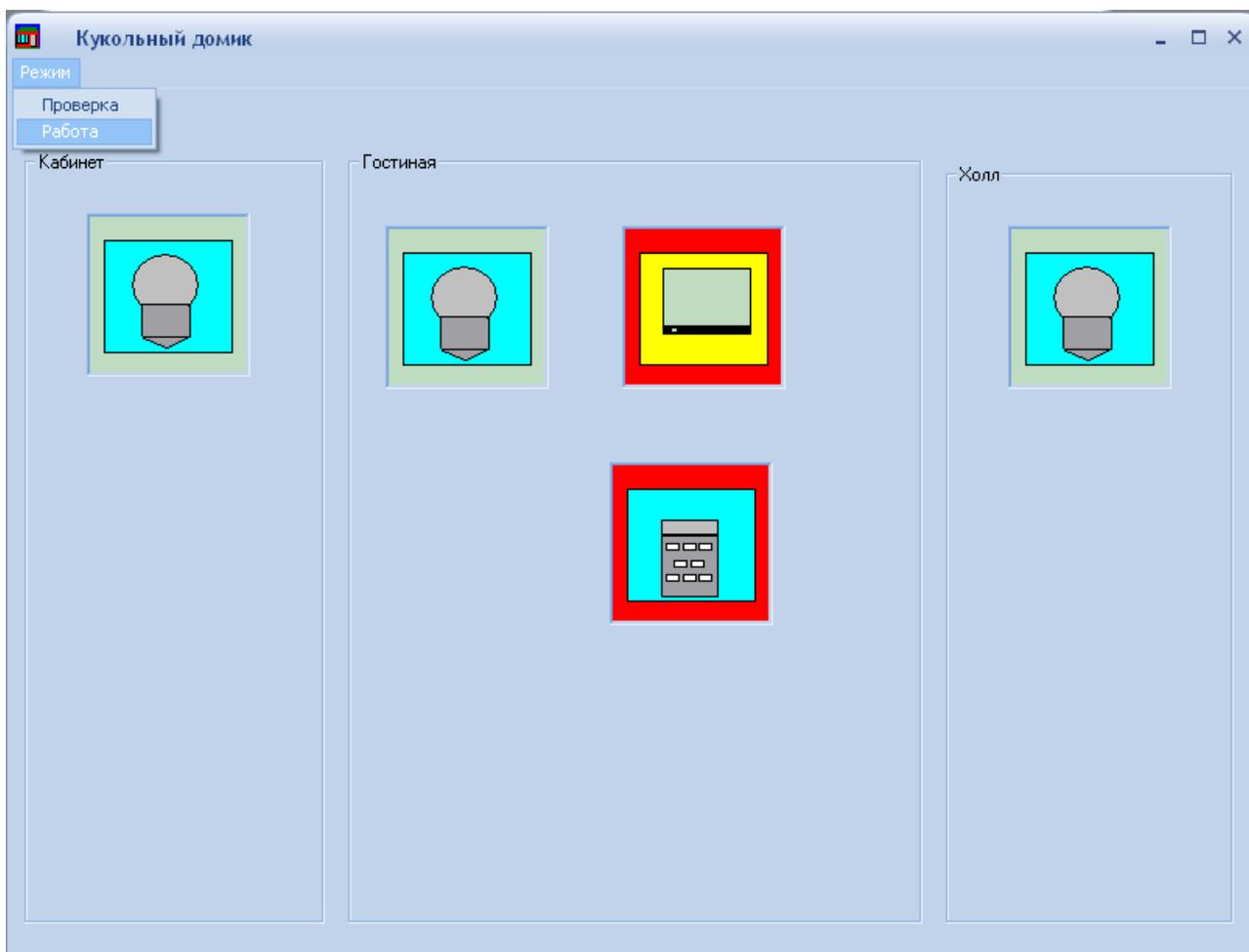


Рис.104

Не трогая режим работы, попробуем организовать работу с COM1 портом. Для этих целей используем возможность добавить элементы управления. Я добавил требуемый элемент управления MSComm на панель Home, которую создал, чтобы не перегружать основную панель. Этот элемент переносим на форму для использования в программе.

Для добавления элемента на инструментальную панель открываем в меню «Проект» раздел «Компоненты» и ставим галочку напротив Microsoft Comm Control 6.0.

## Хобби-электроникс 2. Умный дом.

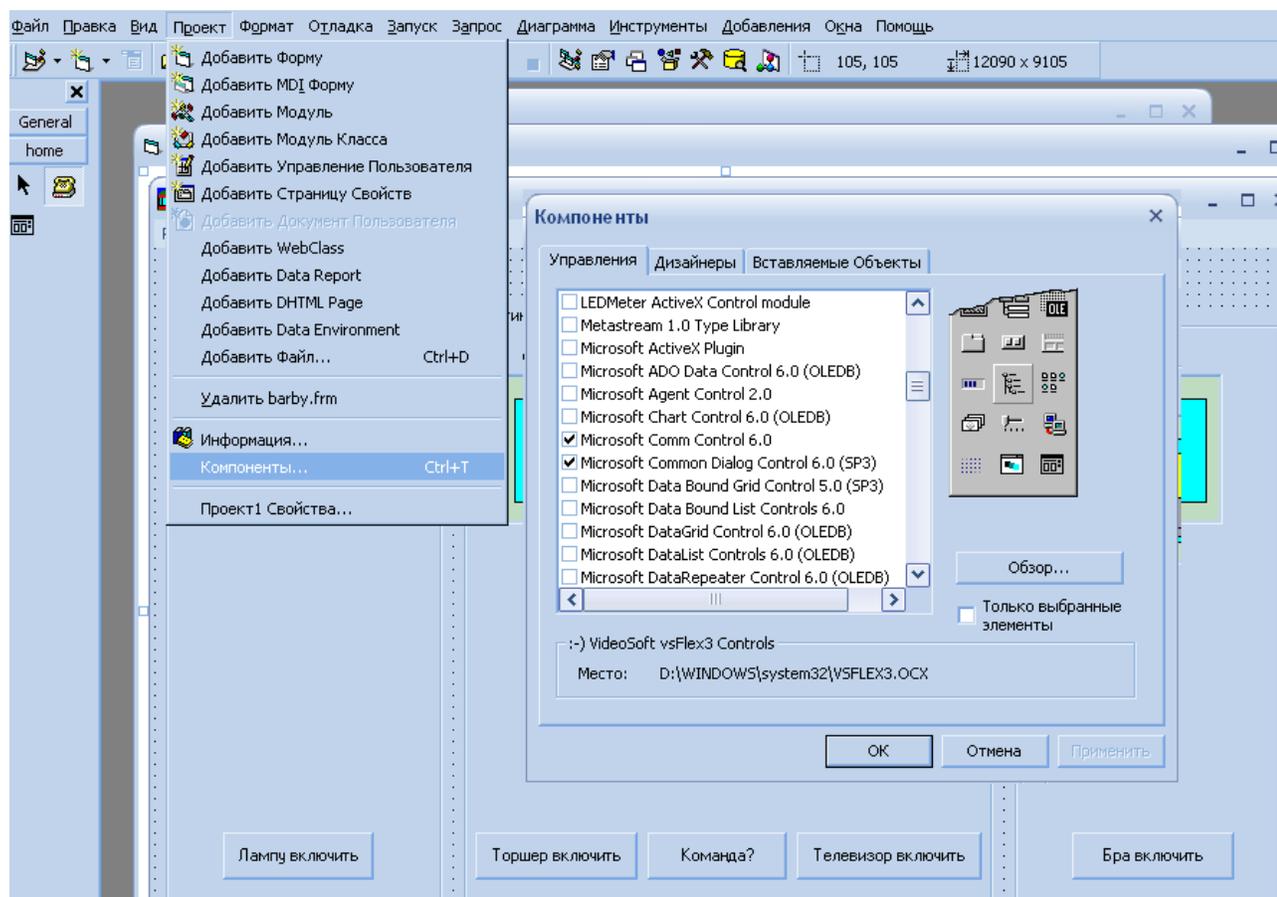


Рис.105

После добавления элемента управления на форму этот элемент управления становится доступен для использования.

*Признаться, я хотел сделать два варианта этой программы на Visual Basic и Delphi, благо есть возможность поработать и с той, и с другой средой программирования. Но в Delphi, установив необходимый компонент VCL, я не смог им воспользоваться по причине отсутствия лицензии на использование компонента. Может быть, я что-то сделал не так, но я не стал без нужды озадачивать тех, кто дал мне возможность пользоваться компьютером, и перешел на Visual Basic, решив, что в этой среде Microsoft не потребует лицензии. Еще раз повторю, что, возможно, я что-то сделал не так, как должно в Delphi, но... а писать работу с COM портом «вручную» в данный момент я считаю преждевременным. Да и зачем, если именно для облегчения работы придуманы VCL, OLE и прочие загадочные составляющие современных сред программирования?*

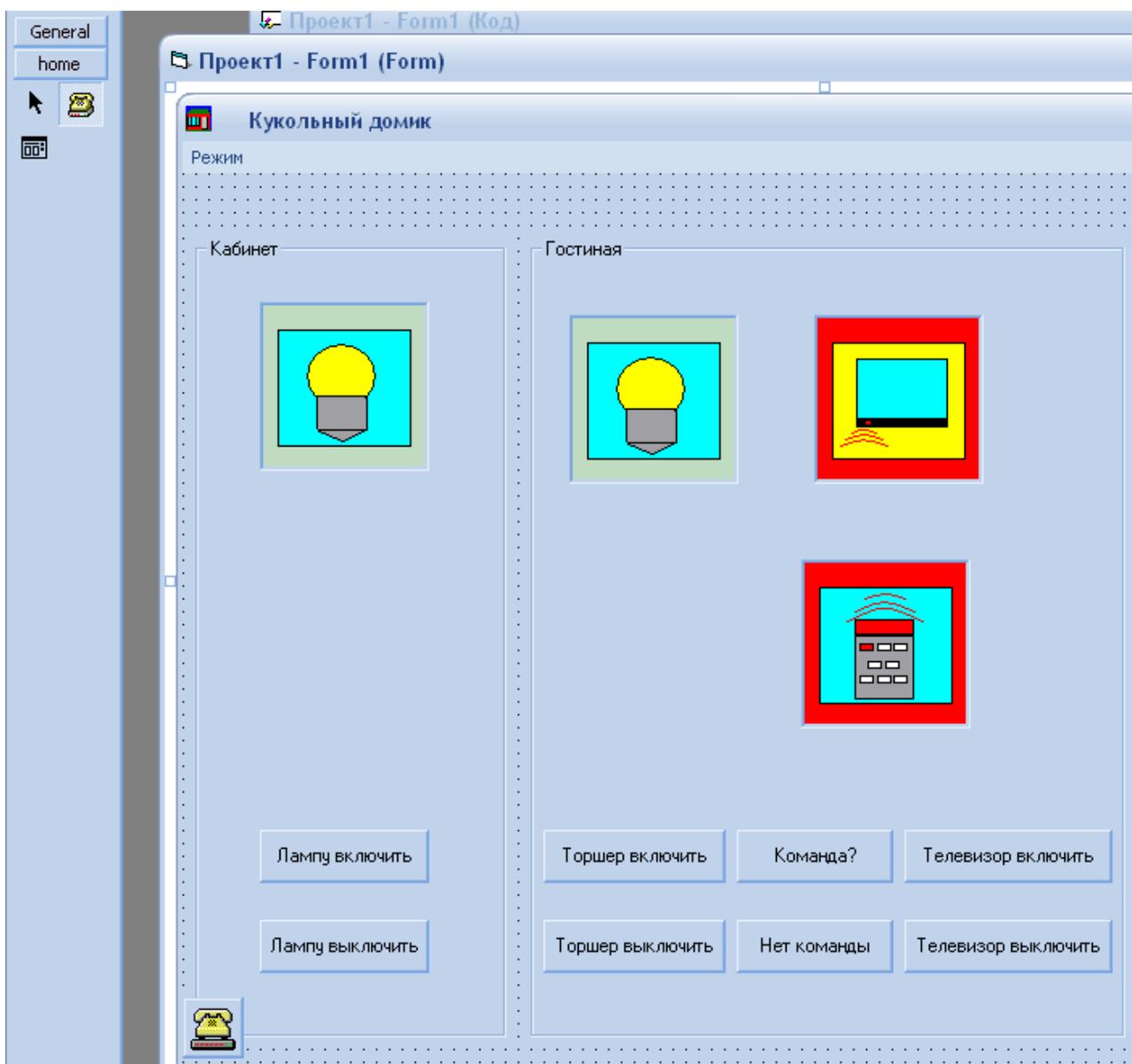


Рис.106

В режим проверки добавим инициализацию порта.

```
Private Sub mnuTest_Click()  
Form1.MSComm1.CommPort = 1  
Form1.MSComm1.Settings = "9600,N,8,1"  
Form1.MSComm1.PortOpen = True  
Form1.Reley01_off.Enabled = True  
Form1.Reley01_off.Visible = True  
И т.д.  
End Sub
```

Как и, впрочем, в рабочий режим.

Изменим управление релейными модулями в режиме тестирования, добавив реальные

## Хобби-электроникс 2. Умный дом.

команды.

Команда «Включить лампу»:

```
Private Sub Reley01_on_Click()  
Form1.MSComm1.Output = "R01$1N"  
Form1.MSComm1.Output = "R01$1S"  
If Form1.MSComm1.Input = "R01#1N" Then  
Form1.Lamp_off.Visible = False  
Form1.Lamp_on.Visible = True  
End If  
End Sub
```

Команда «Выключить лампу»:

```
Private Sub Reley01_off_Click()  
Form1.MSComm1.Output = "R01$1F"  
Form1.MSComm1.Output = "R01$1S"  
If Form1.MSComm1.Input = "R01#1F" Then  
Form1.Lamp_on.Visible = False  
Form1.Lamp_off.Visible = True  
End If  
End Sub
```

Пока я не вставил код инициализации порта при загрузке программы, поэтому при запуске программы и попытках нажать на клавишу «Включить лампу» появляется сообщение об ошибке. Это сообщение пока служит напоминанием, что нужно выбрать режим «Проверки». Вторая особенность работы программы – эта самая лампа, которую я хочу включить или выключить перестала реагировать на команды. Причина в том, что релейный модуль, соответствующим образом настроенный, я пока не включал.

Будем надеяться, что с включенным модулем программа будет работать правильно. С надеждой на это важное допущение исправим работу всех релейных модулей аналогично первому модулю.

Например, для третьего модуля (реле в модуле №1):

```
Private Sub Reley03_off_Click()  
Form1.MSComm1.Output = "R03$1F"  
Form1.MSComm1.Output = "R03$1S"  
If Form1.MSComm1.Input = "R03#1F" Then  
Form1.Bra_on.Visible = False  
Form1.Bra_off.Visible = True  
End If  
End Sub
```

Теперь попробуем обработать прием системной ИК команды фото модулем. Для режима проверки нам достаточно одной системной команды. Пусть это будет команда 001.

## Хобби-электроникс 2. Умный дом.

```
Private Sub Photo01_on_Click()  
Form1.MSComm1.Output = "C01$0S"  
If Form1.MSComm1.Input = "C01001" Then  
Form1.IRcmnd_off.Visible = False  
Form1.IRcmnd_on.Visible = True  
Else  
Form1.IRcmnd_on.Visible = False  
Form1.IRcmnd_off.Visible = True  
End If  
End Sub
```

Работу клавиши «Нет команды» оставим без изменений.

Пришло время разобраться с командой включения телевизора. Подготовим файл, внося, возможно, некоторые исправления в файл input.txt. Мы его использовали с программой MPLAB при разработке модуля ИК команд. Изменим его расширение, превратив в файл input.irc, который я размещаю на диске «D» в папке «barby»:

```
02 02 32 B1 26 5D 26 26 26 5D 26 26 26 5D 26 26 26 26 26 5D 26 5D 26 26 26 5D 26 26 00
```

Здесь я оставил пробелы, чтобы файл был узнаваем, но можно и убрать пробелы, чтобы не заниматься этим в программе.

```
Private Sub IRemitter01_on_Click()  
Form1.TV_off.Visible = False  
Form1.TV_on.Visible = True  
intFH = FreeFile()  
Open "D:\barby\input.irc" For Input As intFH  
Do Until EOF(intFH)  
Line Input #intFH, strString  
strCmnd = strCmnd & strString & vbLf  
Loop  
Form1.MSComm1.Output = "I01$5P"  
Form1.MSComm1.Output = strCmnd  
End Sub
```

Вот такая получилась кнопка. Я честно «срисовал» весь фрагмент программы из полного руководства по Visual Basic 6 Михаэля Райтигера и Геральда Муча (издание ВНУ, «Ирина», Киев, 2000), которое обнаружил на полке. Программа не протестует против этого фрагмента, у меня тоже пока нет оснований для протестов.

Вторую клавишу управления телевизором можно снабдить этим же кодом, или оставить, как есть. Многое зависит от команд включения и выключения. Мой телевизор включается подачей команды канала, а выключается командой power. Пока это не принципиально, важно только, чтобы в файле input.irc была правильная команда. Перед проверкой работы программы в режиме тестирования я удалил строки:

```
Form1.MSComm1.PortOpen = False
```

## Хобби-электроникс 2. Умный дом.

```
Form1.MSComm1.CommPort = 1  
Form1.MSComm1.Settings = "9600,N,8,1"  
Form1.MSComm1.PortOpen = True
```

из Private Sub mnuWork\_Click(). Обозначил ее свойство Visible равным False (в редакторе меню убрал галочку в опции «Включено»). И добавил к фрагменту нажатия на клавишу меню «Проверка» строку Form1.mnuWork.Enabled = True, а к фрагменту нажатия на клавишу меню «Работа» Form1.mnuTest.Enabled = False. Добавил еще один пункт в основное меню «Стоп», нажатие на который описал так:

```
Private Sub mnuStop_Click()  
Form1.mnuTest.Enabled = True  
Form1.mnuWork.Enabled = False  
Form1.MSComm1.PortOpen = False  
End Sub
```

Все это, полагаю, понятно зачем – «заплатки» для правильной работы СОМ порта, который нужно во время открывать и закрывать. Чуть позже поправим (если не забудем) наше лоскутное одеяло, а теперь настал, как говорят: «Момент истины». До написания блока «Работа» основной программы следует проверить тестовую ее часть. Пришла пора собрать модули, подключить конвертор к компьютеру, а модули к конвертеру, и запустить программу. Но это завтра. Сейчас время позднее, я, просто, не успею дома переписать программу, разложиться возле компьютера, и что-то проверить. Однако меня беспокоит фрагмент чтения из файла. Что же я прочитаю? Попробую в очередной раз схитрить.

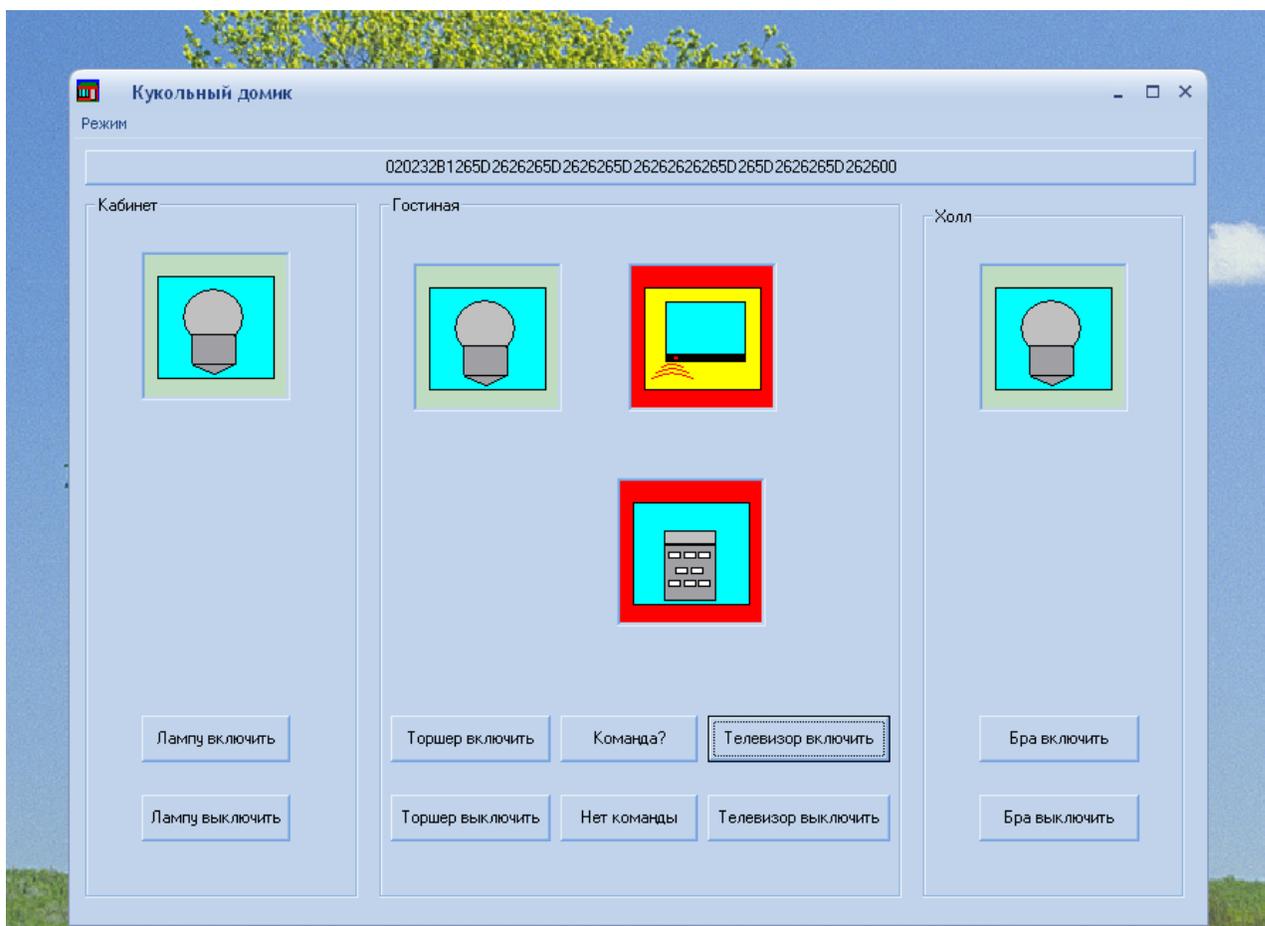


Рис.107

Хитрость в добавленной клавише в верхней части с именем wrem, название которой я убрал. А перед отправкой ИК команды клавишей «Телевизор включить» я добавил строку, выделенную красным цветом:

```
Private Sub IRemitter01_on_Click()
Form1.TV_off.Visible = False
Form1.TV_on.Visible = True
intFH = FreeFile()
Open "D:\barby\input.irc" For Input As intFH
Do Until EOF(intFH)
Line Input #intFH, strString
strCmnd = strCmnd & strString & vbLf
Loop
Form1.wrem.Caption = strCmnd
Form1.MSComm1.Output = "I01$5P"
Form1.MSComm1.Output = strCmnd
End Sub
```

После чтения из файла input.irc и до отправки строки в COM порт все, что было прочитано, выводится на эту клавишу, как название клавиши. Пока у меня не остается сомнений (спасибо Михаэлю Райтигеру и Геральду Мучу), я могу спокойно провести

## Хобби-электроникс 2. Умный дом.

сегодняшний вечер у телевизора, и спать спокойно. До завтрашнего дня. Когда начну проверять модули. Но это будет завтра...

### **Завтра**

Вот оно «Завтра» и пришло.

Ранее уже описаны проверка релейного модуля и модуля приема системных ИК команд. Осталось проверить модуль трансляции ИК кодов. Порядок проверки я предполагаю следующий. Отправляю ИК код, и читаю его с помощью WinLIRC в режиме прямого чтения кода.

Первая правильная мысль – не устанавливать для излучения светодиод ИК-диапазона, а использовать те три светодиода АЛ307, что уже установлены на макетной плате. Возможно, красный светодиод «прихватит» и ИК-диапазон.

Первый результат. После включения клавиши «Телевизор включить» и отправки ИК кода модуль «виснет». А программа WinLIRC хладнокровно выводит результат:

Outputting raw mode2 data.

```
space 6025710
pulse 1086
space 994
pulse 1074
space 1301
pulse 1057
space 978
pulse 1185
space 1011
pulse 1448
space 979
pulse 1147
space 983
pulse 1155
space 1002
pulse 1158
space 1014
pulse 1441
space 998
pulse 1145
space 992
pulse 1133
space 1002
pulse 1163
space 1012
pulse 1442
space 998
pulse 1136
space 976
pulse 1157
space 981
pulse 1156
space 977
```

## Хобби-электроникс 2. Умный дом.

pulse 1158  
space 977  
pulse 1130  
space 1068  
pulse 1467  
space 951  
pulse 1156  
space 1053  
pulse 1427  
space 978  
pulse 1159  
space 999  
pulse 1160  
space 954  
pulse 1183  
space 1013  
pulse 1466  
space 950  
pulse 1104

Я повторяю эксперимент несколько раз, постепенно приходя к выводу, что удобнее фиксировать результат в записи короткие (0) и длинные (1) импульсы, чтобы удобнее было сравнивать с исходным кодом.

Воспроизводимый код: 0001000010000000100000001000000000000100010000000100

Повторное воспроизведение дает тот же код, что уже хорошо.

Оригинальный код имеет вид:

Код клавиши power видеоманитофона:  
Outputting raw mode2 data.

space 12856572  
pulse 2455  
space 532  
pulse 1291  
space 506  
pulse 664  
space 533  
pulse 1266  
space 533  
pulse 665  
space 591  
pulse 1211  
space 533  
pulse 685  
space 515  
pulse 693  
space 526  
pulse 1271  
space 506  
pulse 1265  
197

Хобби-электроникс 2. Умный дом.

space 533  
pulse 666  
space 557  
pulse 1241  
space 533  
pulse 664  
space 24527

Заголовок                    01 0001 0001 000001 01 0001 00

Структура похожа, но код...

Первое, что я делаю, убираю все отладочные вставки в программе модуля. Попутно «смахиваю» функцию `putc()` – модуль работает «молча» - и убираю настройки контроллера, относящиеся к настройке передатчика. Модуль не подает признаков жизни. Возвращаюсь на исходные позиции. Не помогает. Изменяю количество байтов, принимаемых в массив записи команды, с 54 до 29 (реальных). И это не помогает.

Первая «правильная» мысль – а что я читаю из файла `input.irc`? В отладочной программе MPLAB я записывал символы. И читаю, надо понимать, символы. А ожидаю – числа?

Открываем файл в HEX-редакторе и меняем символы на числа. Пробуем:

Outputting raw mode2 data.

space 10739323  
pulse 3650  
space 732  
pulse 1951  
space 733  
pulse 834  
space 755  
pulse 1983  
space 684  
pulse 859  
space 719  
pulse 2009  
space 667  
pulse 833  
space 759  
pulse 811  
space 740  
pulse 1937  
space 742  
pulse 1977  
space 704  
pulse 844

## Хобби-электроникс 2. Умный дом.

space 726  
pulse 1978  
space 705  
pulse 832  
space 2487801

Заголовок 010001000100000101000100

Пытаюсь разобраться с временами, что не сложно – уменьшить числа в файле input.irc. Меняю коэффициент с 13,8 на 19. Беспокоит, что первое нажатие клавиши «Телевизор включить» не воспроизводит код, требуется повторное нажатие. Но времена становятся ближе к исходным.

Делаю попытку включить магнитофон. Неудачно. Возможная причина в единичности кода. «Родные» коды идут сплошным потоком. Возможно, достаточно четырех повторов с паузой в 24505 мс. Пробую переделать программу для реализации четырех повторов. Делаю паузу между повторами с помощью цикла for. Заодно пытаюсь понять, отчего и где зависает модуль?

Дополняю еще одно число в файл irc, чтобы цикл воспроизведения в программе модуля заканчивался правильно. Т.е. чтобы программа не «зависала» на отсутствующем числе длительностью в паузу. Попутно меняю условие завершения чтения с количества чисел (29) на завершение файла «00».

В какой-то момент видеоманитофон даже удается включать-выключать, нажимая клавишу «Телевизор включить». Но модуль продолжает виснуть.

*На все это уходит два дня «борьбы с собственной гениальностью». Но есть и моменты проблесков. Мистика, которой закончилась отладка модуля приема ИК кодов, превратилась в ясное понимание, что лень не всегда хороша.*

*Чтобы не создавать весь проект полностью, я копирую предыдущий в другую папку, правлю имена файлов, и т.д. Но, поправляя программу при отладке, порой забываю, с какой из папок я работаю. Если бы не лень, то создавал новый проект, открывал новые файлы... Словом, получилось, похоже, так, что исходный текст я правил в одной из папок, а компилировал в другую. Вот такая мистика! Даю себе слово впредь удалять в менеджере проекта все файлы, и добавлять заново. Посмотрим!*

За время отладки в голову приходила мысль, что завершение файла в виде «00», может каким-то образом влиять на зависание. Но как? Наконец, когда не остается никаких разумных вариантов, и я устал от бесконечного перепрограммирования микросхемы, я меняю в файле input.irc завершающее число с «00» на «FF», а коэффициент на 25.

Модуль перестает «виснуть», магнитофон исправно включается и выключается. Программа контроллера имеет к этому времени вид:

Файл заголовка:

## Хобби-электроникс 2. Умный дом.

```
#define MODULNAMESIM 'T'
```

```
unsigned char getch(void);  
int init_comms();  
int cmd ();  
void ir_trns();
```

Основной файл:

```
#include <pic16f62xa.h>  
#include <stdio.h>  
#include "ir_trans_fnl.h"
```

```
unsigned char input; // Считываем содержимое приемного регистра  
unsigned char MOD_SIM1; // Первый символ адреса модуля  
unsigned char MOD_SIM2; // Второй символ адреса модуля  
unsigned char command_reciev [6]; // Массив для полученной команды  
int MOD_ADDR; // Заданный адрес модуля, как число  
int MOD_ADDR; // Заданный адрес модуля, как число  
int sim_end_num = 0;  
int MOD_NUM; // Полученный адрес модуля, как число  
int i = 0;  
unsigned char b = 0;  
int c = 0;  
int d = 0;  
int k = 0;  
int l = 0;  
int m = 0;
```

```
unsigned char ir_cmd [60]; // Массив для прочитанной ИК команды
```

```
unsigned char getch()  
{  
    while(!RCIF) /* устанавливается, когда регистр не пуст */  
        continue;  
    return RCREG;  
}
```

```
int init_comms() // Инициализация модуля  
{  
    PORTA = 0x0; // Настройка портов А и В  
    CMCON = 0x7;  
    TRISA = 0x0;  
    TRISB = 0xFE;
```

## Хобби-электроникс 2. Умный дом.

```
RCSTA = 0b10010000;      // Настройка приемника
TXSTA = 0b00000110;      // Настройка передатчика
SPBRG = 0x68;            // Настройка режима приема-передачи
RB0 = 0;                 // Выключаем драйвер RS485 на передачу
CREN = 1;
```

```
    /* Определим номер модуля */
MOD_ADDR = PORTB;        // Номер модуля в старших битах
MOD_ADDR=MOD_ADDR>>4;    // Сдвинем на четыре бита
for (i=0; i<61; ++i) ir_cmd[i] = 0x00;
i = 0;
RA2 = 1;
}
```

```
    /* Преобразуем символьный адрес в число*/
int sim_num_adr()
{
    sim_end_num = 0;
    MOD_SIM1 = command_reciev [1];    // Первый символ номера
    MOD_SIM2 = command_reciev [2];    // Второй символ номера
    MOD_SIM1 = MOD_SIM1 - 0x30;
    MOD_SIM2 = MOD_SIM2 - 0x30;
    sim_end_num = MOD_SIM1*0x0A + MOD_SIM2;
    return sim_end_num;
}
```

```
int cmd()
{
    i=0;
    // Принимаем данные
    while ((input = getch()) != 0xFF)
    {
        ir_cmd[i] = input;
        ++i;
    }
    ir_cmd[i] = input;
}
```

```
void ir_trns()
{
    l = 0;
    for (l = 0; l<4; ++l)
    {
        c = 3;
        while (ir_cmd[c] != 0xFF)    // Наш байт окончания команды
        {
```

## Хобби-электроникс 2. Умный дом.

```
                // Импульс
b = ir_cmd[c];
++c;

while (b != 0)
{
    RA1 = 1;
    RA1 = 1;

    RA1 = 0;
    RA1 = 0;
--b;
}
                // Пауза
b = ir_cmd[c];
++c;

while (b != 0)
{
    RA1 = 0;
    RA1 = 0;

    RA1 = 0;
    RA1 = 0;
--b;
}
}
for (m=0; m<1563; ++m); // Пауза в 25мс
```

## Хобби-электроникс 2. Умный дом.

```
}
  for (i=0; i<61; ++i) ir_cmd[i] = 0x00;
    i = 0;
}

/* Начнем работать */

void main(void)
{
  init_comms();          // Инициализация модуля

  for (k=0; k<6; ++k) command_reciev [k] = ' ';
  command_reciev [0] = 'T';

                                // Прочитаем и преобразуем номер модуля
  MOD_ADDR = PORTB;          // Номер модуля в старших битах
  MOD_ADDR=MOD_ADDR>>4;     // Сдвинем на четыре бита

                                /* Ждем прихода команды и данных */
start: RA2 =1;
      input = getch();

      switch (input)
      {
        case 'T':          // Если обращение к модулю ИК команд
          for (k=1; k<6; ++k) // Запишем команду в массив
          {
            input = getch();
            command_reciev [k] = input;
          }

          MOD_NUM = sim_num_adr(); // Определим к кому адресуются

          if (MOD_NUM != MOD_ADDR) break; // Если не наш адрес
          else
            if (command_reciev [5] == 'P')
            {
              cmd();          // Если команда
              ir_trns();     // Отправим ИК команду на излучатель
            }
            default: goto start;
          }
          goto start;
      }
}
```

HEX-файл для загрузки в программатор:

:10000000830100308A0004282030840078300D20DD

## Хобби-электроникс 2. Умный дом.

:1000100083012D2B04068001840A0406031D0A288F  
:020020000034AA  
:1005520083018C1EA92A1A080800FC01FD01031060  
:10056200FB0CFA0C031CBC2A7808FC077908031858  
:10057200790AFD070310F80DF90D7A087B040319B7  
:100582000034B02A8301AC01AD01D02A2C083C3ED4  
:100592008400831323088000AC0A0319AD0AA92240  
:1005A200A300230FC72A2C083C3E84008313230890  
:1005B200800008008301B401B5013708A0003808A3  
:1005C200A100D030A007A1070A30FA00FB012008E1  
:1005D200F800F901AE2221087C07B4007D08031857  
:1005E2007D0AB500F9003408F8000800830185018E  
:1005F20007309F0083168501FE30860090308312FB  
:10060200980006308316980068309900831206100D  
:1006120018160608A400A5010430F800250DA50C43  
:10062200A40CF80B0F2BAC01AD012D08803AF80099  
:10063200803078023D3003192C020318292B2C0834  
:100642003C3E840083138001AC0A0319AD0A162BC9  
:10065200AC01AD0105150800F722AE01AF013A2B3E  
:100662002E08363E8400831320308000AE0A031920  
:10067200AF0A2F08803AF80080307802063003195A  
:100682002E02031C312B4930B6000608A400A50136  
:100692000430F800250DA50CA40CF80B4B2B7C2B79  
:1006A200AE01AE0AAF012F08803AF800803078021E  
:1006B200063003192E0203186A2BA922A3002E0862  
:1006C200363E8400831323088000AE0A0319AF0A62  
:1006D200542BDB227808A6007908A7002506031D03  
:1006E200742B24082606031D7C2B3B08503A031D5D  
:1006F2007C2BC32283230515A922A300493A03199F  
:10070200512B7C2B8301B001B101B001B101DE2B71  
:100712000330A800A901C52B28083C3E840083139E  
:100722000008A200A80A0319A90AA2080319A92B02  
:1007320085148514851485148514851485148514EF  
:10074200851485148514851485108510A203962BB3  
:1007520028083C3E840083130008A200A80A03195B  
:10076200A90AA2080319C52B8510851085108510CA  
:1007720085108510851085108510851085108510CF  
:1007820085108510A203B22B28083C3E84008313F7  
:10079200000F8D2BB201B3013308803AF800863086  
:1007A20078021B30031932020318DB2BB20A031939  
:1007B200B30ACD2BB00A0319B10A3108803AF80006  
:1007C20080307802043003193002031C892BAC01FB  
:1007D200AD012D08803AF800803078023D300319CF  
:1007E2002C020318FD2B2C083C3E8400831380014D  
:0E07F200AC0A0319AD0AEA2BAC01AD010800F8  
:00000001FF

В основной программе фрагменты для клавиш «Телевизор включить» и «Телевизор

## Хобби-электроникс 2. Умный дом.

выключить» к концу наладки выглядят так:

```
Private Sub IRemitter01_off_Click()
intFH = FreeFile()
Open "D:\barby\input_1.irc" For Input As intFH
Do Until EOF(intFH)
Line Input #intFH, strString
Loop
Form1.MSComm1.Output = "I14$5P"
For i = 0 To 10000
Next i
Form1.MSComm1.Output = strString
For i = 0 To 20000000
Next i
Form1.TV_on.Visible = False
Form1.TV_off.Visible = True
End Sub
```

```
Private Sub IRemitter01_on_Click()
intFH = FreeFile()
Open "D:\barby\input_1.irc" For Input As intFH
Do Until EOF(intFH)
Line Input #intFH, strString
Rem strCmnd = strCmnd & strString & vbLf
Loop
Rem Form1.wrem.Caption = strCmnd
Form1.MSComm1.Output = "I14$5P"
For i = 0 To 10000
Next i
Form1.MSComm1.Output = strString
For i = 0 To 20000000
Next i
Form1.TV_off.Visible = False
Form1.TV_on.Visible = True
End Sub
```

### ***И немного назад***

Поскольку в первой версии единственным средством управления служит старый пульт от видеомэгафона (или телевизора), хотелось бы быть уверенным, что все будет работать правильно (или, хотя бы, работать). Немного переделав основную программу:

```
Dim ext As Boolean
```

```
Private Sub mnuWork_Click()
Form1.mnuTest.Enabled = False
Form1.Command1.Enabled = False
Form1.Command1.Visible = False
```

## Хобби-электроникс 2. Умный дом.

```
Form1.Reley01_off.Enabled = False
Form1.Reley01_off.Visible = False
Form1.Reley02_off.Enabled = False
Form1.Reley02_off.Visible = False
Form1.Reley03_off.Enabled = False
Form1.Reley03_off.Visible = False
Form1.Reley01_on.Enabled = False
Form1.Reley01_on.Visible = False
Form1.Reley02_on.Enabled = False
Form1.Reley02_on.Visible = False
Form1.Reley03_on.Enabled = False
Form1.Reley03_on.Visible = False
Form1.Photo01_on.Enabled = False
Form1.Photo01_on.Visible = False
Form1.Photo01_off.Enabled = False
Form1.Photo01_off.Visible = False
Form1.IRemitter01_off.Visible = False
Form1.IRemitter01_off.Enabled = False
Form1.IRemitter01_on.Visible = False
Form1.IRemitter01_on.Enabled = False
Form1.work
End Sub
```

```
Sub work()
```

```
Do
```

```
    Form1.KeyPreview = True
    Form1.MSComm1.Output = "C14$0S"
    For i = 0 To 10000000
    Next i
    strAnsw = Form1.MSComm1.Input
    Form1.wrem.Enabled = False
    Form1.wrem.Visible = False
    Form1.Refresh
```

```
    Select Case strAnsw
```

```
        Case "C14$0SC14001"
            Form1.Lamp_off.Visible = False
            Form1.Lamp_on.Visible = True
            Form1.Refresh
```

```
        Case "C14$0SC14129"
            Form1.Lamp_off.Visible = True
            Form1.Lamp_on.Visible = False
            Form1.Refresh
```

```
        Case "C14$0SC14193"
```

## Хобби-электроникс 2. Умный дом.

```
Form1.Torsher_off.Visible = False
Form1.Torsher_on.Visible = True
Form1.Refresh
```

```
Case "C14$0SC14033"
Form1.Torsher_off.Visible = True
Form1.Torsher_on.Visible = False
Form1.Refresh
```

```
Case "C14$0SC14161"
Form1.Bra_off.Visible = False
Form1.Bra_on.Visible = True
Form1.Refresh
```

```
Case "C14$0SC14097"
Form1.Bra_off.Visible = True
Form1.Bra_on.Visible = False
Form1.Refresh
```

```
Case "C14$0SC14065"
ext = True
End Select
```

```
Loop While ext = False
```

```
Form1.Command1.Enabled = True
Form1.Command1.Visible = True
Form1.Refresh
End Sub
```

Переделка касается добавления функции work (), где программа «крутится» в цикле после включения режима «Работа», ожидая ввода ИК команд с пульта. Команды включают и выключают «лампы» на форме. Модуль приема ИК команд я проверял, и полагал, что с ним не возникнет проблем. Я ошибался!

При тестировании модуля он правильно отвечал на ИК команды. Но, когда запросы пошли часто, во время прохождения бесконечного цикла основной команды, модуль стал «виснуть». Пришлось вернуться к его наладке. В итоге программа модуля приобрела следующий вид:

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "ir_rec_fnl.h"
```

```
unsigned char input;           // Считываем содержимое приемного регистра
unsigned char MOD_SIM1;       // Первый символ адреса модуля
unsigned char MOD_SIM2;       // Второй символ адреса модуля
```

## Хобби-электроникс 2. Умный дом.

```
unsigned char IRSIM1;
unsigned char IRSIM2;
unsigned char IRSIM3;
unsigned char command_reciev [6]; // Массив для полученной команды
int PHOTOCOME = 0; // Флаг активности фотоприемника
int MOD_ADDR; // Заданный адрес модуля, как число
unsigned char IRCOMMAND = 0;
int MOD_ADDR; // Заданный адрес модуля, как число
int sim1_num = 0;
int sim2_num = 0;
int sim_end_num = 0;
int MOD_NUM; // Полученный адрес модуля, как число
int i;
int k;
int l;
int m;

unsigned char getch()
{
    while(!RCIF)&(RB3 == 1)) /* устанавливается, когда регистр не пуст */
        continue;
    return RCREG;
}

/* вывод одного байта */
void putch (unsigned char byte)
{
    while(!TXIF) /* устанавливается, когда регистр пуст */
        continue;
    TXREG = byte;
}

int init_comms() // Инициализация модуля
{
    PORTA = 0x00;
    CMCON = 0x7; // Настройка портов А и В
    TRISA = 0x00;
    TRISB = 0xFE;

    PCON = 0x8; // Тактовая частота 4 МГц

    RCSTA = 0b10010000; // Настройка приемника
    TXSTA = 0b00000110; // Настройка передатчика
    SPBRG = 0x68; // Настройка режима приема-передачи

    INTCON = 0x0; // Запретить прерывания
    RB0 = 0x0; // Выключим передатчик драйвера RS485
```

## Хобби-электроникс 2. Умный дом.

```
PIE1 = 0x0;           // Настройка таймера 1, запрет прерывания
T1CON = 0x0;          // Выбор внутреннего генератора, бит 1 в ноль
TMR1H = 0x00;
TMR1L = 0x00;

IRCOMMAND = 0x0;

/* Определим номер модуля */
MOD_ADDR = PORTB;     // Номер модуля в старших битах
MOD_ADDR=MOD_ADDR>>4; // Сдвинем на четыре бита
}

/* Преобразуем символьный адрес в число*/
int sim_num_adr()
{
    sim_end_num = 0x0;
    MOD_SIM1 = command_reciev [1]; // Первый символ номера
    MOD_SIM2 = command_reciev [2]; // Второй символ номера
    MOD_SIM1 = MOD_SIM1 - 0x30;
    MOD_SIM2 = MOD_SIM2 - 0x30;
    sim_end_num = MOD_SIM1*0x0A + MOD_SIM2;
    return sim_end_num;
}

int ir_cmd ()
{
    int b = 0;
    CREN = 0;
    while (RB3 == 0) continue; // Ждем окончания заголовка
    for (b=0; b<8; b++)        //Обрабатываем наши импульсы
    {
        while (RB3 != 0)
        continue; // Дождемся импульса
        time (); // Включаем таймер 1
        while (!TMR1IF); // Дождемся такта
        T1CON = 0x0; // Выключаем таймер
        TMR1IF = 0x0; // Сбросим флаг
        if (RB3 == 0) // Если низкий уровень, то "1"
        {
            IRCOMMAND = IRCOMMAND +1; // Запишем это
            time (); // Включаем таймер 1
            while (!TMR1IF); // Дождемся такта
            T1CON = 0x0; // Выключаем таймер
            TMR1IF = 0x0; // Сбросим флаг
        } else // Высокий уровень, значит "0"
        {

```

## Хобби-электроникс 2. Умный дом.

```
        IRCOMMAND = IRCOMMAND; // Запишем это
    }
    if (b<7) IRCOMMAND = IRCOMMAND<<1; // Сместимся влево на бит
}

    PHOTOCOME = 0x0;
    RA0 = 0x0;
    RA1=0x1;
for (m=0; m<4; ++m)
{
    for (l=0; l<10000; ++l);
}
CREN = 1;
RA1 = 0;
}

int cmd()
{
    CREN = 1;
    input = getch();
    switch (input)
    {
        case 'C': // Если обращение к фото модулю
            for (i=1; i<6; ++i) // Запишем команду в массив
            {
                input = getch();
                command_reciev [i] = input;
            }

        MOD_NUM = sim_num_adr(); // Чтение из сети

        if (MOD_NUM != MOD_ADDR) break; // Если не наш адрес
        else
            if (command_reciev [3] = '$') // Если команда
            {
                if (command_reciev [5] = 'S') ir_stat();
            }
            default: break;
        }
    }
}

void time() // Таймер 1 для чтения ИК
{
    TMR1H = 0xFD; // Установка числа циклов до переполнения
    TMR1L = 0x43; // FFFF минус 700 (2BCh)
    T1CON = 0x1; // Включение таймера
}
```

## Хобби-электроникс 2. Умный дом.

```
}
```

```
int ir_stat()
```

```
{
```

```
int ircom;
```

```
int ircom1;
```

```
int ircom2;
```

```
int ircom3;
```

```
command_reciev[0] = 'C';
```

```
command_reciev[1] = MOD_SIM1+0x30;
```

```
command_reciev[2] = MOD_SIM2+0x30;
```

```
ircom = IRCOMMAND; // Преобразуем команду в символы
```

```
ircom1 = ircom/0x64;
```

```
IRSIM1 = ircom1 + 0x30;
```

```
ircom2 = (ircom - ircom1*0x64)/0xA;
```

```
IRSIM2 = ircom2 + 0x30;
```

```
ircom3 = (ircom - ircom1*0x64 - ircom2*0xA);
```

```
IRSIM3 = ircom3 + 0x30;
```

```
if (IRCOMMAND == 0)
```

```
{
```

```
command_reciev[3] = '#';
```

```
command_reciev[4] = 'f';
```

```
command_reciev[5] = 'f';
```

```
CREN = 0x0; // Запрещаем прием
```

```
RB0 = 0x1; // Переключим драйвер RS485 на передачу
```

```
TXEN = 0x1; // Разрешаем передачу
```

```
for (i=0; i<6; ++i) putchar(command_reciev[i]);
```

```
for (i=0; i<1000; i++); // Задержка для вывода
```

```
for (i=0; i<6; ++i) command_reciev [i] = ' ';
```

```
RB0 = 0x0; // Выключаем драйвер RS485 на передачу
```

```
TXEN = 0x0; // Запрещаем передачу
```

```
CREN = 0x1; //Разрешаем прием
```

```
} else // За время между двумя запросами пришла ИК команда
```

```
{
```

```
command_reciev[3] = IRSIM1;
```

```
command_reciev[4] = IRSIM2;
```

```
command_reciev[5] = IRSIM3;
```

```
CREN = 0x0; // Запрещаем прием
```

```
RB0 = 0x1; // Переключим драйвер RS485 на передачу
```

## Хобби-электроникс 2. Умный дом.

```
TXEN = 0x1;          // Разрешаем передачу
for (i=0; i<6; ++i)  putch(command_reciev[i]);
for (i=0; i<1000; i++); // Задержка для вывода
for (i=0; i<6; ++i) command_reciev [i] = ' ';
RB0 = 0x0;          // Выключаем драйвер RS485 на передачу
TXEN = 0x0;        // Запрещаем передачу
CREN =0x1;         // Разрешаем прием
}
IRCOMMAND = 0x0;    // Мы передали команду, она больше не нужна
}
```

*/\* Начнем работать \*/*

```
void main(void)
{
    init_comms();          // Инициализация модуля
    for (i=0; i<6; ++i) command_reciev [i] = ' ';
    command_reciev [0] = 'C';
    PHOTOCOME = 0x0;
    RA0 = 0x00;
        // Прочитаем и преобразуем номер модуля
    MOD_ADDR = PORTB;     // Номер модуля в старших битах
    MOD_ADDR=MOD_ADDR>>4; // Сдвинем на четыре бита

start: RA2 = 0x1;        // Чтобы видно, что модуль включен
        // Нет ИК-сигнала, проверим сеть */
    CREN =0x1;
    if (RCIF)          cmd();
    if (!RB3)          // Появился ИК-сигнал
    {
        CREN =0x0;
        for (k=0; k<30; ++k); // Поставим задержку
        if (!RB3)          // ИК сигнал не пропал?
        {
            PHOTOCOME = 0x1;
            RA0 = 0x01;
                // Обработаем ИК-команду */
            ir_cmd ();
            PHOTOCOME = 0x0;
            RA0 = 0x00;
            goto start;
        }
    } else
    {
        PHOTOCOME = 0x0;
        RA0 = 0x00;
        CREN =0x1;
    }
}
```

## Хобби-электроникс 2. Умный дом.

```
    }  
    goto start;  
}
```

HEX-файл для загрузки в программатор:

```
:10000000830100308A000428203084004C300D2009  
:100010008301102A04068001840A0406031D0A28AD  
:020020000034AA  
:1002D8008301861D71298C1E6C291A080800FD30BF  
:1002E80083018F0043308E009001900A080083013B  
:1002F800CB00831203130C1E7D294B0899000800BC  
:10030800F401F5010310F30CF20C031C9229700898  
:10031800F40771080318710AF5070310F00DF10DC1  
:10032800720873040319003486298301B901BA01DC  
:100338003C08A4003D08A500D030A407A5070A3052  
:10034800F200F3012408F000F10184212508740764  
:10035800B90075080318750ABA00F1003908F000E9  
:1003680008008301850107309F0083168501FE3050  
:10037800860008308E00903083129800063083166D  
:100388009800683099008B018312061083168C013F  
:10039800831290018F018E01A0010608A700A80111  
:1003A8000430F000280DA80CA70CF00BD629080083  
:1003B800830118166C21A6000B2AAD01AD0AAE0107  
:1003C8002E08803AF00080307002063003192D02A2  
:1003D8000318FA296C21A6002D083B3E84008313DC  
:1003E80026088000AD0A0319AE0AE429992170088D  
:0803F800A9007108AA00280603  
:10040000031D042A27082906031D08002430BE0006  
:100410005330C000B62A2608433A031D0800E129DC  
:10042000B521AD01AE011D2A2D083B3E840083138A  
:1004300020308000AD0A0319AE0A2E08803AF00081  
:1004400080307002063003192D02031C142A433039  
:10045000BB00AB01AC0105100608A700A8010430E1  
:10046000F000280DA80CA70CF00B312A0515181662  
:100470008C1ADC218619582A1812AF01B0013008F5  
:10048000803AF000803070021E3003192F020318EA  
:100490004D2AAF0A0319B00A3F2A8619362AAB0142  
:1004A000AB0AAC0105145D22AB01AC010510362A84  
:1004B000AB01AC0105101816362A8301C101C20137  
:1004C0001812861D612AC101C2018619652A73218D  
:1004D0000C1C682A90010C108619752AA00A732139  
:1004E0000C1C702A90010C10762A20084208803AD1  
:1004F000F000803070020730031941020318822A8D  
:100500000310A00DC10A0319C20A4208803AF00084  
:1005100080307002083003194102031C652AAB01C8  
:10052000AC0105108514B301B4013408803AF00021  
:10053000803070020430031933020318B32AB1016A
```

## Хобби-электроникс 2. Умный дом.

:10054000B2013208803AF000A7307002103003196F  
:1005500031020318AF2AB10A0319B20AA12AB30A59  
:100560000319B40A952A1816851008004330830130  
:10057000BB002408303EBC002508303EBD002008EA  
:10058000C500C6016430F200F3014608F1004508D9  
:10059000F000B1237408C7007508C8004708303E52  
:1005A000A1006430F200F3014808F1004708F000B0  
:1005B00084214608F1004508F0007408F002031C8D  
:1005C000F1037508F1020A30F2000030F301B123A3  
:1005D0007408C3007508C4004308303EA2000A3006  
:1005E000F200F3014408F1004308F000842174088C  
:1005F000C9007508CA006430F200F3014808F10030  
:100600004708F00084214608F1004508F00074080E  
:10061000F002031CF1037508F1024908F002031C03  
:10062000F1034A08F1027008C1007108C2004108D4  
:10063000303EA300A008031D632B2330BE006630AC  
:10064000BF00C00018120614831698168312AD015D  
:10065000AE012E08803AF000803070020630031997  
:100660002D0203183D2B2D083B3E84008313000808  
:100670007B21AD0A0319AE0A292BAD01AE012E086C  
:10068000803AF00083307002E83003192D0203181D  
:100690004D2BAD0A0319AE0A3F2BAD01AE012E085A  
:1006A000803AF00080307002063003192D020318E2  
:1006B000AA2B2D083B3E8400831320308000AD0A16  
:1006C0000319AE0A4F2B2108BE002208BF002308E1  
:1006D000C00018120614831698168312AD01AE01DD  
:1006E0002E08803AF00080307002063003192D0287  
:1006F0000318842B2D083B3E8400831300087B21C4  
:10070000AD0A0319AE0A702BAD01AE012E08803A76  
:10071000F00083307002E83003192D020318942B87  
:10072000AD0A0319AE0A862BAD01AE012E08803A40  
:10073000F00080307002063003192D020318AA2B36  
:100740002D083B3E8400831320308000AD0A03193E  
:10075000AE0A962B06108316981283121816A00163  
:100760000800F601F11FBB2BF009F00A0319F10391  
:10077000F1097617F61773088039F606F31FC72BB1  
:10078000F209F20A0319F303F309C72BF601F40186  
:10079000F50172087304031DD02BF001F101003440  
:1007A0001F30F6040310F60AF20DF30D031CD32BD1  
:1007B000F30CF20C73087102031DE02B7208700237  
:1007C000031CE82B7208F0027308031C730AF10281  
:1007D000F40DF50DF60BF61AD82BF61FF42BF409D1  
:1007E000F40A0319F503F5097408F2007508F3001B  
:1007F000761F0034F009F00A0319F103F1090034FF  
:00000001FF

Теперь модуль не «виснет», основная программа «крутится» в цикле. По командам с пульта лампочки на форме включаются и выключаются, а, нажав на цифру «3» на пульте, я

## Хобби-электроникс 2. Умный дом.

покидаю режим работы и могу выключить основную программу.

Здесь тоже есть момент, который меня не устраивает, но с ним, я думаю, можно разобраться во второй части книги. Этот момент относится к выходу из цикла в основной программе. Если я пытаюсь остановить программу с помощью меню, или закрыть ее привычным для Windows образом – не получается. Проблема в том, что программа «крутится» в цикле, из которого может выйти только по событиям, включенным в этот цикл. А единственное событие – прием команды, отправляемой клавишей 3 на пульте. Я думаю, это решаемый вопрос, но к его решению, пожалуй, вернемся позже.

### *Подведем итоги*

Первая версия системы «Кукольный умный домик» завершена.

Конечно, даже после отладки контроллеров модулей в MPLAB пришлось обратиться к отладке на макете. Думаю, это в первую очередь связано с тем, что несколько разработок это еще не опыт работы. Но можно надеяться на изменение ситуации со временем. Полезный вывод из сделанных ошибок – при программировании на языке «С» труднее учитывать реальное время. Например, устанавливая задержку с помощью оператора цикла `for`, который на «С» выглядит, как один оператор, не следует забывать, что на ассемблере операторов будет значительно больше. Наконец, еще один вывод - работать с демо-версиями, создавать себе дополнительные проблемы.

Тем не менее, план, который мы наметили для первой части работы, реализован: есть модули, есть среда программирования системы, пусть не столь красивая и удобная, как в коммерческих реализациях, но работающая.

В кукольном домике, в принципе, можно было бы реализовать ту программу, которую я приводил в качестве примера работы с профессиональными версиями. Но она требует наличия датчиков движения. Для первых экспериментов дороговато.

В целях удешевления первых экспериментов, я уже говорил об упрощении программатора, советую упростить и макет. Пока нет смысла устанавливать реле (дорогостоящий компонент). Все можно собрать на одной макетной плате. Моя макетная плата кроме интерфейсной микросхемы, стабилизатора напряжения и панельки под контроллер, имеет три красных светодиода АЛ307 и фотоприемник TSOP с его «атрибутами». Макетная плата позволила отладить все три модуля. При этом светодиод АЛ307 я использовал в качестве излучателя ИК команд.

Что касается внешнего вида готовых образцов, если вы решите их сделать, сравнительно с изделиями промышленного производства. В первую очередь, значительная часть оборудования может располагаться в шкафах, силовых или слаботочных, т.е. укрыта от глаз. Затем, сегодня можно купить подходящие коробки и коробочки вполне хорошего «товарного» вида. В частности, есть коробки с макетными платами нескольких размеров, предназначенные для установки на DIN-рейку (т.е. в шкаф). Они имеют весьма respectable вид, удобны для сборки в них модулей. А этикетки сегодня можно очень удачно печатать на принтере. Любого вида. Так что, получить вполне приемлемый внешний вид для своих разработок по силам любому. Тем более, что и промышленные образцы

## Хобби-электроникс 2. Умный дом.

зачастую имеют очень «прозаический», хотя и хорошо исполненный, конструктив.

Возвращаясь к предисловию, к тому месту, где говорится о цели написания книги, добавлю небольшое послесловие к первой части книги. Его можно назвать - ода «Ошибкам».

Любого из нас ошибки сердят, вызывают раздражение, могут подорвать веру в собственные силы. Не принято в книге, источнике знаний, делать множество ошибок. Но, если вдуматься, то не наши ли ошибки, хотя мы декларируем, что учиться надо на чужих ошибках, не наши ли ошибки, как наши глаза и уши, ведут нас к знаниям? Как палка слепого, они позволяют определять препятствия, заставляют задуматься, правильным ли путем ты идешь? Собственные ошибки запоминаются лучше, и служат дольше. Когда я попадаю в безвыходную ситуацию, когда отсутствие решения подобно беспросветному мраку, мне помогает простой прием – я не пытаюсь найти правильное решение, я принимаю заведомо ошибочное решение. С ним я работаю до того момента, когда оно либо трансформируется в правильное решение, либо подскажет путь к правильному решению. Самое плохое, что поджидает меня в этом случае, это напрасно потраченное время. Но напрасно ли оно потрачено? Двигаясь неверным путем, я все-таки делаю многое, что позже применяю в других случаях, но, возможно, в схожих ситуациях. Со временем, с опытом приходит интуитивное видение решений. Но это относится в первую очередь к узкой области деятельности. Стоит выйти за пределы этой области и вашей интуиции понадобится палочка, чтобы вы могли пройти весь путь без болезненных падений. Я не люблю ошибаться, не люблю признаваться в ошибках. Я вообще не люблю ошибки! Но понимаю их пользу, как понимаю необходимость и пользу горьких лекарств.

Во второй части мы разработаем еще несколько полезных модулей. Возможно, «причешем» уже готовые модули. Возможно, создадим среду программирования больше похожую на профессиональную версию.

Посмотрим.

## Часть 2. «Детский умный домик»

### **Многословное предисловие**

Во второй части, как и намечалось, я предлагаю расширить набор модулей. В первую очередь за счет модуля цифровых вводов.

Почему я не включил модуль цифровых вводов в первую часть? Модуль, в основном, предназначен для подключения датчиков с «сухими контактами» или аналогичным выходом, имеющих два состояния - включено или выключено. Это противопожарные датчики, датчики охраны. К модулю цифровых входов можно подключить пороговые датчики освещенности – освещенность достигла некоторой величины, датчик замыкает контакты; освещенность упала, датчик выключает контакты. Такими могут быть датчики температуры, влажности, давления, положения и т.д. Большой интерес в этом ряду представляют датчики движения. С их помощью легко организовать автоматическое управление светом в коридоре, прихожей, на лестнице.

Часть датчиков, таких как датчики освещенности и температуры, достаточно просто изготовить самостоятельно. Но пирометрические датчики движения, даже в собственном исполнении, могут оказаться слишком дорогими. Использовать противопожарные или охранные датчики в самостоятельном исполнении я не вижу смысла – эти датчики должны быть очень надежны, требования к устройствам, обрабатывающим их сигналы, очень жесткие. Кроме того, думаю, если вам захочется разработать устройства подобного назначения, то вам не составит труда сделать это самостоятельно. Однако я не рекомендовал бы возлагать большие надежды на использование самодельных противопожарных и охранных устройств.

Тем не менее, во второй части мы разработаем модуль цифровых входов, а в третьей, справочной, я постараюсь привести хотя бы одну схему датчика движения.

Расширение состава модулей для детского умного домика, кроме модуля цифровых вводов, осуществим за счет замены реле в релейном модуле на аналог твердотельного реле, состоящий из оптопары и триака. Подобный модуль, рассчитанный на подключение активной нагрузки, в основном ламп накаливания, хорошо подойдет для включения и выключения света. Мощность нагрузки может составлять 500-600 Вт, габариты позволяют встроить модуль в обычный выключатель света. Попутно мы рассмотрим возможность плавной регулировки яркости света.

Использование релейного модуля для подключения музыкального центра к громкоговорителям, установленным в других помещениях, решает вопрос о распределении звука. Но в ситуации, когда вы намерены в другом помещении не только слушать музыку, но и смотреть фильмы, полезным может оказаться аудио коммутатор. О модуле подобного назначения мы тоже поговорим. Посмотрим, нельзя ли его применить и для цели коммутации видео сигнала?

Конечно, все описанные устройства имеют одно назначение – исследование возможностей микроконтроллеров. Предполагается, что они будут собраны на макетной

## Хобби-электроникс 2. Умный дом.

плате, разложены на столе, а по окончании экспериментов отправятся в ящик с деталями для разборки “по случаю”. Вместе с тем, кому-то, возможно, приглянется идея применить разработки в своей комнате или квартире. Если в квартире не заложено большого количества проводов для организации системной сети, то прокладка проводов может представлять некоторые сложности. Есть несколько решений, позволяющих избежать прокладки проводов для организации системной сети. Я уже упоминал протокол X10 – использование силовых проводов в качестве среды «обитания» системных сигналов. При выборе этого решения настоятельно рекомендую воспользоваться готовыми изделиями, например, обратившись в фирму «Умный дом». Другое решение – радио канал. Или использование инфракрасного излучения для передачи сетевых сигналов. Эти возможности мы обсудим.

И, конечно, постараемся расширить средства управления. Добавить к компьютеру и старому пульту от телевизора что-то, что не менее успешно решало бы эти задачи.

Первоначально я планировал для создания второй часть книги полностью перейти к другой операционной системе. На моем компьютере много лет успешно работает дистрибутив ASPLinux. Linux - операционная система достойная внимания, Она, как правило, продается с огромным количеством программ – текстовые и графические редакторы, поддержка аудио и видео компонент компьютера, среды разработки на разных языках программирования и т.д., и т.п. В Интернете я нашел средства разработки и отладки PIC-контроллеров, что и подало мысль сменить среду работы. Но пока мне не удалось добиться тех же результатов, что и с MPLAB. Если проблемы в моем непонимании, как работать с программами piclab, pikdev или gprsim, то я реализую переход, как только пойму, в чем мои ошибки. Если проблемы глубже, то, вероятно, есть смысл утешить себя мыслью – «от добра добра не ищут!». Посмотрим.

### ***Почему ASPLinux?***

Интерес к Linux возник у меня, скорее, случайно. Первые операционные системы, которые я установил на компьютер, как меня часто поправляли впоследствии, не были, собственно Linux – это QNX и BeOS. Последняя до сегодняшнего дня работает на моем компьютере, хотя я почти не обращаюсь к ней. Позже появилась версия ALTLinux, которая в полной мере меня устраивала, исключая возможность работы в Интернете. Для подключения к Интернету мне приходится использовать VPN. Первая Linux система, которая позволила мне подключиться к Интернету, была ASPLinux 9.0. Выполнение подключения было несуразно “кривым”, однако, работало. После установки следующей версии (ASPLinux 9.2) я повторил все действия, по подключению, которые проделывал раньше, и, их же пытался повторить в версии 10, которой пользуюсь в настоящее время. “Кривая” настройка не удалась, пришлось искать другие варианты, и оказалось, что изобретать велосипед нет смысла – в версии 10 (позже я получил тот же результат и в версии 9.2) достаточно сделать настройки аналогичные настройкам в Windows, чтобы подключиться к Интернету. Практически во всех дистрибутивах Linux есть удобная графическая среда настройки операционной системы, которая называется Webmin. В ней и следовало произвести настройку VPN. Легко и быстро. Но это, когда знаешь.

Итак, ASPLinux. Почему этот дистрибутив? Я не готов оспаривать свою точку зрения, но дистрибутив очень полон, удобен. Скоро выйдет 11 версия, и я думаю, что обязательно

## Хобби-электроникс 2. Умный дом.

куплю новую версию, но причина будет только в любопытстве. Во всем остальном 10 версия меня более, чем устраивает. Нет ни одной причины, по которой ее следовало бы менять на новую.

Для тех, кто никогда не пользовался Linux, я немного расскажу о том, как выглядит эта операционная система (отличие ее в других дистрибутивах будут незначительны). Итак, как выглядит Linux?

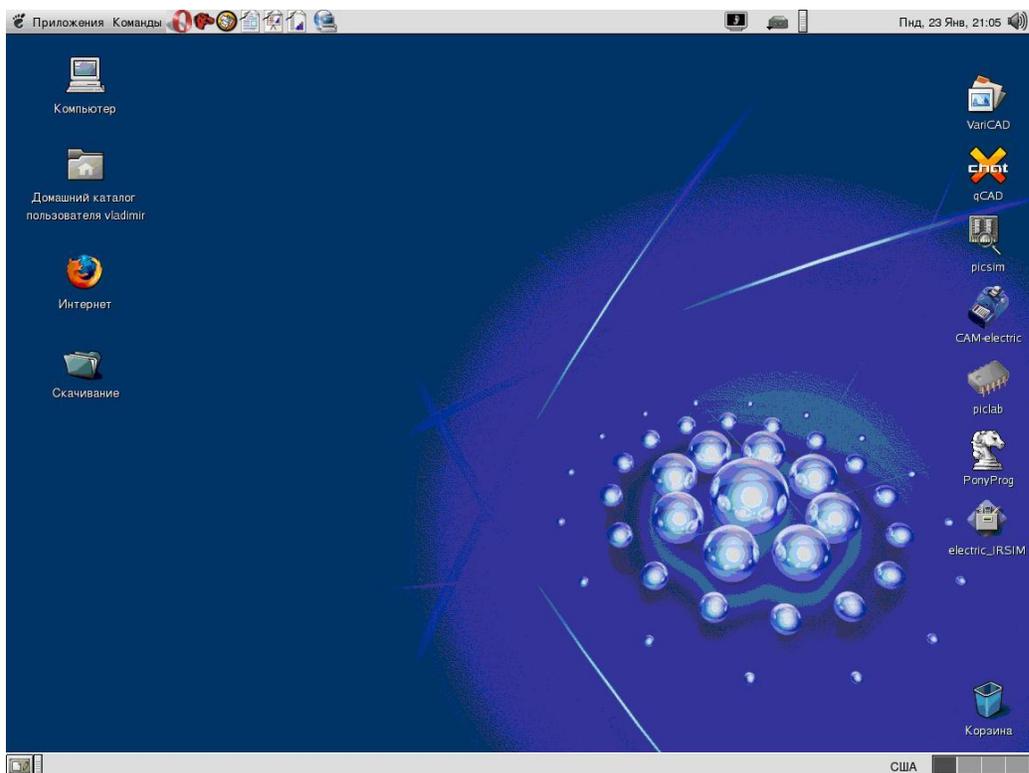


Рис.108

Например, так. С графической оболочкой Gnome. И она же выглядит так:

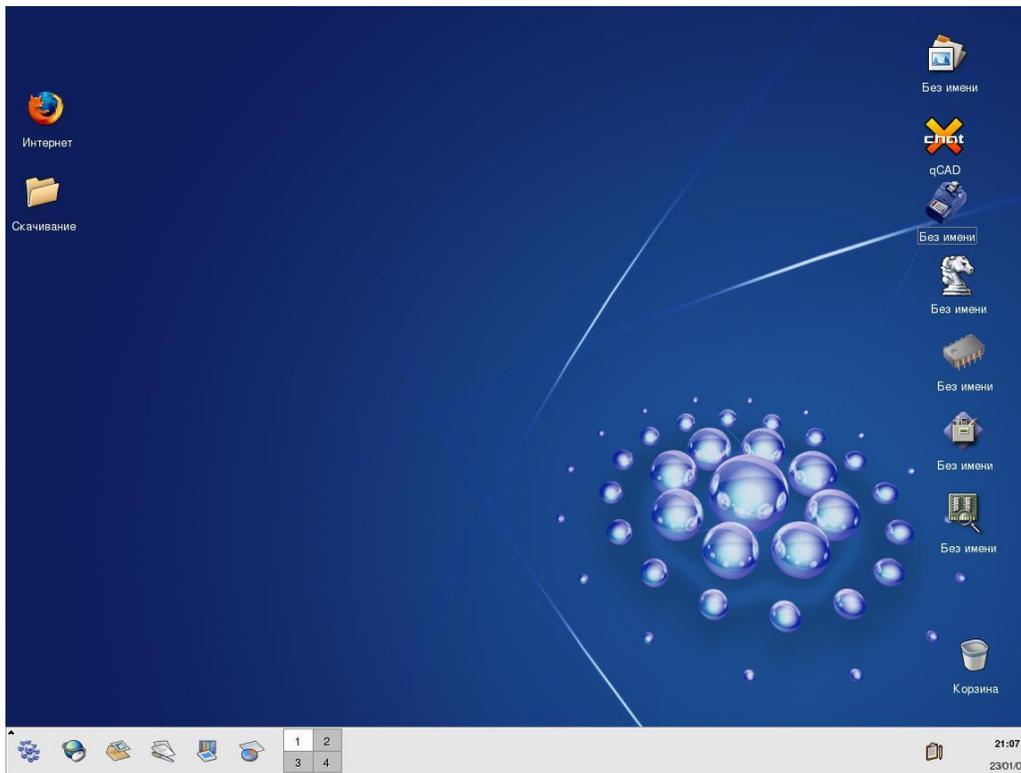


Рис.109

Эта графическая оболочка называется KDE.

Переход от одной графической оболочки к другой происходит после выхода из сеанса и последующего входа в другой сеанс, когда есть возможность выбрать графическую оболочку.

В левом нижнем углу, как и в Windows, есть кнопочка, которая позволяет открыть основное меню и выбрать программу, с которой вы предпочитаете работать (на рисунке открыт раздел программ из дистрибутива для работы с графикой):

## Хобби-электроникс 2. Умный дом.

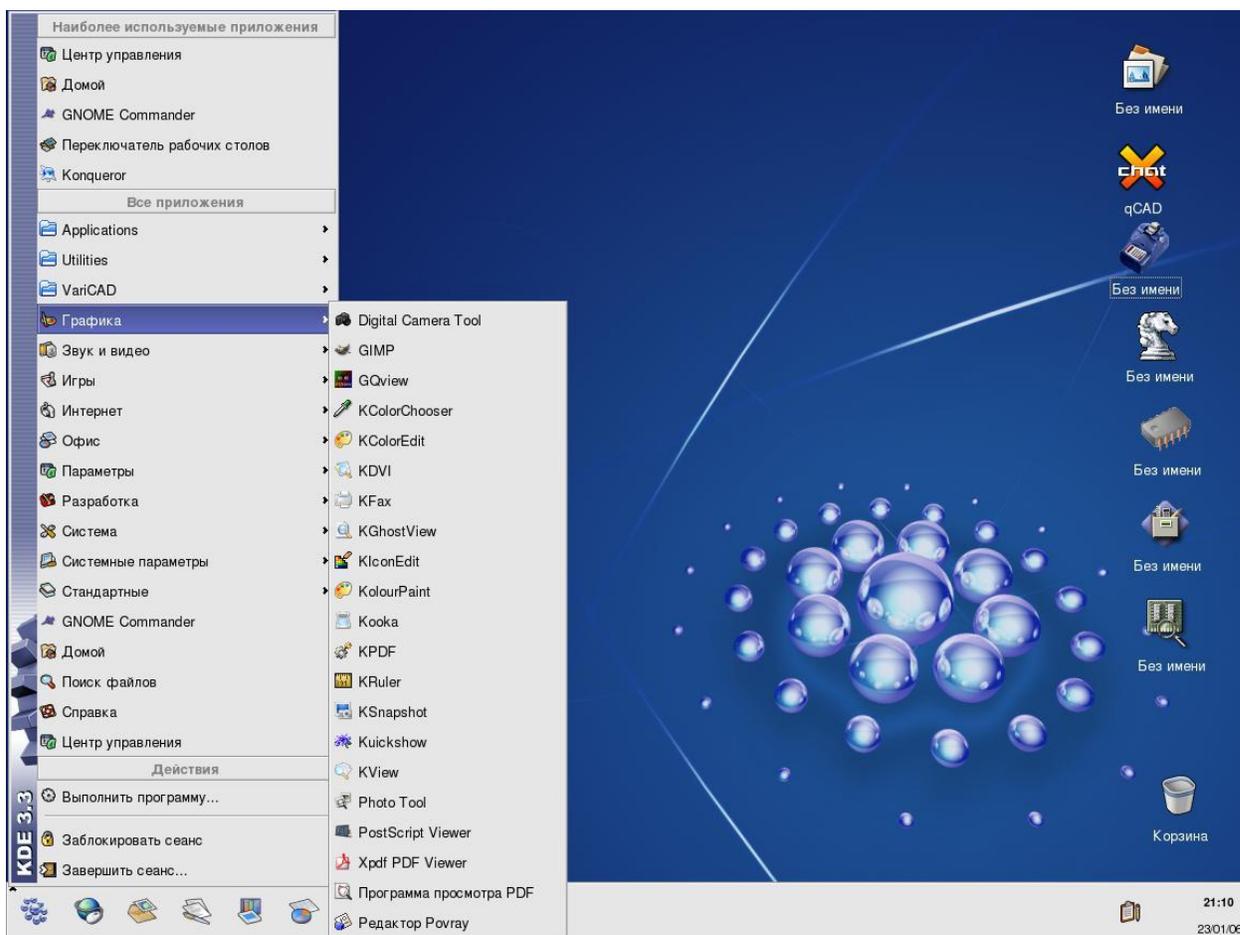


Рис.110

Обе графические оболочки по внешнему виду настолько близки к Windows, что освоение основных приемов работы займет несколько минут. Но это не единственные графические оболочки, которые вы можете менять в зависимости от настроения. Можно использовать экзотические, но не менее интересные и полезные в разные моменты работы. Следующая графическая оболочка отображает сегмент сети, к которому я подключен:

## Хобби-электроникс 2. Умный дом.

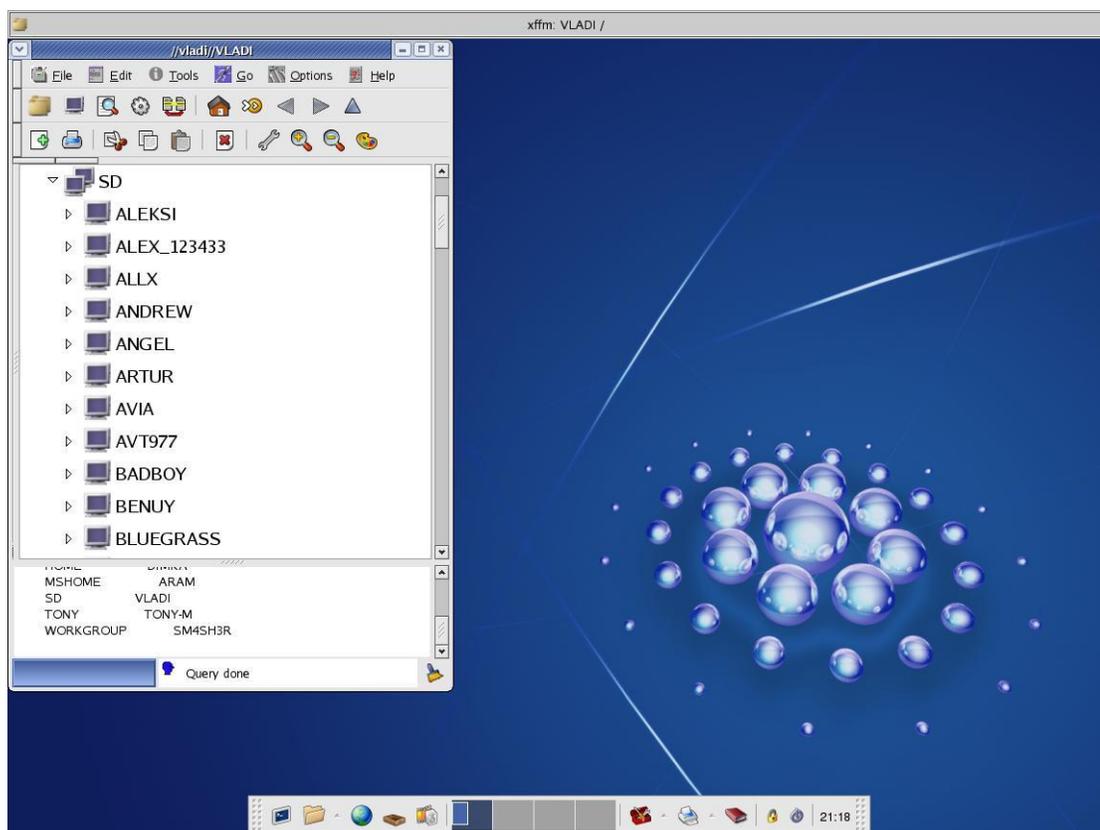


Рис.111

Есть и еще более экзотические графические оболочки, как эта:



Рис.112

## Хобби-электроникс 2. Умный дом.

Я не напрасно упоминал настроение. Настроение, определяющее настрой на работу, очень важный фактор. Если работа безнадежно “застряла”, если задача, которую очень хочется решить, явно не имеет решения, поскольку сделано все возможное и невозможное в поисках решения, то, поменяв фон рабочего стола, можно неожиданно найти решение, о котором ранее и не помышлял. Кроме смены фона Linux позволяет быстро менять тему рабочего стола. Но, главное, можно поменять всю графическую оболочку. И еще. Работая с несколькими программами, можно, как в Windows, переключаться между программами, но можно работать на разных рабочих столах. По умолчанию устанавливается четыре рабочих стола. На Рис.109 и 111 на рабочей панели есть клавиша, состоящая из четырех квадратиков, каждый из которых включает свой рабочий стол. На одном столе можно работать в редакторе, как я это сейчас делаю:

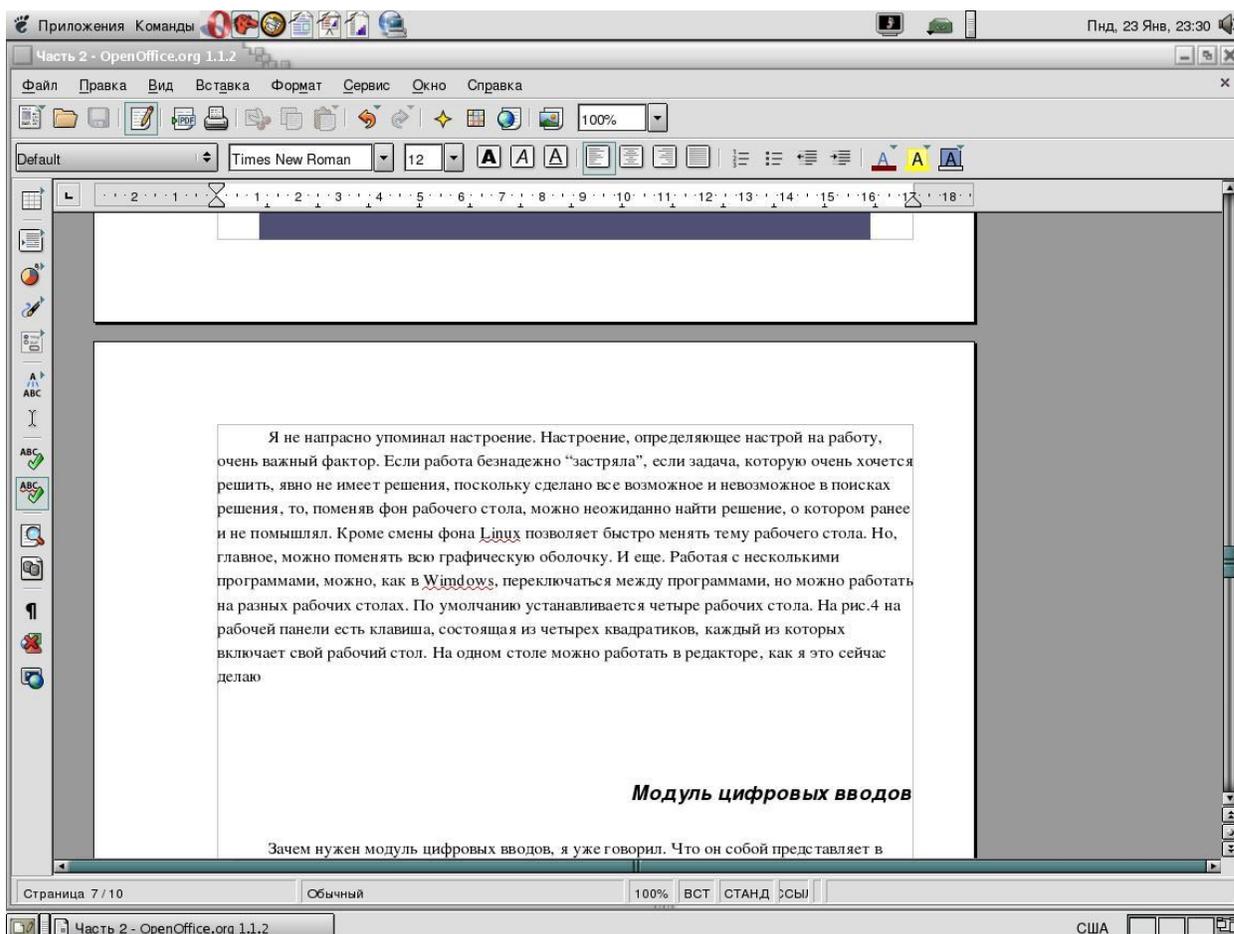


Рис.113

На другом рабочем столе можно работать с программой моделирующей электрические цепи, подобно Multisim из Части 1 книги:

## Хобби-электроникс 2. Умный дом.

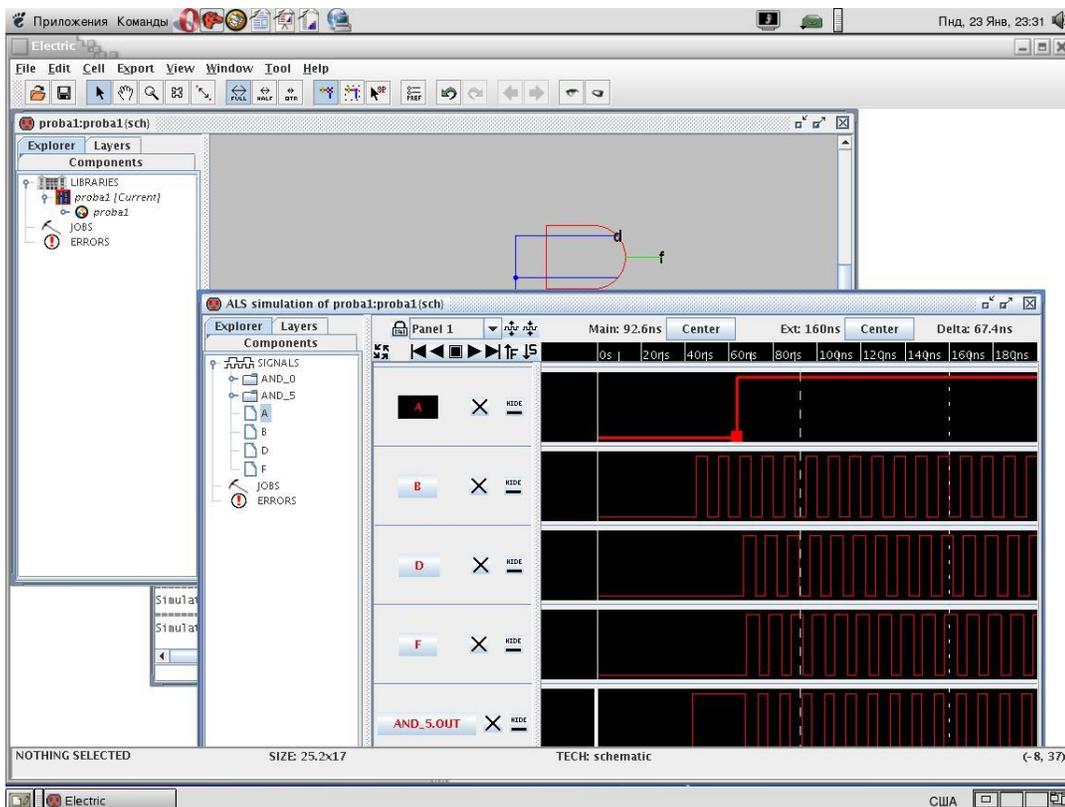


Рис.114

На следующем столе я расположился с редактором piclab (аналог MPLAB):

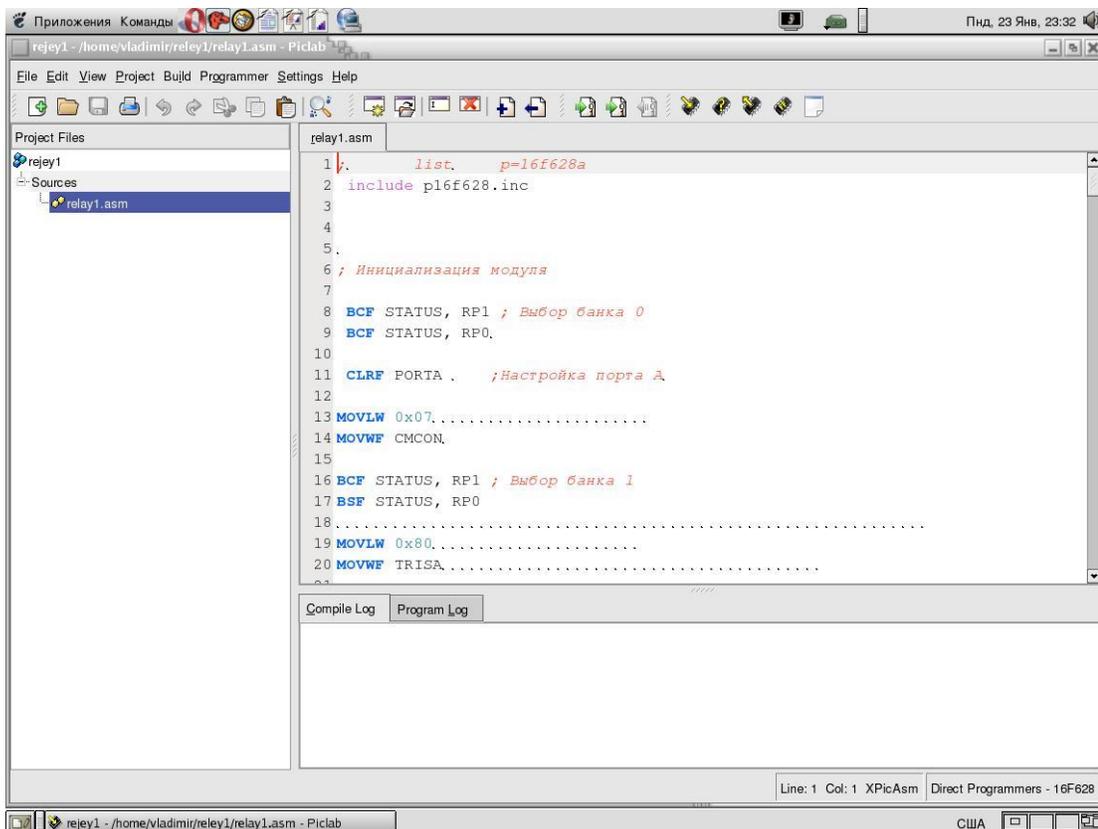


Рис.115

## Хобби-электроникс 2. Умный дом.

А за последний рабочий стол я сажусь для отладки этой программы:

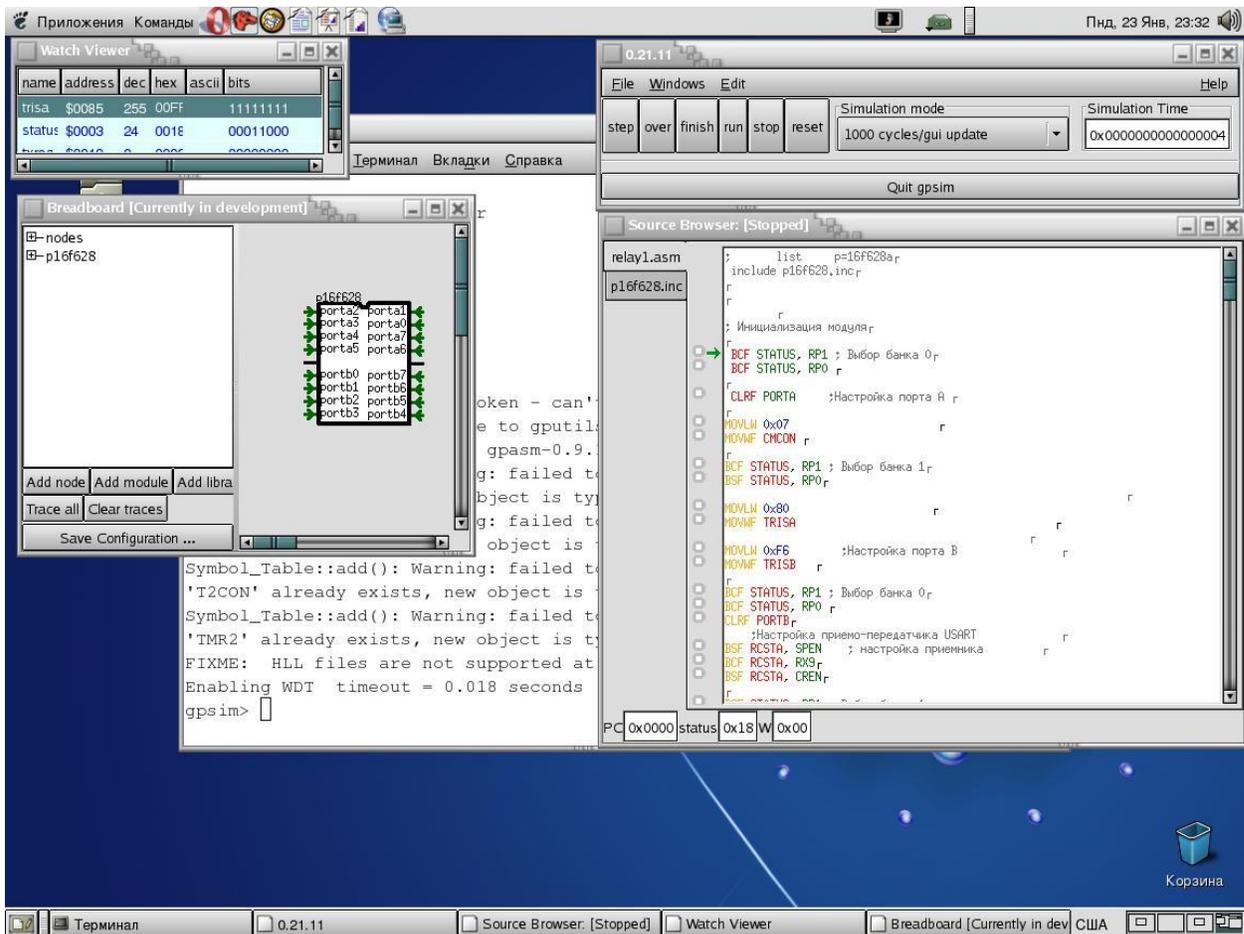


Рис.116

Переключение между столами происходит с помощью одного щелчка мышки. На каждом столе я могу пользоваться несколькими программами, как обычно. Но несколько столов – это несколько столов, что бы кто-то не говорил! Удобно-с!

В завершение первой части многословного предисловия немного расскажу о программе Electric VLSI Design System (аналог Multisim), но прежде еще несколько слов о Linux.

Есть два подхода к этой операционной системе. Две точки зрения. Одна точка зрения больше подходит профессионалам, но не обязательно. Люди, разделяющие ее, ценят в Linux быстроту и мощь использования современных компьютеров. Что я имею в виду?

Сегодняшний компьютер имеет тактовую частоту больше 2 ГГц, оперативную память порядка 1 Гбайт, видео память 256 Мбайт, винчестер емкостью более 100 Гбайт. Но, работая в Windows, я, а я могу говорить только о себе, порой диву даюсь, насколько мало моя работа отличается от работы за компьютером десятилетней давности. Компьютером, который называли “трешкой” за процессор 386. В один из дней, огорченный этим обстоятельством, я решил установить MS DOS и Нортон-командер, и никогда больше не пользоваться Windows. Но DOS 6.22 на компьютер уже не устанавливалась.

## Хобби-электроникс 2. Умный дом.

Linux позволяет работать без графической оболочки, и, думаю, работает очень быстро. Есть и аналог Нортон-командер. В таком варианте Linux действительно выглядит очень привлекательно – быстрая и компактная операционная система. С подобной точки зрения, во многом справедливой, графическая оболочка – пустая трата ресурсов компьютера.

Другая точка зрения – пользовательская. Для пользователя, к их числу я отношу в первую очередь себя, привычная среда обитания и ленивое пощелкивание по картинкам на экране дороже скорости и компактности. Для пользователей и создано большое количество графических оболочек, о которых я упоминал выше. Одни из них мало отличаются от привычной среды Windows, их и создавали с целью облегчить переход пользователям с Windows на Linux. Другие разительно отличаются от Windows. Есть графические оболочки еще более непохожие на те, что я упоминал. Сказать по правде, только выяснение, как работать с экзотическими графическими оболочками, или какие чудесные вещи можно сделать в WM (Windows Maker), могут дать много радостных часов, проведенных за компьютером с операционной системой Linux. Я уж не говорю о том, что можно создать свой вариант Linux – исходные тексты доступны, есть руководства, рассказывающие о том, как это сделать. Словом, какую точку зрения на Linux вы ни примете, если вы не работали с Linux, попробуйте, мне кажется, вы не пожалеете. Например, на моем компьютере, начиная с ASPLinux9.0 установлена программа VariCAD – очень хороший аналог AutoCAD для Linux. Думаю, последние версии, не уступают AutoCAD ни в чем, кроме цены. Я установил VariCad из первого дистрибутива Linux, коробочной версии. Программа оказалась демо-версией, работающей в течение 30 дней, пользоваться я ею не пользовался, но и “сносить” не собираюсь. Итак, первая программа, по аналогии с Частью 1 этой книги, Electric VLSI Design System.

### ***Работа в среде Electric***

Сразу оговорюсь, хотя я подозреваю, что в программе можно моделировать великое множество схем, я попробовал работу только простейшей схемы. Я не уверен, что обращение к программе потребуется в дальнейшем, и, следовательно, не хочу тратить время на ее освоение до того момента, когда в этом будет необходимость. Программу можно найти на сайте: <http://www.staticfreesoft.com/>

Для работы программы необходимо установить пакет Java. Полагаю, как и у меня, он найдется в дистрибутиве под именем `jre-1.5.0-1asp.i386.rpm` или аналогичным. Программы для установки, почти без головной боли, в Linux имеют расширение `rpm`, но не для всех дистрибутивов. После установки пакета можно запускать программу. Я установил ее первоначально в своей домашней папке, но затем перенес в папку `/usr/etc`. Для запуска создал кнопку запуска (ярлык) с командой `java -jar /usr/etc/electric.jar -mdi`, где последние символы (`-mdi`) означают, что я хотел бы видеть все части программы в едином окне. Что совсем не обязательно.

После запуска программы появляется окно:

## Хобби-электроникс 2. Умный дом.

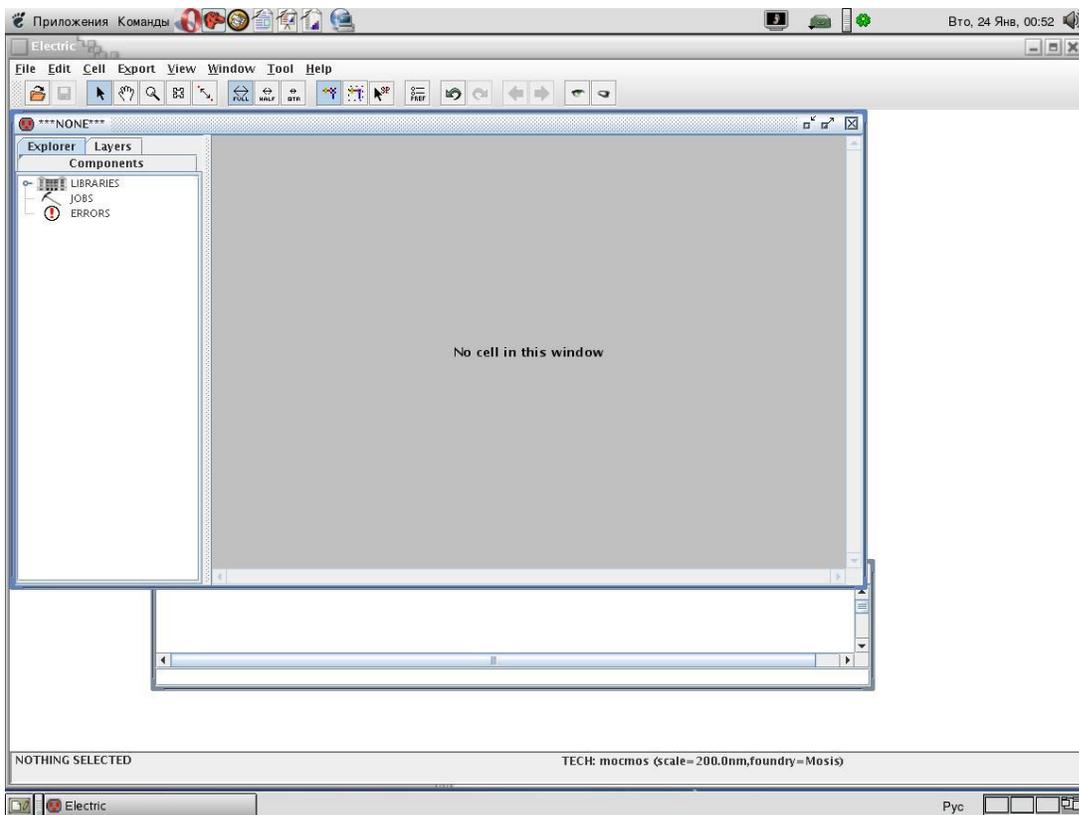


Рис.117

Можно сразу обратиться к руководству пользователя, которое находится в разделе Help->User's Manual:

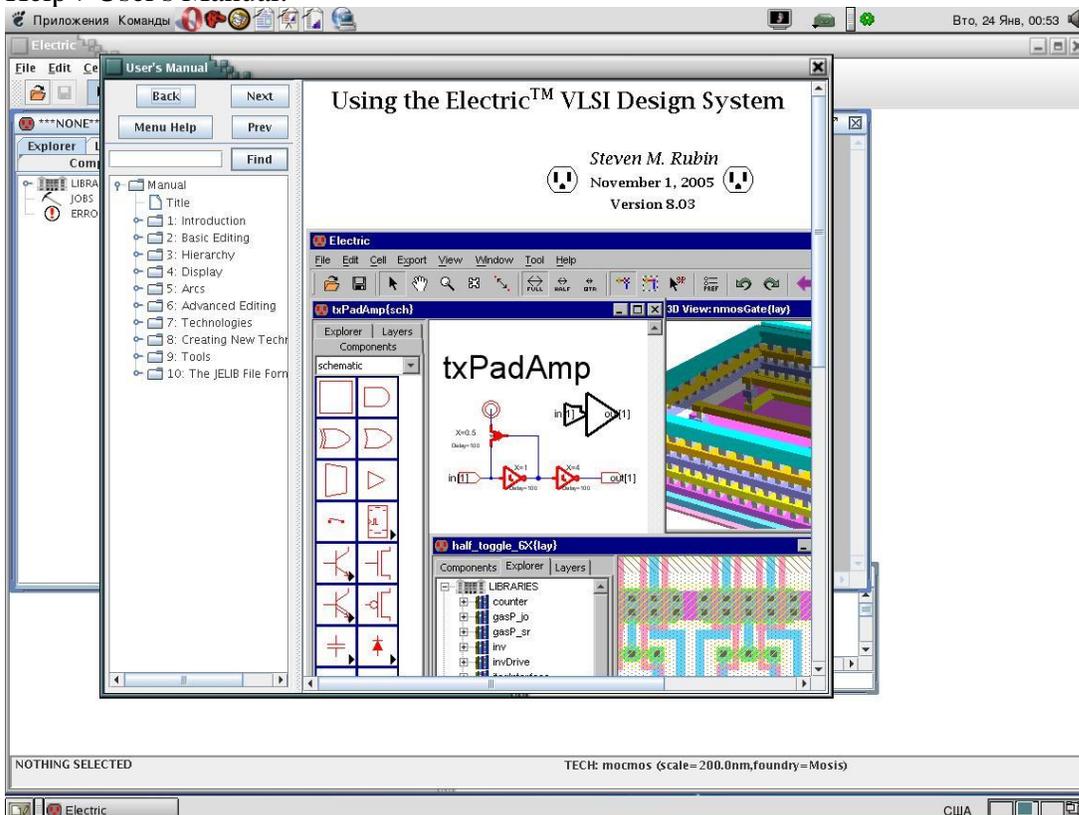


Рис.118

## Хобби-электроникс 2. Умный дом.

Надеюсь, вам это руководство поможет больше, чем мне. Оно достаточно подробно, хорошо иллюстрировано, удобно в обращении. Но написано, как мне кажется, для пользователей Windows, хотя есть все уточнения и для MAC, и для пих-пользователей.

Чтобы убедиться в том, что программа работает, можно сделать следующие, возможно, и не правильные шаги.

Сохраним библиотеку с помощью меню File->Save Library As..., где в открывшемся диалоговом окне выберем заранее созданную в своей домашней папке папку под названием, положим, Electric\_projects. Открывать папку лучше клавишей Open. Затем изменим поиме.jelib на любое имя первого проекта, например, first.jelib, и нажмем на появившуюся клавишу Save. Теперь выберем в меню раздел Cell->New Cell...; в диалоговом окне, которое появится, выберем schematic и впишем имя first в окне для имени первой схемы. Подтвердим выбор клавишей ОК.

Надпись об отсутствии Cell пропадает, вместо нее появляется курсор в виде крестика. Выберем слева в проводнике ярлычок Components, а в открывшемся меню компонентов выберем схему И (на рис.119 видно меню компонентов). Схема И – верхняя правая, которую мышкой перенесем на рабочее поле чертежа:

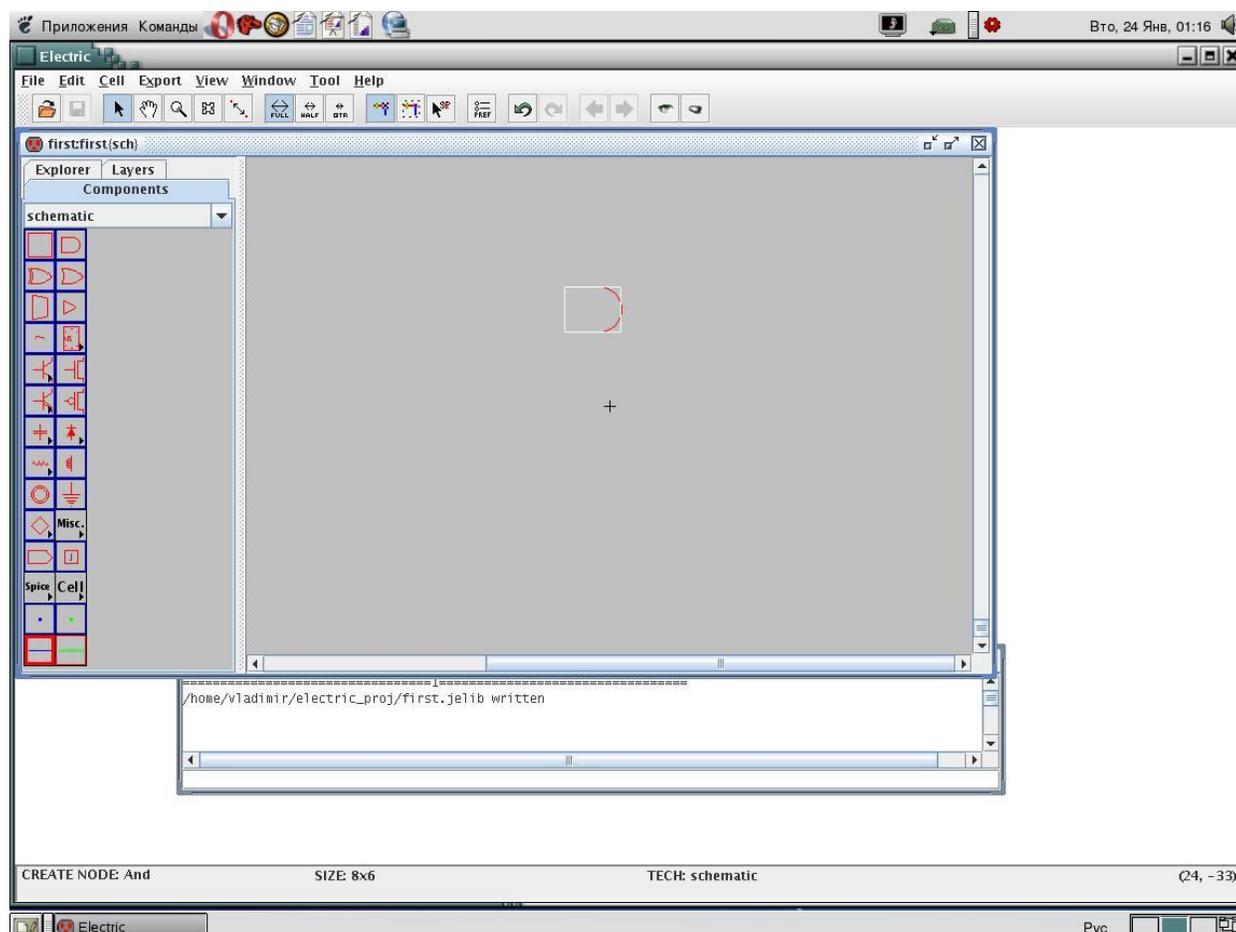


Рис.119

Теперь, если поводить маркером по картинке, то видно, как на полуокружности задней

## Хобби-электроникс 2. Умный дом.

части объекта появляются три крестика – один в верхней части, второй посередине, третий в нижней части. Если выделить верхнюю метку левой клавишей мышки, а затем провести, удерживая правую клавишу в сторону передней линии фигуры линию, которую, в свою очередь, возможно, потом нужно будет выделить, нажав на метку левой клавишей мышки, то... Иногда мне удается добиться этого, просто, сразу щелкнув правой клавишей мышки, когда маркер выделяет метку... то получим следующего вида картинку, которую я для наглядности увеличил:

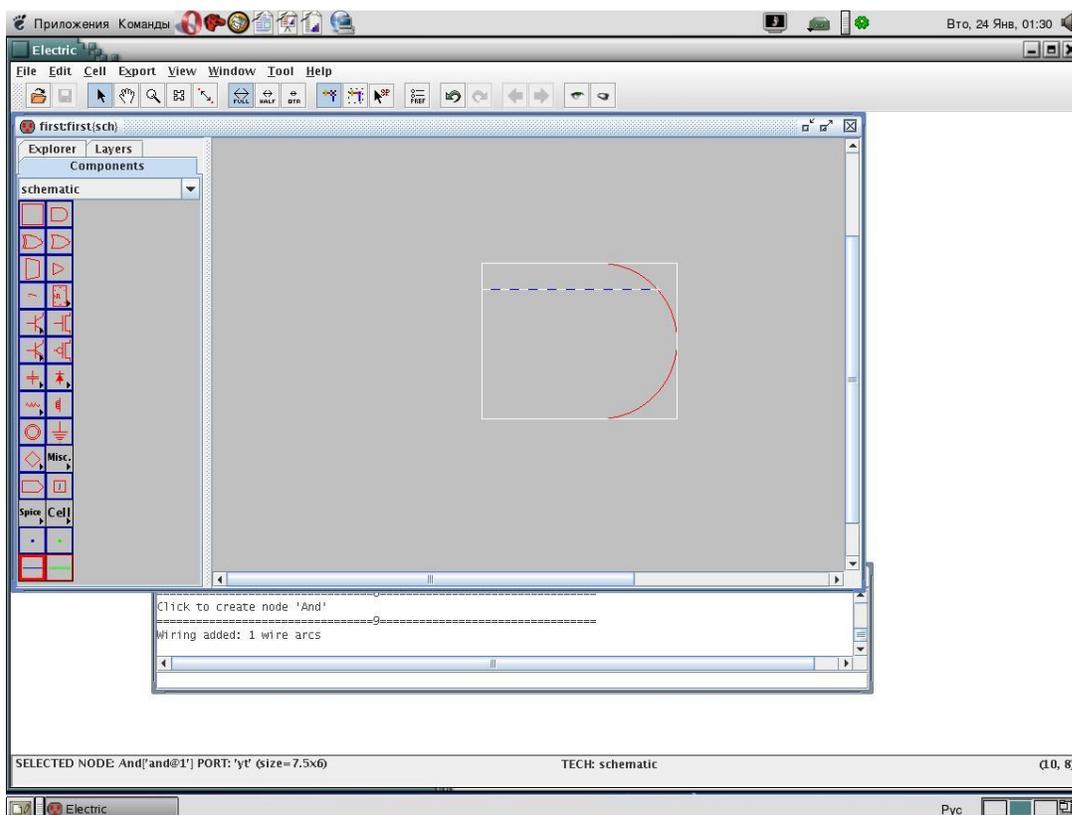


Рис.120

В разделе меню Export->Create Export, в диалоговом окне для Export Name зададим «а». В окошке же ниже выберем «input»:

## Хобби-электроникс 2. Умный дом.

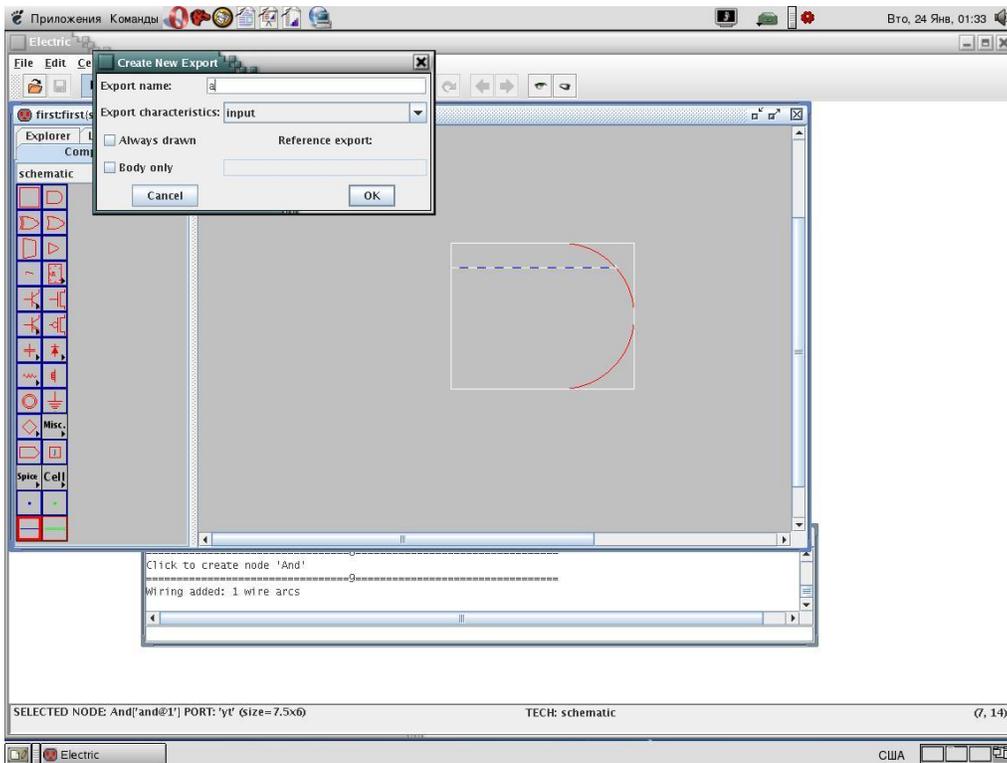


Рис.121

Ну, и по доброй уже традиции нажмем ОК. Теперь в разделе Tool->Simulation (Built-in)->ALS: Simulate Current Cell, нажав все в правильной последовательности, получим следующее:

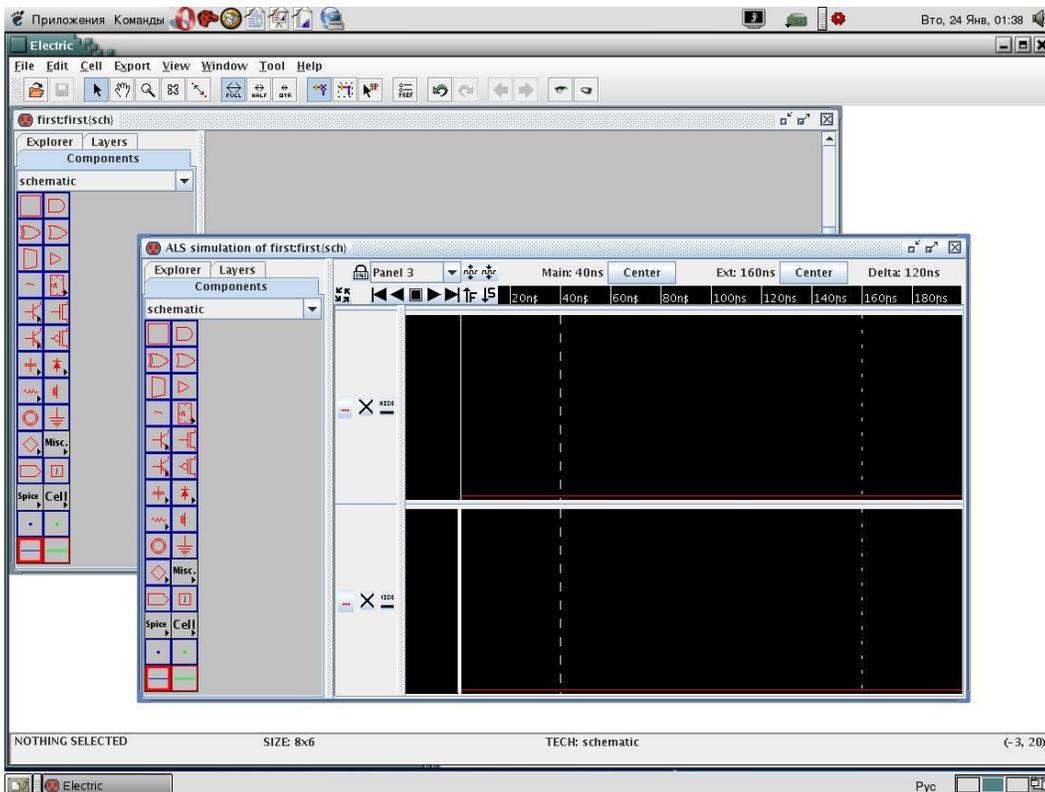


Рис.122

## Хобби-электроникс 2. Умный дом.

Теперь можно закрыть окно наблюдения; выделить нижнюю метку, аналогично верхней, и аналогично верхней через разделы меню Export и Tool получить отображение обоих входов. Выделив метки входов курсором и щелчком левой кнопки мышки, правой кнопкой мышки можно “удлиннить” входы микросхемы. Можно перенести буквы, обозначающие входы, вперед, чтобы микросхема приобрела более привычный вид. Чтобы упростить последнюю процедуру, я на вкладке проводника проекта Explorer в верхней части левого окна раскрываю проект first (Current), first, выделяю название окна схемы first(sch), правой клавишей мышки получаю выпадающее меню, в котором выбираю раздел Edit in New Window. После того, как окно открывается, можно закрыть предыдущее, а в новом легче выделить буквы и перетащить мышкой в новое место.

Микросхема не обязательно должна выглядеть иначе, чем выглядит на Рис.120 или 121. Это дело привычки и удобства. Чтобы оживить окно наблюдения, выделяем вход, например, b (я полагаю, что раздел меню Tool->Simulation (Built-in)->ALS: Simulate Current Cell уже пройден) через выделение метки входа b на дуге. Затем в меню выбираем Tool->Simulation (Built-in), где, в свою очередь, выбираем пункт Set Clock on Selected Signal. Соглашаемся с предложенной частотой (ОК). Вид окна наблюдения сразу изменится:

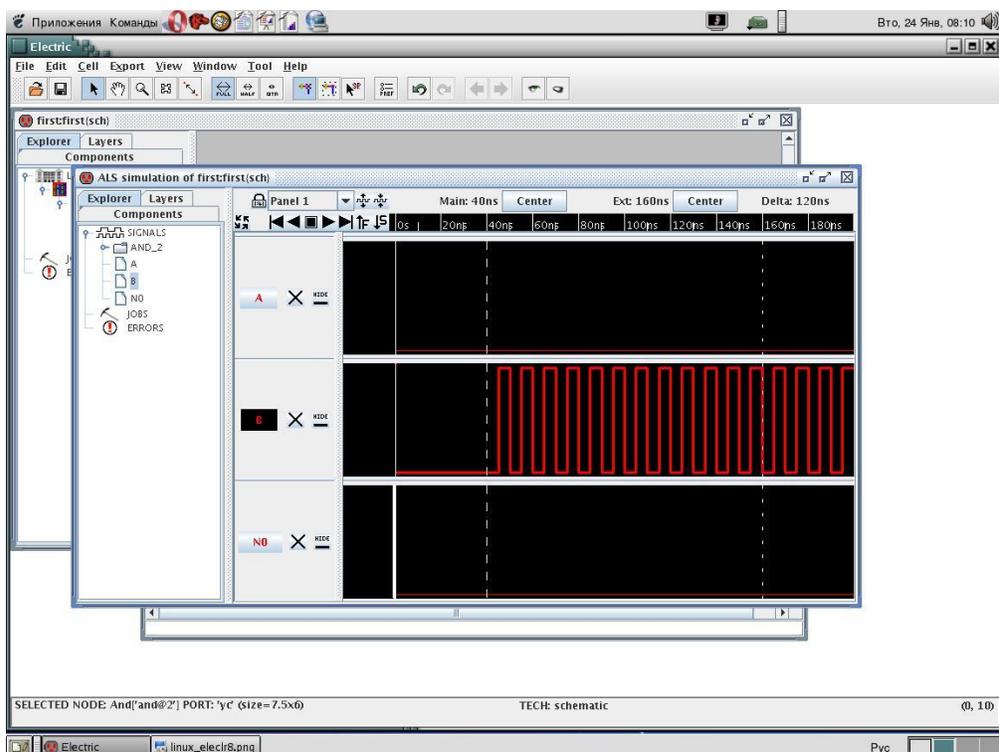


Рис.123

Воспользуемся клавишами управления, подобными клавишам аудио и видео проигрывателей под надписью Panel1, запустим время клавишей ►. И, когда подвижной маркер дойдет до временной отметки 60-80 ns, остановим его клавишей стоп ■. Мне приходится быстро нажимать ее несколько раз. Теперь вернемся к чертежу, выделим вход «а», войдем в меню Tool->Simulation (Built-in) и выберем пункт Set Signal High at Main Time. Вид окна наблюдения изменится, что с моей точки зрения свидетельствует о правильном отображении событий. Сигнал на выходе появляется в тот момент, когда он принимает

## Хобби-электроникс 2. Умный дом.

значение логической “1” на входе «а»:

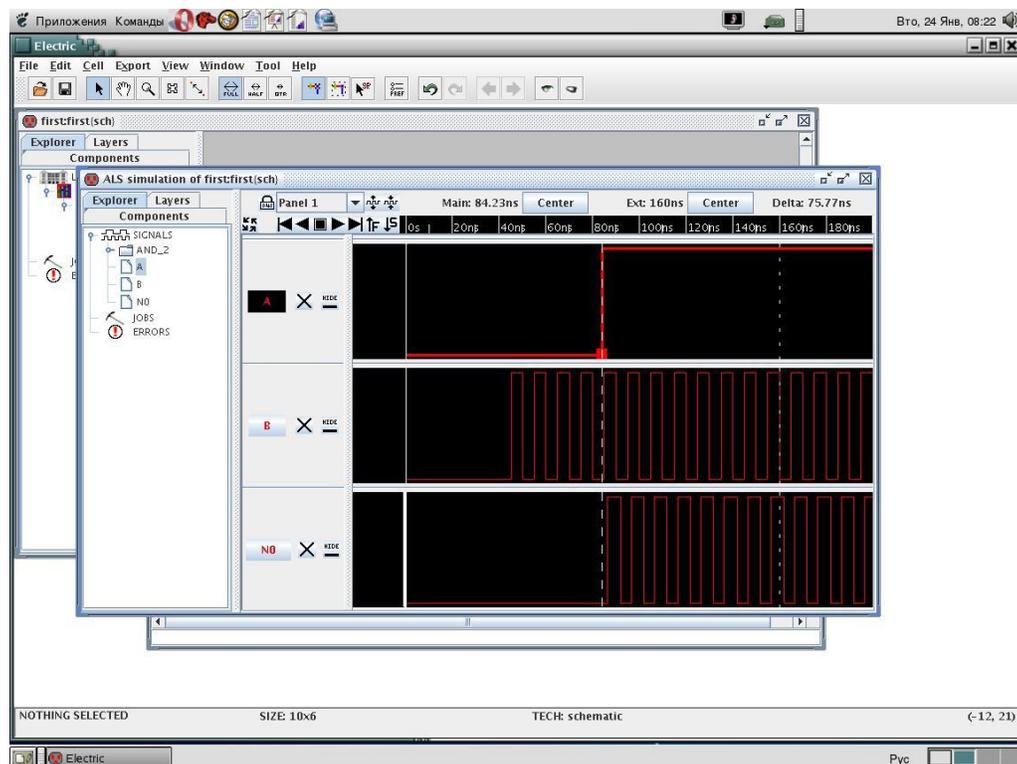


Рис.124

Повторно запуская маркер клавишей ►, и останавливая через некоторое время у отметки, например, близкой к 120 ns (мы оставили вход «а» на схеме в выделенном состоянии), входим в раздел меню Tool->Simulation (Built-in), но теперь выбираем пункт Set Signal Low at Main Time. Вид окна наблюдения меняется, и мы получили вариант работы схемы, управляемой по входу «а» и тактовым генератором на входе «b». Если временной интервал задавать с помощью таймера на микросхеме 555, думаю, он будет работать с нужными временами, а этих таймеров сделать несколько. Если в свой черед к выходу микросхемы “И” подключить светодиод, ИК или АЛ307, как это было в Части 1, а частоту тактового генератора выбрать равной 37 кГц, то можно придумать альтернативную схему излучения ИК команды. Если это кому-то надо?

По крайней мере, мы получили пачку импульсов, готовую к употреблению:

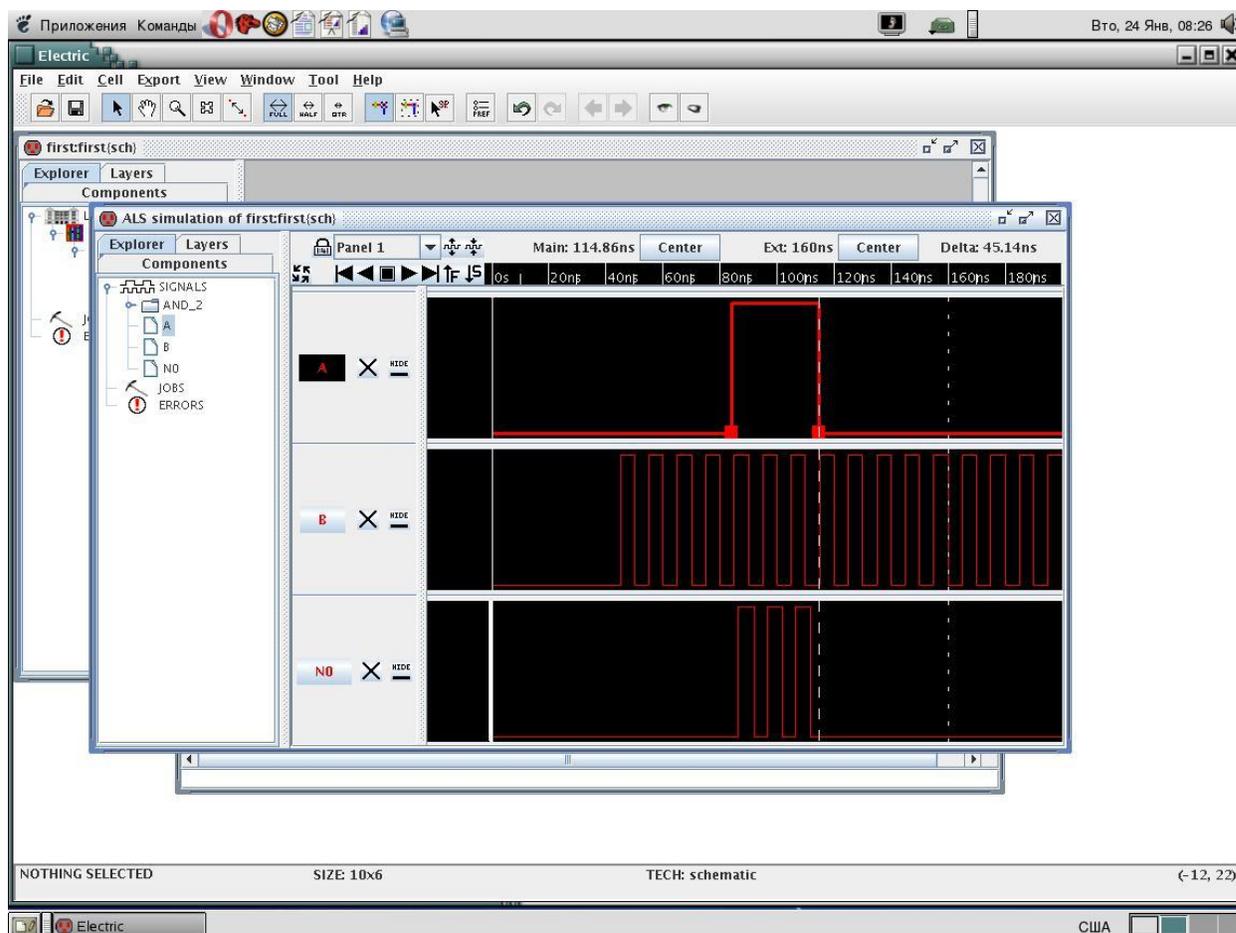


Рис.125

Добавлю еще одно замечание – программа позволяет создать, как pcb-программа, вид печатной платы, но я не пробовал. Лично я предполагаю вернуться к разработке модуля, о котором уже говорил, с помощью программы piclab. А именно

### ***Модуль цифровых вводов***

Зачем нужен модуль цифровых вводов, я уже говорил. Что он собой представляет в плане постановки задачи? Модуль должен иметь некоторое количество входов, каждый из которых может быть замкнут, или разомкнут, а модуль, в ответ на запрос центрального управляющего устройства, передает состояние своих входов. К входам присоединяются датчики. Кроме уже упомянутых выше датчиков, это могут быть датчики, регистрирующие состояние бытовой аппаратуры. Например, датчик состояния телевизора, регистрирующий, включен он или выключен. Применение подобного датчика особенно актуально, если телевизор, как это чаще всего получается, управляется с помощью ИК команд.

Стандартная ситуация: систему можно построить таким образом, что управление будет происходить по времени суток. В час ночи система, если вы ложитесь спать раньше, может выключить все бытовые приборы, весь свет в доме (или квартире), всю аудио и видео аппаратуру, которые могли остаться включенными. Есть одно «но». ИК-сигнал включения некоторых телевизоров, музыкальных центров, видео магнитофонов полностью совпадает с сигналом выключения. Обычно в программе управления можно устанавливать флаги

## Хобби-электроникс 2. Умный дом.

состояния. Каждое включение телевизора устанавливает флаг «TV\_ON», а выключение сбрасывает его. А если телевизор выключен из сети? Флаг будет успешно устанавливаться и сбрасываться.

Для решения этой проблемы можно применить релейный модуль, подключая телевизор, музыкальный центр и остальное к сети через контакты реле. Но хотелось бы иметь запасной вариант решения. Таким вариантом будет датчик, который фиксирует ток в проводах подключения бытовой техники. Лучше использовать, например, датчик Холла для определения тока в цепи, но можно применить и более простой датчик. Об этом мы поговорим ниже, но подобный датчик к системе будет подключаться с помощью модуля цифровых вводов. Или модуль цифровых вводов станет частью датчика.

Есть очень удобные и недорогие датчики, которые называют «герконовыми». Они удобны для установки на дверь, фиксируя каждое открывание двери. На основе этого датчика можно организовать автоматическое включение и выключение света в ванной или туалете. При первом открывании двери свет включается, при втором выключается. «Герконовый» датчик, конечно, к системе подключается через модуль цифровых вводов. Если установить датчик на входную дверь, а в основной программе поставить счетчик, то можно организовать подпрограмму определения момента, когда все покидают квартиру или дом. Этот момент может быть отправной точкой для принудительного отключения всех электроприборов и перекрытия вводов воды, чтобы избежать протечек. Или включения системы имитации присутствия людей в доме, что используется, как часть подсистемы охраны. Или позже, когда кто-то приходит домой, использовать для сцены, которую я описал в виде программы в рассказе о системах автоматизации быта промышленного производства.

Таким образом, модуль цифровых вводов вполне можно отнести к базовым и обязательным модулям системы «Умный дом».

Я хочу обсудить еще одно использование модуля цифровых вводов - в качестве интерфейса к клавишным устройствам управления. Имея 8 входов, подобный модуль может работать с устройством управления, снабженным 8 клавишами. Этого достаточно для многих задач управления. Если же использовать матрицу 4x4 для построения клавиатуры, то количество подаваемых команд увеличивается до 16, а количество используемых выводов порта остается равным восьми.

Модуль цифровых вводов в плане программирования контроллера легко сделать из программы для релейного модуля. Перенаправим все выходы, предназначенные для подключения реле, на вход. Будем отслеживать состояние всех входов, и записывать их. Остается только передавать их состояние в ответ на запросы главной программы, как это делалось с системными ИК командами.

Присвоим командам модуля цифровых входов префикс D, например, и получим формат запроса:

Dxx\$0S

где xx – два символа номера модуля от 0 до 15, а 0 после \$ - «заставка» для поддержки формата.

Формат ответа модуля тоже, очевидно, будет:

Dххууу - где ууу символьное представление десятичного числа, отображающего состояние входов (как это было сделано для приемника ИК команд).

### ***Работа с программой piclab***

Программ аналогичных MPLAB для Linux я нашел несколько. Это и MPLAB (терминальный вариант), и pikdev, и piclab. Последние две очень похожи. И обе по внешнему виду похожи на MPLAB (Windows), исключая то, что отладчик существует в виде отдельной программы - gpsim. Но о ней несколько позже.

Я использовал установочный пакет из дистрибутива Mandrake – piclab-0.1-1mdk.i586.rpm, который нашел на сайте производителя дистрибутива. Можно попробовать найти его с помощью поисковиков. Для работы пакета, если я все не перезабыл, мне потребовалось установить пакет утилит gprutils-0.10.0-1.i386.rpm. И здесь появились первые «мягкие грабли». Взяв текст на ассемблере релейного модуля, который я транслировал в Windows с помощью MPLAB без особых проблем, я долго не получал положительного эффекта от своих попыток повторить это с помощью программы piclab. Помогло то, что я «снес» (выбросил в корзину) модуль grasm, входящий в состав пакета gprutils, а вместо него добавил grasm более старой версии. А именно, grasm-0.8.16.tar.gz. Этот пакет следовало распаковать в какую-либо папку, а затем установить с помощью команд из терминала: ./configure, make и make install – типовой набор команд, если это не оговорено, для установки пакета из исходных текстов. Можно создать некую (любую) папку в своей домашней директории, распаковать все туда, а затем перенести полученную после распаковки папку с grasm в основную домашнюю папку. Можно и не переносить, но адрес в терминале (аналог командной строки в Windows) по команде перехода в папку, например, cd /home/vladimir/grasm/grasm-0.8.16 удлинится. Если все получится с первого раза, то проблем нет. Если эту строку придется набирать несколько раз, то, чем она короче, тем лучше. На последней стадии установки иногда требуются права администратора компьютера (root). А для этого нужно либо запустить консоль суперпользователя командой kdesu konsole, либо сменить сеанс на сеанс root. Впрочем, и здесь есть варианты, но это другой рассказ. Итак.

Вторые «мягкие грабли» подстерегали меня в тексте releyl.asm, который я, естественно, скопировал из Windows. Несколько меток в тексте сдвинулись с позиции первой колонки на вторую. MPLAB игнорировала этот факт, а piclab, вернее grasm, указала мне на несоответствие.

Следующая неприятность, «грабли железные» - piclab поддерживает работу программаторов. Есть возможность работать с Direct Programmers, но прочитать что-либо из программатора по схеме, собранной для PonyProg, мне не удалось. Это не существенно, поскольку последняя программа есть для Linux, имеет вид тот же, что и Windows, и работает совершенно так же, как в Windows (с теми же особенностями). Но для тех, кто хотел бы поработать с микроконтроллерами в среде Linux, и кто не собрал еще программатор, может быть, есть смысл собрать программатор по схемам, рекомендованным для piclab. Меньше хлопот, если программатор будет работать с piclab. Но для этого шага следует проверить

## Хобби-электроникс 2. Умный дом.

работу самой программы piclab и программы симулятора.

Есть еще одна неприятность, скорее для меня – поддержка языка “С”. Вернее ее отсутствие. Ситуация, впрочем, схожая с MPLAB для Windows. Microchip предлагает компиляторы, поддерживающие серии PIC17 и PIC18 в Windows. Других я не нашел, кроме демо-версии Hi-Tech. Для Linux'a есть компилятор, поддерживающий серию PIC18 - crik. В данном случае есть несколько возможностей – пытаться отыскать нужный компилятор, перейти на микроконтроллер PIC18, попробовать работать с PIC18, внося затем исправления (пока я этот вариант не пробовал) или работать на ассемблере, что, конечно, предпочтительнее, но уж очень лень. Да, я ленив. Увы!

После запуска программы, для которой я создал кнопку запуска, командой piclab, вид программы мало отличается от вида MPLAB, и первые действия – Project->New Project в основном меню, тоже привычны. Создав новый проект, можно добавить файл текста (правой клавишей мышки получаем выпадающее меню, когда маркер в окне навигатора проекта установлен на имени проекта – Add File), написанный на ассемблере для MPLAB, и запустить трансляцию клавишей Assemble File. Затем собрать проект (или сделать все сразу).

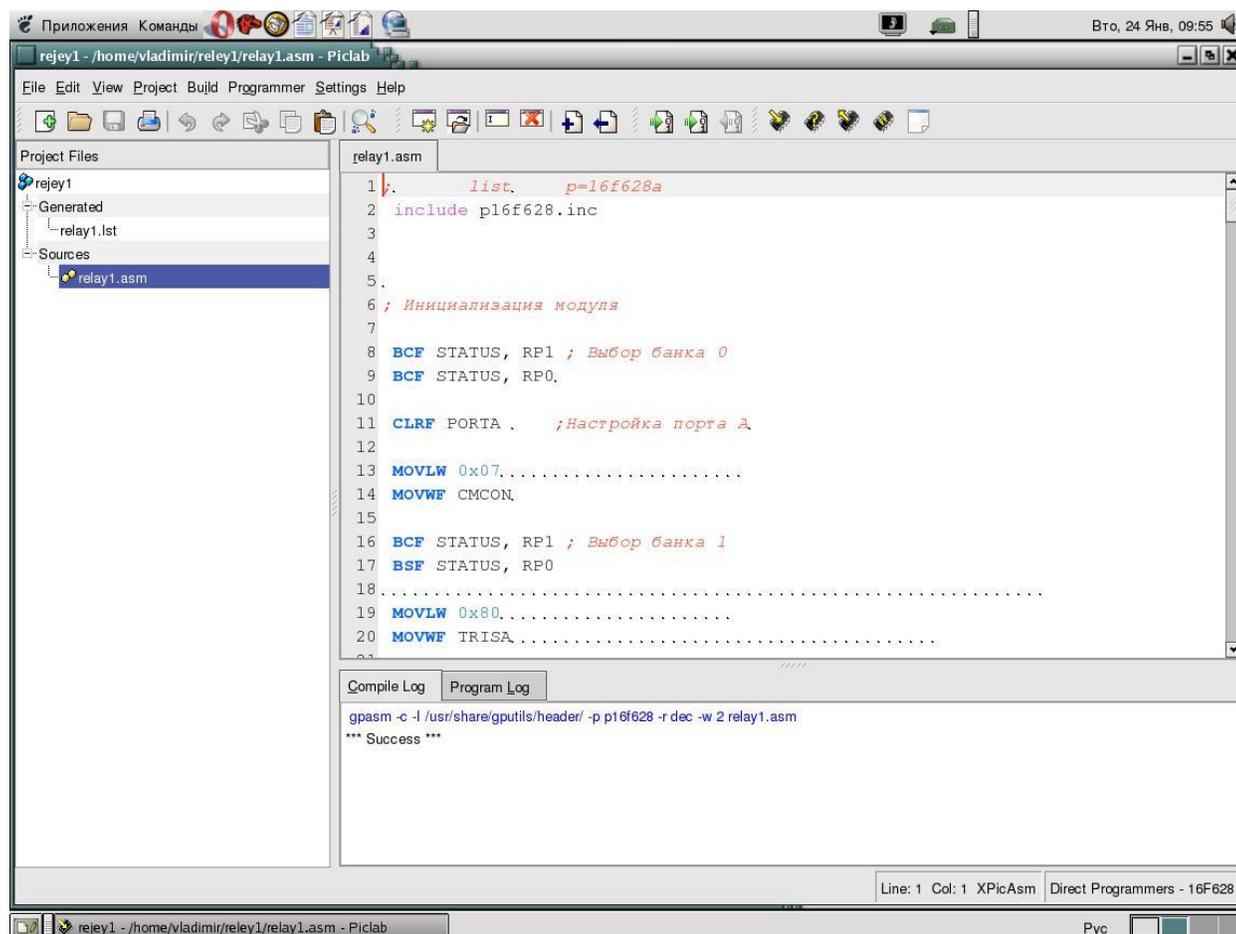


Рис.126

## Хобби-электроникс 2. Умный дом.

Думаю, пришло время определиться с дальнейшей работой – ассемблер или “С”, или как? Попробуем установить компилятор для PIC18, но вначале посмотрим, что собой представляет это микроконтроллер. Я, пожалуй, приостановлю работу, попробую решить, что делать с языком “С”, а те, кто намерен работать на ассемблере, что похвально, могут провести первые эксперименты.

### **Результаты тайм-аута**

Если дорога в ад вымощена благими намерениями, то путь к благим намерениям выстлан «граблями».

Как установить компилятор `spic-0.1` в `piclab` я не нашел. В итоге перебежал на `picdev`, где при создании нового проекта можно выбрать, будет ли проект на ассемблере или языке “С”. Но для начала я решил повторить трансляцию ассемблерного модуля, как это сделал в `piclab`. Работать оно не захотело. Пришлось вернуться к исходному пакету `gprutils`. И получать ассемблерный код из hex-файла. Причины отыскиались, но это не главное. Для работы симулятора `gpsim` нужно получить файлы `.cod` и `.lst`, имея файл `.asm`. И, когда я, наконец, их получил, оказалось, что ассемблерный файл не желает работать с симулятором. Попутно я еще раз попытался “оживить” работу из программы `picdev` с программатором, но безуспешно. Я даже скачал исходные коды программы `picdev`, чтобы попытаться изменить настройки. Так и не ожило. Компилятор “С” работает, но при всей схожести микроконтроллеров, даже простейшая программа на “С” после трансляции затрагивает регистры, которых нет у контроллера PIC16F628. Менять контроллер я не захотел. В итоге - вернулся к программе `piclab`.

К моему удивлению она одумалась и согласилась работать с ассемблером из пакета `gprutils` без особых проблем.

Ассемблерный код имел вид:

```
processor p16f628

org      0
clrf    0x3
movlw   0
movwf   0xa
goto    0x4
clrf    0x3
goto    0x3dd  и так далее...
```

где “`processor p16f628`”, возможно, не обязательно, в отличие от “`org 0`”, без которой ассемблер работать не желает.

Для запуска отладчика `gpsim` пришлось полученные файлы (три выше упомянутых) установить в домашнюю директорию, создать кнопку запуска программы `gpsim`, где установить опцию “Запускать в терминале” (в оболочке Gnome это не составляет проблем, думаю и в KDE тоже) и добавить файл для чтения. Строка запуска получилась такой:

## Хобби-электроникс 2. Умный дом.

```
/usr/bin/gpsim -s file_name.cod
```

Запустить можно из терминала, а затем открыть файл .cod. Но, это кому что больше нравится:

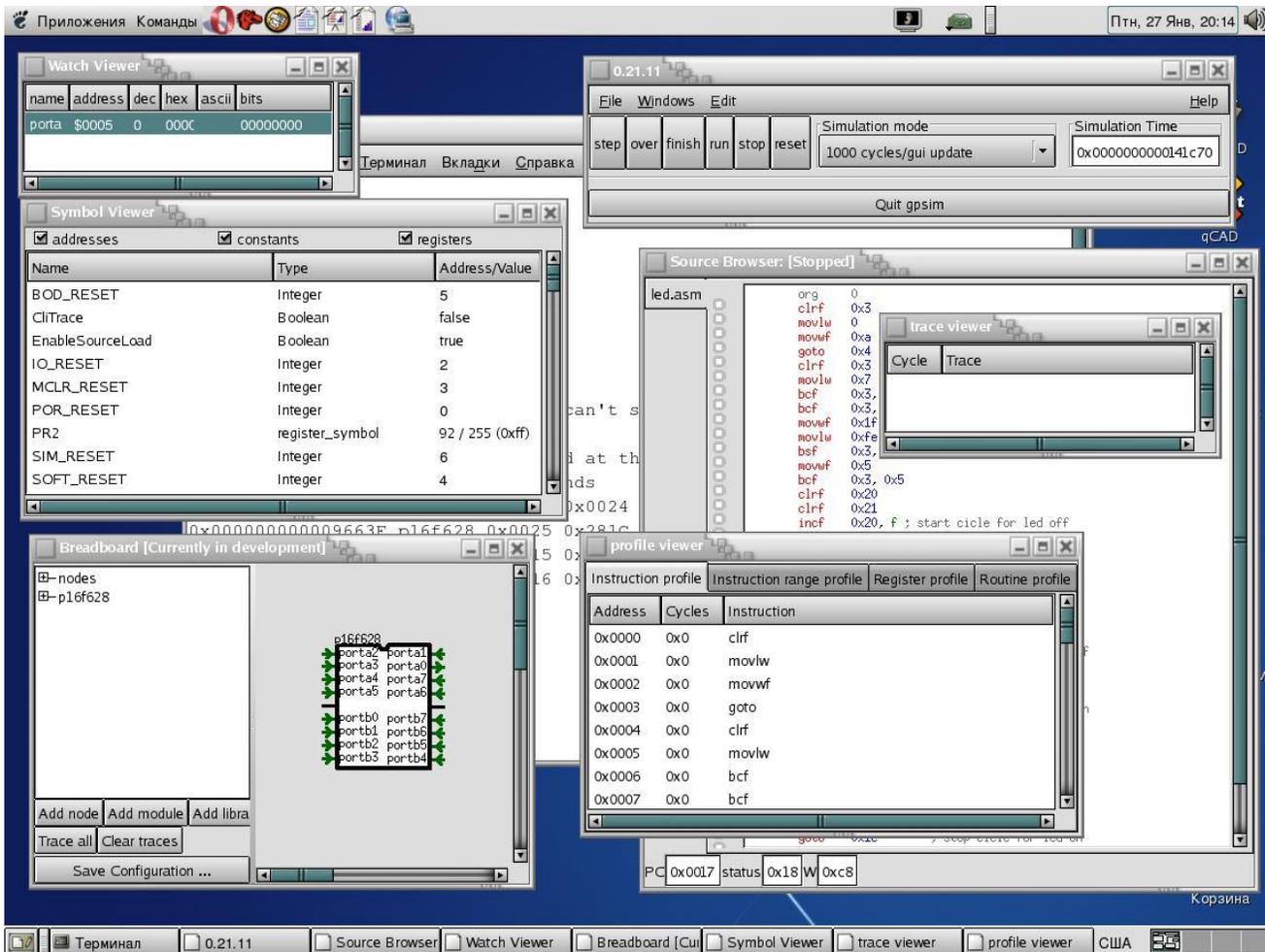


Рис.127

Попытка открыть все окна наблюдения в моем случае приводит к сбою.

## Хобби-электроникс 2. Умный дом.

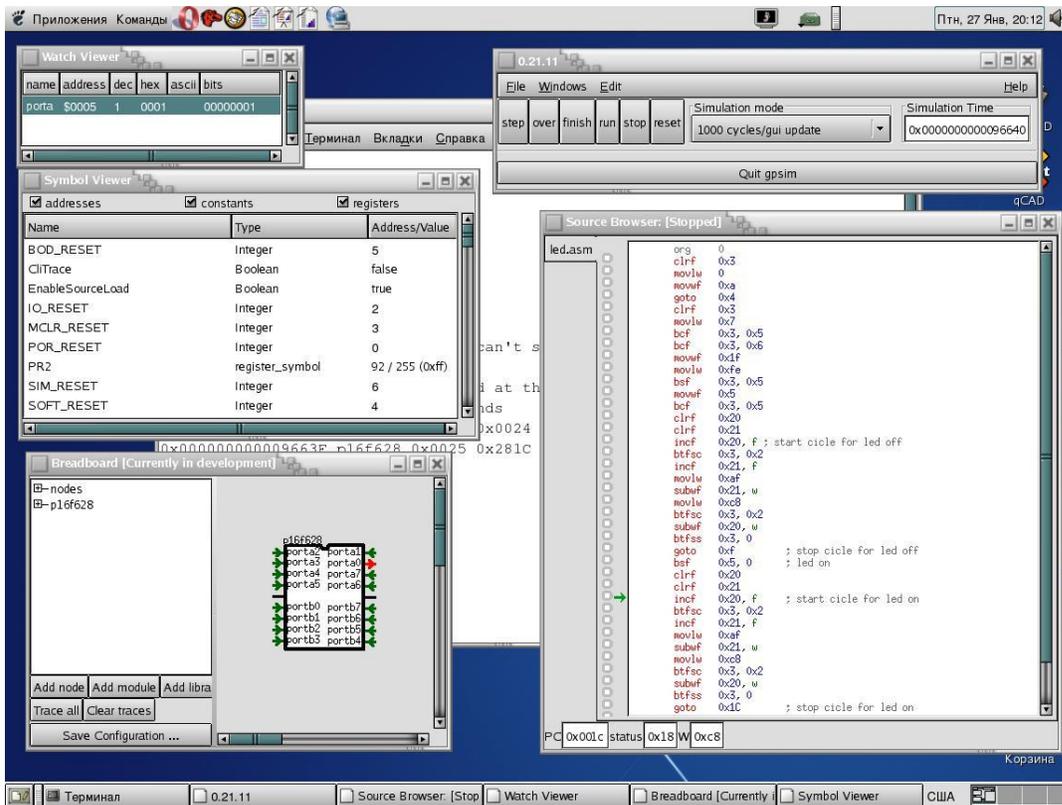


Рис.128

Показан режим отладки простой программы - зажечь индикатор, как на Рис.128, и

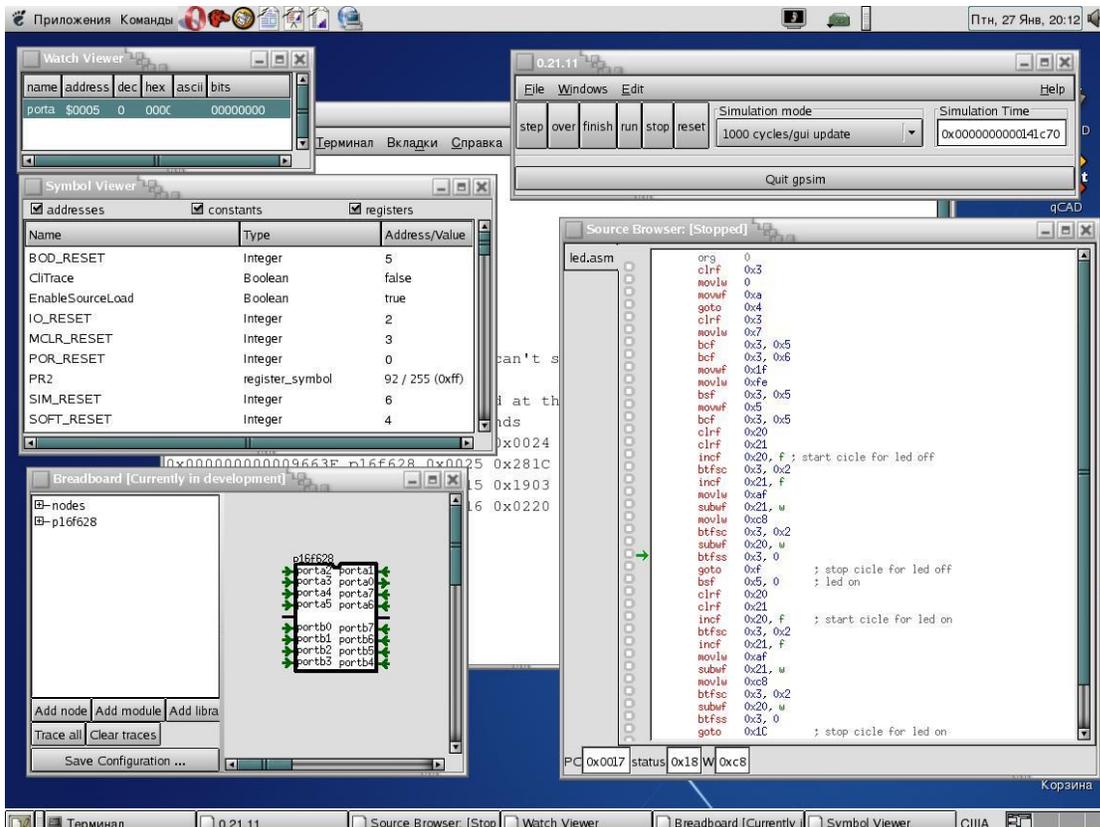


Рис.129

## Хобби-электроникс 2. Умный дом.

погасить индикатор, как на Рис.129 (там, где нарисована сама микросхема). Маркер (в правом окне наблюдения за ассемблерным кодом) показывает работающие операторы.

Для тех, кого заинтересовала работа в среде Linux с микроконтроллерами, в Часть 3 книги я добавлю свой конспект оригинального описания работы с отладчиком gpsim.

Сам же, предпочитая продолжить работу с языком «С», вернусь в Windows к MPLAB, чтобы закончить описание оставшихся модулей.

Итак, переделанная программа для модуля цифровых вводов (входы RA0-RA2 я оставил для выхода, к ним подключены индикаторы):

### Файл digin.h

```
void putch(unsigned char);
unsigned char getch(void);
int init_comms();
int sim_num_adr();
int cmd();
int din_stat();
```

### Файл digin.c

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "digin.h"

unsigned char input; // Для считывания приемного регистра
unsigned char MOD_SIM1; // первый символ адреса модуля
unsigned char MOD_SIM2; // второй символ адреса модуля
unsigned char command_reciev [6]; // Массив для полученной команды
int MOD_ADDR; // Заданный адрес модуля, как число
int sim_end_num = 0; // Полный символьный номер модуля
int MOD_NUM; // Полученный адрес модуля, как число
unsigned char DINSTAT = 0xF8; // Статус входа (позиционно): 1 - вкл, 0 - выкл.
unsigned char DINSIM1; // Символы состояния для передачи
unsigned char DINSIM2;
unsigned char DINSIM3;
int i;

/* получение байта */
unsigned char getch()
{
    while(!RCIF) /* устанавливается, когда регистр не пуст */
```

## Хобби-электроникс 2. Умный дом.

```
        continue;
    return RCREG;
}

    /* вывод одного байта */
void putch(unsigned char byte)
{
    while(!TXIF) /* устанавливается, когда регистр пуст */
        continue;
    TXREG = byte;
}

/* Преобразуем символьный адрес в число*/

int sim_num_adr()
{
    sim_end_num = 0;
    MOD_SIM1 = command_reciev [1];    // первый символ номера
    MOD_SIM2 = command_reciev [2];    // второй символ номера
    MOD_SIM1 = MOD_SIM1 - 0x30;
    MOD_SIM2 = MOD_SIM2 - 0x30;
    sim_end_num = MOD_SIM1*0x0A + MOD_SIM2;
    return sim_end_num;
}

/* Получение и выполнение команды */

int cmd()
{
    switch (command_reciev [5]) {
        case 'S': din_stat();
            break;
    }
}

/* Выполнение команды передачи состояния */

int din_stat()
{
    int din;
    int din1;
    int din2;
    int din3;

    din = DINSTAT; // Преобразуем состояние в символы
    din1 = din/0x64;
```

## Хобби-электроникс 2. Умный дом.

```
DINSIM1 = din1 + 0x30;
din2 = (din - din1*0x64)/0xA;
DINSIM2 = din2 + 0x30;
din3 = (din - din1*0x64 - din2*0xA);
DINSIM3 = din3 + 0x30;
```

```
command_reciev[0] = 'D';
command_reciev[1] = MOD_SIM1+0x30;
command_reciev[2] = MOD_SIM2+0x30;
command_reciev[3] = DINSIM1;
command_reciev[4] = DINSIM2;
command_reciev[5] = DINSIM3;
```

```
CREN =0; // Запрещаем прием
RB0 = 1; // Переключим драйвер RS485 на передачу
TXEN = 1; // разрешаем передачу
for (i=0; i<6; ++i)    putchar(command_reciev[i]);
for (i=0;i<1000;i++); // Задержка для вывода
for (i=0; i<6; ++i) command_reciev [i] = ' ';
RB0 = 0; // Выключаем драйвер RS485 на передачу
TXEN = 0; // запрещаем передачу
CREN =1; // Разрешаем прием
RA1 = 0;
DINSTAT = 0xF8; // Сбросим состояние
```

```
}
```

```
int init_comms() // Инициализация модуля
```

```
{
PORTA = 0xF8; // Настройка портов А и В
CMCON = 0x7;
TRISA = 0xF8;
TRISB = 0xFE;
```

```
RCSTA = 0b10010000; // настройка приемника
TXSTA = 0b00000110; // настройка передатчика
SPBRG = 0x68; // настройка режима приема-передачи
RB0 = 0; // Выключаем драйвер RS485 на передачу
RA0 = 1;
}
```

```
void main(void)
```

```
{
// Инициализация модуля
    init_comms();
    for (i=0; i<6; ++i) command_reciev [i] = ' ';
```

Хобби-электроникс 2. Умный дом.

```
command_reciev [0] = 'D';
// Прочитаем и преобразуем номер модуля
MOD_ADDR = PORTB; // Номер модуля в старших битах
MOD_ADDR = MOD_ADDR>>4; // Сдвинем на четыре бита

// Начинаем работать
start: CREN =1;
      if ((PORTA&0xF8) != 0xF8)
      {
          DINSTAT = DINSTAT&PORTA;
          RA1 = 1;
      }

      if (RCIF) // Если пришел запрос по сети
          input = getch();
          switch (input)
          {
              case 'D': // Если обращение к модулю цифрового ввода
                  for (i=1; i<6; ++i) // Запишем команду в массив
                  {
                      input = getch();
                      command_reciev [i] = input;
                  }

                  MOD_NUM = sim_num_adr(); // чтение из сети

                  if (MOD_NUM != MOD_ADDR) break; // Если не наш адрес
                  else
                      if (command_reciev [3] = '$') cmd(); // Если команда
                      default: goto start;
          }

          goto start;
      }
}
```

***Примечание:***

*Выводы порта А RA0-RA2 я оставил для индикации на время работы с макетом. Индикатор на RA0 включается при включении модуля, индикатор на RA1 должен отображать изменение входов.*

При настройке программы в MPLAB я использую таблицу:

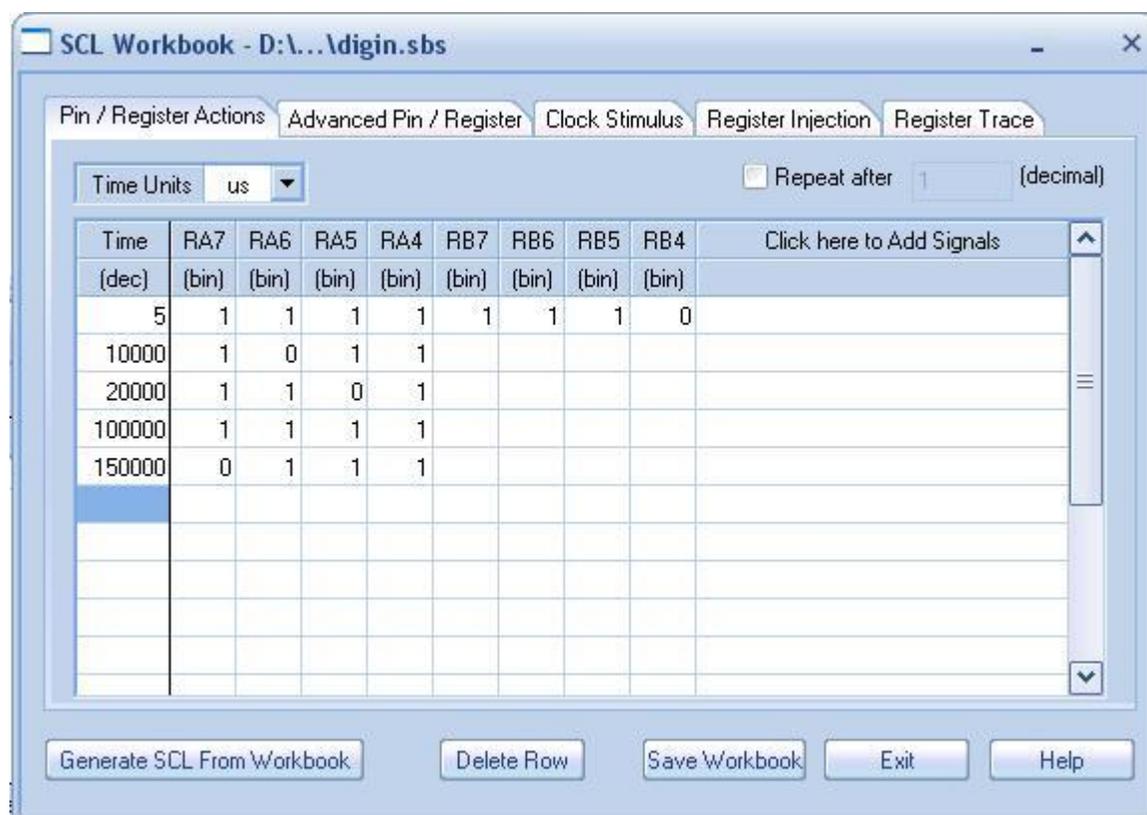


Рис.130

В этом файле я многие переменные и функции оставил в обозначениях, близких к исходным. Я уже говорил, что программу собираюсь переделать из программы релейного модуля. Часть текста, подправив, я взял от модуля приемника системных ИК команд. Правка была минимальной, чтобы было понятнее, откуда я взял текст.

Здесь мне хотелось бы еще раз вернуться к вопросу о написании текста программы. Если посмотреть на вышеприведенный текст, то многие переменные обозначены так, что трудно понять для каких целей они служат. Функции расположены так, что не облегчают понимание их назначения в программе. Вернувшись к тексту после нескольких недель перерыва, начинаешь понимать, что лучше было бы все сделать иначе. Удобно обозначать переменные так, чтобы легко было понять их назначение. Это же касается и функций. Для этого есть много правил, которые лучше найти в руководствах по программированию. Но, есть такой момент в работе над текстом – удобно переменную обозначить, как, например, `int digital_input_close`. Однако, используя эту переменную десятки раз, переделывая программу столько же раз, ты понимаешь, что каждый раз вписывать это многословное название переменной занятие утомительное. Можно, конечно, копировать это название из текста программы. Но для этого приходится возвращаться к началу текста, а затем искать рабочее место в программе. Можно поступить так, и это будет полезно во всех отношениях – использовать файл `todo.txt`. В нем перечислить все переменные и функции, а, попутно, описать их. Переключение между файлами быстрое, а копирование не сложное.

При первой проверке макета, а для его проверки я использую тестовую часть основной программы, исправив обращение на обращение к модулю цифровых входов - D14\$0S, первое, что мне не нравится, но что не является неожиданностью, это поведение входов. Я

## Хобби-электроникс 2. Умный дом.

получаю не то, что ожидаю. Думаю, причина в отсутствии «подтягивающих» резисторов.

Я, как получилось, спаиваю на макет подтягивающие резисторы, и картина действительно меняется. Макет полностью перестает реагировать на тестовые команды. Создается впечатление, что или конвертер RS232-RS485, или приемник макета вышли из строя. В моем распоряжении только мультиметр. Не самый удачный вариант. Первое, что приходит в голову после нескольких неудачных попыток оживить модуль, внести в программу изменение – при инициализации переключить драйвер RS485 на передачу (бит RB0). Бит переключается, что уже хорошо. Затем, манипулируя индикаторами на выводах RA0-RA2, я начинаю проверять, в чем дело. И проблема оказывается в простой халатности – при переделке программы релейного модуля, я, хотя и намеревался, не изменил основную часть программы, которую намеревался взять от программы фотоприемника. После изменения программы модуль цифровых входов стал работать вполне убедительно.

### *Примечание:*

*В тексте выше приведена уже исправленная версия программы.*

Теперь осталось проверить его работу в основной программе в режиме «Работа». В этом режиме программа не реагирует ни на что, кроме сетевых событий. Ранее для выхода из режима «Работа» я использовал ИК команду. Сейчас я хочу использовать изменение входа RA3.

Для этой цели я в основную программу вношу изменения:

Sub work()

Do

```
Form1.KeyPreview = True
```

```
Form1.MSComm1.Output = "D14$0S"
```

```
For i = 0 To 10000000
```

```
Next i
```

```
strAnsw = Form1.MSComm1.Input и т.д. до строки
```

```
Case "D14$0SD14240"
```

```
ext = True
```

```
End Select
```

Таким образом, я поменял обращение (было обращение к модулю фотоприемника) к модулю, и условие выхода.

Программа работает. В итоге HEX-файл для загрузки в программатор модуля цифровых вводов (выводы RA0-RA2 используются в «макетном варианте» для индикации!):

Хобби-электроникс 2. Умный дом.

:10000000830100308A00042820308400403016200C  
:1000100083010330C200FF30C10040308400413012  
:100020001A2083019E2A04068001840A0406031D07  
:1000300013280034F00026208000840A040870068B  
:10004000031900341B2883120313C100C21B31287B  
:10005000421B392842088A004108C10A0319C20A12  
:1000600082008313421883174108C10A84000008E4  
:02007000080086  
:1004860083018C1E432A1A08080083013408533A54  
:10049600031D0800FC2A8301BF00831203130C1EF0  
:1004A600502A3F0899000800F401F5010310F30CE7  
:1004B600F20C031C652A7008F40771080318710A08  
:1004C600F5070310F00DF10D7208730403190034DB  
:1004D600592AF8308301850007309F00F8308316CB  
:1004E6008500FE3086009030831298000630831611  
:1004F60098006830990083120610051408008301DD  
:10050600AD01AE013008A4003108A500D030A40723  
:10051600A5070A30F200F3012408F000F101572282  
:1005260025087407AD0075080318750AAE00F100BA  
:100536002D08F00008006C22AB01AC01AB2A2B0899  
:100546002F3E8400831320308000AB0A0319AC0AC7  
:100556002C08803AF00080307002063003192B0216  
:10056600031CA22A4430AF000608A700A8010430E5  
:10057600F000280DA80CA70CF00BBC2AEA2AAB0148  
:10058600AB0AAC012C08803AF000803070020630CD  
:1005960003192B020318DB2A4322A6002B082F3E41  
:1005A6008400831326088000AB0A0319AC0AC52A07  
:1005B60082227008A9007108AA002806031DE52AF0  
:1005C60027082906031DEA2A2430B20048221816F5  
:1005D6000508F839F83A0319F32A0508C005851401  
:1005E6008C1EF72A4322A6002608443A0319C22A7B  
:1005F600EA2A83014008B900BA016430F200F30127  
:100606003A08F1003908F000B0237408BB007508F9  
:10061600BC003B08303EA0006430F200F3013C0809  
:10062600F1003B08F00057223A08F1003908F000C3  
:100636007408F002031CF1037508F1020A30F20097  
:100646000030F301B0237408B7007508B800370806  
:10065600303EA1000A30F200F3013808F1003708F5  
:10066600F00057227408BD007508BE006430F20021  
:10067600F3013C08F1003B08F00057223A08F1006C  
:100686003908F0007408F002031CF1037508F10242  
:100696003D08F002031CF1033E08F1027008B500A4  
:1006A6007108B6003508303EA2004430AF00240879  
:1006B600303EB0002508303EB1002008B2002108C7  
:1006C600B3002208B4001812061483169816831273  
:1006D600AB01AC01772B2B082F3E84008313000857  
:1006E6004E22AB0A0319AC0A2C08803AF00080307F  
:1006F6007002063003192B02031C6E2BAB01AC01F2

## Хобби-электроникс 2. Умный дом.

:100706002C08803AF00083307002E83003192B027F  
:100716000318912BABA0A0319AC0A832BAB01AC016E  
:100726002C08803AF00080307002063003192B0244  
:100736000318A72B2B082F3E84008313203080003C  
:10074600AB0A0319AC0A932B061083169812831270  
:1007560018168510F830C0000800F601F11FBA2BF4  
:10076600F009F00A0319F103F1097617F617730871  
:100776008039F606F31FC62BF209F20A0319F303B2  
:10078600F309C62BF601F401F50172087304031D83  
:10079600CF2BF001F10100341F30F6040310F60AE6  
:1007A600F20DF30D031CD22BF30CF20C730871023D  
:1007B600031DDF2B72087002031CE72B7208F00280  
:1007C6007308031C730AF102F40DF50DF60BF61A05  
:1007D600D72BF61FF32BF409F40A0319F503F509D1  
:1007E6007408F2007508F300761F0034F009F00A69  
:0A07F6000319F103F1090034F8348F  
:00000001FF

Пришло время поговорить о датчиках, с которыми можно провести эксперименты, используя модуль цифровых вводов.

Я уже говорил, что наиболее интересными, в плане применения, датчиками будут датчики движения, противопожарные датчики, датчики температуры (пороговые измерители температуры), датчики положения (в частности герконовые датчики).

Рассмотрим их по порядку, начиная с простейшего – герконового датчика.

Герконовый датчик (название происходит от сокращения «герметизированные контакты») представляет собой пару: контакты в стеклянной колбе и постоянный магнит. Реальный датчик, конечно, имеет пластмассовые корпуса и для контактов, и для магнита. Когда контакты попадают в поле постоянного магнита, они замыкаются (или переключаются для варианта с контактами на переключение). Естественно, соединив один вывод контактов с выводом модуля цифровых вводов, а другой с общим проводом модуля, мы получаем сигнал о взаимном положении контактов и магнита. Если контакты укрепить на дверной короб, а магнит на дверь, то при закрытой двери контакты замкнуты, а при открытой размыкаются. Этот сигнал можно использовать в основной программе, чтобы инициировать любые события.

Положим, мы установили герконовый датчик на входную дверь. Теперь вполне можно реализовать программу «Я пришел домой», о которой я говорил в разделе, посвященном промышленным разработкам. Напомню:

«Я пришел домой, система включает воду для кофе, а спустя некоторое время включает телевизор, чтобы я мог выпить чашечку кофе и посмотреть новости».

Для реализации подобной программы все, собственно, есть.

Можно установить герконовые датчики на дверь туалета, а основную программу построить так, чтобы при первом открывании двери свет в туалете включался, а при втором

## Хобби-электроникс 2. Умный дом.

открывании двери выключался. Естественно, в систему нужно будет добавить релейный модуль для управления светом. Или, не релейный, а модуль с триаком, о котором речь пойдет ниже.

Герконовые датчики, несмотря на всю свою простоту, могут найти множество применений в экспериментах с системой «Умный дом».

Рассмотрим, как можно сделать датчик температуры. В качестве датчика температуры можно использовать терморезистор. Сделав делитель из терморезистора и обычного резистора, подключив напряжение к делителю, можно снимать сигнал изменяющегося при изменении температуры напряжения на терморезисторе. Теперь осталось подключить компаратор и реле, чтобы при достижении некоторого значения напряжения реле включалось (с модулем цифровых вводов можно использовать вместо реле транзистор с открытым коллектором). Используя в качестве обычного резистора пару из переменного резистора и постоянного, можно изменять заданное напряжение срабатывания. Схем подобных описанной мной множество. Можно построить свою схему, в основе которой может лежать чувствительность к температуре не только терморезистора, но транзистора и т.д.

Как в системе можно использовать подобный датчик? Например, можно настроить датчик на срабатывание при температуре ниже -15 градусов и поместить термочувствительный элемент за окном. В качестве индикатора «похолодания» можно использовать свет в прихожей. Действия программы могут быть следующими:

Температура за окном упала ниже -15 градусов. Когда открывается входная дверь, снабженная герконовым датчиком, свет в прихожей выключается и включается вновь – не забудьте теплое пальто!

Я уже говорил, что, управляя бытовой аппаратурой с помощью ИК кодов, сталкиваешься с проблемой определения текущего состояния аппаратуры – включен телевизор или выключен? Здесь тоже можно использовать датчик температуры для безусловного выключения телевизора в заданное время, поскольку включенный телевизор греется, а выключенный нет.

О противопожарной и охранной системах, в которых применяются противопожарные датчики, я говорил.

Датчики движения. Насколько я понимаю, они появились, как датчики охранных систем. Есть несколько разновидностей подобных датчиков. С моей точки зрения наиболее интересны пирометрические датчики движения. Но они достаточно дороги. Более дешевыми могут оказаться микроволновые датчики. Схемы подобных датчиков, которые я нашел в Интернете, я приведу в Части 3.

Есть еще один вид датчиков, которые могут найти практическое применение даже при самостоятельном изготовлении. Это датчики протечек. Датчики, реагирующие на появление воды там, где ее не должно быть. Установив подобные датчики возле батарей отопления, в местах ввода горячей и холодной воды, возле стоков раковин, можно избежать серьезных неприятностей при появлении протечек воды. Сигнализируя об аварии, система выручит вас

## Хобби-электроникс 2. Умный дом.

на этапе, когда еще не поздно перекрыть вентили или перестать пользоваться раковиной и вызвать специалиста для устранения неисправности. Датчики протечек, я думаю, лучше использовать готовые.

Есть еще одно применение модуля цифровых вводов, о котором я упоминал раньше – системное устройство управления на базе модуля цифровых вводов.

Само системное устройство управления – это клавишный пульт. Контакты при нажатии клавиши замыкают один из входов модуля на общий провод. Восемь входов модуля позволяют сделать простой пульт с восьмью клавишами, что позволяет подать восемь команд. Достаточно ли этого? Достаточно для управления всеми светильниками в комнате. Или всеми основными светильниками в квартире. Этого хватит для управления одним аудио- или видео устройством, например, видео магнитофоном. Но управлять телевизором будет не удобно, поскольку трудно будет ввести номер канала. Для ввода номера канала обычные пульты имеют цифровую клавиатуру с цифрами от 0 до 9. Чтобы увеличить количество клавиш управления достаточно добавить кнопки не с одним контактом на замыкание, а с двумя. Если первые восемь кнопок будут менять состояние одного бита, то остальные могут менять состояние двух бит одновременно. Для этого случая в программу контроллера следует добавить задержку на 10-50 мс перед определением состояния входов:

```
if ((PORTA&0xFF) != 0xFF)
{
    for (i=0; i<1000; i++);
    DINSTAT = DINSTAT&PORTA;
}
```

Что-нибудь в этом роде. Необходимость в задержке связана с одновременным замыканием контактов у двухконтактных кнопок.

Применение кнопок с двумя замыкающими контактами, т.е. замыкающими первый вход и второй, первый и третий, и т.д., одновременно позволит добавить на пульт еще семь клавиш управления. Одновременное замыкание второй и третий, второй и четвертый, и т.д. – добавит еще шесть клавиш. Даже 21 клавиши управления для пульта, мне кажется, более, чем достаточно для любых применений этого решения к построению клавишного пульта управления.

Позже мы обсудим вопрос о том, что можно сделать в экспериментах, если в квартире не заложено достаточно большого количества проводов, а менять проводку нет возможности. Я имею в виду использование радиоканала. Это же решение можно применить и к пульту управления, построенному на основе модуля цифровых вводов, для превращения его в переносной пульт управления.

### *Резюме:*

Мы расширили возможности системы за счет нового устройства – модуля цифровых вводов. Рассмотрели и несколько применений этого модуля. И, как мне кажется, с появлением нового модуля система приобрела новое качество.

**Модуль с триаком**

Релейный модуль, описанный в Части 1 «Кукольный умный домик», универсален во многих отношениях. Он позволяет, выбрав соответствующее реле, коммутировать и настольные лампы, и электрический чайник; и переключать громкоговорители, и коммутировать входы усилителя (возможно, с не очень высоким качеством).

На базе релейного модуля можно изготовить выключатель света, который вполне можно разместить на месте обычного. Конечно, в этом случае потребуется подвести провода системной сети. Но, если говорить о замене обычного выключателя света на системный, то лучше изменить способ управления светильником. Конечно, если это не лампа дневного света, которую лучше включать с помощью реле.

Есть хороший коммутатор ламп накаливания – триак. Триак, или семистор – управляемый полупроводниковый прибор, который в отличие от тиристора может использоваться в цепях переменного тока. При мощности нагрузки до 500-600 Вт, ему достаточно небольшого радиатора в виде металлической пластины по размерам выключателя света. В первой книге «Хобби-электроникс» я писал о том, что задача управления этим коммутатором тока, при всей ее видимой простоте, может оказаться весьма интересной. Очень хорошее решение эта задача получила с появлением оптопары – светодиод и фототриак. Подобные микросхемы, насколько я знаю, выпускает фирма Toshiba в серии TPL (TPL3061, TPL3041) и несколько производителей – серия МОС:

| ТРИСТОРНЫЕ ОПТОПАРЫ |         |                 |                 |             |          |             |
|---------------------|---------|-----------------|-----------------|-------------|----------|-------------|
| Наименование        | СХЕМА № | U комм. пик., В | I сраб. вх., мА | Zero-Cross* | U из. кВ | Тип корпуса |
| МОС3020             | 10      |                 |                 |             |          | PDIP 6      |
| МОС3021             | 10      | 400             | 15              |             | 7.5      | PDIP 6      |
| МОС3023             | 10      | 400             | 5               |             | 7.5      | PDIP 6      |
| МОС3041             | 11      | 400             | 15              | +           | 7.5      | PDIP 6      |
| МОС3042             | 11      | 400             | 10              | +           | 7.5      | PDIP 6      |
| МОС3061             | 11      | 600             | 15              | +           | 7.5      | PDIP 6      |
| МОС3062             | 11      | 600             | 10              | +           | 7.5      | PDIP 6      |
| МОС3063             | 11      | 600             | 5               | +           | 7.5      | PDIP 6      |

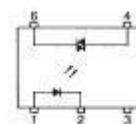


Схема 10

Рис.131

Светодиод оптрона включается в модуль управления светом, как включаются индикаторы, при этом полностью изолируя модуль от силовой сети. Триак оптрона подключается к управляющему электроду мощного триака. Стоимость же микросхемы и триака может оказаться меньше, чем стоимость реле.

Схема модуля с триаком может выглядеть следующим образом:

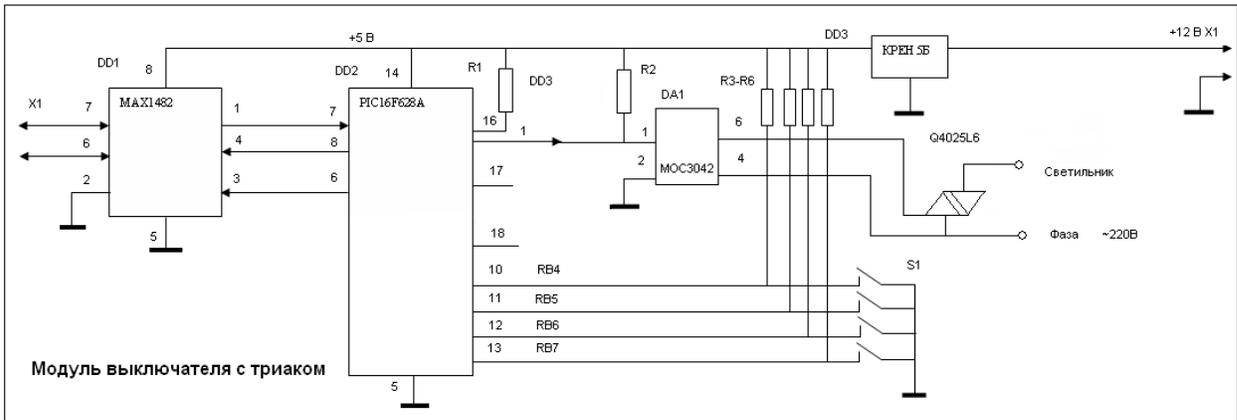


Рис..132

С моей точки зрения подобного выключателя достаточно для системы. Но применение триака в качестве коммутатора для управления светом открывает возможность плавной регулировки яркости света. Для тех, кого заинтересует подобная возможность, можно предложить следующее решение.

### ***Модуль с триаком и плавной регулировкой яркости***

Плавная регулировка яркости света при использовании триака связана с тем, что яркость ламп накаливания зависит от эффективного напряжения. Чем оно ниже, тем меньше яркость. Устройства подобного типа имеют название «диммер», возможно, от глагола dim – терять яркость.

Тиристор или триак, открывается сигналом, подаваемым на управляющий электрод. Осциллограмма напряжения на нагрузке, лампе накаливания, выглядит следующим образом:

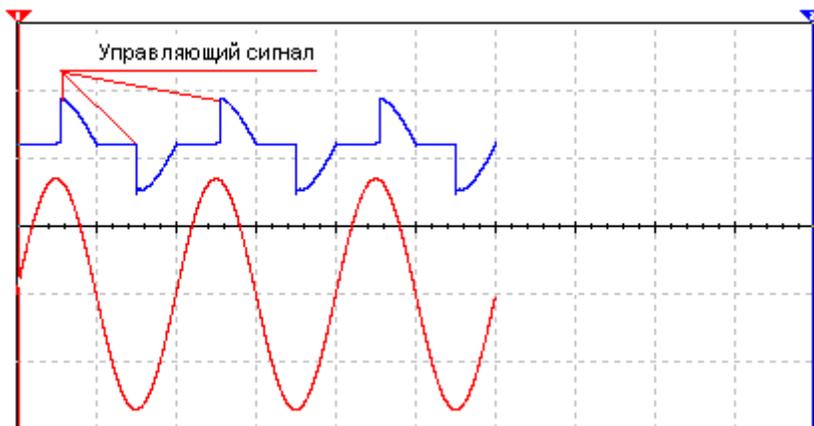


Рис.133

Синим цветом здесь и далее отображено напряжение, интересующее нас, а красным переменное силовое напряжение 220В.

При подаче управляющего сигнала триак открывается, и напряжение полностью

## Хобби-электроникс 2. Умный дом.

падает на нагрузку. При переходе напряжения через ноль триак закрывается. Напряжение на лампе накаливания появится только после прихода следующего управляющего сигнала. В данном случае частота управляющих сигналов 100 Гц. Усеченное синусоидальное напряжение на лампе накаливания в данном случае равнозначно подключению лампы к напряжению  $\sim 110$  В. Яркость лампы уменьшается.

Сдвигая момент подачи управляющего сигнала ближе к моменту перехода напряжения через ноль:

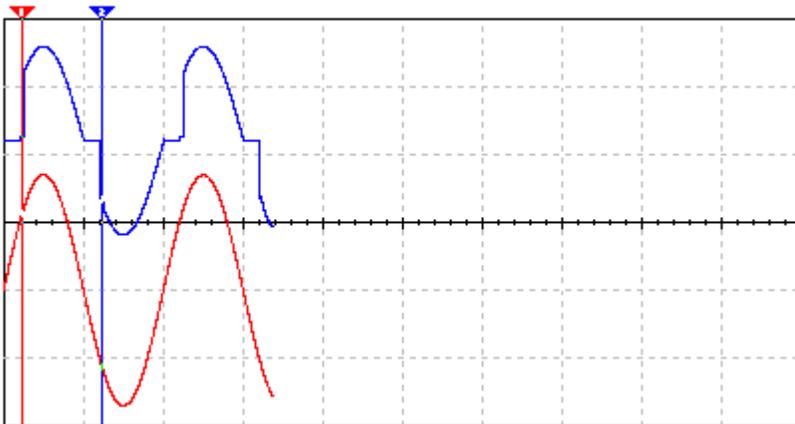


Рис.134

(маркерами помечены моменты подачи управляющего сигнала), мы получим большую яркость. Удлиняя время подачи управляющего сигнала после перехода напряжения через ноль:

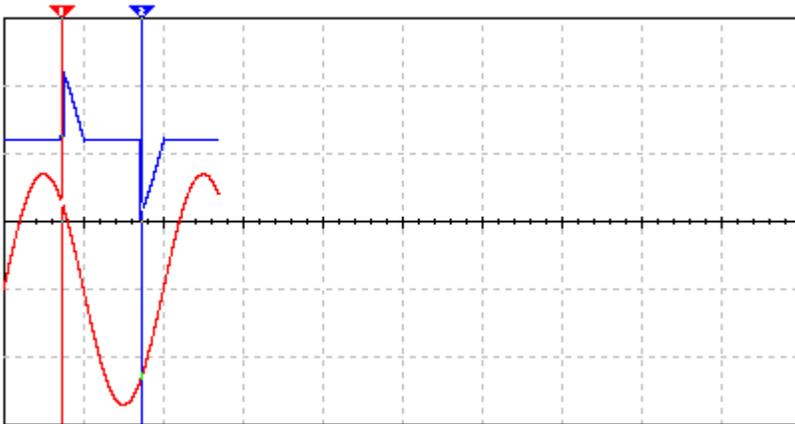


Рис.135

мы получим меньшую яркость свечения лампы.

При построении модуля следует включать светодиод оптрона в нужные моменты времени, изменяя яркость свечения настольной лампы или торшера. И настольная лампа и торшер включаются в розетку, имеющую фазовый и нулевой провод.

А, вот, при использовании модуля вместо обычного выключателя, мы столкнемся с

## Хобби-электроникс 2. Умный дом.

трудностью – отсутствие нулевого провода в зоне выключателя. Без него мы не можем синхронизировать подачу управляющих сигналов на триак с моментами перехода напряжения через ноль. Если бы нулевой провод был в зоне выключателя, то для получения синхронизирующих импульсов можно было бы использовать другой тип оптрона – светодиод-фототранзистор. Светодиод через конденсатор (чтобы не устанавливать резистор большой мощности) и резистор (ограничивающий ток через светодиод) мы подключили бы между фазой и нулем силового напряжения. При обычном светодиоде (не двух полярном) мы получили бы возможность отмечать один из двух переходов через ноль за период силового напряжения. Но как быть, если нулевой провод отсутствует? Можно поступить двояко – в том месте, где будет установлен системный блок питания 12В, соединить общий схемный провод С ЗАЩИТНЫМ ЗАЗЕМЛЕНИЕМ. При этом в месте установки модуля на место выключателя мы будем иметь возможность сделать схему синхронизации. Второй вариант – подключение оптрона для синхронизации управления с силовым напряжением к триаку. Напряжение на нем выглядит так:

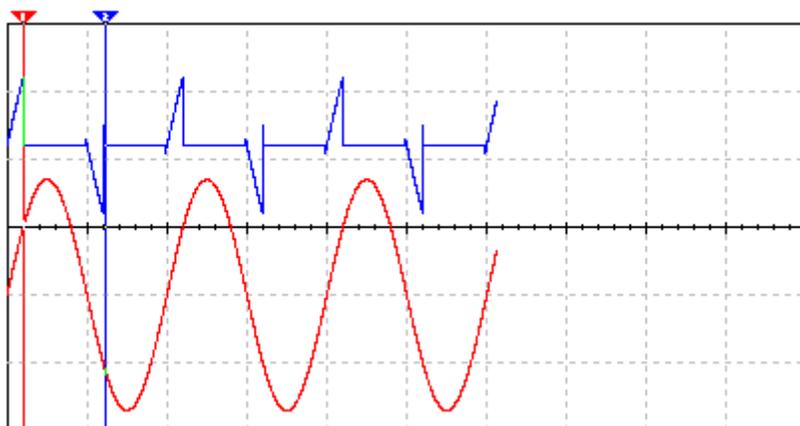


Рис.136

Этот рисунок соответствует Рис.134 выше в тексте. Или так, соответственно Рис.135:

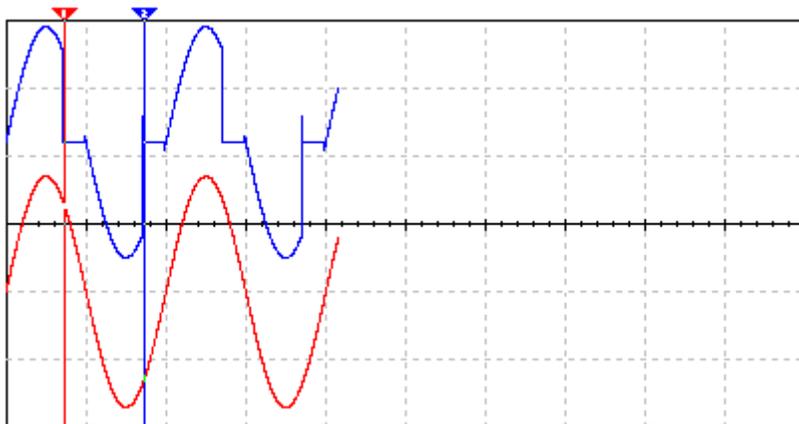


Рис.137

Как видно из рисунков, мы можем осуществить синхронизацию.

Итак. Определившись со схемой регулировки, постараемся решить, как мы будем

## Хобби-электроникс 2. Умный дом.

регулировать яркость – плавно или ступенями. В любом случае мы будем регулировать яркость ступенями, но их можно сделать маленькими или большими. На мой взгляд, градация переходов с тремя-четырьмя уровнями яркости в полной мере устроит любого. А большинству пользователей достаточно будет двух ступеней – полная яркость и треть яркости. Конечно, я имею в виду систему «Детский умный домик». Вопрос этот не принципиальный, и каждый решит его сам.

Второй аспект этой проблемы – использовать ли временную задержку подачи управляющего импульса, используя встроенный таймер или пустой цикл в программе? Или реализовать эту процедуру с использованием встроенного в микроконтроллер PIC16F628A блока ШИМ. ШИМ – аббревиатура названия способа Широтно-Импульсной Модуляции. Я остановлюсь на максимально наглядном варианте, используя цикл задержки.

Наконец, последнее, что меня в настоящий момент может беспокоить – не будет ли свет «мигать»? Причина этого беспокойства в том, что контроллер будет осуществлять несколько процессов:

- Сканировать напряжение сети ~220 В для определения моментов перехода напряжения через ноль. Это нужно для синхронизации.
- Запускать управляющий импульс, который включит триак, по истечении времени задержки после обнаружения перехода через ноль напряжения сети.
- Прослушивать порт USART с целью выявления команд, адресованных модулю.

Системные команды следуют одна за другой. Модуль будет реагировать на все команды, даже тогда, когда они адресованы не ему. Пока модуль разбирается с системными командами, он может пропускать управление триаком, а светильник будет выключен.

Попробуем «прикинуть», как это будет выглядеть. Длительность полуволны частотой 50 Гц составляет 10 мс. Возьмем средний интервал «свободного» от управления времени 5мс. С другой стороны при скорости в сети 2400 бит/с прием одного байта займет около 4мс. Остается надеяться, что я неправильно понимаю работу USART. Проверим. Если не получится, я готов отказаться от регулировки яркости. Но тем, кому очень хочется иметь модуль выключателя света с регулируемой яркостью, можно предложить:

- Поэкспериментировать с изменением тактовой частоты контроллера до 20 МГц и скоростью сетевой работы 115 200 бит/с.
- Поэкспериментировать с работой по прерыванию.
- В основной программе все команды и запросы перемежать паузами. Это замедлит сетевую работу системы, но может оказаться не столь ощутимо, в конечном счете.
- Если совсем ничего не получится, то использовать внешнюю микросхему обслуживания управления включения света. Например, использовать контроллер для связи с системой, а полученное время задержки переписывать во внешний счетчик, который будет синхронизирован с частотой сети 50 Гц, и подавать управляющие импульсы на оптрон.
- Поискать, нет ли еще каких-либо интересных решений. Это-то, по моему мнению, и есть суть и смысл всей книги.

## Хобби-электроникс 2. Умный дом.

Тем не менее, попробуем реализовать программу и проверить ее работу в MPLAB.

### *Требования к модулю диммера:*

1. Регулирование яркости лампы накаливания на 3 уровнях: полная яркость, половинная яркость, минимальная яркость.
2. Получение команд от центрального управляющего устройства в формате L14\$1N, где 1 – практически, полная яркость. Уровень – 5, практически, выключение.
3. Статус модуль не передает.

Для тех, кто хотел бы запрашивать состояние диммера, я думаю, не составит труда взять эту часть программы из предыдущих решений. Аналогично обстоит и с уровнями яркости, командой выключения и полного включения.

### **Реализация модуля**

Вывод RA3 подключим к сканирующему оптрону. Переход через ноль будет отображаться состоянием «1». Для основного программного модуля два события – появление системной команды и импульс синхронизации с силовой сетью – будут служить рабочей частью.

Переделав текст программы, немного «повозившись» с устранением остаточных ошибок после переделки, я вначале сталкиваюсь с проблемой (в отладчике MPLAB) плохой обработки сканирования. Справившись с этим, я никак не могу получить изменения яркости (предполагаемой) из-за того, что сетевые команды «пропадают». В конечном счете программа приобретает следующий вид:

### **Файл dimmer.h**

```
void putch(unsigned char);
unsigned char getch(void);
int init_comms();
int sim_num_adr();
int cmd();
```

### **Файл dimmer.c**

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "dimmer.h"

unsigned char input;           // Для считывания приемного регистра
unsigned char MOD_SIM1;       // первый символ адреса модуля
unsigned char MOD_SIM2;       // второй символ адреса модуля
int LEVEL = 100;              // Уровень яркости
unsigned char command_reciev [6]; // Массив для полученной команды
unsigned char COMMAND;        // Флаг прихода сетевой команды
int MOD_ADDR;                 // Заданный адрес модуля, как число
```

Хобби-электроникс 2. Умный дом.

```
int sim_end_num = 0;           // Полный символьный номер модуля
int MOD_NUM;                 // Полученный адрес модуля, как число
int i;
int k;

    /* получение байта */
unsigned char getch()
{
    while(!RCIF) /* устанавливается, когда регистр не пуст */
        continue;
    return RCREG;
}

    /* вывод одного байта */
void putch(unsigned char byte)
{
    while(!TXIF) /* устанавливается, когда регистр пуст */
        continue;
    TXREG = byte;
}

/* Преобразуем символьный адрес в число*/

int sim_num_adr()
{
    sim_end_num = 0;
    MOD_SIM1 = command_reciev [1];    // первый символ номера
    MOD_SIM2 = command_reciev [2];    // второй символ номера
    MOD_SIM1 = MOD_SIM1 - 0x30;
    MOD_SIM2 = MOD_SIM2 - 0x30;
    sim_end_num = MOD_SIM1*0x0A + MOD_SIM2;
    return sim_end_num;
}

/* Выполнение команды */

int cmd()
{
    command_reciev [0] = 'L';
    for (i=1; i<3; ++i)           // Запишем номер модуля
    {
        input = getch();
        command_reciev [i] = input;
    }

    MOD_NUM = sim_num_adr();    // Полученные номер модуля, как число
}
```

## Хобби-электроникс 2. Умный дом.

```
if (MOD_NUM == MOD_ADDR)          // Если номер модуля совпадает
{
    MOD_NUM = 0;
    for (i=3; i<6; ++i)            // Запишем команду в массив
    {
        input = getch();
        command_reciev [i] = input;
    }
    command_reciev [4] = '3';      // Для получения иллюстрации
    LEVEL = (command_reciev [4] - 0x30) * 0x64; // Уровень яркости
    for (i=0; i<6; ++i) command_reciev [i] = ' ';
    COMMAND = 0;                   // Сбросим флаг команды
}
else
{
    for (i=0; i<6; ++i) command_reciev [i] = ' '; // Очистим массив
    COMMAND = 0;                   // Сбросим флаг
}
}
```

```
int init_comms() // Инициализация модуля
{
    PORTA = 0;          // Настройка портов А и В
    CMCON = 0x7;
    TRISA = 0b00001000;
    TRISB = 0xFE;

    RCSTA = 0b10010000; // настройка приемника
    TXSTA = 0b00000110; // настройка передатчика
    SPBRG = 0x68;       // настройка режима приема-передачи
    RB0 = 0;           // Выключаем драйвер RS485 на передачу
    CREN = 1;
    RA0 = 1;
}
```

```
void main(void)
{
    // Инициализация модуля
    init_comms();
    for (i=0; i<6; ++i) command_reciev [i] = ' ';
    COMMAND = 0;
```

```
// Прочитаем и преобразуем номер модуля
```

Хобби-электроникс 2. Умный дом.

```
MOD_ADDR = PORTB;           // Номер модуля в старших битах
MOD_ADDR = MOD_ADDR>>4;     // Сдвинем на четыре бита
```

```
// Начинаем работать
```

```
start:
```

```
    if ((RA3 == 1)&(COMMAND == 0)) // Сканирование ~220В, когда нет
    {                               // сетевой команды
        for (k=0; k<LEVEL; k++) RA1 = 0;
        RA1 = 1;
        if (RCIF) // Если пришла команда по сети
        {
            COMMAND = 1; // Устанавливаем флаг команды
            input = getch();
            if (input == 'L') cmd(); // Если наш модуль, обработаем
команду
            else COMMAND = 0; // Иначе сбросим флаг команды
        }
    }
    goto start;
}
```

***Примечание:***

*Отмеченная красным цветом строка добавлена только для получения иллюстрации, как это описано ниже.*

Вначале приведу таблицу настроек стимулов:

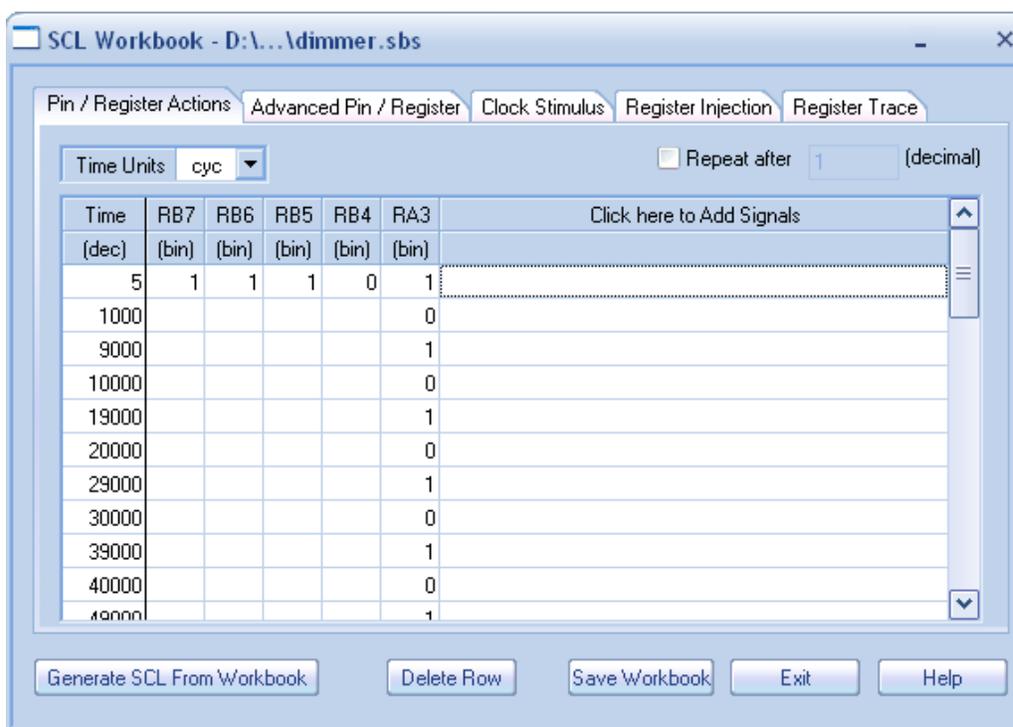


Рис.138

Эти настройки пришлось продолжить до времени 140000. При наладке я выбрал единицы измерения – циклы.

Этим не ограничилось – пришлось изменить частоту с 4 МГц на 20 МГц. Изменить основной текст в функции приема команды, закомментировав часть текста:

```
int cmd()
{
/*      command_reciev [0] = 'L';
for (i=1; i<3; ++i) // Запишем номер модуля
{
    input = getch();
    command_reciev [i] = input;
}

MOD_NUM = sim_num_adr(); // чтение из сети

if (MOD_NUM == MOD_ADDR)
{
    MOD_NUM = 0;
for (i=3; i<6; ++i) // Запишем команду в массив
{
    input = getch();
    command_reciev [i] = input;
}
*/
command_reciev [4] = '3';
LEVEL = (command_reciev [4] - 0x30) * 0x64;
```

## Хобби-электроникс 2. Умный дом.

```
        for (i=0; i<6; ++i) command_reciev [i] = ' ';  
        COMMAND = 0;  
/*      }  
else  
{  
        for (i=0; i<6; ++i) command_reciev [i] = ' ';  
        COMMAND = 0;  
      }  
*/  
  
}
```

Только после всех этих обманных маневров, мне удалось получить иллюстрацию к тому, что я хотел сказать. После получения команды, интервал от момента перехода напряжения сети ~220В через ноль, до момента включения триака, отмеченный на рисунке 32 маркером, меняется на новый интервал, отмеченный на рисунке 33:



Рис.139



Рис.140

В конечном счете, что можно сказать. Во-первых, отработка сканирования и включения выглядит нормально (Рис.141 – яркость минимальна, Рис.142 - максимальна):



Рис.141

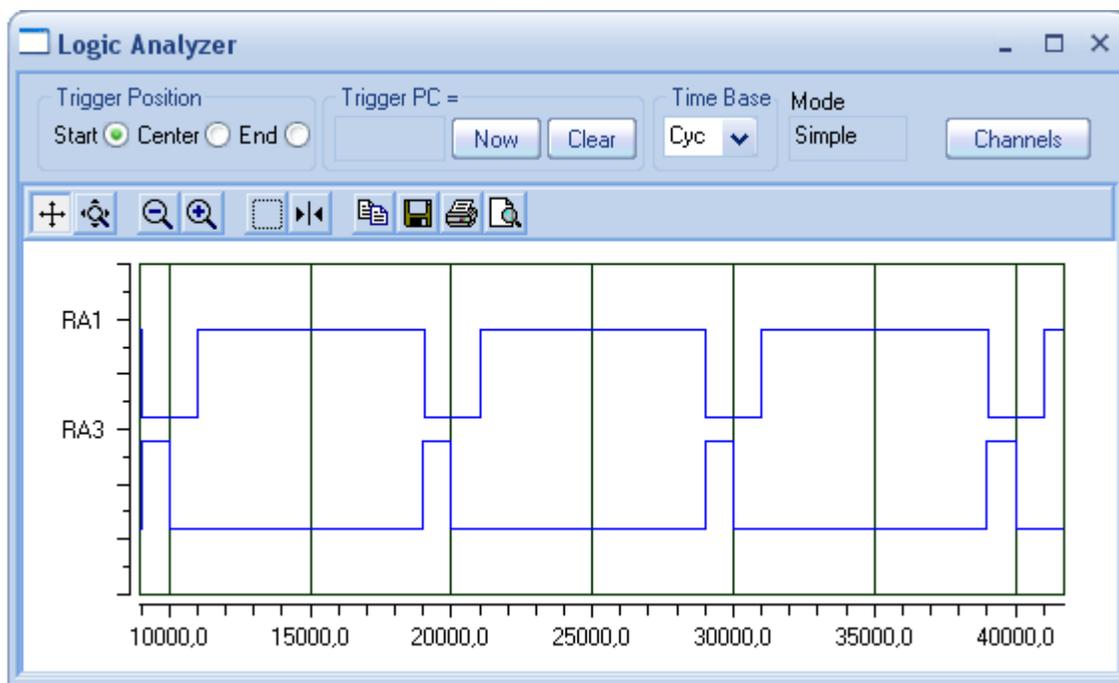


Рис.142

Уровень яркости определяется переменной LEVEL, которая изменяется командой от компьютера. На этих рисунках видно, что импульсы управления идут с частотой 100 Гц (10000 мкс). Выше приведенные картинки, полученные «жульническим путем», дают надежду на то, что команду от компьютера все-таки удастся прочесть. Мои опасения по поводу «мигания» света, скорее всего, беспочвенны. Их лучше отнести к опасениям по поводу отработки сетевых команд.

Для очистки совести мне следовало проверить работу модуля на макете. Но осуществлять реальное сканирование сети ~220В мне совсем не хочется. Можно бы подключить генератор с частотой 100 Гц к выводу RA3 через транзистор с открытым коллектором, но «смотреть» все нужно осциллографом, который у меня пока «гуляет на стороне».

И, наконец, самое главное – я вовсе не хочу давать готовые решения с описанием деталей работы схемы. Я хочу, и это то, ради чего вся эта книга, я хочу показать, насколько увлекательное занятие - придумывать и реализовывать свои электронные проекты!

Завершая разговор о диммере, приведу HEX-файл:

```
:10000000830100308A000428203084003530162017
:1000100083010330B800FE30B7003530840037303C
:100020001A2083012E2B04068001840A0406031D76
:1000300013280034F00026208000840A040870068B
:10004000031900341B2883120313B700B81B31288F
:10005000381B392838088A003708B70A0319B80A44
:1000600082008313381883173708B70A8400000802
:02007000080086
:1005B20083018C1ED92A1A0808008301B400831211
```

## Хобби-электроникс 2. Умный дом.

:1005C20003130C1EE02A340899000800F401F50117  
:1005D2000310F30CF20C031CF52A7008F4077108DF  
:1005E2000318710AF5070310F00DF10D7208730478  
:1005F20003190034E92A8301850107309F0008307E  
:1006020083168500FE3086009030831298000630F3  
:100612008316980068309900831206101816051484  
:1006220008008301AC01AD012F08A1003008A2002F  
:10063200D030A107A2070A30F200F3012108F0002E  
:10064200F101E72222087407AC0075080318750A45  
:10065200AD00F1002C08F0000800FC22A801A9015D  
:100662003B2B28082E3E8400831320308000A80AEA  
:100672000319A90A2908803AF00080307002063076  
:1006820003192802031C322BA0010608A400A501AD  
:100692000430F000250DA50CA40CF00B4B2B83129B  
:1006A2000313851D502BA008031D522BAA01AB0179  
:1006B2002B08803AF0003608803A7002031D632B43  
:1006C20035082A028312031303186C2B8510AA0A19  
:1006D2000319AB0A592B85148C1E502BA001A00ABA  
:1006E200D922A3004C3A031D782B7A23502BA00168  
:1006F200502B4C308301AE00A801A80AA9018C2B13  
:10070200D922A30028082E3E8400831323088000E8  
:10071200A80A0319A90A2908803AF0008030700259  
:10072200033003192802031C812B12237008A60030  
:100732007108A7002506031DA02B24082606031D09  
:10074200E62BA601A7010330A800A9012908803AD7  
:10075200F000803070020630031928020318BD2B06  
:10076200D922A30028082E3E840083132308800088  
:10077200A80A0319A90AA72B6430F200F301320870  
:10078200F000F101D030F0070318F10AFF30F10751  
:10079200E7227408B5007508B600A801A901290866  
:1007A200803AF000803070020630031928020318E4  
:1007B200E42B28082E3E8400831320308000A80AF0  
:1007C2000319A90AD02BA0010800A801A901290830  
:1007D200803AF000803070020630031928020318B4  
:1007E200FC2B28082E3E8400831320308000A80AA8  
:0E07F2000319A90AE82BA001080064340034A2  
:00000001FF

Файл предназначен для экспериментальных целей. Добавлю еще несколько слов.

- Если эксперименты с диммером не приведут к успеху, то можно рассмотреть вариант применения более мощного микроконтроллера.
- Или использовать те, решения, о которых я говорил выше.
- Или применить, что проще, два контроллера, один из которых будет общаться с системной сетью, а второй управлять триаком.
- И не забудьте. Если вы решите сделать выключатель с регулируемой яркостью, то (что никак не отражено в реализации) следует предусмотреть одну или две кнопки для ручного управления!

## Хобби-электроникс 2. Умный дом.

- В режиме ручного управления можно сделать следующее – одна из кнопок при кратковременном нажатии полностью включает свет, при нажатии и удержании медленно увеличивает яркость. Вторая кнопка проделывает то же самое, но с выключением. Думаю, вам не составит труда изменить программу для поддержки этих добавлений.

А я продолжу рассказ о модулях, которые могут войти в систему.

### ***Модуль последовательного интерфейса***

Некоторые устройства позволяют управлять ими через интерфейс RS232, что предпочтительней ИК управления, если известны команды управления устройством. Это может быть проигрыватель CD-дисков, или проектор. Модуль позволяет пересылать команды по системной сети к устройству, с преобразованием их в интерфейс RS232.

В данном случае, я думаю, можно применить обратный конвертер RS485-RS232. Схему модуля, без применения контроллера, можно получить из схемы конвертора RS232-RS485.

Если есть желание расширить возможности модуля, то можно сделать модуль с несколькими выходами RS232, добавив контроллер и мультиплексор, переключающий порты RS232. Это только идеи, а не решения. Решения, я думаю, вам интереснее найти самим.

### ***Модуль аудио коммутатора***

Еще один модуль, который может найти применение в системе «Детский умный домик» - это модуль аудио коммутатора. В принципе, достаточно коммутации громкоговорителей через релейный модуль. Но можно коммутировать линейный выход, например, CD-проигрывателя и радиоприемника, с аудио входом телевизора. Для этой цели, аналогично релейному модулю, можно включать соответствующие электронные ключи. Дешевле всего использовать ключи цифровой 561 серии. Микросхема K561TK3 (функциональный аналог 4066) может использоваться в качестве управляемого ключа по схеме аналогичной следующей:

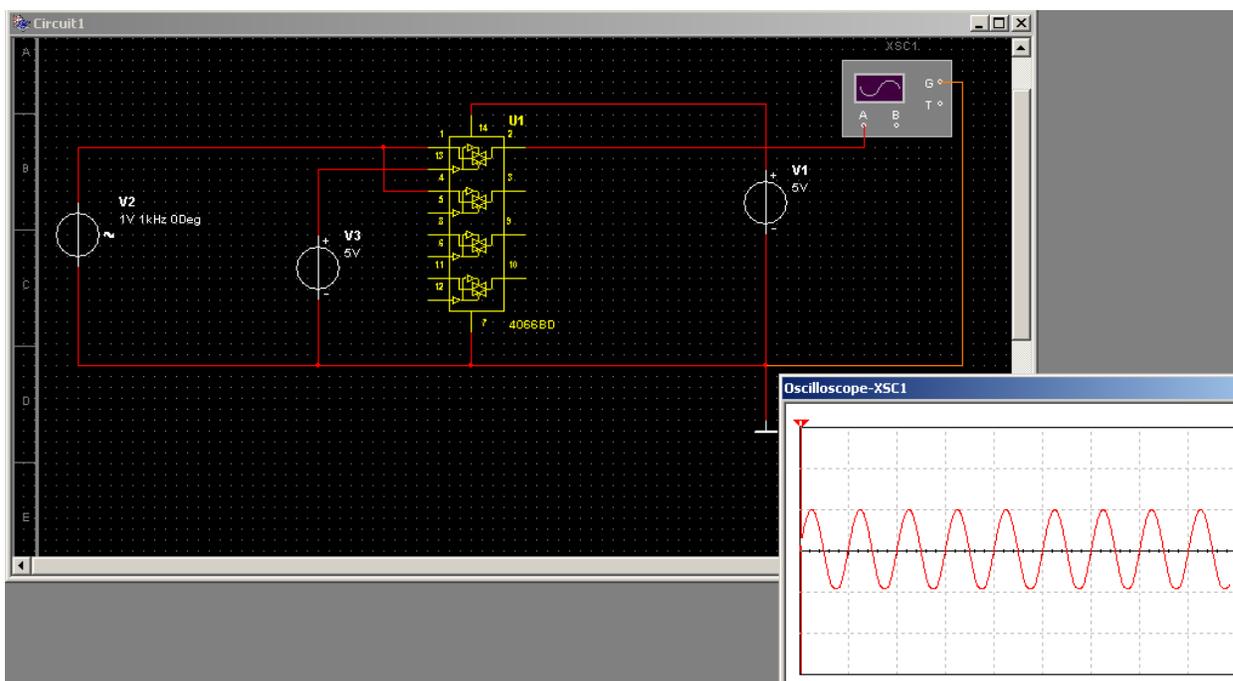


Рис.143

Микросхема допускает двухполярное питающее напряжение, при этом она передает низкие частоты от «0». Но микросхема допускает и однополярное включение питающего напряжения. При этом сигнал, может статься, лучше передавать через конденсатор большой емкости. Я уверен, нет смысла приводить схему и текст программы для контроллера, вам будет интереснее проверить все самим. При реализации не следует забывать, что коммутатор должен позволять подключение к одному выходу источника сигнала нескольких входов приемников, но не наоборот. Если вам понадобится сделать микшер (смеситель сигналов), то несколько выходов следует подключать к одному входу через развязывающие резисторы.

Для аудио коммутатора важно проверить, не ухудшается ли в значительной мере соотношение сигнал/шум. Конечно, я не имею в виду применение подобного модуля для аппаратуры класса Hi-End.

## **Модуль видео коммутатора**

Еще более интересными мне представляются эксперименты по превращению модуля аудио коммутатора в видео коммутатор. Я не уверен, что это осуществимо, но попробовать стоило бы. Если использовать новые цифровые микросхемы серии 1561 или 1564. Собственно, видео коммутатор в отношении логики ничем не отличается с аудио коммутатором. Необходимо только согласование нагрузки. Видео цепи работают на сопротивление 75 Ом, т.е. подразумевается коаксиальный кабель с волновым сопротивлением 75 Ом, и разъемы желательно применять соответствующие. И коммутатор должен быть согласован. Этому можно достигнуть, применяя входные и выходные усилители на микросхемах, хорошо работающих с видео сигналом. Его второе отличие от аудио коммутатора - полоса частот. Для видео коммутатора она должна быть не менее 0-6 МГц для композитного сигнала.

## **Модуль управляемого усилителя**

На основе микросхем усилителей в сочетании с управляющим контроллером, реле для подключения питающего напряжения к усилителям и схемами ключей на микросхемах К561ТК3 (или другими удобными электронными ключами) можно реализовать модуль управляемого усилителя (в частности многоканального). Реле будет включать и выключать усилитель по команде компьютера по системной сети. Ключи будут коммутировать резистивный делитель для ступенчатой регулировки громкости по командам компьютера или системного устройства управления.

С введением в систему умного домика управляемого усилителя расширяются возможности отображать информацию. Когда я говорил о применении уличного термометра в системе, то предлагал «помигать светом», если температура за окном ниже, чем -15 градусов. Гораздо приятнее получить сообщение от системы в виде звукового: «Не забудьте одеться теплее! На улице мороз!»

Сейчас можно недорого приобрести микросхемы вполне качественных усилителей небольшой (по сегодняшним меркам) мощности. С мощностью усилителей вопрос особый. Необходимая для создания нормальной звуковой картинке мощность очень зависит от применяемых громкоговорителей. Вернее, от их, фактического, КПД. Тяга к усилителям большой мощности возникла из-за желания, в первую очередь, передать низшие частоты звукового диапазона. Чем ниже желаемая частота, тем труднее ее воспроизвести, что требует большей мощности от громкоговорителя и усилителя. Но не следует забывать, что в реальном помещении небольшой площади без специальной и достаточно сложной (и дорогой) обработки помещения воспроизвести частоты ниже 40-50 Гц слишком сложно. А, если ограничиться нижней частотой воспроизведения 40-50 Гц, то при правильной конструкции акустического оформления, необходимая мощность усилителя может не превышать 5-10 Вт.

Но это тема другого разговора.

Что же касается системы «Детский умный домик», то мне хотелось бы добавить некоторые соображения, которые вполне можно отнести и к недетским умным домикам. Давайте подумаем, в каких случаях нам требуется звук в помещениях без источников звука?

- Есть большие любители послушать музыку в ванной – полежать в теплой ванне и послушать любимую музыку.
- Приятно, когда к вам приходят гости, включить музыку в прихожей или гостиной, чтобы ваши гости раздевались и пили коктейли в ожидании обеда под музыку.
- В столовой, где собрались гости, можно включить музыку, которая слегка заглушит стук ножей и вилок, и позволит соседям по столу переговорить без того, чтобы привлекать внимание всего стола.

Я привел несколько примеров применения аудио дистрибуции в реальной системе умного дома. Какой в данных случаях разумно реализовать вариант физического построения

## Хобби-электроникс 2. Умный дом.

аудио подсистем. Во-первых, во всех вышеперечисленных случаях разумно использовать встроенные потолочные громкоговорители. И, я почти уверен, работающие в режиме «моно». Получить от стерео системы с потолочными громкоговорителями качественную звуковую картину мне кажется слишком сложно, да еще и на большой площади.

Нужно ли заботиться о большом динамическом диапазоне звука? А какой в этом прок, если музыка должна оставаться «мягким фоном». Т.е. не нужна большая мощность.

Нужно ли передавать весь звуковой частотный спектр? Не думаю. Скорее следует позаботиться о том, чтобы, ограничив частотный спектр снизу частотой 60-70 Гц, не забыть ограничить его сверху частотой 10-12 кГц. И при этом учесть особенность восприятия звука при небольшой громкости, внося частотную коррекцию. Правильно построив эту часть устройства звуковоспроизведения, можно ограничиться уровнем громкости, имеющим одно приемлемое значение, что избавит от необходимости регулировать громкость.

Словом, не следует переоценивать возможности «звука сверху», затрачивая силы на получение очень хороших параметров звучания, если они никогда не будут применены. Лучше позаботиться о правильном построении всего тракта, и подобрать музыку, которая была бы приятна гостям, не отвлекала их ни от беседы, ни от вкусной еды за столом.

И, даже если вы намерены озвучить свою ванную комнату, не пытайтесь получить нечто небывалое, используя самое высококачественное оборудование. Условия воспроизведения звука не позволят реализовать все эти преимущества.

Высококачественное оборудование потребует специального помещения с хорошей акустической обработкой. И, если вы можете себе позволить подобное помещение, то было бы хорошо, если бы специалисты из лаборатории акустики проверили помещение с помощью своих приборов. Это позволит внести коррекцию в оборудование с целью получения максимального качества звука в данном конкретном случае.

### ***Модуль системного ИК пульта управления***

На базе решений для модуля цифровых вводов и модуля управляющих ИК кодов (Часть 1) можно разработать системный пульт с небольшим количеством команд. В качестве излучающего светодиода можно использовать либо светодиоды от старых пультов управления (или купить аналогичный), либо использовать любой светодиод ИК диапазона (или захватывающего ИК диапазон). В последнем случае их следует включать короткими импульсами с током, превышающим средний допустимый ток, но с большой скважностью. Так, кстати, работают и промышленные пульты управления, если не все, то некоторые.

Конечно, возможности такого пульта будут ограничены, как в количестве управляющих клавиш, так и в количестве системных кодов. Но о том, как можно увеличить количество клавиш для управления, мы говорили в разделе, относящемся к модулю цифровых вводов. А увеличение количества системных кодов зависит от реализации. Взяв за основу простейшие ИК коды, можно создать функцию воспроизведения такого кода, где параметр, меняющий код, может иметь значение, уместяющееся в одном байте. В этом случае энергонезависимая часть контроллера уместит достаточно много кодов. Я не знаю готового решения, но, полагаю, вам будет интересно попробовать реализовать этот вариант модуля в

составе системы. Подразумевается, что пульт управления будет переносным, а в этом случае не забудьте, что контроллер предлагает энергосберегающий режим управления!

### ***Модуль аналогового ввода для термометра***

Модули аналоговых вводов, как мне кажется, больше требуется в профессиональной деятельности, в технологических процессах. Там, где есть необходимость считывать показания датчиков, отображающих непрерывно меняющиеся параметры в широком диапазоне. Для автоматики более характерно применение пороговых устройств типа термостата. Есть некоторый задатчик параметра, есть датчик и есть устройство сравнения. Включить термостат в систему можно с помощью модуля цифровых вводов.

Я не исключаю интерес со стороны любителей к созданию системных устройств, отображающих непрерывные значения меняющихся параметров. Но, с моей точки зрения, подобное устройство лучше сделать в виде самостоятельного модуля целевого назначения. И только после того, как продумано его применение в системе. Даже достаточно простая задача отображения непрерывно меняющейся информации, например, температуры, может быть проще решена с помощью специализированного АЦП, соединенного с дисплеем. Как это сделано у мультиметров. Получается простая, надежная и многофункциональная конструкция. Реализовать подобное даже в промышленных системах умного дома, о которых я говорил в первой части, даже при наличии модуля аналогового ввода достаточно не просто.

Микроконтроллер PIC16F628A имеет встроенные аналоговые компараторы, на базе которых можно построить устройство аналогового ввода. Это упростит схему датчика. Но следует иметь в виду, что датчик температуры или освещенности в последнем случае, упрощенный до одного или двух элементов, может для связи с модулем потребовать экранированного кабеля, чтобы избежать влияния наводок. Если кому-то хочется реализовать подобное устройство, то я предложил бы разработать датчик с микроконтроллером, который был бы размещен в одном корпусе с термо- или светочувствительным элементом. И, возможно, был бы оснащен АЦП. Хотя я не исключаю возможности построения датчика полностью на микроконтроллере. Во всяком случае, это была бы интересная задача.

### ***Замена проводного канала RS485***

Для многих, кто, проведя первые эксперименты с системой «Детский умный домик», и пожелал использовать что-то из возможностей системы, препятствием к реализации задуманного может стать отсутствие в доме развитой кабельной системы, которую можно было бы использовать для создания сети. Прокладывать провода поверх плинтуса или по стенам не каждый захочет. В этом случае можно, проведя предварительные эксперименты, попробовать заменить провода радио каналом. В качестве приемопередатчиков можно использовать схемы первой книги «Хобби-электроникс» или готовые модули, используемые в моделизме. В продаже есть и одноканальные, и многоканальные радио модули для радиоуправляемых моделей. Они настроены на разрешенные для этого радиочастоты, не велики по габаритам, и, я думаю, удобны в применении. Заменяв микросхему интерфейса RS485 на подобные модули можно попытаться обойтись без проводов. Сам я не пробовал, но желание купить подобные модули, и испытать их в работе было.

При отказе от проводной связи потребуется снабдить каждый модуль системы собственным блоком питания, но эта проблема может оказаться более простой, чем прокладка проводов по комнате или по квартире. Хотя и с проводами – в настоящее время есть плоские кабели очень небольшой толщины. Такие кабели можно аккуратно разместить под обоями возле пола, и они не испортят вида комнаты.

Другим решением этой проблемы может стать использование ИК канала. При этом для ретрансляции сигнала между комнатами потребуются специальные модули ретрансляции. Их легко сделать на базе модулей ИК-приемников и ИК-излучателей. К недостаткам решения следует отнести необходимость в большом количестве ретрансляторов. Но в продаже есть ИК зеркала, которые можно использовать в качестве ретрансляторов ИК-сигналов. Словом, здесь тоже большой простор для экспериментов.

### ***Усовершенствование базовых модулей***

Наконец, в части модулей системы, следует отметить, что все описанные в книге модули далеки от совершенства. Хотя большая их часть была мной проверена на макете, я не проверял их совместную работу. Подобная проверка может открыть большой простор для творчества и по устранению недостатков, и по усовершенствованию самих устройств. При этом усовершенствование может касаться как увеличения функциональности, так и надежности. Сама микросхема контроллера настолько надежна, что хотя бы из уважения к этому факту, следует не оставлять без внимания вопрос надежности устройства в целом.

Одним из усовершенствований, о котором я говорил, может быть использование встроенной EEPROM памяти для задания адреса модуля. В промышленных разработках адрес устройства задается программно. Для этой цели один из вводов устройства используется для перехода к режиму настроек. Когда он соединен с общим проводом, при включении питающего напряжения модуль переходит в специальный режим настроек. По сети ему передаются параметры – адрес модуля, скорость сетевой работы – которые модуль запоминает в энергонезависимой памяти. Если применить построение модуля с внешним кварцем на 20 МГц, то диапазон сетевых скоростей расширяется до 115200 бит/сек. Сами по себе эксперименты с разной скоростью сетевой работы могут оказаться не менее увлекательными, чем разработка модулей. В этом случае возможность программного изменения скорости работы USART очень полезна.

Наконец, микроконтроллер PIC16F628A не является самым мощным в серии PIC контроллеров. Можно провести эксперименты с контроллерами PIC18Fxxx, или другими контроллерами. Выбор контроллера вне профессиональной разработки может определяться разными факторами. Наличием подходящего контроллера в продаже, доступностью его по цене. Может играть роль и простой интерес к конкретной микросхеме. Главное, что я хотел показать на протяжении книги, что освоение работы с микроконтроллерами доступно любому, кому это интересно. Что это интересно любому, кому нравится творить, созидать, изобретать. Даже если изобретаешь велосипед, это занятие оказывается много интереснее, чем покупка готового велосипеда.

Последнее, о чем осталось рассказать в этой части, это другая среда разработки основной программы.

## ***Вторая версия основной программы***

Причина, по которой я хочу рассказать о другой среде программирования, именно KDevelop, работающей с операционной системой Linux, в ее большей доступности для многих, чем Visual Basic. Эта среда программирования входит в состав многих дистрибутивов. Хотя она и не единственная в них.

Итак.

### ***Основная программа в Kdevelop***

Я предполагаю работать с релейным модулем. По этой причине я программирую микроконтроллер соответствующим образом и загружаю ASPLinux. Я уже говорил, что эта операционная система поставляется с огромным количеством приложений, в частности и для разработки программ.

Запускаем программу KDevelop. Моя версия в данный момент предоставляет в меню запуска выбрать язык программирования и вариант проекта, хотя это можно сделать и несколько позже. Итак, я запускаю KDevelop в разделе основного меню - “Приложения-Разработка-KDevelop KDE/C++”. Если это первый запуск, то ни один проект не открывается, если повторный запуск, то открывается последний проект, над которым вы работали. Для создания нового проекта предыдущий следует закрыть, выбрать в меню KDevelop “Проект-Новый проект”:

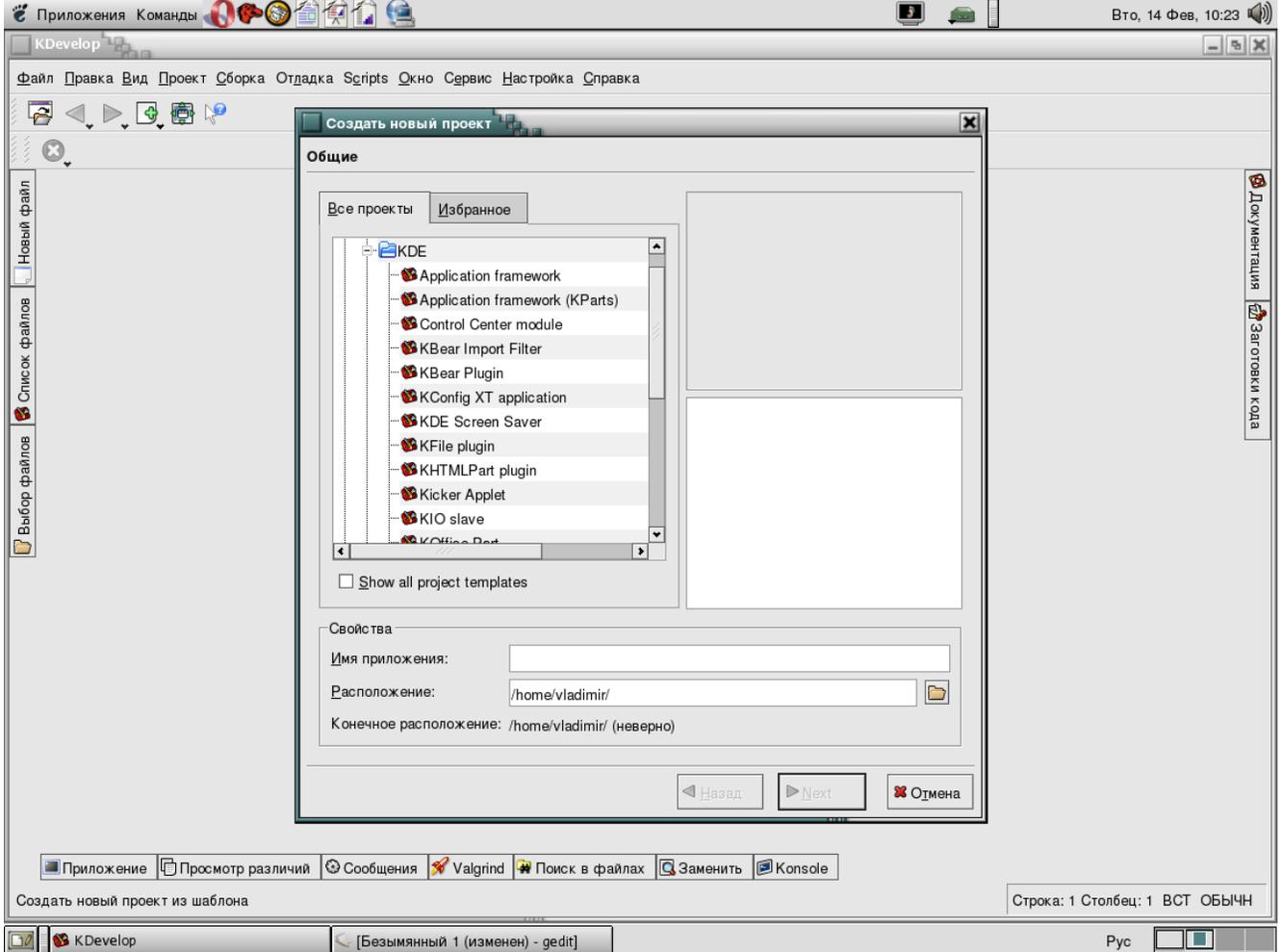


Рис.144

В диалоговом окне выбираем “C++ - KDE – Simple Designer based KDE Application” и задаем имя проекта (в данном случае book), и место его расположения (в данном случае домашняя папка пользователя):

## Хобби-электроникс 2. Умный дом.

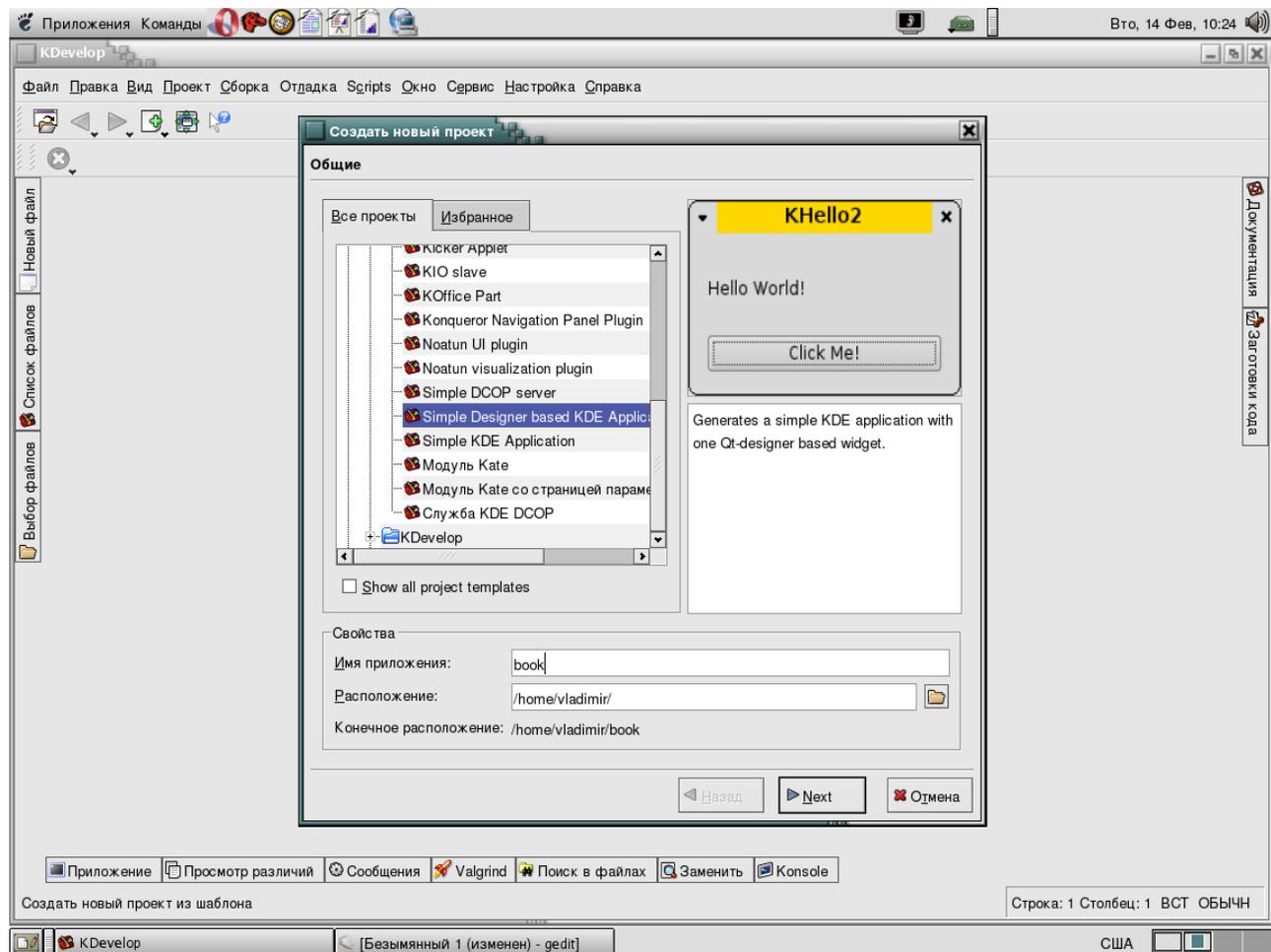


Рис. 145

Теперь нам остается нажать несколько раз на клавишу “Next”, при этом можно сделать уточнения, относящиеся к проекту, но их можно сделать и позже. Заканчивается оформление проекта тогда, когда клавиша “Next” превращается в клавишу “Готово”, которую и нажимаем в последний раз, получая проект:

## Хобби-электроникс 2. Умный дом.

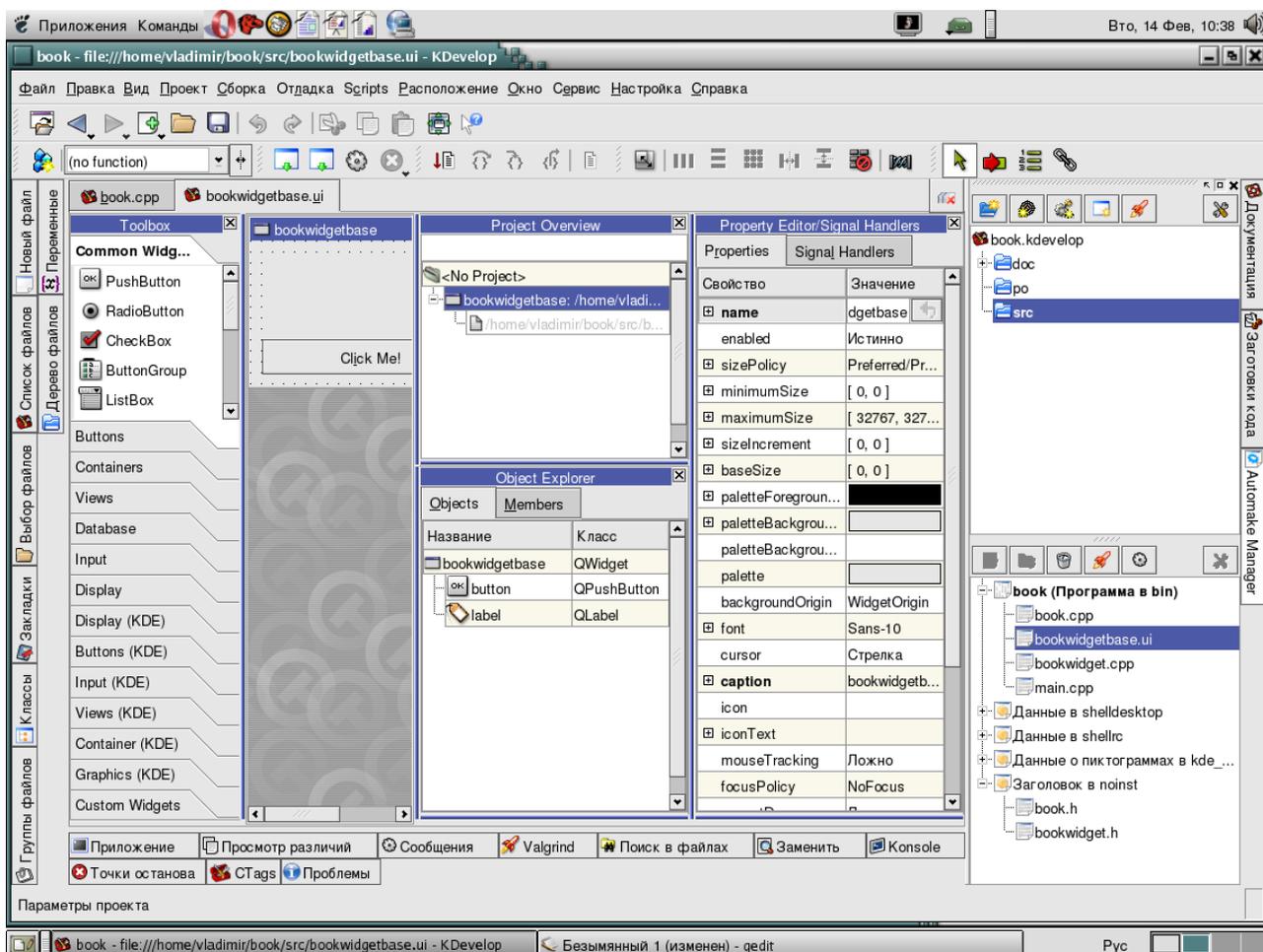


Рис.146

Нажав на правой инструментальной панели ярлычок “Automake Manager”, открываем окно менеджера, в котором можно открыть все файлы проекта и файлы заголовков двойным щелчком правой клавиши мышки. Я открыл файл book.cpp:

## Хобби-электроникс 2. Умный дом.

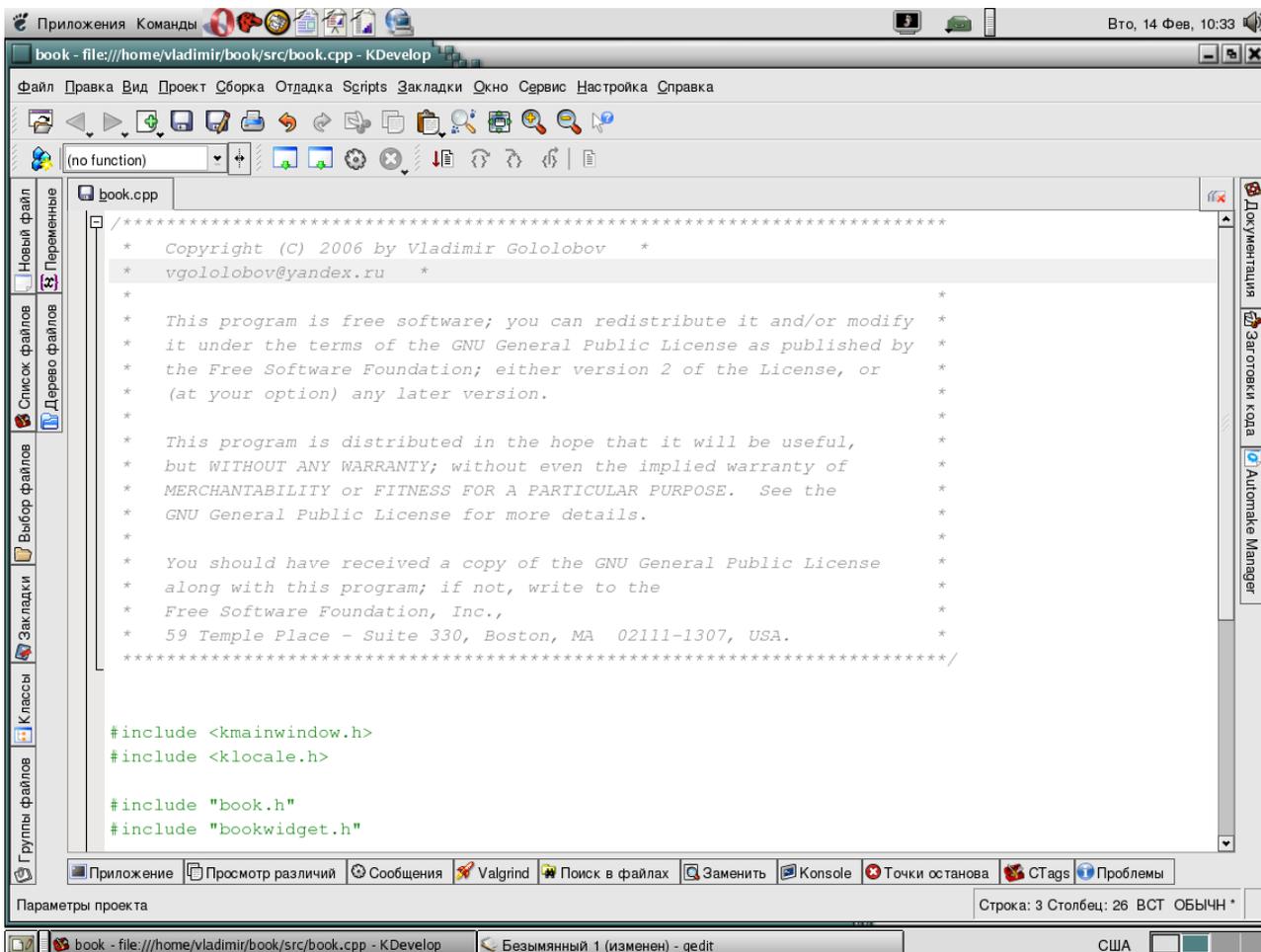


Рис.147

Файл интересен тем, что относится к встроенному редактору интерфейса программы. Окно с клавишей “Click Me!” появится после компиляции и сборки проекта при запуске программы. Для компиляции и сборки сначала в основном меню выбираем “Сборка-Запустить automake и др.”. В нижней части открывается окно сообщений, где показывается процесс сборки:

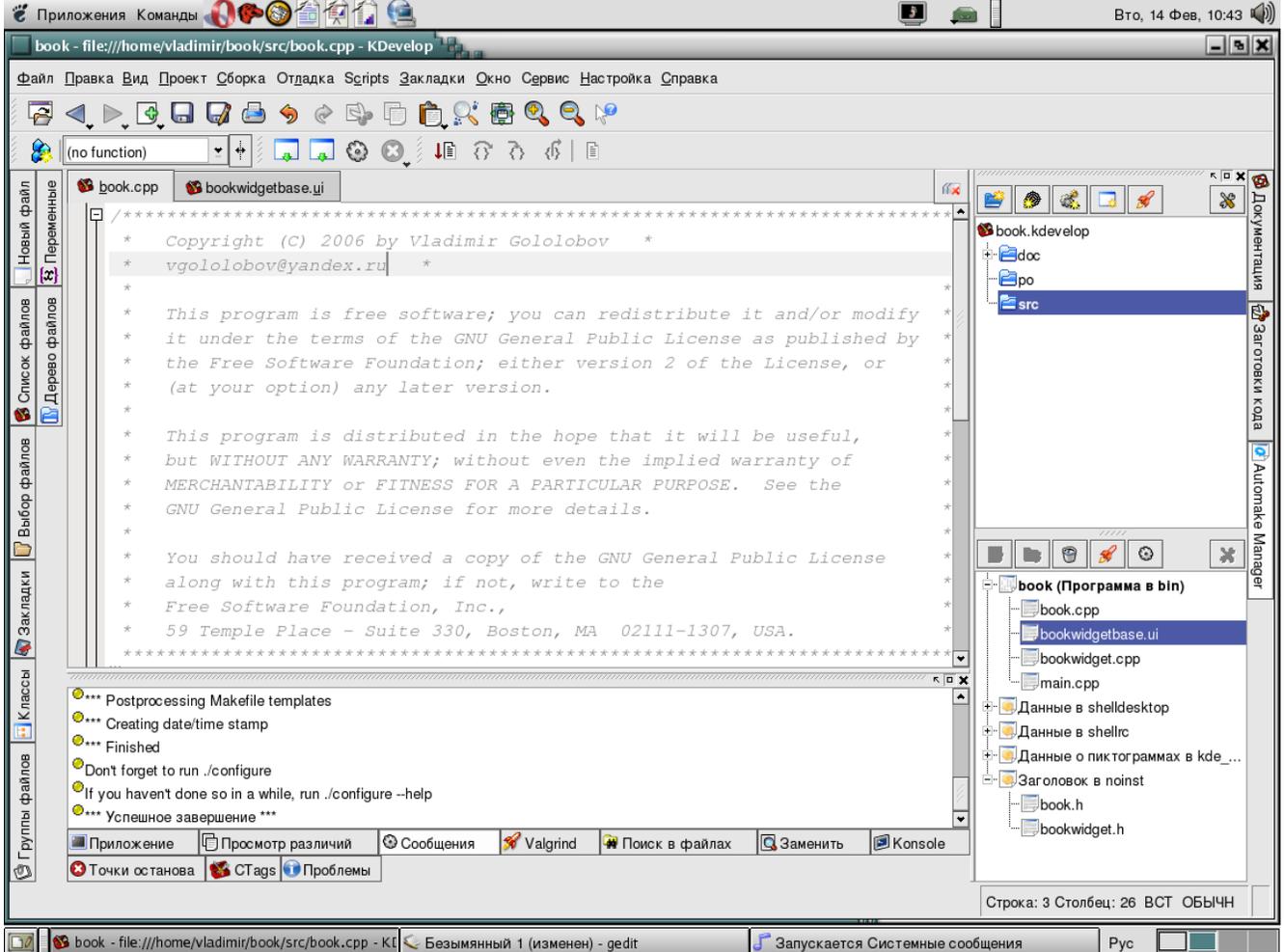


Рис.148

Затем мне приходится сделать лишнее “телодвижение” - после одного из экспериментов в KDevelop с разными языками программирования мне приходится поправлять файл `configure.sh`, который находится в папке проекта `book`. Я открываю файл текстовым редактором, удаляю показанную часть текста почти в самом начале файла и сохраняю файл:

```
# Support unset when possible.
if ( (MAIL=60; unset MAIL) || exit) >/dev/null 2>&1; then
    as_unset=unset
else
    as_unset=false
fi
```

Без этой операции появляются сообщения об ошибках при конфигурировании проекта, которое запускается в разделе основного меню “Сборка-Запустить `configure`”. После завершения этого процесса можно запустить сборку “Сборка-Собрать проект” или сразу запустить программу “Сборка-Выполнить программу”, что приведет к сборке проекта и запуску программы.

Есть еще один момент, который привел меня в замешательство, но, возможно, это тоже результат моих экспериментов. Не получалась работа с графическим редактором интерфейса. Для его нормальной работы в опции проекта: Проект-Project Options, раздел “Поддержка C++” на вкладке “Автодополнение кода” в части Code Completion Databases понадобилось с помощью клавиши “Добавить” добавить поддержку Qt:

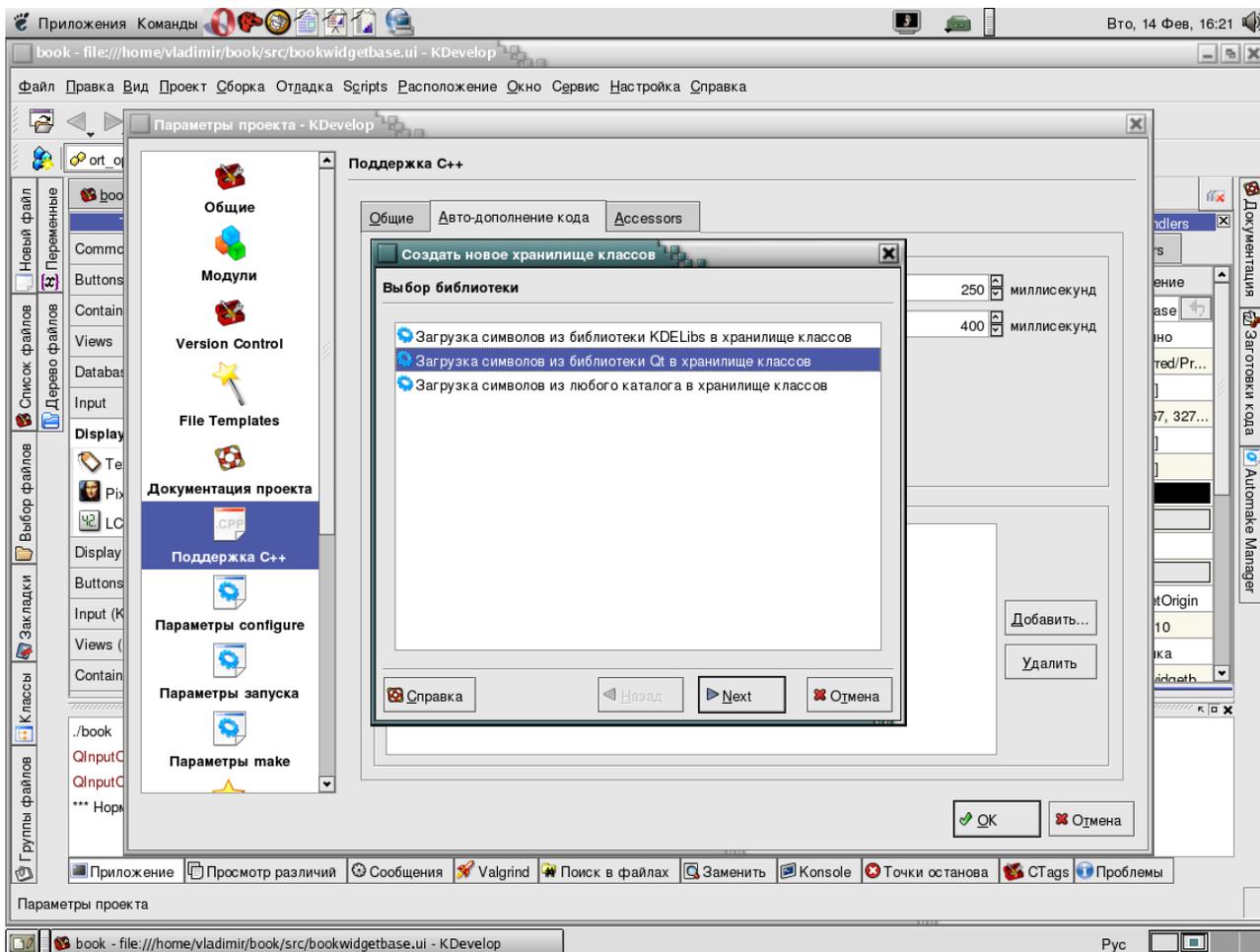


Рис.149

После создания базы проблема перестала беспокоить. Qt, насколько я понимаю, это мощная библиотека, поддерживающая графику. Разработана она фирмой Trolltech. Все графические объекты интерфейса – объекты этой библиотеки. Но вернемся к проекту.

## Хобби-электроникс 2. Умный дом.

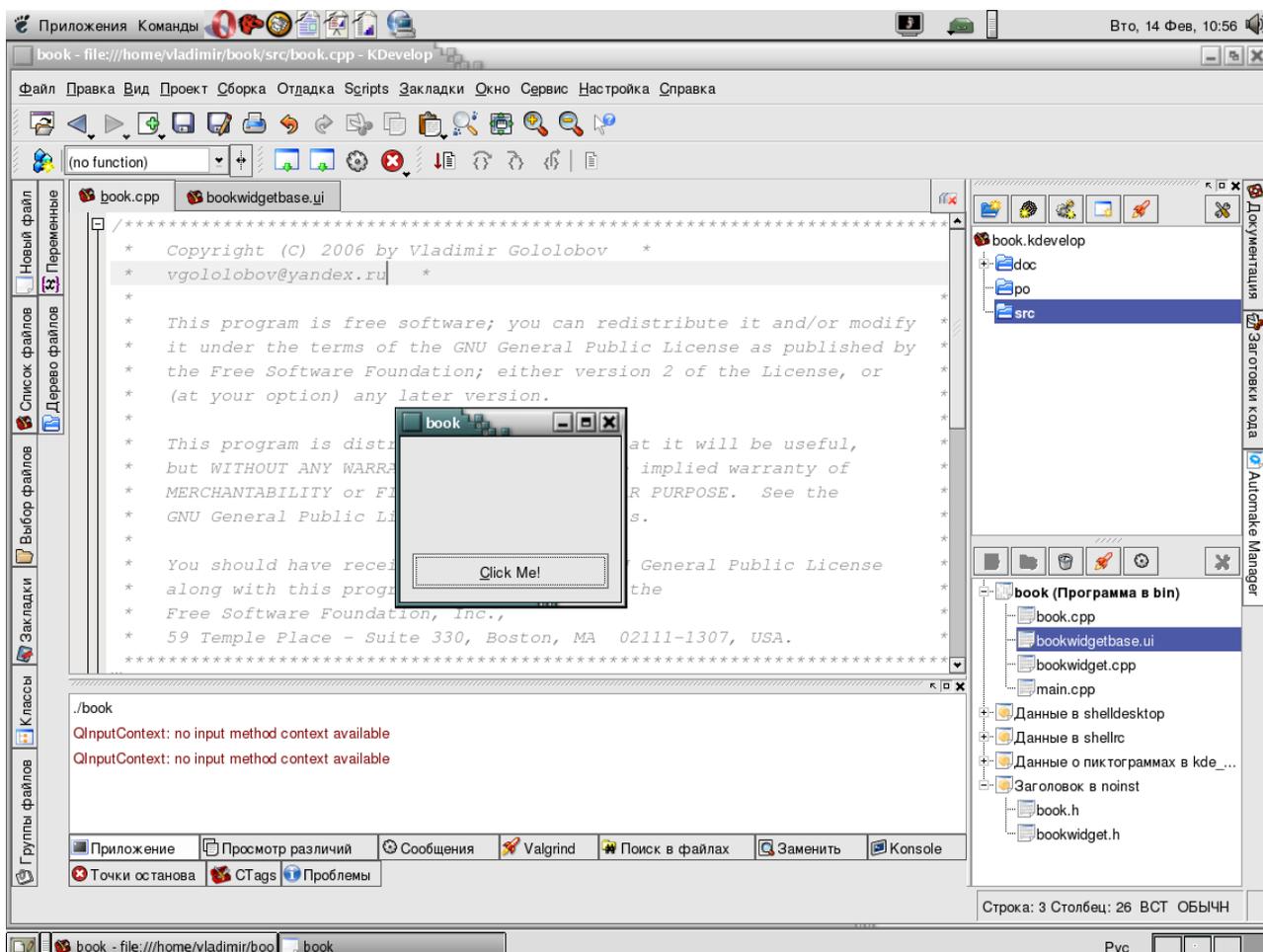


Рис.150

Если теперь нажать на клавишу появившегося окна book – Click Me!, то появится традиционное для языка C – Hello World! Таким образом, почти незаметно для себя мы написали первую программу на языке C++, работающую с операционной системой Linux.

Теперь хотелось бы эту программу превратить в то, что нужно нам. Меня беспокоят некоторые аспекты предстоящей работы, но для начала я, выключив работающую программу, возвращаюсь в редактор интерфейса к файлу bookwidgetbase.ui. С этой небольшой проблемой я столкнулся позже, но хочу устранить ее сейчас. Существо проблемы в том, что возникли некоторые неприятности с изменением интерфейса, а, изменив интерфейс, я никак не мог справиться с размерами окна при запуске программы. Окно оставалось в том же первоначальном виде, что и при первом запуске. Немного о том, что мы видим в редакторе. Основу интерфейса, как и в других средах программирования, составляет форма, на которой размещаются кнопки, этикетки и т.д. - все составляющие формы. Кроме нее открыты инструментальная панель (слева) и менеджеры проекта и свойств формы и ее составляющих (справа).

Для устранения проблемы, вероятно, есть более правильные пути, но я пошел следующим – щелкнув правой клавишей мышки по форме, в выпадающем меню выбираем пункт “Разорвать расположение”.

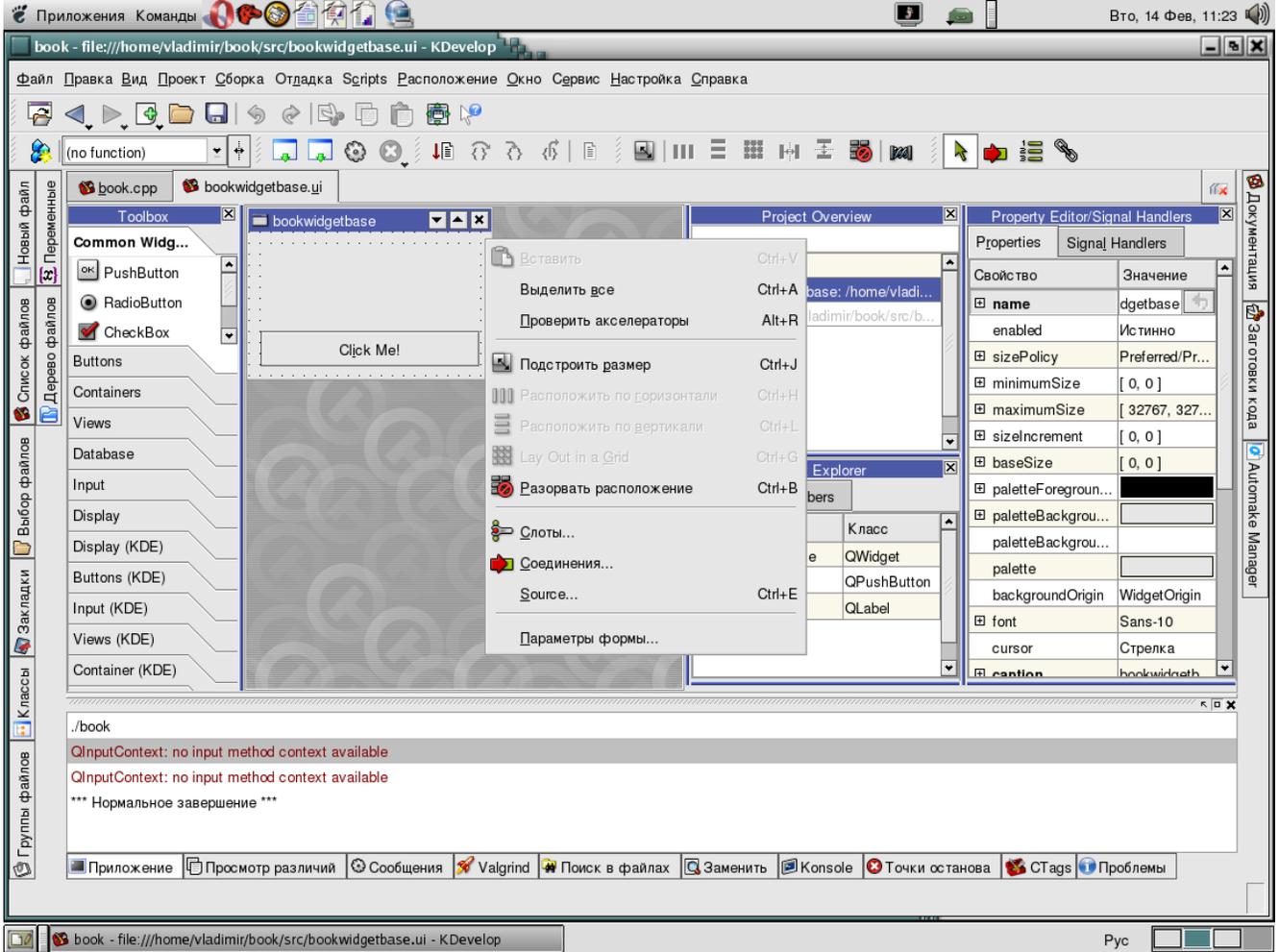


Рис.151

Теперь я могу работать с формой. На форме в данный момент одна этикетка – label, и одна клавиша – button. Для начала я увеличу размер формы и поменяю местами, изменив размер, компоненты.

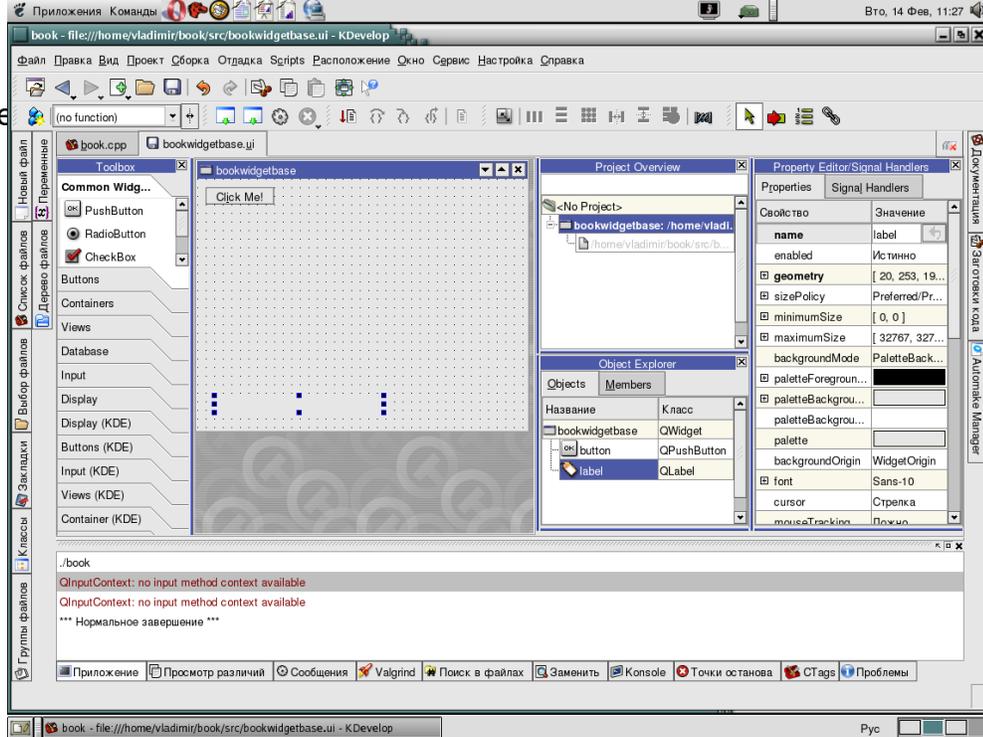


Рис.152

Кнопку я позже использую для работы с портом, этикетку для вывода сообщений о состоянии порта. Добавим разделители, как показано ниже. Щелкнув правой клавишей мышки по форме, выбираем в выпадающем меню “Lay Out in a Grid”. Теперь при запуске программы окно приобретает тот вид, который бы мне хотелось видеть.

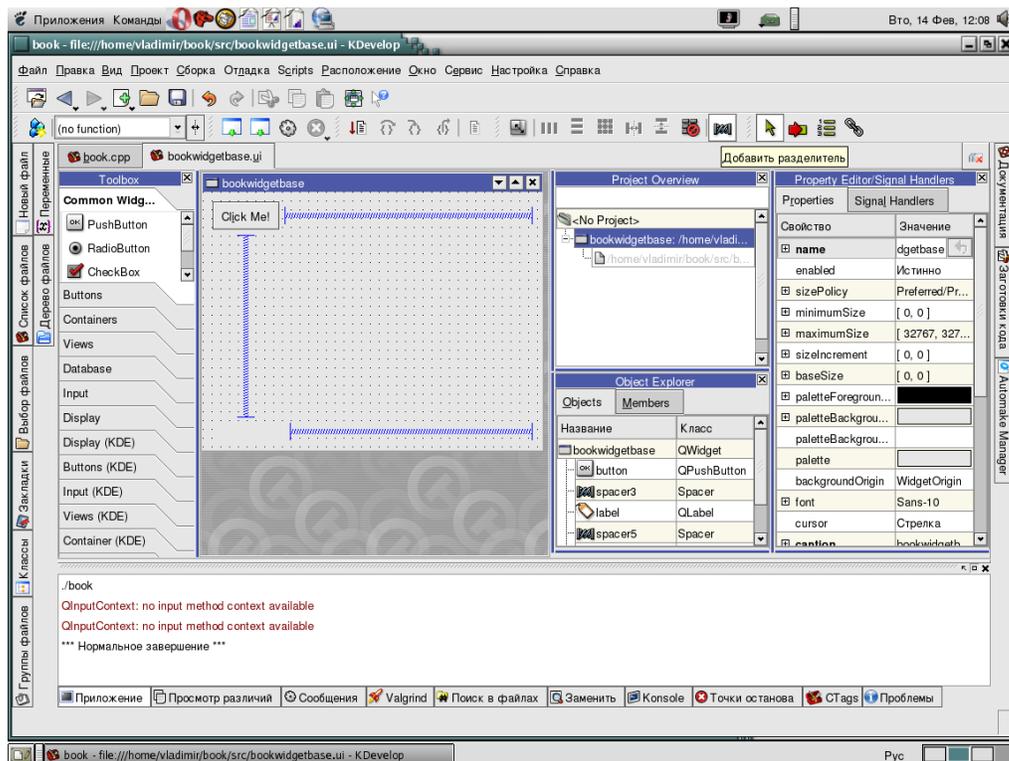


Рис.153

Однако вернемся к проблемам, вызывающим у меня беспокойство. Я бы сформулировал это так - “Плохо, когда забудешь то, что не знаешь. Особенно, если забудешь, что ты этого не знаешь!”

Производственная необходимость некогда заставила меня обратиться к языку C++, на котором я написал небольшую программку. Для знакомства с языком я законспектировал оригинальную версию книги Липпмана “Основы C++”. С тех пор прошло достаточно времени, чтобы я все забыл. По этой причине, я буду добавлять “шпаргалки”, постаравшись не использовать сложные синтаксические или программные конструкции.

Первое, что я запишу, поскольку всегда путаюсь:

### **Указатели в C.**

*Пусть, мы имеем переменную с именем var типа int (целое): int var;  
Теперь создадим другую переменную, которую назовем: int\* p\_var;  
Появившаяся звездочка говорит, что переменная p\_var – это указатель. Если теперь мы присвоим этой переменной следующее значение:  
p\_var = &var; то, в переменной p\_var будет храниться адрес первой переменной. Положим, есть еще одна переменная: int new\_var;  
Мы можем добавить к первой переменной единицу, присвоив результат новой переменной:  
new\_var = var + 1;  
Но в языке C то же можно записать, используя указатель:  
new\_var = \*p\_var + 1; звездочка в данном случае говорит о том, что мы используем значение переменной, адрес которой хранит переменная p\_var.  
Таким образом:  
&var – указывает, что используется адрес переменной var.  
int\* p\_var; - p\_var это указатель.  
\*p\_var – показывает, что используется значение переменной, на которую указывает p\_var. Указатели могут указывать на переменные, на функции, на другие указатели и т.д.*

Не знаю, поможет ли вам моя шпаргалка, но для себя я ее оставлю, тем более что в C++ без звездочек, разбросанных по всему тексту, шагу не ступишь.

Второй аспект. Когда я впервые столкнулся с KDevelop, то это была версия, в которую не интегрировался редактор интерфейса. Для этих целей использовалась отдельная программа Qt Designer (она и сейчас входит в дистрибутив). И это не единственная возможность создать графический интерфейс, как и KDevelop не единственная среда разработки на C. Объединение программы на C++ и графического интерфейса выглядело несколько иначе, что заставляет меня, как мне кажется, сейчас делать много ненужного. Но я не собираюсь поразить воображение пользователей необычным дизайном программы, по причине чего не хочу пока разбираться с этим вопросом. В памяти осталось, что Qt Designer использует понятие слотов. В моем представлении, это нечто в роде гнезда, в которое

## Хобби-электроникс 2. Умный дом.

вставляется модуль, как в материнскую плату вставляется модуль памяти. Используется и понятие сигналов, что для меня равносильно понятию использования выводов разъема. Через его выводы модуль получает сигналы от материнской платы, или сам передает сигналы материнской плате.

И, действительно, открыв в KDevelop файлы `bookwidget.cpp` и `bookwidget.h`, в последней находим слот:

```
public slots:
    /*$PUBLIC_SLOTS$*/
    virtual void button_clicked();
```

А в первой находим сигнал, относящийся к этому слоту:

```
void bookWidget::button_clicked()
{
    if ( label->text().isEmpty() )
    {
        label->setText( "Hello World!" );
    }
    else
    {
        label->clear();
    }
}
```

Этим для начала я и воспользуюсь, хотя, подозреваю, есть более простые способы реализовать то, что мне надо. В первую очередь я хочу переименовать клавишу из `button` в `port_open`. Я делаю это на вкладке `bookwidgetbase.ui`. В Object Explorer'e выбираю `button`, а в Property Editor в графе `name` меняю имя с `button` на `port_open`. Щелчком правой клавиши мышки по форме открываю меню, выбираю “Слоты”, где меняю имя слота. Повторяю это, выбирая “Соединения”, где в разделе `slot` указываю новое имя слота. Затем меняю, чтобы не путаться в файлах `.cpp` и `.h` ИМЯ:

```
void bookWidget::port_open_clicked()
virtual void port_open_clicked();
```

Сохраняю эти файлы. Пытаюсь сохранить файл `bookwidgetbase.ui`, но он не хочет сохраняться. Тогда в выпадающем меню формы выбираю “Разорвать расположение”, сохраняю файл, и повторяю “`Layout in a Grid`”, сохраняя файл.

В итоге проект собирается и запускается. В редакторе интерфейса я добавляю кнопку, которую называю `port_close`, и `ButtonGroup` (и то, и другое из меню `Common Widgets` слева). Текст группы кнопок я меняю на “Гостиная” (двойной щелчок по тексту на форме открывает редактор текста).

Пока не забыл, я вношу добавления в файлы `bookwidget.cpp`:

Хобби-электроникс 2. Умный дом.

```
void bookWidget::port_close_clicked()  
{
```

и bookwidget.h:

```
virtual void port_close_clicked();
```

Затем открываю в редакторе интерфейса раздел “Слоты” и добавляю новый слот с помощью клавиши “Добавить функцию”:

```
port_close_clicked()
```

Отказываюсь генерировать новый класс. Добавляю новое соединение с помощью раздела “Соединения” и клавиши “Новое”, щелкая левой клавишей мышки по ячейкам таблицы:

```
Sender – port_close  
Сигнал – clicked()  
Receiver - bookwidgetbase  
Слот – port_close_clicked()
```

И все сохраняю.

Для завершения этого этапа работы я добавляю еще две клавиши “Свет включить”, “Свет выключить” и из раздела инструментального меню Display этикетку QPixmapLabel. Их я пока не оформляю. С QPixmapLabel история простая – я попытался сделать включение и выключение ламп, как это делал в Visual Basic, но не получилось. Я не стал на этом “зацикливаться” и решил обойти проблему. В графическом редакторе KolourPaint я делаю полосу 350x30, которую закрашиваю в желтый цвет, а картинку сохраняю, как light (Рисунок Portable QPixmap). Сохраняю в папке проекта. Теперь картинку по умолчанию я поменяю на свой рисунок, который и будет изображать включенный свет:

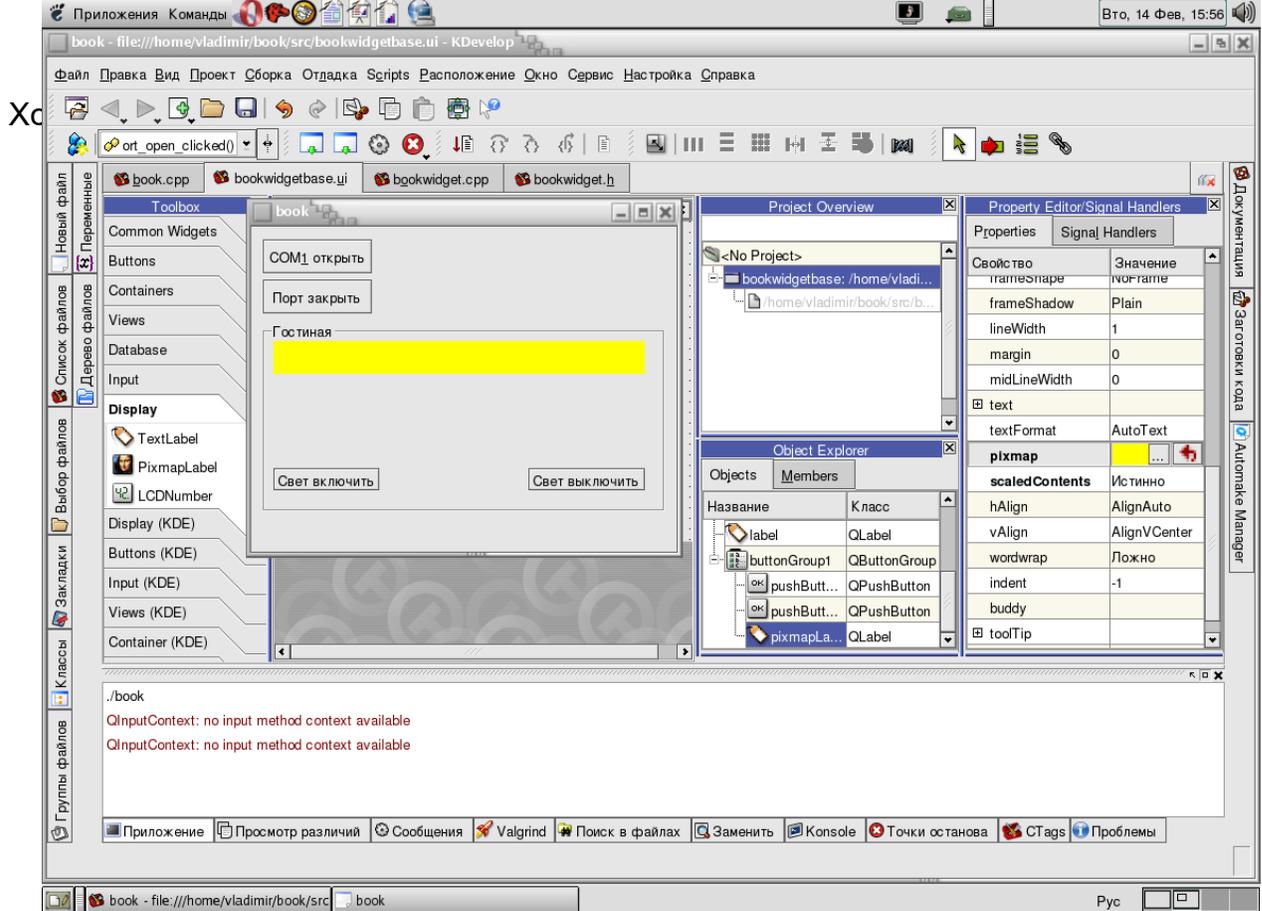


Рис.154

Под клавишей “Свет выключить” на основной форме я расположил еще одну TextLabel, которую назвал answer. Ее я предполагаю использовать для получения ответа от релейного модуля для запроса статуса реле.

Вот, собственно, что я предполагал сделать в графическом плане. И хотя по дороге я несколько раз наступал на грабли, это были еще не “те” грабли.

Что меня беспокоило с самого начала, и продолжает беспокоить сейчас, так это возможность работы с COM-портом. Среда программирования в явном виде не предоставляет мне такой возможности, как это было в Visual Basic. Первое, что приходит в голову – поискать в Интернете. И не напрасно. Я нахожу пакет qextserialport -0.9.0 автор которого:

```

\class Posix_QextSerialPort
\version 0.70 (pre-alpha)
\author Wayne Roth

```

предлагает исходные тексты классов объектов для кросс-платформенного применения при работе с последовательным портом.

С объектами разных классов мы, в сущности, уже работали, когда создавали интерфейс. Почитав у Липпмана то, что относится к классовой сущности языка C++, я, в какой-то мере, осознал, что класс – это классно. Но как применить наверняка очень полезное и грамотное творение Вэйна Рота к моей простенькой программе, я понять не могу. Но почти уверен, что есть разные пути и способы. Но почти уверен, что я их не знаю. Для начала я просто создаю свой собственный класс, который добавляю в программу. Поскольку этот

## Хобби-электроникс 2. Умный дом.

механизм работает, я, не мудрствуя лукаво, добавляю в программу исходные тексты программы, написанные Вэйном Ротом. Для моих целей хватает `posix_qextserialport` и `qextserialbase`. Я добавляю в проект и файлы заголовков, и основные файлы.

Второй, быть может не менее важный для меня момент осознания, куда и что добавлять? Я понимаю, что следует записать

```
#include <posix_qextserialport.h>
#include <qextserialbase.h>
```

но куда? И опять, я, не мудрствуя лукаво, добавляю эти записи в файл, где происходят все события моей программы – `bookwidget.cpp` и его файл заголовка. Не знаю, насколько это правильно, но это срабатывает. Программа транслируется без ошибок, проект собирается. Кстати, проект, который я собирал, я назвал так же, как в Visual Basic – `barby`.

К этому проекту и обратимся в дальнейшем. Его графический интерфейс выглядит, практически так же, как вышеприведенный. Файл `barbywidget.cpp` (аналогично файлу `bookwidget.cpp`) выглядит следующим образом:

```
/*
 * Copyright (C) 2006 by Vladimir Gololobov *
 * vgololobov@yandex.ru *
 *
 * This program is free software; you can redistribute it and/or modify *
 * it under the terms of the GNU General Public License as published by *
 * the Free Software Foundation; either version 2 of the License, or *
 * (at your option) any later version. *
 *
 * This program is distributed in the hope that it will be useful, *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the *
 * GNU General Public License for more details. *
 *
 * You should have received a copy of the GNU General Public License *
 * along with this program; if not, write to the *
 * Free Software Foundation, Inc., *
 * 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA. *
 */
```

```
#include <qlabel.h>
#include <posix_qextserialport.h>
#include <qextserialbase.h>
```

```
#include "barbywidget.h"
```

```
char command_on[7] = "R14$1N";
char command_of[7] = "R14$1F";
```

Posix\_QextSerialPort comPort("/dev/ttyS0"); // Это позволяет использовать готовый класс

```
barbyWidget::barbyWidget(QWidget* parent, const char* name, WFlags fl)
    : barbyWidgetBase(parent,name,fl)
{
    light->setEnabled(FALSE); // Здесь light – этикетка с желтой картинкой
}

barbyWidget::~barbyWidget()
{}

/*$SPECIALIZATION$*/
void barbyWidget::port_open_clicked()
{
    comPort.setName("/dev/ttyS0"); // Далее следуют установки порта COM1
    comPort.setDataBits(DATA_8);
    comPort.setStopBits (STOP_1);
    comPort.setBaudRate (BAUD2400);
    comPort.open(0);

    if (comPort.isOpen())
    {
        label->setText( "port open" ); // Это переделанная этикетка
    }
}

void barbyWidget::port_close_clicked()
{
    comPort.close();
    if (!comPort.isOpen())
        label->setText( "port close" );
}

void barbyWidget::light_on_clicked()
{
    comPort.writeBlock(command_on, 6);
    light->setEnabled(TRUE);
}

void barbyWidget::light_off_clicked()
{
    comPort.writeBlock(command_of, 6);
    light->setEnabled(FALSE);
}
#include "barbywidget.moc"
```

## Хобби-электроникс 2. Умный дом.

Файл заголовка barbywidget.h:

```
/******  
 * Copyright (C) 2006 by Vladimir Gololobov *  
 * vgololobov@yandex.ru *  
 * *  
 * This program is free software; you can redistribute it and/or modify *  
 * it under the terms of the GNU General Public License as published by *  
 * the Free Software Foundation; either version 2 of the License, or *  
 * (at your option) any later version. *  
 * *  
 * This program is distributed in the hope that it will be useful, *  
 * but WITHOUT ANY WARRANTY; without even the implied warranty of *  
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the *  
 * GNU General Public License for more details. *  
 * *  
 * You should have received a copy of the GNU General Public License *  
 * along with this program; if not, write to the *  
 * Free Software Foundation, Inc., *  
 * 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA. *  
*****/
```

```
#ifndef _BARBYWIDGET_H_  
#define _BARBYWIDGET_H_
```

```
#include "barbywidgetbase.h"  
#include <posix_qextserialport.h>  
#include <qextserialbase.h>
```

```
class barbyWidget : public barbyWidgetBase  
{  
    Q_OBJECT
```

```
public:  
    barbyWidget(QWidget* parent = 0, const char* name = 0, WFlags fl = 0 );  
    ~barbyWidget();  
    /*$PUBLIC_FUNCTIONS$*/
```

```
public slots:  
    /*$PUBLIC_SLOTS$*/  
    virtual void port_open_clicked(); // Это бывшая клавиша button  
    virtual void port_close_clicked(); // Далее добавленные клавиши  
    virtual void light_on_clicked();  
    virtual void light_off_clicked();
```

```
protected:  
    /*$PROTECTED_FUNCTIONS$*/
```

## Хобби-электроникс 2. Умный дом.

protected slots:

```
/*$PROTECTED_SLOTS$*/
```

```
};
```

```
#endif
```

После запуска программы и нажатия на клавишу «COM1 открыть»:

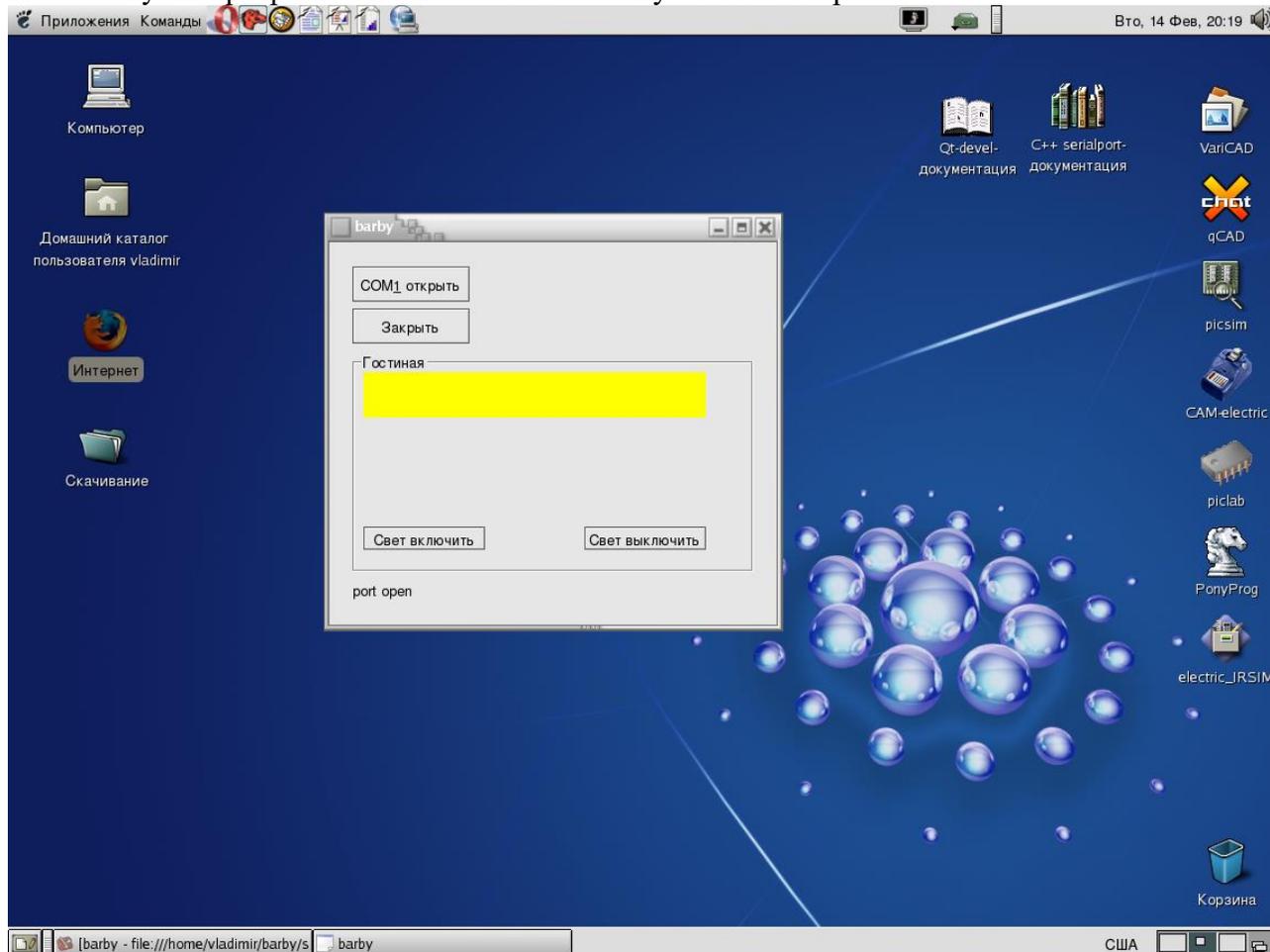


Рис.155

Нажатие на клавишу “Свет включить” дает картинку:

## Хобби-электроникс 2. Умный дом.

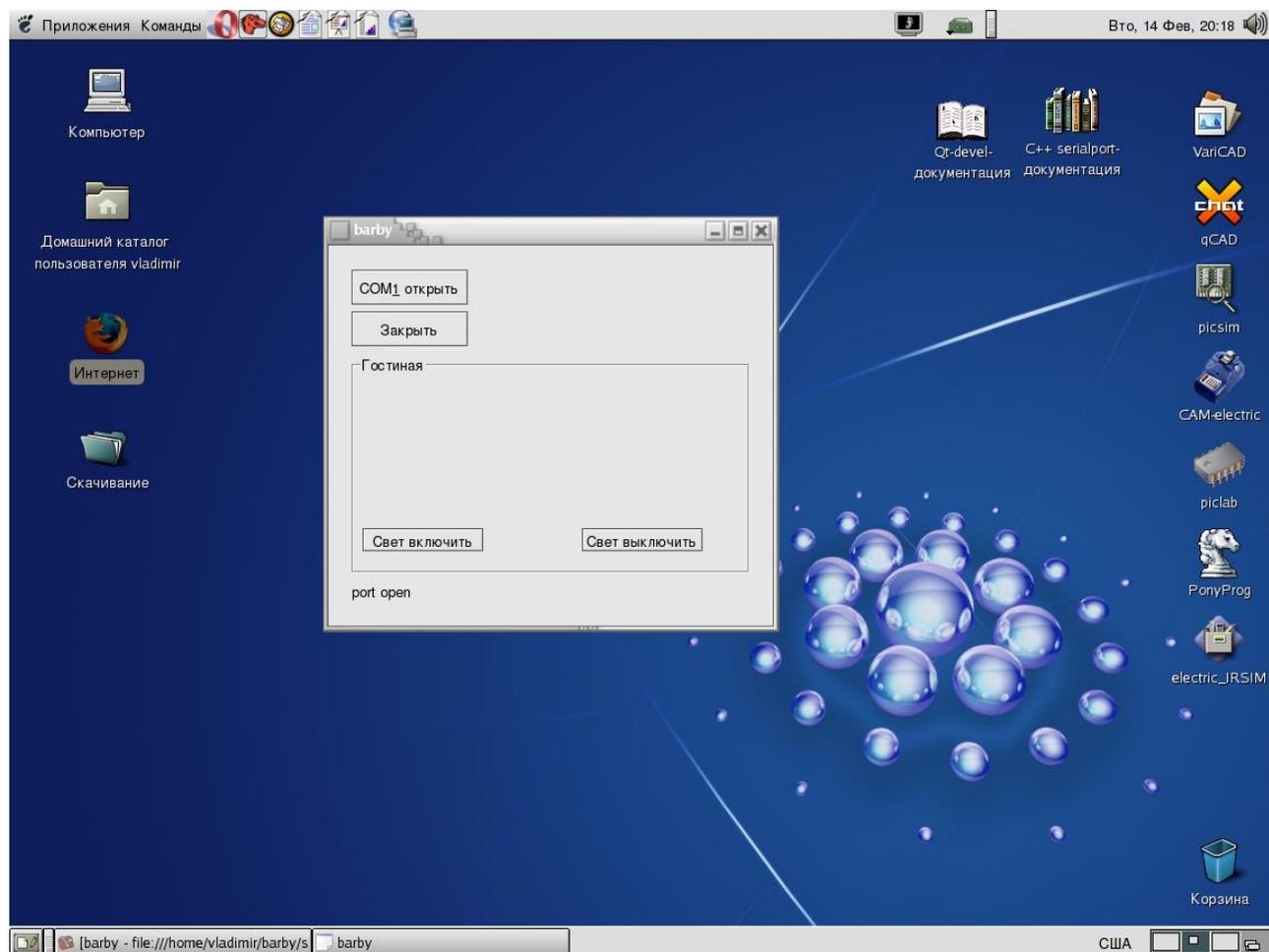


Рис.156

Первая неожиданность – программа работает, но модуль не реагирует на команды. Ситуация повторяет начало работы с портом в части проверки конвертера. Первое, что удастся проверить с помощью мультиметра - наличие изменений сигналов на входе конвертера. Если изменить порт (т.е. /dev/ttyS1), напряжение на сигнальном выводе остается неизменным. Это хорошо. Но плохо, что модуль не реагирует на команды. На всякий случай я возвращаюсь к программе на Visual Basic и проверяю работу модуля. Сомнений не остается – модуль обрабатывает команды.

Пользуясь тем, что модуль работает, проверяю сигналы интерфейса RS485 в программе работающей с Visual Basic. Затем я перезагружаю операционную систему и запускаю программу из KDevelop. Спустя некоторое время, удастся заметить, что сигнал RTS на 3 выводе микросхемы MAX1483 сбрасывается в “0” при открытии порта. Добавление в программу в разделе настроек после открытия порта:

```
comPort.open(0);
```

```
comPort.setRts (FALSE); // Добавленная строка
```

исправляет ситуацию. Модуль выполняет команды.

## Хобби-электроникс 2. Умный дом.

Теперь остается разобраться с командами запроса статуса реле. Модифицированная программа выглядит следующим образом:

```
#include <qlabel.h>
#include <posix_qextserialport.h>
#include <qextserialbase.h>

#include "barbywidget.h"

char command_on[7] = "R14$1N";
char command_of[7] = "R14$1F";
char command_st[7] = "R14$1S";
char answer_st[19] = "                ";
int i = 0;

Posix_QextSerialPort comPort("/dev/ttyS0");

barbyWidget::barbyWidget(QWidget* parent, const char* name, WFlags
fl)
    : barbyWidgetBase(parent, name, fl)
{
    light->setEnabled(FALSE);
}

barbyWidget::~barbyWidget()
{}

/*$SPECIALIZATION$*/
void barbyWidget::port_open_clicked()
{
    comPort.setName("/dev/ttyS0");
    comPort.setDataBits(DATA_8);
    comPort.setStopBits (STOP_1);
    comPort.setBaudRate (BAUD2400);
    comPort.open(0);
    comPort.setRts (FALSE);
    answer->setText("                ");
    if (comPort.isOpen())
    {
        label->setText( "port open" );
    }
}

void barbyWidget::port_close_clicked()
{
    comPort.close();
    if (!comPort.isOpen())
        label->setText( "port close" );
}
```

## Хобби-электроникс 2. Умный дом.

```
answer->setText("                ");
}

void barbyWidget::light_on_clicked()
{
comPort.writeBlock(command_on, 6);
comPort.writeBlock(command_st, 6);
while (!comPort.atEnd());
for (i=0;i<100000000; ++i);
comPort.readBlock(answer_st, 18);
answer->setText(answer_st);
if (answer_st[17] == 'N')
light->setEnabled(TRUE);
}

void barbyWidget::light_off_clicked()
{
comPort.writeBlock(command_of, 6);
comPort.writeBlock(command_st, 6);
while (!comPort.atEnd());
for (i=0;i<100000000; ++i);
comPort.readBlock(answer_st, 18);
answer->setText(answer_st);
if (answer_st[17] == 'F')
light->setEnabled(FALSE);
}
#include "barbywidget.moc"
```

В основном изменения касаются чтения ответа в буфер порта после отправки запроса:

```
while (!comPort.atEnd());
for (i=0;i<100000000; ++i);
comPort.readBlock(answer_st, 18);
```

Пустой цикл for добавлен, чтобы ответ был полностью получен. Массив

```
char answer_st[19]
```

служит для чтения буфера ввода порта, и сохраняемые им команды содержат все, что было отправлено:

## Хобби-электроникс 2. Умный дом.

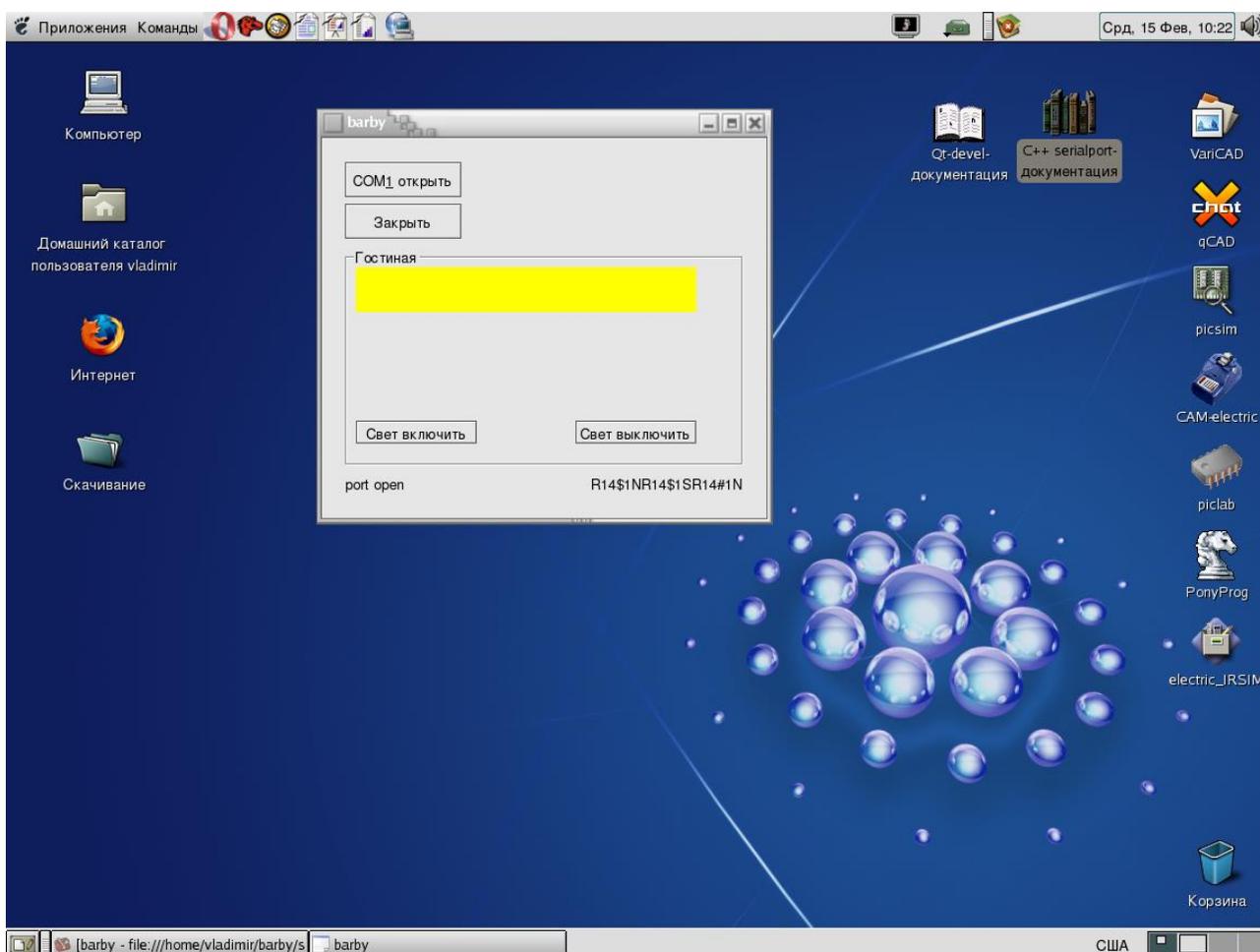


Рис.157

В правом нижнем углу отображается содержание этого массива. Поэтому в проверке его содержимого я проверяю только последний символ:

```
if (answer_st[17] == 'N')
```

Возможно, не лучшим образом, но механизм работает. По команде выключения света...

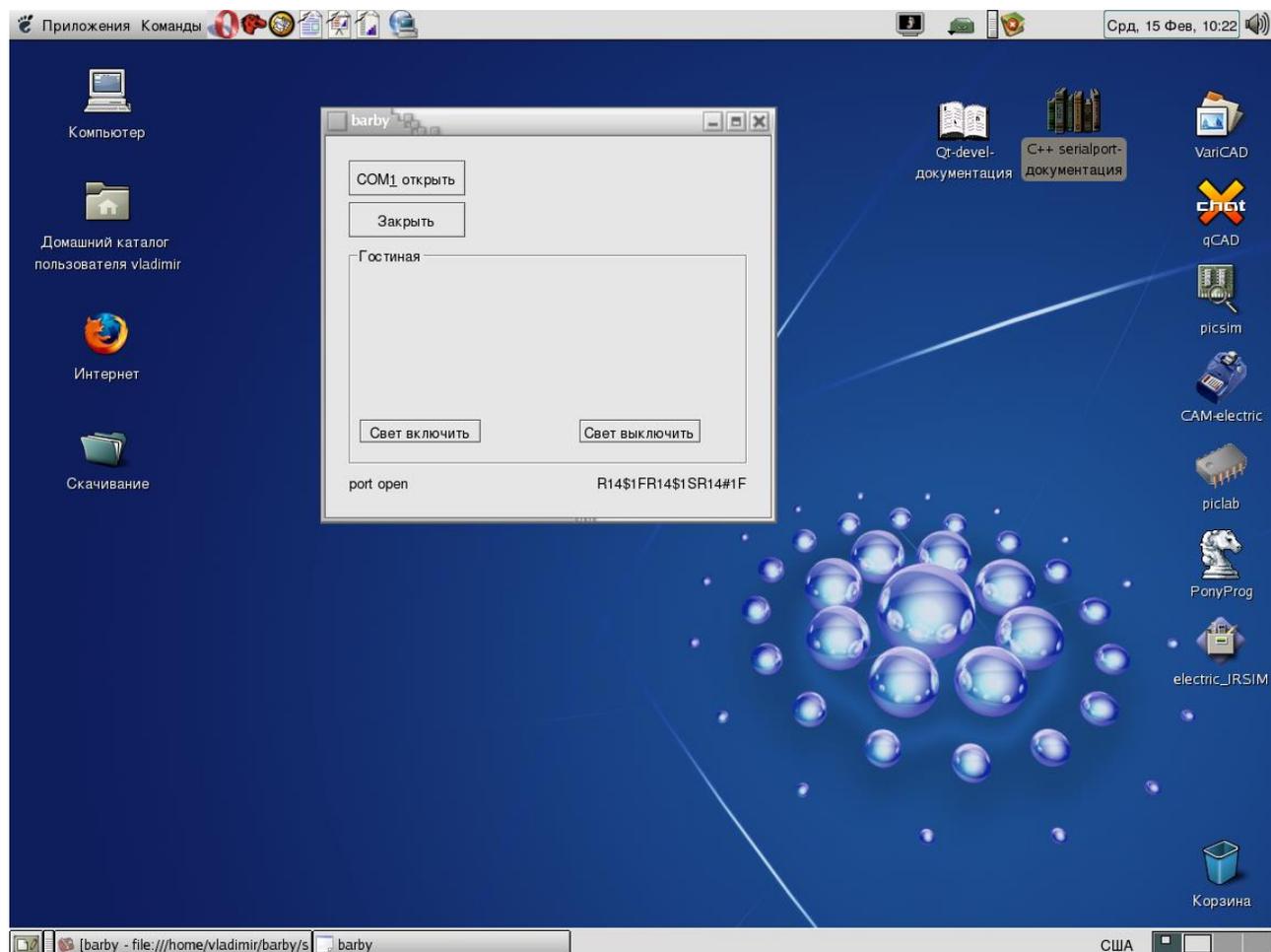


Рис.158

свет на картинке выключается. На макете, естественно, по команде включения света загорается индикатор, и гаснет по команде выключения света.

Запланированная часть работы выполнена.

### ***Последние замечания***

Как аппаратная, так и программная разработка, мною выполнена в макетном варианте. Я говорил, что не намерен создавать промышленный вариант – моя задача показать, что есть такое интересное занятие, как придумывать и создавать, используя возможности компьютера, интересные устройства. По этой причине компьютер выбран в качестве основного управляющего устройства. Но, если кому-то захочется, не ограничиваясь экспериментами, использовать систему в своей комнате или квартире, то я бы посоветовал рассмотреть возможность замены компьютера, как центрального управляющего устройства, на автономное управляющее устройство, выполненное на базе, например, того же микроконтроллера PIC16F628A. Его программная память позволяет разместить до 2 Кбайт управляющей программы, что вполне может оказаться достаточным для управления модулями в комнате, или даже квартире. Можно, если памяти окажется недостаточно, использовать другой микроконтроллер, или внешнюю память программы.

## Хобби-электроникс 2. Умный дом.

Программу для создания основной программы можно написать в виде близком к профессиональному – очень интересная задача. И очень интересно придумывать новые модули, которые были бы полезны, будучи включены в состав системы.

В книге я почти не затронул вопросы наблюдения за системой после ее сборки. В процессе эксплуатации появляются как сбои в работе, так и неполадки. Выявление этих неприятных моментов – предмет особого разговора. Возможно, сюжет еще одной книги?

### Часть 3. Справки и конспекты

Как и обещал, в эту часть я собираюсь включить свои конспекты и некоторые справочные материалы. Работая с микроконтроллером PIC16F628A, я, подобно тем, для кого написал этот «дневник», начинал «с нуля». Я отыскивал программу для программирования контроллера, которую нашел на сайте Microchip. Я отыскивал схему программатора, не дорогую, и такую, которую можно быстро собрать.

Многие из описаний, в частности описание самого контроллера, можно в переводе на русский язык скачать с сайта российского представительства производителя микросхемы. Другие описания я нашел только в англоязычной версии. Те, что мне казались интересными для других любителей электроники, я законспектировал. Их я и включу в третью часть.

В первую очередь это описание программы gpsim, о которой я говорил во второй части. Ее описание я привожу в виде близком к оригиналу, что скажется на нумерации страниц в первую очередь. Я сохраню эту нумерацию с тем, чтобы была возможность распечатать это руководство в отдельном виде.

Кроме того, я обещал привести схемы датчиков движения. Два найденных в Интернете варианта я привожу. Один из них в виде моего конспекта, второй в оригинале.

Я привожу таблицу основных команд микроконтроллера PIC16F628A, скорее для полноты.

***gpsim***

Т. Скотт Даттало

05 ИЮНЯ 2005

## Содержание

|          |                                     |           |
|----------|-------------------------------------|-----------|
| <b>1</b> | <b>gpsim – Обзор</b>                | <b>6</b>  |
| 1.1      | Создание исполняемого файла         | 6         |
|          | Задание деталей - ./configure опции | 6         |
|          | RPMs                                | 7         |
|          | Windows                             | 7         |
| 1.2      | Запуск                              | 7         |
| 1.3      | Требования                          | 9         |
| <b>2</b> | <b>Интерфейс командной строки</b>   | <b>10</b> |
| 2.1      | attach                              | 11        |
| 2.2      | breack                              | 11        |
| 2.3      | clear                               | 14        |
| 2.4      | disassemble                         | 14        |
| 2.5      | dump                                | 15        |
| 2.6      | echo                                | 15        |
| 2.7      | frequency                           | 15        |
| 2.8      | help                                | 15        |
| 2.9      | icd                                 | 16        |
| 2.10     | list                                | 16        |
| 2.11     | load                                | 16        |
| 2.12     | macros                              | 17        |
| 2.13     | module                              | 19        |
| 2.14     | node                                | 20        |
| 2.15     | processor                           | 21        |
| 2.16     | quit                                | 21        |
| 2.17     | run                                 | 21        |
| 2.18     | step                                | 21        |

|          |   |           |
|----------|---|-----------|
| 2.19     | symbol  | 22        |
| 2.20     | stimulus  | 22        |
| 2.21     | stopwatch                                       | 23        |
| 2.22     | trace   | 23        |
| 2.23     | version   | 24        |
| 2.24     | x   | 24        |
| <b>3</b> | <b>Графический интерфейс пользователя (GUI)</b> | <b>25</b> |
| 3.1      | Основное окно                                   | 25        |
| 3.1.1    | Меню  | 25        |
| 3.1.2    | Клавиши   | 25        |
| 3.1.3    | Режим симуляции                                 | 26        |
| 3.2      | Проводники исходного кода                       | 26        |
| 3.2.1    | .asm Проводник                                  | 26        |
| 3.2.2    | Обозреватель Opcode - .obj Проводник            | 27        |
| 3.3      | Обозреватели регистров                          | 28        |
| 3.4      | Обозреватель символов                           | 29        |
| 3.5      | Обозреватель наблюдения                         | 30        |
| 3.6      | Обозреватель стека                              | 31        |
| 3.7      | Макет   | 31        |
| 3.8      | Обозреватель трассировки                        | 32        |
| 3.9      | Обозреватель профиля                            | 32        |
| 3.10     | Остановка наблюдения                            | 33        |
| <b>4</b> | <b>Контроль выполнения: Точки Останова</b>      | <b>34</b> |
| 4.1      | Выполнение точек останова                       | 34        |
| 4.1.1    | Точки останова неправильных команд              | 34        |
| 4.2      | Точки останова регистра                         | 35        |
| 4.3      | Точки останова цикла                            | 35        |
| <b>5</b> | <b>Трассировка: Что случилось?</b>              | <b>36</b> |
| <b>6</b> | <b>Симуляция реального мира: Стимулы</b>        | <b>38</b> |
| 6.1      | Как это работает                                | 38        |
| 6.1.1    | Соединение между стимулами                      | 39        |
| 6.2      | Выводы ввода-вывода (I/O)                       | 39        |
| 6.3      | Асинхронные стимулы                             | 40        |
| 6.3.1    | Аналоговые асинхронные стимулы                  | 40        |

|           |                               |           |
|-----------|-------------------------------|-----------|
| <b>7</b>  | <b>Модули</b>                 | <b>42</b> |
| 7.1       | gpsim Модули                  | 43        |
| 7.2       | Написание новых модулей       | 43        |
| <b>8</b>  | <b>Символьная отладка</b>     | <b>43</b> |
| <b>9</b>  | <b>Макросы</b>                | <b>43</b> |
| <b>10</b> | <b>HEX-файлы</b>              | <b>43</b> |
| <b>11</b> | <b>ICD</b>                    | <b>44</b> |
| <b>12</b> | <b>Теория операций</b>        | <b>46</b> |
| 12.1      | Задний план                   | 46        |
| 12.2      | Инструкции                    | 46        |
| 12.3      | Основные регистры файла       | 47        |
| 12.4      | Специальные файловые регистры | 47        |
| 12.5      | Пример инструкции             | 48        |
| 12.6      | Трассировка                   | 49        |
| 12.7      | Точки останова                | 50        |

gpsim – это программный симулятор, с полной поддержкой всех свойств, для PIC микроконтроллеров Microchip, распространяемый под GNU General Public License (смотрите раздел COPYING).

gpsim был разработан со всей возможной точностью. Точностью, включающей весь PIC – от ядра до выводов ввода-вывода, и включающей ВСЮ внутреннюю периферию. Таким образом, возможно открыть стимулы и связать их с выводами ввода-вывода, и проверить PIC, тот самый PIC, и тем же манером, как вы сделали бы в реальном мире.

gpsim был разработан, чтобы быть столь быстрым, сколь возможно. Симуляция реального времени со скоростью в 20 МГц возможна.

gpsim может управляться либо через графический пользовательский интерфейс (GUI), через интерфейс командной строки (CLI), либо с помощью процесса управления. Типичные средства отладки, подобно точкам останова, пошаговому режиму, дисассемблированию, проверке памяти и модификации, и т.д., все поддерживаются. Вдобавок комплексные отладочные средства, как трассировка реального времени, заявки, условные остановки и подключаемые модули для нескольких наименований, также поддерживаются.

## Часть 1

### gpsim – Обзор

Если вы не намерены брести через все детали, то этот раздел поможет вам получить необходимое для начала работы. Файлы INSTALL и README предоставят больше последней информации, чем этот документ, поэтому, пожалуйста, обратитесь сначала к ним.

#### 1.1 Создание исполняемого файла

Исполняемый файл `gpsim` создается на манер других программ, представленных исходными текстами:

| <i>Команда</i>                            | <i>Описание</i>                                 |
|---|---|
| <code>tar -xvzf gpsim-x.y.z.tar.gz</code> | разархивирует сжатый tar файл                   |
| <code>./configure</code>                  | создает “makefile” уникальный для вашей системы |
| <code>make</code>                         | компилирует <code>gpsim</code>                  |
| <code>make install</code>                 | устанавливает <code>gpsim</code>                |

Последний шаг требует привилегий `root`.

##### 1.1.1 Задание деталей - `./configure` опции

###### **gui-less**

Конфигурация по умолчанию будет поддерживать `gui` (графический пользовательский интерфейс). `cli` (интерфейс командной строки) также доступен, поскольку много людей предпочитают пользоваться им. Эти стойкие души могут построить интерфейс только командной строки конфигурируя `gpsim`:

```
./configure --disable-gui
```

###### **debugging**

Если вам угодно отладить `gpdim`, тогда вы, возможно, воспользуетесь `gdb`. Следовательно, вы захотите отвергнуть общие библиотеки:

```
./configure --disable-shared
```

Этим создается единственный, большой исполняемый файл с символической информацией.

### 1.1.2 RPMs

gpsim также распространяется в форме RPM. В текущих версиях два RPM – gpsim-devel и gpsim. Оба должны быть установлены. Есть так же RPM для исходных кодов. Он может использоваться для построения двоичного RPM уникального для вашей системы. Обратите, пожалуйста, внимание на последние INSTALL и README файлы для получения последней информации.

### 1.1.3 Windows

gpsim работает и на Windows. Борут Разем смонтировал gpsim Windows web-сайт:

<http://gpsim.sourveforge.net/gpsimWin32/gpsimWin32.html>

Вы можете найти детальные инструкции по инсталляции gpsim и ее зависимостям. Внешний вид можно найти:

<http://gpsim.sourveforge.net/snap.php>

## 1.2 Запуск

Исполняемый файл, открытый как описано выше, называется: gpsim. Следующие опции командной строки могут быть установлены, когда gpsim запускается.

```
gpsim [-?] [-p <device> [<hex_file>]] [-c <stc_file>]
-p, --processor=<processor name> processor (e.g. -pp16c84 for the 'c84)
-c, --command=STRING startup command file
-s, --symbol=STRING .cod symbol file
-L, --sourcepath=STRING colon separated list of directories to
search.
```

|                     |  |
|---------------------|--|
| -v, --version       | gpsim version  |
| -E, --echo          | Echo lines from a command file to the console.   |
| -i, --cli           | command line mode only   |
| -S, --source=STRING | 'enable' or 'disable' the loading of source code. Default is 'enable'. Useful for running faster regression tests.   |
| -d, --icd=STRING    | use ICD (e.g. -d /dev/ttyS0).  |
| -D, --define=STRING | define symbol with value that is added to the gpsim symbol table. Define any number of symbols.  |
| -e, --exit=STRING   | Causes gpsim to auto exit on a condition. Specifying onbreak will cause gpsim to exit when the simulation halts, but not until after the current command script completes. |

## Examples:

```
gpsim -s myprog.cod      <-- loads a symbol file
gpsim -p p16f877 myprog.hex <-- select processor and load hex
gpsim -c myscript.stc   <-- loads a script
```

## Help options:

```
-?, --help      Show this help message
--usage        Display brief usage message
```

Обычно gpsim запускается подобным образом:

```
[My-Computer]$ gpsim -s mypic-program.cod
```

([My-Computer]\$ текст – пример типичной командной строки bash – вы только вводите текст после этой подсказки). Команда загружает .cod файл (mypic-program.cod), созданный с помощью gputils. Под Windows gpsim также может открыть окно DOS или CygWin сессию bash (терминальный интерпретатор командной строки), и запустить gpsim из него.

### 1.3 Требования

gpsim была разработана под Linux. Она должна встраиваться и запускаться прекрасным образом под всеми популярными дистрибутивами Linux, подобными Redhat. gpsim также был портирован в MAC, MicroSoft Windows, Solaris и BSD. Два пакета gpsim требуются постольку, поскольку не все дистрибутивы Linux имеют readline и qtk (инструментальные средства gimp). Скрипт ./configure подскажет, если эти пакеты не установлены на вашей системе, или устарели.

Необходимо удовлетворить минимальные требования по оборудованию для запуска gpsim. Но чем оно быстрее, тем лучше!

Утилиты grutils и gnuric также очень полезны. gpsim может использовать сразу hex-файлы, но, если вы хотите использовать символьную отладку, тогда вам потребуется .cod-файл, который производит пакет grutils. cod-файл имеет тот же формат, что производится MPASM. (.cod-файл – это символьный файл созданный ByteCraft и используемый Microchip, а MPASM – ассемблер Microchip).

## Часть 2

### Интерфейс командной строки

Интерфейс командной строки, в известной мере, первичен. Таблица ниже суммирует допускаемые команды. Краткое описание этих команд может быть выведено при вводе `help` в командной строке.

| <i>Команда</i>           | <i>Суммарно</i>                                       |
|--------------------------|---|
| <code>attach</code>      | присоединяет стимул к узлу                            |
| <code>break</code>       | устанавливает точку останова                          |
| <code>bus</code>         | добавляет или показывает базовый узел                 |
| <code>clear</code>       | удаляет точку останова                                |
| <code>disassemble</code> | дисассемблирует текущий <code>cru</code>              |
| <code>dump</code>        | отображает RAM или EEPROM                             |
| <code>frequency</code>   | устанавливает частоту процессора                      |
| <code>help</code>        | введите <code>help</code> для подсказки               |
| <code>icd</code>         | поддержка отладки в схеме                             |
| <code>list</code>        | отображает исходные и <code>list</code> файлы         |
| <code>load</code>        | загружает либо <code>hex</code> , либо командный файл |
| <code>log</code>         | Log/record события в файл                             |
| <code>node</code>        | добавляет или отображает узлы стимула                 |
| <code>module</code>      | Выбор&Отображение модулей                             |
| <code>processor</code>   | Добавить/Список процессоров                           |
| <code>quit</code>        | покинуть <code>gpsim</code>                           |
| <code>reset</code>       | сбросить все или часть симуляций                      |
| <code>run</code>         | выполнить <code>pic</code> программу                  |
| <code>set</code>         | отображение и управление флагами <code>gpsim</code>   |
| <code>step</code>        | выполнить одну или более инструкций                   |

| <i>Команда</i> | <i>Суммарно</i>                     |
|----------------|-------------------------------------|
| stimulus       | открыть стимул                      |
| stopwatch      | измерение времени между событиями   |
| symbol         | Добавить/Список символов            |
| trace          | дамп последовательности трассировки |
| version        | отображает версию gpsim             |
| x              | проверка и/или модификация памяти   |

Встроенный help предоставляет дополнительную online информацию.

## 2.1 attach

```
attach node1 stimulus1 [ stimulus2 stimulus_N]
```

attach используется для определения связи между стимулами и узлами. Хотя бы один узел и один стимул должны быть определены. Если определено больше стимулов, тогда они все будут связаны с узлом, например:

```
gpsim> node n1 # Определяет новый узел
gpsim> attach n1 porta4 portb0 # Связывает два вывода I/O с узлом
gpsim> node # Отображает новый "список
связей"
```

## 2.2 break

Команда break используется для установки и проверки точек останова. Новым точкам останова присваиваются уникальные номера. Эти номера могут быть использованы при запросах или удалении точек останова. Точки останова останавливают симуляцию, когда ассоциированные с ними условия выполняются (становятся истинными). Точки останова игнорируются в пошаговом режиме.

### Проверка точек останова

`break [bp_number]` (номер точки останова)

Точки останова могут проверяться командой `break` без дополнительных опций. Определенная точка останова может быть проверена заданием номера.

### Память программы/Выполнения останова (Program Memory/Execution breaks)

Ощие точки останова – это точки останова выполнения. Последние останавливают выполнение в тот момент, когда программный счетчик достигает адреса, на котором установлена точка останова. Синтаксис следующий:

`break e | r | w ADDRESS [expr (выражение)]`

Симуляция останавливается, когда адрес выполнен, прочитан или записан. `ADDRESS` может быть символьным или числовым. Если дополнительное выражение определено, тогда оно должно стать истинным, до останова симуляции. Опции чтения и записи относятся только к тем процессорам, которые могут работать с их собственной программной памятью.

### Точки останова регистровой области памяти (Register Memory)

`gpsim` может также ассоциировать точки останова с доступом к регистрам. Это полезно для выявления ошибок, которые связаны с RAM. Т.е. вы можете сказать что-нибудь вроде “остановить выполнение, когда бит 4й регистра 42 сбрасывается”. Синтаксис команды:

`break r | w REGISTER [expr]`

Симуляция останавливается, когда `REGISTER` читается или записывается и дополнительное выражение становится истинным. Поддерживаются два стиля выражений. Один обращается только к выражениям `REGISTER`, другой проверяется комплексно. Пример ниже иллюстрирует разницу. Вот пример останова при записи в регистр. Он остановит симуляцию, если

любое значение записывается в регистр для переменной, названной *temp1*.

```
break w temp1
```

Вот условная запись для случая только некоторых значений:

```
break w temp1==0x22
```

Вот условие, затрагивающее определенные биты:

```
break w temp1 & 0b11110000 == 0b11000000
```

Остановка происходит только в случае, когда шестнадцатеричное число “С” записывается в старшие биты переменной *temp1*.

### Булевы выражения

Иногда необходимо определить вспомогательное условие с точкой останова. Например, есть временная переменная, которая обобщается через код. Вы можете захотеть прервать запись в эту переменную только, когда выполняется определенная подпрограмма. Например, следующая точка останова переключается, когда *temp1* записывается и пока счетчик программы находится между метками *func\_start* и *func\_end*:

```
break w temp1 (pc >= func_start && pc < func_end)
```

**ПРЕДУПРЕЖДЕНИЕ:** Используйте этот тип точки останова, если вы подозреваете прерывание программы через запись в переменную. Другой ситуацией может оказаться такая, где вы хотите прервать запись в переменную только, если какая-то другая переменная имеет то же значение:

```
break w temp1 (CurTask & 0x0f != 0b101)
```

Если программа записывает в переменную *temp1*, тогда симуляция останавливается, если младшие биты *CurTask* не эквивалентны 5.

## Точки останова атрибута

`gpsim` также поддерживает концепцию точек останова атрибута. Атрибуты – параметры, которые `gpsim` и ее модули предоставляют пользовательскому интерфейсу. Например, остановка наблюдения симулятора предоставляет атрибуты, которые поддерживают точки останова. Это свойство предназначено в основном для создателей модулей, для поддержки механизма, который позволяет пользователю управлять модулем.

### 2.3 clear

`clear bp_number`

Команда `clear` используется для удаления точек останова. Номер точки останова должен быть задан. Команда `break` без аргументов отображает все в настоящий момент определенные точки останова. Это может использоваться для уточнения номера точки останова. При удалении, точка останова удаляется.

### 2.4 disassemble

`disassemble [begin:end] | [length]`

Команда `disassemble` раскодирует `opcodes` программной памяти в их стандартную мнемонику. Без опций команда дисассемблирует инструкции вокруг текущего значения счетчика программы:

```
gpsim> disassemble
current pc = 0x1c
  0012 2a03 incf reg3,f,0
  0014 0004 clrwdt
  0016 5000 movf reg,w,0
  0018 1001 iorwf reg1,w,0
  001a 1002 iorwf reg2,w,0
===> 001c 1003 iorwf reg3,w,0
      001e e1f4 bnz $-0x16 ; (0x8)
      0020 d7ff bra $-0x0 ; (0x00020)
```

С единственной числовой опцией команда дисассемблирования будет

## 2.5 dump

`dump [r | e]`

`dump r` or `dump` with no options will display all of the file registers and special function registers.

`dump e` will display the contents of eeprom (if the pic being simulated contains any)

Обратитесь к команде 'x' для проверки и модификации конкретных регистров.

## 2.6 echo

Команда `echo` используется подобно заявлению печати в файлах конфигурации. Она только позволяет вам отобразить информацию о вашем файле конфигурации.

## 2.7 frequency

Эта команда устанавливает тактовую частоту. По умолчанию `gpsim` использует частоту 4 МГц. Частота используется для вычисления времени в секундах. Используйте эту команду для задания этого значения. Если не предоставлено никакого значения эта команда печатает текущее время. Заметьте, что PIC имеют инструкцию `clock`, которая продвигает внешнее время. Это значение внешнего времени.

## 2.8 help

Само по себе `help` будет отображать все команды вместе с краткими описаниями, как они работают. Команда 'help' предоставляет более богатую online помощь. Команда может также отображать информацию об атрибутах.

## 2.9 icd

```
icd [open <port>]
```

Команда *open* используется для включения режима ICD и определяет последовательный порт, в который включено ICD (т.е. “*icd open /dev/ttyS0*”). Без опций (и после включения *icd*) она будет печатать некоторую информацию об ICD.

## 2.10 list

```
list [[s | l] [*pc][line_number1 [,line_number2]]]
```

Отображает содержимое исходного файла и файла листинга. Без опций команда будет использовать последние определенные опции.

*list s* будет отображать строки в исходном (или *.asm*) файле.

*list l* будет отображать линии в *.lst* файле.

*list \*pc* будет отображать либо *.asm*, либо *.lst* линии вокруг *pc* (счетчика программы).

Команда *list* позволяет вам видеть исходный код при отладке.

## 2.11 load

Команда *load* используется для загрузки *hex*-файла, файла конфигурации или *.cod* файла. *Hex*-файл обычно используется для программирования физической части. Следовательно, он не поддерживает символьную информацию. *.cod* файл, с другой стороны, поддерживает символьную информацию. Единственная причина для использования *hex*-файла, полное отсутствие *.cod* файла.

Синтаксис загрузки файла исходного кода:

```
load [processortype] file
```

*gpsim* автоматически определит, файл *hex* или *.cod* формата. Опция *processortype* позволит переопределить процессор, обозначенный в *.cod* файле.

Файл конфигурации – это скрипт, содержащий `grsim` команды. Крайне полезно создавать окружение отладки, которое будет использовано повторно.

## 2.12 macros

Макросы определяются подобно:

```
name macro [arg1, arg2, ..., argN]
    macro body (тело макроса)
endm
```

И они вызываются:

```
name param1, param2, ..., paramN
```

Макросы – способ коллекционировать некоторые параметризованные команды в одной короткой команде. Первая строка определения макроса определяет имя макроса и дополнительные аргументы. *name* используется для вызова макроса. Аргументы – текстовая строка удерживающая место. Когда макрос вызывается, параметры соотносятся с аргументами. Т.е. *param1* в вызове может быть ассоциирован с *arg1* в определении. Параметры замещаются аргументами в теле макроса. В следующем примере переменная или атрибут названный *mac\_flags* будет использован в выражении. Аргументы *add* и *mask* появляются в теле макроса и предоставляют параметризованный путь для изменения этого выражения.

```
mac_exp macro add, mask
    mac_flags = (mac_hflag+add) & mask
endm
```

Заметьте, что идентификация произвольна. Макрос вызывается:

```
mac_exp 1, 0b00001111 # наращивает младшие биты
```

Параметр *add* заменен числом 1, тогда как *mask* замещена бинарным

числом *0b00001111*. Вызов возвращает в *gpsim* команду:

```
mac_flags = (mac_flags+1) & 0b00001111
```

### Вложенные макросы

Тело макроса может состоять из любых команд *gpsim*. Частный интерес представляет вызов макроса внутри другого макроса. Вот другой макрос, который вызывает макрос, определенный выше:

```
# Пример вложенного макроса
mac1 macro p1, p2
    run
    mac_exp p1, p2
endm
```

И он может использоваться подобным образом:

```
mac1 1,      0b00001111 # проверяет младшие биты
mac1 (1<<4), 0b00001111 # проверяет старшие биты
```

Первый вызов начинает симуляцию с выполнения команды *run*. Когда точка останова обнаруживается, управление возвращается командной линии и вызывается макрос *mac\_exp*.

### Отображение определенных макросов

Все определенные к настоящему времени макросы могут отображаться вводом команды *macro* без имени или аргументов:

```
gpsim> macro
mac1 macro p1, p2
    run
    mac_exp p1, p2
endm
mac_exp macro add mask
    mac_flags = (mac_flags+add) & mask
endm
```

## 2.13 module

Команда *module* используется для загрузки и запросов к внешним модулям. Модуль – специальная часть программы, которая может расширять *gpsim* некоторым образом. Индикаторы и переключатели – примеры модулей. Библиотека модулей – коллекция модулей.

### Загрузка библиотек модулей

```
module lib lib_name
```

Опция *lib* используется для загрузки библиотеки модуля. Библиотеки модуля – зависимые от системы общие библиотеки. Т.е. на Windows – это DLL, а в UNIX – общие библиотеки. Это означает, что, либо библиотеки должны располагаться в директории, где ОС знает, что расположены библиотеки, или следует задать полный путь для *lib\_name*. *gpsim* предоставляет библиотеку модулей с несколькими модулями:

```
gpsim> module lib libgpsim_modules
```

Отображение доступных модулей

```
module list
```

Опция *list* отобразит все модули, которые могут быть загружены. Вот пример встроенных *gpsim* в модулей.

```
gpsim> module list
Module Libraries libgpsim_modules.so
  binary_indicator
  pullup
  pulldown
  usart
  parallel_interface
  switch
  and2
  or2
```

```
xor2  
not  
led_7segments  
led  
PAL_video  
Encoder
```

### Загрузка обозначенного модуля

```
module load module_type [module_name]
```

После того, как библиотека была загружена, обозначенный модуль может быть проиллюстрирован. `module_type` – это то, что отображено командой `module list`. В качестве опции может выступать имя модуля. Вот пример:

```
gpsim> module load led D1
```

### Отображение загруженных модулей Запрос модулей

#### 2.14 node

```
node [new_node1 new_node2 ...]
```

Если не обозначено `new_node`, то все узлы, которые были определены, отобразятся. Если `new_node` обозначено, тогда он будет добавлен в список узлов. Посмотрите на команды 'attach' и 'stimulus', чтобы увидеть, как стимул добавляет узлы.

Примеры:

```
node // отображает список узлов  
node n1 n2 n3 // открывает и добавляет 3 новых узла в  
список
```

## 2.15 processor

`processor [new_processor_type [new_processor_name]] | [list] | [dump]`

Команда *processor* используется либо для определения нового процессора, либо для запроса одного из уже определенных. Обычно нет необходимости специально определять процессор, поскольку символьный файл уже содержит эту информацию. Но есть два исключения – символьная информация не доступна, или вы хотите переопределить процессор, обозначенный в файле. (Посмотрите команду `load` для понимания, как процессор может быть переопределен).

Чтобы посмотреть список процессоров, поддерживаемых *gpsim*, введите '*processor list*'. Для отображения состояния вводов-выводов процессора – '*processor pins*'. Вот, например, это отобразит номера выводов и их текущее состояние.

Примеры:

```
processor                // Отображает уже определенные
процессоры
processor list           // Отображает список поддерживаемых
процессор pins         // Отображает цоколевку и состояние выводов
processor p16cr84 fred  // Создает новый процессор
processor p16c74 wilma  // и другой
processor p16c65        // Создает процессор без имени
```

## 2.16 quit

Покидаем *gpsim*.

## 2.17 run

Начинает (или продолжает) симуляцию. Симуляция будет продолжена, пока следующая точка останова не обнаружится.

## 2.18 step

`step [over | n]`

нет аргументов: шаг на одну инструкцию  
 числовой аргумент: шаг на количество (число) инструкций  
 аргумент 'over': шаг через следующую инструкцию

## 2.19 symbol

symbol [symbol\_name[symbol\_type value]]

Команда `symbol` используется для запроса и определения символов. Если нет опций, отобразится вся таблица символов. Создание определенных пользователем символов ограничено в настоящий момент (сверьтесь с 'help' для уточнения положения дел).

## 2.20 stimulus

stimulus [{type} options]

Команда `stimulus` создает сигнал, который может быть привязан к узлу или атрибуту. Если нет опций, тогда отображаются все в настоящий момент определенные стимулы. Заметьте, что в большинстве случаев легче создать файл стимула, чтобы не вводить команду вручную.

| <i>состояние_при_инициализации</i> | <i>состояние при старте и выполнении</i> |
|------------------------------------|--|
| start_cycle                        | cycle симуляции, когда начинается стимул |
| period                             | период стимула                           |
| name                               | определенное имя стимула                 |

Вот пример стимула, который будет генерировать два импульса и повторение этого через 1000 циклов.

```
stimulus asynchronous_stimulus # Состояние инициализации AND
                                # состояние стимула при выполнении
initial_state 0
start_cycle 0 # асинхронный стимул будет выполнен в 'period' циклов.
              # Удалите эту линию, если не хотите выполнения.
```

```
period 1000
{ 100, 1,
  200, 0,
  300, 1,
  400, 0
} # Даем имя стимулу: name two_pulse_repeat
end
```

Стимул может быть запрошен введением его имени в командной строке:

```
gpsim> two_pulse_repeat
two_pulse_repeat attached to pulse_node
Vth=0V Zth=250 ohms Cth=0 F nodeVoltage= 7.49998e-07V
Driving=0 drivingState=0 drivenState=0 bitState=0
statetes = 5
100 1
200 0
300 1
400 0
1000 0
initial=0
period=1000
start_cycle=0
Next break cycle=100
```

Даже в этом примере, использующем 1 и 0 для данных, можно вместо них применить целые, с плавающей точкой числа, или выражения. Целые полезны для привязки стимулов к атрибутам. Выражения полезны для абстрагирования данных. Загляните в Часть 6, где больше внимания уделяется обсуждению, и приведено больше примеров стимулов.

## 2.21 stopwatch

## 2.22 trace

```
trace [dump_amount]
```

*trace* будет распечатывать самые последние “dump\_amount” трассировки.

Если нет специфицированных `dump_amount`, тогда полный буфер трассировщика будет отображен.

## 2.23 version

`version`

Отображает версию `gpsim`. Заметьте, что эта команда будет, возможно, получена через атрибут с тем же (или подобным) именем.

## 2.24 x

`x [file_register][new_value]`

опции:

`file_register` – место `ram`, которое будет проверено или модифицировано

`new-value` – новое значение записанное в `file_register`.

Если опции не заданы, все содержимое `file_register`'ов будет отображено (`dump`).

## Графический пользовательский интерфейс

gpsim предоставляет также графический пользовательский интерфейс, который несколько облегчает жизнь по сравнению с cli. Возможно открыть окно для обзора всех деталей вашего окружения отладки. Чтобы получить максимум от вашей сессии отладки вам следует откомпилировать ваш код с помощью gprasm (gpic ассемблер) и использовать символьные .cod файлы, им производимые.

### 3.1 Главное окно

#### 3.1.1 Меню

|            |                          |
|------------|--------------------------|
| File->Open | .stc или .cod файлы      |
| File->Quit | покидаем gpsim           |
| Windows->* | Открываем/Закрываем окна |

#### 3.1.2 Клавиши

(Так же есть привязка к клавиатуре в исходном окне)

|        |   |
|--------|---|
| Step   | Шаг на одну инструкцию                    |
| Over   | Шаг до перехода pc к следующей инструкции |
| Finish | Выполнение до возврата к адресу           |
| Run    | Постоянное выполнение                     |
| Stop   | Остановка выполнения                      |
| Reset  | Сброс CPU                                 |

### 3.1.3 Режим симуляции

Этим управляется то, как работает gpsim, и как обновляется GUI.

|              |  |
|--------------|--|
| Never        | Не обновлять GUI при симуляции. Это самый быстрый режим. Вы должны остановить симуляцию нажатием Ctrl-C в интерфейсе командной строки. |
| x cycles     | Обновляет GUI каждые x циклов симуляции.   |
| every cycle  | Обновляет GUI в каждом цикле. (вы увидите все, если пополните запасы кофе :-)  |
| x ms animate | Здесь вы можете замедлить симуляцию введением задержек между каждым циклом   |
| realtime     | Это даст возможность gpsim попытаться синхронизировать скорость симуляции с настенными часами  |

## 3.2 Проводники исходного кода

### 3.2.1 .asm Проводник

Когда загружен .cod файл с исходным кодом, что-нибудь появится в этом окне.

Есть пространство слева от текста, где символы представляют программный счетчик (pc), точки останова, и т.д. Двойной клик в этом пространстве переключает точки останова. Вы можете перетаскивать эти символы выше или ниже, чтобы переместить их и изменить PC и переместить точку останова. Клик правой клавишей на тексте вызывает выпадающее меню с шестью пунктами (слово “здесь” в некоторых пунктах меню означает линию в тексте, на которую был установлен маркер мышки, когда правая клавиша была нажата):

| Пункты меню        | Описание  |
|--------------------|---|
| Find PC            | Этот пункт будет искать PC и менять таблицу страниц и прокручивать обзор текста до текущего (значения) PC |
| Run here           | Этим установится точка останова “здесь” и стартует симуляция, пока не попадет на точку останова           |
| Move PC here       | Этим просто изменяется PC на адрес, который имеет “эта” линия текста.                                     |
| Breajpoint here    | Устанавливает точку останова “здесь”.   |
| Profile start here | Устанавливает старт маркера для профилирования  |

правила здесь.

**Profile stop here** Устанавливает стоп маркера. (Посмотрите раздел по окну профилирования)

**Select symbol** Этот пункт меню доступен только, когда некоторый текст

выбран в текстовом поле. Что он делает, так это ищет в списке символов выбранное слово, и если находит, выбирает его в символьном окне. В зависимости от типа символа еще кое-что делается, тоже самое, что при выборе символа в символьном окне:

- ◆ Если это адрес, тогда opcode и проводник отображают этот адрес.
- ◆ Если это регистр, проводник регистров отмечает ячейку.
- ◆ Если это константа, адрес, регистр или порт, он отмечается в символьном окне.

**Find text** Этим открывается диалог поиска. Каждое нажатие клавиши “Find” текущая страница блокнота находится и текст на этой странице используется.

**Settings** Диалог, в котором вы можете изменить шрифт.  
**Controls** Подменю, содержащее команды симуляции. (также привязка клавиатуры (рекомендуется), или в основном окне).

Вот привязка к клавиатуре:

| Клавиша | Команда                         |
|---------|---------------------------------|
| s,S,F7  | Пошаговое выполнение            |
| o,O,F8  | Перешагнуть инструкцию          |
| r,R,F9  | Постоянное выполнение           |
| Escape  | Остановить симуляцию            |
| f,F     | Выполнять до возвращения адреса |

### 3.2.2 Проводник Opcode - .obj проводник

Это окно имеет две таблицы. Одна с ячейкой памяти, адресом, шестнадцатеричным значением и декодированной инструкцией на линии, другая с программной памятью, отображенной по шестнадцать ячеек в строке

и конфигурируемым ASCII столбцом.

### **С таблицей ассемблера вы можете:**

- Переключать точки останова двойным кликом на линии.
- Использовать те же клавишные команды, что и в проводнике исходного кода.
- Правым кликом мышки получить меню, где можно изменить шрифт.

### **Таблица Opcode.**

Здесь программная память организована в колонки по шестнадцать ячеек в колонке, а строк столько, чтобы поместить всю память.

Семнадцатая колонка отведена под ASCII представление памяти. Вы можете конфигурировать эту колонку для использования в трех разных режимах:

- Один байт на ячейку
- Два байта на ячейку, старшие биты впереди
- Два байта на ячейку, младшие биты впереди

Вы можете изменить шрифт, используя меню “Settings” (Установки).

Вы можете задать точки останова на одном или больше (помечая мышкой выбранные ячейки) адресах меню, вызываемом кликом правой клавиши мышки.

## **3.3 Обзоратель регистров**

В вашем распоряжении два похожих окна регистров. Один для RAM, другой для EEPROM данных, когда доступны.

Здесь вы увидите все регистры текущего процессора. Клик на ячейке отображает его имя и значение над всей таблицей. Вы можете изменить значение здесь, или ввести его в ячейку таблицы.

Следующие процедуры могут выполняться с одним регистром или с несколькими. (Выбор нескольких регистров выполняется с удержанием левой клавиши мышки перемещением маркера до нужного места, где клавиша отпускается).

- Установить и сбросить точки останова. Используйте выпадающее меню, вызываемое правой клавишей мышки, где вы можете прочитать, записать, прочитать значение и записать значение точек

останова. Вы можете также “сбросить точки останова”, но, заметьте, что при “сбросе точек останова” каждая точка останова на регистрах будет удалена.

- Установить и сбросить логи регистров. Вы можете прочитать лог, записать, прочитать/записать специфические значения и для выбранных бит с помощью специальной маски. Вы можете выбрать разные имена файлов через “set log filename...” (установить имя лог-файла). По умолчанию это “gpsim.log”. Вы можете выбрать LXT или ASCII формат. LXT может быть прочитан программой gtkwave. По умолчанию установлен ASCII формат.
- Можно копировать ячейки. Копирование доступно перетаскиванием за рамку выбранной ячейки (ячеек).
- Можно заполнить ячейки. Переместите мышь к нижнему правому углу окна выбранной ячейки, и перетащите ее. Содержимое ячейки будет скопировано в другие ячейки.
- Можно наблюдать за ними. Выберите пункт меню “Add Watch” (Добавить наблюдение).

Ячейки имеют разный фон, в зависимости от того, что в них представлено:

- Файл регистр (т.е. RAM): светло голубой
- Регистры специальных функций (т.е. STATUS, TMR0): темно голубой
- Переобозначенный регистр (т.е. INDF, расположенный по адресу 0x80, тот же, что 0x00): серый
- Неправильный регистр: черный. Если все шестнадцать регистров в строке неисправны, тогда строка не показывается.
- Регистр с одной или больше точкой останова: красный. Регистры с лог-файлами тоже красные.

gpsim динамически обновляет регистры в процессе симуляции. Регистры, которые меняют значение между обновлениями окна в процессе симуляции подсвечиваются синим цветом фона.

В меню есть пункт 'settings', с помощью которого вы можете изменить шрифт.

### 3.4 Символьный обозреватель

В этом окне, как это следует из названия, отображаются символы. Все

регистры специальных функций допускают ввод в символьном проводнике. Если вы используете .cod файлы, тогда вы дополнительно имеете файловые регистры (определенные в cblocks), эквиваленты и метки адресов.

Вы можете фильтровать некоторые типы символов, используя клавиши в верхней части окна, и вы можете сортировать строки кликом на клавишах колонок (читающие символы (symbol), тип (type), адрес (address)).

Вы можете добавить символ в окно наблюдения щелчком правой клавиши мышки и выбором пункта “Добавить в окно наблюдения” (Add to watch window). Этим добавится RAM регистр с адресом, эквивалентным значению символа.

Символьный проводник связан с другими окнами. Например, если вы кликнете на символе и:

- Если это адрес, тогда проводники orcode и исходного текста отобразят адрес.
- Если это регистр, проводник регистров выберет ячейку.

### 3.5 Обзоратель наблюдения

Это не только окно вывода, что предполагает название. Вы можете и видеть, и менять данные. Двойной щелчок по биту меняет его значение. Вы добавляете переменные сюда, отмечая в проводнике регистров и выбирая “Добавить в наблюдение” (Add watch) из меню. Выпадающее меню по щелчку правой клавиши мышки имеет следующие пункты:

- Удалить из наблюдения
- Установить значение регистра
- Сбросить точки останова
- Установить остановку по чтению
- Установить остановку по записи
- Установить остановку по чтению значения
- Установить остановку по записи значения
- Колонки

Последнее открывает окно, где вы можете выбрать, какие из следующих данных отображать:

- Точка останова

- Тип
- Имя
- Адрес
- Десятичное
- Шестнадцатеричное
- Вх (биты слова)

Вы можете сортировать список наблюдаемых щелчком по клавиши колонки. Двойной щелчок сортирует список в обратном порядке.

### 3.6 Обзоратель стека

Это окно отображает текущий стек. Выбор ввода создает окно кода, отображающее адрес возврата. Двойной клик устанавливает точку останова на адрес возврата.

### 3.7 Макет (breadboard)

Здесь вы можете создавать/модифицировать и проверять окружение PIC. Выводы отображены, как стрелки. Направление стрелки показывает, это вывод ввода или вывода. Цвет стрелки отображает состояние (зеленый – низкое, красный – высокое).

Вы не можете иллюстрировать поведение процессора отсюда, вы должны сделать это из командной строки, или с помощью .stc файла.

Вы можете создать узел щелчком по клавише “новый узел” (new node). (Узел – часть схемы, к которой вы можете присоединить стимулы). Вы можете видеть список созданных узлов в пункте “узлы” в верхнем левом окне дерева макета. Вы можете создать соединение с узлом, щелкнув по выводу, а, затем, щелкнув по клавише “Присоединить стимул к узлу”. Это выведет список узлов.

Выберите один двойным щелчком на нужном вам.

Если вы кликнете по выводу, что уже присоединен к узлу, тогда вы увидите узел и его соединение в нижней левой части окна. Вы можете отключить стимул, щелкнув по нему, и нажав клавишу “удалить стимул” (remove stimulus).

Когда вам захочется добавить модуль к симуляции, вы вначале должны определить библиотеку, которая содержит требуемый модуль. Щелкните по “Добавить модуль” (add module) и введите имя библиотеки (т.е. “libgpsim\_mofules.so”). Теперь вы можете щелкнуть по клавише “Добавить

модуль”. Выберите нужный модуль из списка двойным щелчком по нему. Введите имя для модуля (оно должно быть уникально и не использоваться ранее). Теперь вы можете позиционировать модуль. Перемещайте указатель мышки туда, куда хотите добавить модуль и щелкните левой клавишей мышки.

Если щелкнуть средней клавишей по выводу, вы увидите, как вывод присоединен. Нажмите “Трассировать все” (trace all) для того, чтобы увидеть все сразу, и “Очистить трассировку” для удаления всего (вы удалите только графическое представление, а не соединения!). Если трассировка не работает, постарайтесь разместить элементы так, чтобы оставалось побольше места вокруг выводов.

Когда вы все сделаете, вы можете сохранить все щелчком по клавише “Сохранить конфигурацию” (save configuration). Позже вы можете загрузить этот файл из командной строки подобно тому, как загружали .cod файл, и файл “mynet.stc”:

```
gpsim -s mycode.cod -c mynet.stc
```

Вы можете загрузить только .stc файл поскольку он не содержит тип процессора и код. Вы можете создать (в редакторе) ваш собственный .stc файл и в этом файле дать команду “load с mynet.stc” после загрузки .cod файла. Вы затем только должны загрузить этот файл (gpsim -c my\_project.stc).

### 3.8 Обзоратель трассировки

Это окно показывает трассировку выполнения инструкций. См. 5.

### 3.9 Обзоратель профиля

В этом окне показывается счетчик для адресов программной памяти. Окно профиля должно быть открыто перед стартом симуляции, поскольку трассировка не установлена по умолчанию.

#### Профиль инструкций

Здесь показано значение времени выполнения каждой инструкции.

#### Профиль ряда инструкций

Здесь вы можете группировать ряд инструкций в один ввод. Выпадающее меню (правый клик мышки) содержит:

|                  |   |
|------------------|---|
| Remove range     | Удалить ряд   |
| Add Range ...    | Открывает диалог, в котором вы можете добавить ряд  |
| Add all labels   | Добавляет все метки кода, как ряд   |
| Snapshot to plot | Открывает окно, содержащее граф данных. Из этого нового окна вы можете также сохранить (postscript) или распечатать их. |

### Профиль регистра

Здесь показано число чтений или записи сделанных симулятором в регистр.

### Профиль программного блока

Здесь вы можете увидеть статистику времени выполнения для выбранного программного блока. Вы отмечаете точки входа и выхода в проводнике исходного текста (profile start/stop). Если блок, который вы хотите измерить, имеет несколько точек входа и/или точек выхода, тогда вы должны отмаркировать каждую точку входа, так же, как (и обязательно) каждую точку выхода. Иначе вы получите неверные данные.

Когда вы сделаете это, `gprsim` будет (с началом симуляции) сохранять времена выполнения этих блоков и вычислять мин/макс/среднее/и т.д. Вы можете также использовать пункт меню “Печатать распределение” (plot distribution) для открытия окна, показывающего гистограмму данных. Из этого нового окна вы можете также сохранить (в postscript) или распечатать ее.

Вы можете также измерить период вызова переключением между точкой входа и выхода. Если также захотите измерить время от сброса (или эквивалентной точки) до первого входа, тогда вы должны также отметить точку входа здесь.

### 3.10 Остановка наблюдения (stopwatch)

Это окно показывает счетчик циклов и переустанавливаемый счетчик. Счетчик циклов тот же, что и в окне регистров. Он в основном считает инструкции. Другой счетчик считает так же, как счетчик циклов, но может быть очищен щелчком по клавише “clear” (или переустановлен вводом числа в окне ввода).

Индикатор вверх/вниз отмечает направление счета.

Прокручиваемое значение определяет пределы счетчика циклов (модульный счетчик). Например, если прокручиваемое значение должно быть 0x42, тогда в какой бы момент переустанавливаемый счетчик не достиг величины 0x42, он провернется к нулю. Если счетчик считает вниз, тогда после установки нуля следующее значение будет 0x41. Если вы не хотите, чтобы так было, установите значение прокрутки несколько больше.

## Контроль над выполнением: Точки останова

Одним из сильных свойств `gpsim` является гибкость, предоставляемая точками останова. Большинство симуляторов ограничено в выполнении типов точек останова.

Если вы хотите установить точки останова на регистры, на выполняемые циклы, неправильное размещение программы, переполнение стека, и т.д., тогда вы, обычно, усиливаете отладчик для вашего кода применением ICE.

### 4.1 Точки останова выполнения

Точки останова выполнения – это то, что остановит выполнение программы, когда адрес программной памяти, на котором она установлена, будет достигнут. Например, если вы отлаживаете PIC среднего класса, и хотите остановить выполнение при прерывании, вы должны задать точку останова в программной памяти по адресу `0x04`:

```
gpsim> break e 4
```

(Более точно, прерывание не должно обнаружиться, поскольку эта точка останова будет вычислена – код обработки должен тоже перейти сюда). Точка останова обнаруживается до выполнения инструкции. Другие симуляторы, как MPLAB останавливаются после того, как инструкция выполнена. Во многих случаях эта разница не существенна. Однако если остановка задана на инструкциях `'goto'` или `'call'`, тогда предпочтительнее сделать остановку до того, как обнаружится ветвление программы. Этим путем легче определить место ветвления.

#### 4.1.1 Точки останова неправильных инструкций

`gpsim` автоматически будет останавливать выполнение программы при попытке выйти за ее пределы. Расположение памяти программы, которое не определено вашим исходным кодом, будет инициализировано, как неправильная инструкция ('Invalid instruction'). Это хорошо видно, когда вы дисассемблируете программу.

## 4.2 Регистровые точки останова

gpsim предоставляет возможность останавливаться при обращении к регистру, как для чтения, так и для записи или в обоих случаях. Более того, возможно останавливаться при специфическом значении читаемом или записываемом в регистр.

## 4.3 Цикловые точки останова

Цикловые точки останова позволяют остановить программу определенной цикловой инструкцией. Представьте, что вы имеете 20 МГц PIC и хотите остановить программу через одну секунду симуляции. Вы должны задать точку останова инструкцией в 5 миллионов циклов. (При 4 циклах на выполнение одной инструкции. В дальнейшем предполагается ввести в gpsim возможность задавать инструкцию точки останова в терминах секунд).

## Часть 5

### Трассировка: Что случилось?

36

Проверка текущего состояния вашей программы временами недостаточна для выявления причины ошибки. Часто полезно выяснить условия, при которых индикатор переходит в текущее состояние. `gpdim` предоставляет историю или трассировку всего, что происходит – хотите вы этого, или нет – чтобы помочь в диагностике этих другими методами плохо анализируемых ошибок.

| <i>Что трассируется</i> | <i>Примечания</i>                   |
|-------------------------|-------------------------------------|
| Счетчик программы (PC)  | Адрес выполнения                    |
| Инструкции              | opcode (код операции)               |
| Чтение регистра         | Значение и положение                |
| Запись регистра         | Значение и положение                |
| Счетчик циклов          | Текущее значение                    |
| Пропущенные инструкции  | Пропущенные адреса                  |
| Статус регистра         | В течение потенциальной модификации |
| Прерывания              |                                     |
| Точки останова          | Тип                                 |
| Сбросы                  | Тип                                 |

Команда 'trace' будет записывать содержимое буфера трассировки. Большой замкнутый буфер (чей размер предустановлен) сохраняет информацию для буфера трассировщика. Когда он заполняется, он возвращается к началу и записывает поверх старой записи. Содержимое буфера трассировщика разбирается по фреймам, где один фрейм соответствует циклу симуляции.

Вот пример вывода трассировки:

```
gpsim> trace
0x00000000000025F6 p18f452 0x001C 0x1003 iorwf reg3,w,0
Read: 0x00 from reg3 (0x0003)
Wrote: 0xE7 to W (0xFE8) was 0xE7
Wrote: 0x18 to status (0xFD8) was 0x18
```

## ЧАСТЬ 5. ТРАССИРОВКА: ЧТО СЛУЧИЛОСЬ ?

37

```
0x00000000000026F7 p18f452 0x001E 0xE1F4 bnz $-0x16 ;  
(0x8)0x00000000000026F8 p18f452 0x0008 0x3E00 incfsz reg,f,0  
Read: 0xE4 from reg (0x0000)  
Wrote: 0xE5 to reg (0x0000) was 0xE4  
0x00000000000026F9 p18f452 0x000A 0xD004 bra $+0xa ; (0x00014 0x000000C  
0x00000000000026FB p18f452 0x0016 0x5000 movf reg,w,0  
Read: 0xE5 from reg (0x0000)  
Wrote: 0xE5 to W(0x0FE8) was 0xE7  
Wrote: 0x18 to status(0x0FD8) was 0x18  
0x00000000000026FC p18f452 0x0018 0x1001 iorwf reg1,w,0  
Read: 0x03 from reg1 (0x0001)  
Wrote: 0xE7 to W(0x0FE8) was 0xE5  
Wrote: 0x18 to status(0x0FD8) was 0x18
```

Каждый фрейм начинается с нового цикла симуляции. Обычно это включается в инструкцию симуляции. Вот каждое из полей:

| 64-bit simulation cycle | processor | PC     | opcode | instruction    |
|-------------------------|-----------|--------|--------|----------------|
| 0x00000000000026F6      | 16f452    | 0x001C | 0x1003 | iorwf reg3,w,0 |

Другие события, обнаруженные в процессе трассировочного фрейма, вырезаны. Обычно это будут трассировки чтения или записи регистра. Трассировки чтения показывают читаемое значение. Трассировки записи показывают записываемые значения и предыдущие значения регистра.

## Симуляция реального мира: стимулы (воздействия)

Стимулы крайне полезны, если не необходимы, для симуляции. Они предоставляют значения для симуляционного взаимодействия с реальным миром.

Возможности стимулов `gpsim` разработаны точными, эффективными и гибкими. Модели для выводов ввода-вывода PIC имитируют реальные устройства. Например, открытый коллектор вывода порта А PIC16C84 может быть только в низком состоянии. Множество выводов ввода-вывода могут соединяться между собой так, что открытый коллектор порта А может получить “подтягивающий” резистор порта В. В первую очередь стимулы только обнаруживают, когда стимул меняет состояние. Другими словами, стимулы не выбирают определения их статуса.

Аналоговые стимулы тоже допустимы. Возможно создать ссылки на напряжение и уровни напряжения для симуляции, практически, любого рода вещей реального мира. Например, возможно комбинировать два аналоговых стимула вместе, чтобы создать сигналы подобные DTMF тонам.

### 6.1 Как они работают

В простейшем случае стимул действует как источник для вывода ввода-вывода PIC. Например, вы можете захотеть симулировать часы и измерять их период, используя TMR0 (таймер 0). В этом случае стимулом будет источник и TMR0 входной вывод PIC'a как цепь. В `gpsim` вы создадите стимул для часов, используя команды стимула и соединив их с выводом ввода-вывода, используя команду узла.

В общем, вы можете иметь несколько “источников” и несколько “цепей”, которые соединены с узлами. (Хотя `gpsim` в данный момент ограничена “одно-портовыми” устройствами. Другими словами, она принимает, что земля обслуживается, как общая ссылка для источников и цепей). Хорошим аналогом будет цепь `spice`. `spice` список цепей (`netlist`) обращается к списку узлов в `gpsim` и элементы `spice` обращаются к источникам и цепям стимулов. Этот главный подход делает возможным создавать разнообразнейшее окружение симуляции. Вот список разных путей, которыми стимулы могут быть соединены:

1. Стимул соединен с одним I/O выводом
2. Стимул соединен с несколькими I/O выводами
3. Несколько стимулов присоединены к одному I/O выводу

4. Несколько стимулов соединены с несколькими I/O выводами
5. I/O вывод соединен с I/O выводом

Основная техника для реализации стимулов следующая:

1. Определить стимулы или стимул.
2. Определить узел.
3. Присоединить стимулы с узлами.

Чаще бывает, чем нет, определение стимула в файле.

### 6.1.1 Соединения между стимулами

Одной из проблем с этим узловым подходом к моделированию стимулов является возможность для соединения существующих. Например, если два I/O вывода соединены один с другим и работают в противоположных направлениях, соединение получится. `gprsim` принимает соединение с суммой атрибутов. Каждый стимул, даже если это ввод, производит влияние на узел. Это влияние заданной силы. Когда узел обновляется, `gprsim` будет просто добавлять силу всех стимулов вместе и применять это числовое значение к узлу. Сила со значением нуль относится к отсутствию цепи. Большая положительная сила используется стимулом для положительного движения узла, большая отрицательная сила используется для отрицательного движения. Суммирование атрибутов полезно для “подтягивающих” резисторов. В примере с открытым коллектором порта А/ порт В слабое “подтягивание”, `gprsim` задает относительно слабую силу “подтягивания” для резисторов и большую отрицательную силу для открытого коллектора, если он активен, или не задает силу, если нет движения (изменения). Эффект конденсатора (пока не поддерживается) может быть симулирован с динамически меняющимся значением силы.

## 6.2 Выводы I/O

`gprsim` моделирует выводы I/O, как стимулы. Таким образом, где бы стимулы не использовались, I/O выводы могут быть заменены. Например, вы можете захотеть присоединить два I/O вывода один к другому, как присоединился подтягивающий резистор порта В к порту А с открытым коллектором. `gprsim` автоматически создает стимул I/O вывода при создании процессора. Все, что

вам нужно, так это обозначить узел и затем назначить стимул для него. Имена для этих стимулов формируются связыванием имени порта с позицией бита I/O вывода. Например, бит 3 порта B называется portb3. Вот список поддерживаемых типов стимулов выводов I/O:

| <i>Тип I/O выводов</i>                 | <i>Функция</i>  |
|--|---|
| INPUT_ONLY (только ввод)               | Принимает только ввод (подобно MCLR)                      |
| BI_DIRECTIONAL (двунаправленные)       | Может быть источником или цепью (большинство выводов I/O) |
| BI_DIRECTIONAL_PU (двунаправленные PU) | PU – подтягивающие резисторы (Порт B)                     |
| OPEN_COLLECTOR (открытый коллектор)    | Могут двигаться только вниз (RA4 у c84)                   |

Нет специального типа выводов для аналоговых выводов I/O. Все аналоговые входы PIC мультиплексируются с цифровыми входами. Определение I/O выводов будет всегда для цифровых входов. gpsim автоматически определяет, если I/O вывод – аналоговый ввод.

### 6.3 Асинхронные стимулы

Асинхронные стимулы – аналоговые или цифровые стимулы, которые могут изменять состояние в любой заданный момент (ограниченный разрешением счетчика циклов). Они могут также определяться как повторяющиеся.

| <i>Параметр</i>                     | <i>Функция</i>                          |
|-------------------------------------|---|
| start_cicle (начать цикл)           | Количество циклов перед началом стимула |
| cycles[] (циклов)                   | Массив номеров циклов                   |
| data[] (данные)                     | Состояние стимула для цикла             |
| period (период)                     | Количество циклов в одном периоде       |
| initial_state (начальное состояние) | Начальное состояние перед data[0]       |

Когда стимул впервые инициализируется, он будет приведен в движение в 'initial state' (начальном состоянии) и будет оставаться в нем, пока CPU счетчик команд не станет равен 'start' (начать) циклу. После этого два массива cycles[] и data[] определяют выход стимула. Размер массивов тот же, что относится к количеству событий, которые создавались.

```

stimulus asynchronous_stimulus # or we could've used asy
# The initial state AND the state the stimulus is when
# it rolls over
initial_state 1
# all times are with respect to the cpu's cycle counter
start_cycle 100
# the asynchronous stimulus will roll over in 'period'
# cycles. Delete this line if you don't want a roll over.
period 5000
# Now the cycles at which stimulus changes states are
# specified. The initial cycle was specified above. So
# the first cycle specified below will toggle this state.
# In this example, the stimulus will start high.
# At cycle 100 the stimulus 'begins'. However nothing happens
# until cycle 200+100.
{ 200, 0,
  300, 1,
  400, 0,
  600, 1,
  1000, 0,
  3000, 1 }
# Give the stimulus a name:
name asy_test
# Finally, tell the command line interface that we're done
# with the stimulus
end

```

## ЧАСТЬ 6. СИМУЛЯЦИЯ РЕАЛЬНОГО МИРА: СТИМУЛЫ

41

Таким образом, количество событий, если было, обслуживает индекс этих двух массивов. Массив `cycles[]` определяет, когда событие обнаруживается, тогда как массив `data[]` определяет, какое состояние стимула будет введено. `cycles[]` измеряется в согласии с циклом `start`. Асинхронные стимулы могут быть сделаны периодическими через задание числа циклов в параметре `period`.

Вот пример, который генерирует три импульса, а затем повторяется:

### **Аналоговый асинхронный стимул**

Аналоговый асинхронный стимул идентичен синхронным стимулам, исключая тот момент, что данные точки – это числа с плавающей точкой.

## Модули

gpsim был разработан для отладки микропроцессоров. Однако микропроцессоры всегда часть системы. И неизменной часто встречающейся ошибкой становится интерфейс с системой. Модули предоставляют пользователю путь расширения gpsim и симуляции системы. Например, *системой* может быть процессор с несколькими подтягивающими резисторами и переключателями, или может быть процессор с LCD дисплеем. gpsim предоставляет несколько модулей, каждый может использоваться как для отладки, так и в качестве шаблона для создания новых модулей.

### 7.1 Модули gpsim

gpsim предоставляет следующие модули:

- binary\_indicator
- pullup
- pulldown
- usart
- parallel\_interface
- switch
- and2
- or2
- xor2
- not
- led\_7segments
- led
- PAL\_video
- Encoder

### 7.2 Написание новых модулей

Модули – это библиотека кодов. В Windows библиотека это dll, а в Unix общие библиотеки. Есть несколько деталей, которых модуль должен

придерживаться, но в основном модуль должен иметь полный доступ к gpsim API.

## Часть 8

### Символьная отладка

gpsim поддерживает таблицу символов.

## Часть 9

### Макросы

## Часть 10

### Нех файлы

Целевой код, симулируемый gpsim, может содержаться в hex файле, или более точно в Intel Hex файле. gpsim принимает формат hex, предоставляемый программами gasm и miasm. HEX файл не предоставляет никакой символьной информации. Рекомендуется, чтобы hex файлы использовались только, если вы обнаруживаете проблемы с использованием .cod файла, генерируемого вашим ассемблером или компилятором, или ваш ассемблер или компилятор не генерирует .cod файлы.

Также вы должны запастись процессором, когда загружаете hex файл. Посмотрите команду load.

## ICD

gpsim поддерживает (частично) первую версию ICD (в противоположность ICD2 (с формой хоккейной шайбы))

### Специальная конфигурация кода

Прочитайте руководство пользователя по MPLAB ICD.  
Вот краткая версия:

невозможно, по крайней мере: обнаружение неопределенного выхода, низковольтное программирование и полная защита кода. Возможно, лучше выключать также watchdog. Обратитесь к руководству пользователя по MPLAB ICD за более подробной информацией.

иметь NOP в качестве первой инструкции.

Не трогайте RB6 или RB7.

Не используйте последний уровень стека.

Не используйте следующие регистры и программные слова:

| <i>Процессор</i> | <i>Регистр</i>                         | <i>Программа</i> |
|------------------|--|------------------|
| -870/1/2         | 0x79,0xBB-0xBF                         | 0x6E0-0x7FF      |
| -873/4           | 0x6D, 0x1FD, 0xEB-0xF0,<br>0x1EB-0x1F0 | 0xEE0-0xFFF      |
| -876/7           | 0x70, 0x1EB-0x1EF                      | 0x1F00-0x1FFF    |

### icdprog

Загрузите и установите icdprog.

Используйте icdprog для программной цели с hex файлом (icdprog mycode.hex).

### Использование ICD

Запустите gpsim, как указано ниже:

```
gpsim -d /dev/ttyS0 -s mycode.cod
```

принимается, что ICD присоединено к первому СОМ-порту.  
Теперь вы можете напечатать `icd`, чтобы получить информацию:

```
**gpsim> icd  
ICD version "2.31.00" was found.  
Target controller is 16F877 rev 13.  
Vdd: 5.2 Vpp: 13.3  
Debug module id present
```

- это версия внутреннего программного обеспечения. Я только попробовал эту частную версию...

Вы можете шагать, сбросить, запустить, остановить, задать точки останова и прочитать файловые регистры. Это работает как в командной строке, так и с графической оболочкой.

## ICD TODO

MPLAB имеет установку для частоты целевого CPU, я только попробовал с 20 МГц кристаллом, так что возможно придется согласовывать тайм-аут последовательного порта с установками `gpsim`.

Окна исходного кода, дизассемблера, наблюдения, символьное и RAM работают. А остальные нет. Я полагал, что макет должен быть работоспособен хотя бы для PIC, но этого нет.

EEPROM поддерживается

Модифицируются данные

Зафиксировано, UI должно давать больше сигналов обратной связи в части, что случилось во время длинных задержек.

Лучшее обнаружение ошибок. `gpsim` не всегда видит, что цель не функциональна.

## Теория операций

Этот раздел предназначен только для тех, кого может заинтересовать, как `gpsim` оперирует. Информация здесь максимально точна. Однако, поскольку `gpsim` развивается, тоже должны делать детали теории операций. Используйте информацию, предложенную здесь, как введение высокого уровня, и используйте (хорошо закомментировано :) исходный текст, чтобы выяснить детали.

### 12.1 Задний план

`gpsim` написан в основном на C++. Почему? Хорошо, основным соображением была легкость разработки иерархической модели PIC. Если вы подумываете о микроконтроллере, так действительно легко конструировать различные компоненты. C++ сам предоставляет все для этой концептуализации. Более того, `Microchip`, как и другие производители микроконтроллеров, создал семейство устройств, которые очень похожи одно на другое. Опять же, C++ предоставляет “наследование”, которое допускает, чтобы взаимосвязь была обобщена на различные модели PIC.

### 12.2 Инструкции

Представлен базовый класс для 14-битовых инструкций (Я планирую сделать в будущем следующий шаг и создать базовый класс, из которого можно получить все PIC инструкции). Он в первую очередь служит двум целям: хранить все общее для каждой инструкции и дать значение виртуальным функциям основного доступа. Общая информация состоит из имени – или более точно, мнемоники инструкции, `opcode`, и указатель на процессор, пользующийся инструкцией. Некоторые из виртуальных функций выполняемые и именованные. После декодирования hex файла содержимое инструкций создается и сохраняется в массиве, названном `program_memory`. Индексом массива служит адрес, по которому пребывает инструкция. Для выполнения инструкции вызывается следующая последовательность кодов:

```
program_memory[pc.value]->execute();
```

которая говорит, взять инструкцию по текущему состоянию счетчика команд (pc.value) и вызвать через виртуальную функцию execute(). Такой подход позволяет, чтобы точки останова при выполнении были легко заданы. Инструкции специальных точек останова могут замещать соответствующие, находящиеся в массиве памяти программы. Когда вызывается execute, точка останова может появиться.

### 12.3 Основные регистры файла

Регистр файла симулируется классом file\_register. Есть один образец объекта file\_register для каждого регистра файла PIC. Все регистры собраны вместе в массив, названный register, который индексируется регистрами, обращающимися к адресам PIC. Массив линейный и не разбит на банки, как в реальном PIC. (Использование банков памяти поддерживается при симуляции).

### 12.4 Специальные файловые регистры

Специальные файловые регистры все остальные регистры, которые не относятся к основным файловым регистрам. Это включает регистры ядра, как status и option, а также периферийные регистры, как eeadr для eeprom. Специальные файловые регистры производные от основных файловых регистров и также хранятся в массиве register. Есть один образец для каждого регистра – даже, если регистр доступен более, чем в одном банке. Так, например, есть один образец для регистра status, хотя он может быть доступен через массив register более, чем в одном месте.

Все файловые регистры доступны через виртуальные функции put и get. Это сделано по двум основным причинам. Во-первых, это удобно для инкапсуляции перезаписи точек останова (для регистровых точек останова) в файловые регистры, а не в инструкции. Во-вторых, что более важно, допускают производные классы для реализации put и get более специфично. Например, put для indf регистра имеет целый ряд отличий от put для intcon регистра. В каждом случае put инициирует действия между просто сохраняемыми байтами данных в массиве. Это также позволяет использовать следующую последовательность кодов для легкой реализации:

```

movlw trisa      ; получает адрес trisa
movwf  fsr
movf   indf,w    ; читает trisa косвенно

```

## 12.5 Пример инструкции

Вот пример кода для инструкции `movf`, который иллюстрирует то, что обсуждалось выше. Где-нибудь в `gpsim` выполняется последовательность кодов:

```
program_memory[pc.value]->execute();
```

Давайте скажем, что РС указывает на инструкцию `movf`. Виртуальная функция `->execute()` вызовет `MOVF::execute`. Я добавил особые комментарии (их нет в основном коде) для детальной иллюстрации того, что происходит.

```

void MOVF::execute(void)
{
    unsigned int source_value;

    // Все инструкции traced (обсуждается ниже). Достаточно
    // только сохранить opcode. Однако даже этого может оказаться
    // не надо, если счетчик команд тоже трассируется. Ожидается,
    // что это исчезнет в дальнейшем...
    trace.instruction(opcode);

    // 'source' – это указатель на объект 'file_register'. Он
    // инициализируется чтением массива 'registers'. Заметьте, что
    // индекс зависит от бит 'rp' (в действительности от одного) в
    // регистре status. Время сохраняется кэшированием rp в
    // противоположность декодированию регистра status.
    source = cpu->registers[cpu->rp |
        opcode&REG_IN_INSTRUCTION_MASK];

    // У нас нет соображений по тому, до какого регистра мы пытаемся

```

```

// добраться и как он будет достигнут или есть ли на нем
// точка останова. Нет проблем, виртуальная функция 'get'
// разрешит все эти детали и “сделает все правильно”
source_value = source->get();

// Если мы обращаемся к W, тогда конструктор уже
// инициализировал 'destination'. Иначе source и destination
// те же самые.
if(opcode&DESTINATION_MASK)
    destination = source; // Результат отправится в source

// Запишем значение source в destination. Вновь у нас нет
// соображений, где может быть destination или
// как данные будут там записаны.
destination->put(source_value);

// Инструкция movf установит Z (ноль) бит в регистре status,
// если значение source было нулем.
cpu->status.put_Z(0==source_value);

// Окончательно, наращиваем PC на единицу.
cpu->pc.increment();
}

```

### Трассировка

Все, что симулируется, трассирует -all все время. Буфер трассировки - единый огромный циклический буфер целых. Информация после OR с символами трассировки сохраняется в буфере трассировки. Не делается попыток ассоциировать пункты в буфере трассировщика, пока симулятор симулирует PIC. Хотя, если вы взглянете на строки буфера, то увидите нечто похожее: cycle counter = ..., opcode fetch = ..., register read = ..., register write = ..., и т.д. Однако эта информация – пост процесс для того, чтобы удостовериться, что происходит и когда оно происходит. Можно использовать эту информацию для “undo” симуляции, другими словами, вы можете сделать шаг назад. Хотя я не разрабатывал этого совсем.

## Точки останова

Точки останова можно разбить на три категории: выполнения, регистровые и циклов.

### Выполнения:

Для точек останова выполнения специальная инструкция, соответственно названная 'Breakpoint\_instruction', создана и размещена в массиве программной памяти в месте, где желательна точка останова.

Оригинальная инструкция сохраняется во вновь открытой инструкции точки останова. Когда точка останова очищается, оригинальная инструкция извлекается из инструкции точки останова и помещается назад в массив программной памяти.

Заметьте, что эта схема имеет нулевую перезапись. Симулятор тревожится только, когда точка останова встречается.

### Регистровые:

Есть, по крайней мере, четыре разных типа точек останова, которые могут применяться к регистрам: чтение любого значения, запись любого значения, чтение определенного значения, запись определенного значения. Подобно точкам останова выполнения, есть специальные регистровые точки останова, которые замещают объект регистра. Так что, когда пользователь устанавливает точку останова на регистр 0x20, например, создается новый объект точки останова и вставляется в массив файла регистров в позицию 0x20. Когда симулятор пытается получить доступ к регистру по адресу 0x20, вместо него получается доступ к объекту точки останова.

Заметьте, что эта схема также имеет нулевую перезапись, принимая, когда встречается точка останова.

### Цикловые:

Цикловые точки останова позволяют `gpsim` изменять выполнение в обозначенных циклах инструкций. Это полезно для выполнения вашей

симуляции на определенные промежутки времени. Внутренне `gprsim` применяет широкое использование цикловых точек останова. Например, объект `TMR0` может программироваться на генерацию периодических цикловых точек останова.

Цикловые точки останова имплементируются с сортированным списком двойных связей. Список связей содержит две части информации (кроме связей): цикл на котором обнаруживается остановка и функция возврата из вызова (Функция возврата из вызова – указатель на функцию. В этом контексте `gprsim` получает указатель на функцию, которая должна появиться, когда обнаруживается цикл. Логика останова крайне проста. Когда счетчик циклов наращивается (инкрементируется), он сравнивается со следующей желаемой цикловой точкой останова. Если НЕТ совпадения, тогда инкрементируем. Так что перезапись для цикловых остановок требует времени для реализации и сравнения. Если совпадение ЕСТЬ, тогда функция возврата из вызова ассоциируется с появлением этой точки останова в двусвязном списке обслуживания, как ссылка на следующую цикловую остановку.



Updated October 2005

## **Как работают компоненты ИК-датчика движения**

### **ИК-излучение**

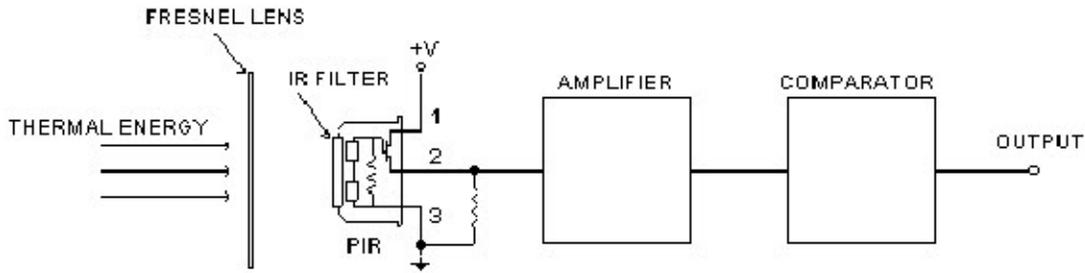
ИК-излучение содержится в электромагнитном спектре с длиной волны большей, чем видимый свет. Оно не видимо, но может быть обнаружено. Объекты, излучающие тепло, также излучают ИК, и эти объекты включают животных и людей, чье излучение максимально на длине волны 9.4 мкм. ИК-излучение в этом диапазоне не проходит через многие типы материалов, которые пропускают видимый свет, как обычное оконное стекло и пластик. Однако оно проходит, с некоторым ослаблением, через материал непрозрачный для видимого света, как германий и кремний. Необработанная кремниевая пластина служит хорошим ИК окном в защищенном от непогоды изделии уличного использования.

### **Пирозлектрические сенсоры**

Пирозлектрический сенсор изготовлен из кристаллического материала, который генерирует поверхностный электрический заряд, когда подвергается нагреву ИК-излучением. Когда какое-то количество излучения порождает изменения кристалла, количество заряда также изменяется, и может быть измерено чувствительным FET устройством, встроенным в сенсор. Элементы сенсора чувствительны к излучению в широком диапазоне, так что в корпус TO5 добавлен светофильтр для ограничения проникающего излучения диапазоном от 8 до 14 мкм, который наиболее подходит для обнаружения излучения от тела человека.

Как правило, вывод 2 FET соединен через отводящий резистор примерно в 100 кОм с общим проводом. И приходит на двухполярный усилитель, имеющий цепь ограничения сигнала. Усилитель обычно бывает фильтрующим с частотой среза 10 Гц для удаления высокочастотных шумов и сопровождается компаратором, реагирующим как на положительные, так и на отрицательные переходы выходного сигнала сенсора. Хорошо отфильтрованное напряжение питания от 3 до 15 вольт должно быть присоединено к выводу 1 FET.

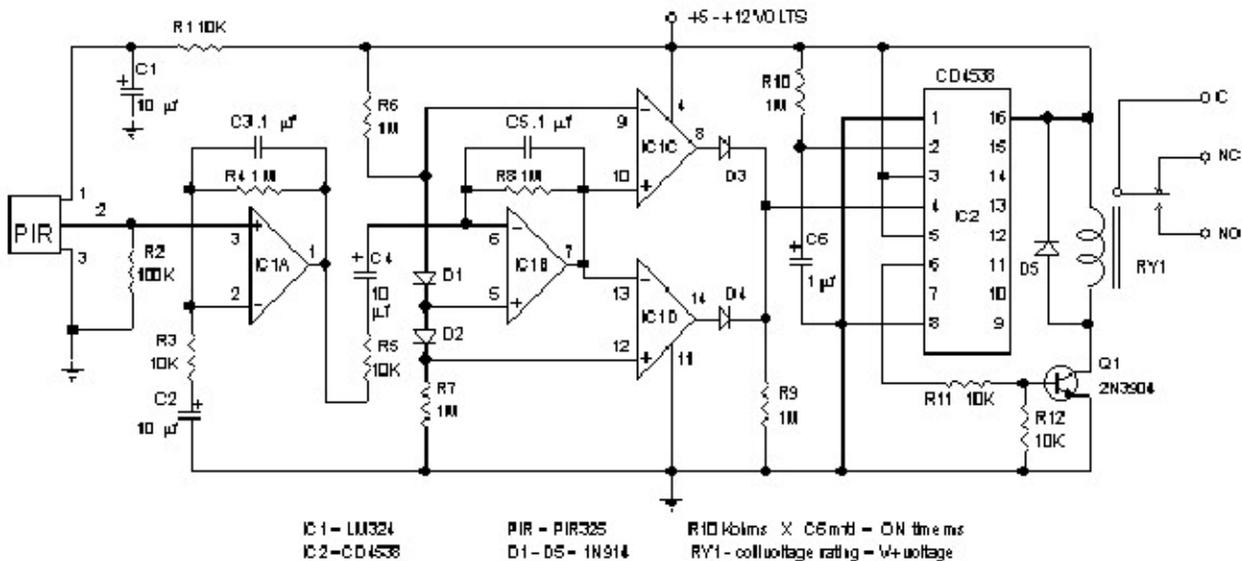
## TYPICAL CONFIGURATION



Сенсор PIR325 имеет два чувствительных элемента, объединенных в вольтокомпенсирующую конфигурацию. Это предотвращает воздействие сигналов при вибрации, изменении температуры и засветке. Человек, проходя перед сенсором, активизирует первый элемент, а затем второй, тогда как другие факторы воздействуют на оба элемента сразу, и компенсируются ими. Источник излучения должен пересекать сенсор в горизонтальном направлении, когда выводы 1 и 2 расположены горизонтально, так что элементы последовательно подвергаются воздействию ИК-излучения. Перед сенсором обычно устанавливается фокусирующее устройство.

На рисунке выше показана электрическая спецификация и внешний вид PIR325 в корпусе TO5. Отметьте широкий угол обзора без внешней линзы.

Типовое применение - в схеме с исполняющим реле. Резистор R10 и конденсатор C6 изменяют время, в течение которого реле RY1 активизировано после обнаружения движения.



## MOTION DETECTOR

### Линзы Френеля

Линзы Френеля – это выпуклые в разрезе линзы, сжатые в себя так, чтобы принять форму плоской линзы с сохранением оптических характеристик, но меньшей толщины. Благодаря этому они менее подвержены абсорбционным потерям.

Наша линза Френеля FL65 сделана из прозрачного для ИК-излучения материала в диапазоне 8-14 мкм, наибольшей чувствительности к излучению тела человека. Она разработана так, чтобы ее прорези смотрели на ИК чувствительный элемент. При этом гладкая поверхность обращена к объекту наблюдения и была вне корпуса, в котором располагается сенсор.

Линза закруглена с диаметром в один дюйм и имеет фланец в виде квадрата со стороной в 1.5 дюйма. Фланец используется для крепления линзы в соответствующее окно корпуса. Крепление может быть лучше и проще выполнено с помощью скотча. Силиконовая резина тоже может применяться, если она перекрывает кромку, формируя надежное крепление. Не известно надежных клеев, связывающих материал линзы.

FL65 имеет фокусное расстояние 0.65 дюймов от линзы к чувствительному элементу. Оно было определено экспериментально, чтобы иметь поле обзора приблизительно в 10 градусов, когда используется совместно с пироэлектрическим сенсором PIR325.

Эти относительно недорогие и легкие в применении пироэлектрический сенсор и линза Френеля могут широко использоваться в научной работе, робототехнике, и других полезных устройствах.

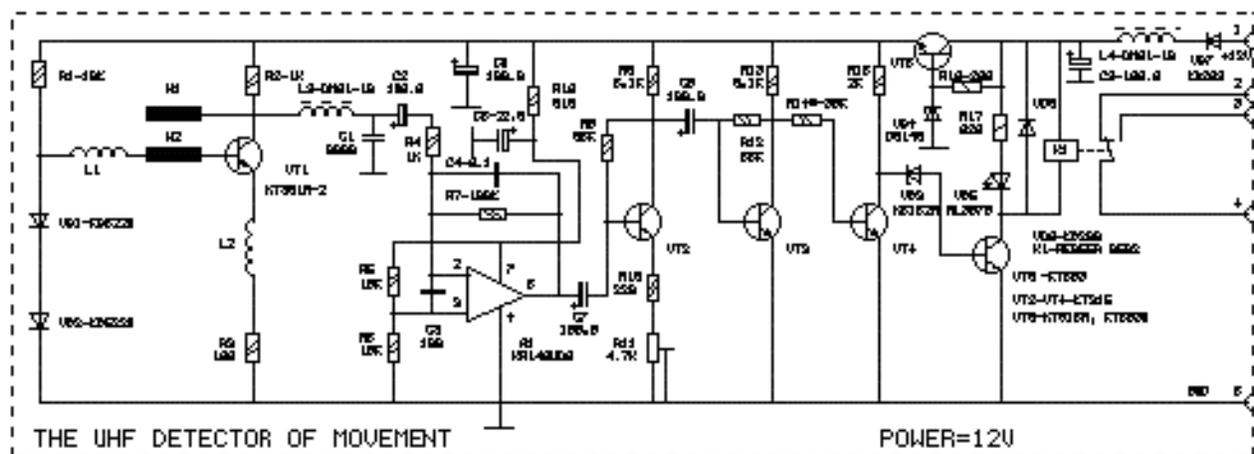
---

## СВЧ - датчик движения для охранной сигнализации .

<http://isaev51.narod.ru>

При разработке датчика ставилась задача создания альтернативы импортным датчикам движения. Ставилась задача создать датчик буквально из "мусора", простой, надежный и дешевый, технологичный в изготовлении и почти не уступающий импортным по габаритно-массовым характеристикам. Датчик реализован полностью на старой советской элементной базе, имеющейся у радиолюбителей в большом количестве. Корпусом датчика является обыкновенная мыльница с размерами полости внутренней части 54x95 мм. Если датчик установлен на диэлектрическом основании, то диаграмма направленности есть сфера с надежной чувствительностью 2-3 метра. Если датчик установлен на алюминиевом основании с размерами в полтора раза большими платы датчика, то диаграмма направленности есть конус 120 градусов, а надежная чувствительность возрастает вдвое. Датчик не чувствителен к большим перепадам температуры, а импульсы выходного реле совместимы с приемно-контрольными приборами охраны, рассчитанными на импульсные магнито- контактные датчики. Датчик опубликован в журнале Радио №12/2002г. стр. 41.

Схема датчика:

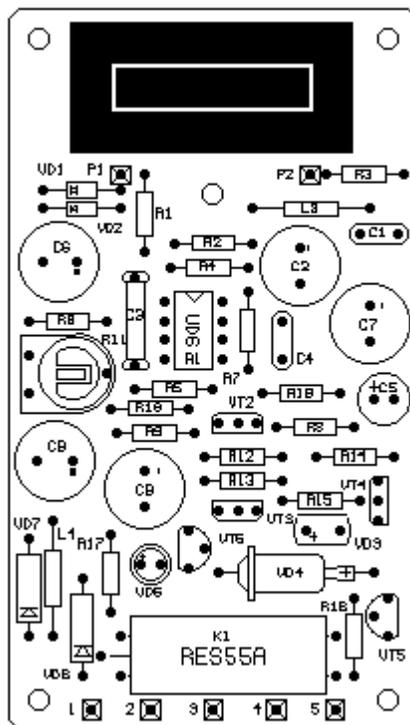


На транзисторе VT1 собран автодин - автогенератор частотой 2.4 ГГц с мягким самовозбуждением. Он же является гетеродином и смесителем для отраженного сигнала. При появлении в зоне охраны движущегося человека частота принятого сигнала изменяется на величину доплеровского смещения, которое составляет единицы герц. Этот сигнал через ФНЧ L3, C1 и конденсатор C2 поступает на вход каскада на A1, который одновременно является и усилителем и инфра низкочастотным фильтром. Далее сигнал усиливается усилителем переменного тока, что обеспечивает высокую термостабильность. Подстроечный резистор R11 - регулятор чувствительности. Роль компаратора выполняют стабилитрон VD3 и реле K1. Так как корпорация происходит на большом сигнале, то вопрос о стабильности порога корпорации отпадает сам собой. Недостатком схемы является чувствительность к понижению напряжения питания- оно не должно быть ниже 11 вольт. Если охранная система питается от аккумулятора

12 вольт, то для того, чтобы при просадке напряжения аккумулятора датчик продолжал нормально работать, в состав системы можно включить:

Повышающий стабилизатор питания.

Печатная плата:



Изображенная в верхней части платы щелевая антенна является не деталью, а частью рисунка печати. При изготовлении платы щелевая антенна должна быть отполирована до зеркального блеска и покрыта слоем ацетонового или спиртового раствора канифоли для предотвращения ее окисления в процессе эксплуатации. Катушки L1,L2 намотаны проводом ПЭЛ-0.23 на оправке диаметром 0.8 мм. и имеют по 12 витков, растянутых на длину 10 мм. Через отверстие в середине платы винтом М3 со стороны деталей крепится втулка со сквозной резьбой М3. В крышке, мыльницы напротив стойки сверлится отверстие диаметром 3 мм. Через это отверстие крышка мыльницы винтом М3 притягивается к торцу стойки и тем самым крепится. По углам мыльницы, против угловых отверстий вложенной в мыльницу платы, сверлятся отверстия на 3 мм. для крепления платы винтами М3. И сама стойка и крепежные винты могут быть из любого материала. Отверстие в крышке мыльницы напротив светодиода VD5 можно не делать, так как его вспышки просвечивают через крышку, а в процессе регулировки чувствительности крышка все равно снимается.

Внешний вид закрытого датчика:



Внешний вид открытого датчика:



Для изготовления платы вручную можно воспользоваться материалами журнала Радио. Для изготовления платы фотоспособом или способом термопереноса с помощью лазерного принтера и утюга потребуются файлы высококачественных изображений слоев печатной платы:

Схема, плата: [uhfs.zip\(268Kb\)](#)

---

[isaev51@bk.ru](mailto:isaev51@bk.ru)

## Таблица основных команд микроконтроллера PIC16F628A.

TABLE 15-1: ОПИСАНИЕ ПОЛЕЙ КОДОВ ОПЕРАЦИЙ

| Поле    | Описание  |
|---------|---|
| f       | Адрес регистра (от 0x00 до 0x7F)  |
| W       | Рабочий регистр (аккумулятор)   |
| b       | Бит, адресуемый внутри 8-битового регистра  |
| k       | Константа или метка   |
| x       | Не обслуживаемое расположение (= 0 или 1) Ассемблер будет генерировать код с x = 0. Это рекомендованная форма использования для совместимости со всеми программными средствами Microchip. |
| d       | Выбор адресата: d = 0, сохранить результат в W; d = 1, сохранить результат в регистре f. По умолчанию d = 1   |
| label   | Имя метки   |
| TOS     | Top of Stack (Верх стека)   |
| PC      | Program Counter (Счетчик команд)  |
| PCLATH  | Program Counter High Latch (Защелка старшего байта счетчика команд)   |
| GIE     | Global Interrupt Enable bit (Бит глобального разрешения прерываний)   |
| WDT     | Watchdog Timer/Counter (Watchdog Таймер/Счетчик)  |
| TO      | Time out bit (Бит окончания по времени)   |
| PD      | Power-down bit (Бит сброса по питанию)  |
| dest    | Адресуется либо регистр W, либо определенный положением регистр   |
| [ ]     | Опции   |
| ( )     | Содержимое  |
| →       | Назначить   |
| < >     | Поле бита регистра  |
| €       | Во множестве  |
| italics | Пользовательский термин (шрифт courier)   |

TABLE 15-2: PIC16F627A/628A/648A команды

| Мнемоника,<br>Операнды                    | Описание | Циклы         | 14-битовый код |   |     |     | Статус<br>Касается | Прим   |     |
|---|----------|---------------|----------------|---|-----|-----|--------------------|--------|-----|
|   |          |               | MSb            |   | LSb |     |                    |        |     |
| БАЙТ-ОРИЕНТИРОВАННЫЕ РЕГИСТРОВЫЕ ОПЕРАЦИИ |          |               |                |   |     |     |                    |        |     |
| ADDWF                                     | f, d     | Сложить W и f | 1              | 0 | 011 | dff | fff                | C,DC,Z | 1,2 |
| ANDWF                                     | f, d     | И W с f       | 1              | 0 | 010 | dff | fff                | Z      | 1,2 |
| CLRF                                      | f        | Очистить f    | 1              | 0 | 000 | 1ff | fff                | Z      | 2   |

|        |      |  |      |   |     |     |     |        |       |
|--------|------|--|------|---|-----|-----|-----|--------|-------|
| CLRW   | —    | Очистить W   | 1    | 0 | 1   | f   | f   | Z      |       |
| COMF   | f, d | Дополнение f (двоичное - инвертировать все цифры и добавить 1) | 1    | 0 | 100 | dff | fff | Z      | 1,2   |
| DECF   | f, d | Декремент f  | 1    | 0 | 001 | dff | fff | Z      | 1,2   |
| DECFSZ | f, d | Декремент f, Пропустить, если 0                                | 1(2) | 0 | 101 | dff | fff |        | 1,2,3 |
| INCF   | f, d | Инкремент f  | 1    | 0 | 101 | dff | fff | Z      | 1,2   |
| INCFSZ | f, d | Инкремент f, Пропустить, если 0                                | 1(2) | 0 | 111 | dff | fff |        | 1,2,3 |
| IORWF  | f, d | Включающее ИЛИ W с f   | 1    | 0 | 010 | dff | fff | Z      | 1,2   |
| MOVF   | f, d | Переместить f  | 1    | 0 | 100 | dff | fff | Z      | 1,2   |
| MOVWF  | f    | Переместить W в f  | 1    | 0 | 000 | lff | fff |        |       |
| NOP    | —    | Нет операции   | 1    | 0 | 000 | 0xx | 000 |        |       |
| RLF    | f, d | Сдвинуть влево через перенос                                   | 1    | 0 | 110 | dff | fff | C      | 1,2   |
| RRF    | f, d | Сдвинуть вправо через перенос                                  | 1    | 0 | 110 | dff | fff | C      | 1,2   |
| SUBWF  | f, d | Вычесть W из f   | 1    | 0 | 001 | dff | fff | C,DC,Z | 1,2   |
| SWAPF  | f, d | Поменять местами полубайты f                                   | 1    | 0 | 111 | dff | fff |        | 1,2   |
| XORWF  | f, d | Исключающее ИЛИ W с f  | 1    | 0 | 011 | dff | fff | Z      | 1,2   |
|        |      |  |      | 0 | 0   | f   | f   |        |       |

#### БИТ-ОРИЕНТИРОВАННЫЕ РЕГИСТРОВЫЕ ОПЕРАЦИИ

|       |      |                                 |      |   |     |     |     |  |     |
|-------|------|---------------------------------|------|---|-----|-----|-----|--|-----|
| BCF   | f, b | Очистить бит f                  | 1    | 0 | 00b | bff | fff |  | 1,2 |
| BSF   | f, b | Установить бит f                | 1    | 0 | 01b | bff | fff |  | 1,2 |
| BTFSC | f, b | Проверить бит f, Если сброшен   | 1(2) | 0 | 10b | bff | fff |  | 3   |
| BTFSS | f, b | Проверит бит f, Если установлен | 1(2) | 1 | 11b | bff | fff |  | 3   |
|       |      | Пропустить                      |      | 1 | b   | f   | f   |  |     |

#### ОПЕРАЦИИ УПРАВЛЕНИЯ И С КОНСТАНТАМИ

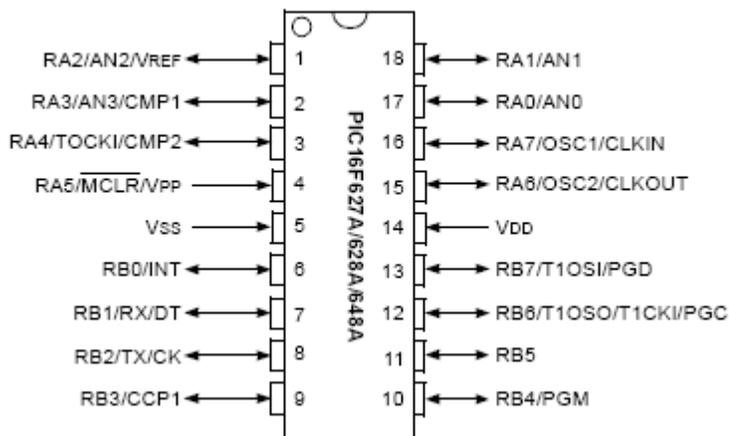
|        |     |                                       |   |   |     |     |     |        |  |
|--------|-----|---------------------------------------|---|---|-----|-----|-----|--------|--|
| ADDLW  | k   | Сложить константу и W                 | 1 | 1 | 111 | kkk | kkk | C,DC,Z |  |
| ANDLW  | k   | И константы с W                       | 1 | 1 | 100 | kkk | kkk | Z      |  |
| CALL   | k — | Вызвать подпрограмм                   | 2 | 1 | 0kk | kkk | kkk |        |  |
| CLRWDT |     | Очистить сторожевой таймер (Watchdog) | 1 | 0 | k   | k   | k   |        |  |
|        |     |                                       |   | 0 | 000 | 011 | 010 |        |  |
|        |     |                                       |   | 0 | 0   | 0   | 0   | TO,PD  |  |
| GOTO   | k   | Перейти к адресу                      | 2 | 1 | 1kk | kkk | kkk |        |  |
|        |     |                                       |   | 0 | k   | k   | k   |        |  |
| IORLW  | k   | Включающее ИЛИ константы с W          | 1 | 1 | 100 | kkk | kkk | Z      |  |
|        |     |                                       |   | 1 | 0   | k   | k   |        |  |

|        |   |                               |     |                          |  |  |  |                 |
|--------|---|-------------------------------|-----|--------------------------|--|--|--|-----------------|
| MOVLW  | k | Переместить константу в W     | 1   | 1 00x kkk kkk<br>1 x k k |  |  |  |                 |
| RETFIE | — | Возврат из прерывания         | 2   | 0 000 000 100<br>0 0 0 1 |  |  |  |                 |
| RETLW  | k | Возврат с константой в W      | 2   | 1 01x kkk kkk<br>1 x k k |  |  |  |                 |
| RETURN | — | Возврат из подпрограммы       | 2 1 | 0 000 000 100<br>0 0 0 0 |  |  |  |                 |
| SLEEP  | — | Переход в режим Standby       |     | 0 000 011 001<br>0 0 0 1 |  |  |  | TO,PD<br>C,DC,Z |
| SUBLW  | k | Вычесть W из константы        | 1   | 1 110 kkk kkk<br>1 x k k |  |  |  |                 |
| XORLW  | k | Исключающее ИЛИ константы с W | 1   | 1 101 kkk kkk<br>1 0 k k |  |  |  | Z               |

- Прим 1:** Когда модифицируется регистр I/O, как функция сама по себе ( т.е., `MOVF PORTB, 1`), используемое значение будет то же, что значение на самом выводе. Например, если данные на выводе, сконфигурированном, как ввод, «1», и сбрасываются внешним устройством, возвращаемые данные – это «0».
- 2:** Если эта инструкция выполняется на регистре TMR0 (и применено  $d = 1$ ), предделитель будет очищен, если назначен для Timer0 Module.
- 3:** Если счетчик команд (PC) модифицируется или проверяемое условие истинно, инструкция требует двух циклов. Второй цикл выполняется, как `NOB`.

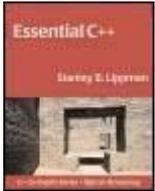
## Цоколевка контроллера PIC16F628A

PDIP, SOIC



## **С.Липпман. Основы C++**

Тем, кто будет программировать контроллер на языке «С», а особенно тем, кто выберет в качестве среды написания основной программы KDevelop, может пригодиться мой конспект книги С. Липпмана «Основы C++». Конспектировать вообще полезно, а для меня, порой, это единственный способ преодолеть более десятка страниц. Есть грамотный перевод, изданный, если не ошибаюсь издательством Williams.



**Essential C++**

By [Stanley B. Lippman](#)

Publisher : Addison Wesley

Pub Date : September 12, 2002

ISBN : 0-201-48518-4

Pages : 416

«Читатели могут проштудировать эту книгу и быстро освоить C++. Стен смог рассказать об очень многих и сложных вопросах в их сущности, которую, начинающие работать на C++ программисты, должны знать для написания реальных программ. Его стиль обучения эффективен и создает ясное понимание предмета до конца книги» - Steve Vinoski, IONA

Для программистов-практиков, не располагающих избытком свободного времени, книга «Основы C++» предлагает быстрый путь освоения языка. Книга специально разработана так, чтобы вы набрали нужную скорость за самое короткое время. Она фокусирует ваше внимание на тех элементах программирования на C++, которые вам, скорее всего, понадобятся в первую очередь, и вы опробуете возможности и технику, помогающие решать проблемы мира программирования.

Книга «Основы C++» представляет основы языка в контексте процедурного, группового, объектно-базируемого и объектно-ориентированного программирования. Все это собрано вокруг серии комплексных проблем программирования, а возможности языка проявляются, как решения. Благодаря этому вы не только узнаете о функциях и структуре языка C++, но начинаете понимать его назначение и природу.

Вы найдете далее такие ключевые моменты, как:

Программирование с использованием классов и стандартной библиотеки шаблонов (STL).

Объектно-базируемое программирование и построение классов.

Объектно-ориентированное программирование и построение иерархии классов.

Разработка шаблонов функций и классов, и их использование.

Поддержка исключений и идентификация типов в режиме выполнения программы.

И еще, бесценное приложение, предоставляющее полное решение и детальное разъяснение упражнений в программировании, расположенных в конце каждой части. И второе приложение, дающее быстрый обзор общих алгоритмов, с показом примеров их использования.

Это сжатое руководство даст вам умение работать с C++ и прочный фундамент, на котором выстроится ваш профессиональный опыт.

## Содержание

|  |     |
|--|-----|
| Оглавление                                       | 349 |
| Права  | 354 |
| Посвящение                                       | 8   |
| Предисловие                                      | 8   |
| Структура этой книги                             | 9   |
| Замечания по исходному коду                      | 10  |
| Благодарности                                    | 10  |
| Где найти дополнительную информацию              | 11  |
| Типографские соглашения                          | 11  |
| <br>   |     |
| Часть 1. Основы программирования на C++          | 12  |
| 1.1 Как писать программы на C++                  | 12  |
| 1.2 Определение и инициализация объектов данных  | 17  |
| 1.3 Написание выражений                          | 21  |
| 1.4 Написание условий и создание циклов          | 25  |
| 1.5 Как использовать массивы и векторы           | 31  |
| 1.6 Указатели дают больше гибкости               | 35  |
| 1.7 Запись и чтение файлов                       | 39  |
| <br>   |     |
| Часть 2. Процедурное программирование            | 44  |
| 2.1 Как писать функции                           | 44  |
| 2.2 Активизация функций                          | 49  |
| 2.3 Установка значений параметров по умолчанию   | 58  |
| 2.4 Использование локальных статических объектов | 60  |
| 2.5 Объявление Inline (встроенных) функций       | 61  |
| 2.6 Использование перезаписываемых функций       | 63  |
| 2.7 Определение и использование функций шаблонов | 64  |
| 2.8 Указатели на функции – добавление гибкости   | 66  |
| 2.9 Установка файла заголовка                    | 68  |
| <br>   |     |
| Часть 3. Групповое программирование              | 72  |
| 3.1 Арифметика указателей                        | 72  |
| 3.2 Считывание итераторов                        | 77  |
| 3.3 Операции Common для всех контейнеров         | 80  |
| 3.4 Использование последовательности контейнеров | 81  |
| 3.5 Использование общих алгоритмов               | 85  |
| 3.6 Как разрабатывать общий алгоритм             | 87  |
| 3.7 Использование отображения                    | 92  |
| 3.8 Использование множества                      | 94  |
| 3.9 Как использовать установщиков итераторов     | 96  |
| 3.10 Использование итераторов iostream           | 97  |
| <br>   |     |
| Часть 4. Объектно-базовое программирование       | 100 |
| 4.1 Как внедрять класс                           | 101 |
| 4.2 Что такое конструктор и деструктор класса    | 104 |

|  |   |     |
|--|---|-----|
| 4.3  | Что такое изменяемый и постоянный                     | 109 |
| 4.4  | Что такое this указатель                              | 112 |
| 4.5  | Члены статического класса                             | 114 |
| 4.6  | Построение класса итератора                           | 117 |
| 4.7  | Сотрудничество иногда требует дружбы                  | 120 |
| 4.8  | Внедрение оператора присваивания копии                | 122 |
| 4.9  | Внедрение объекта функции                             | 123 |
| 4.10   | Употребление классовых исключений операторов iostream | 125 |
| 4.11   | Указатели на функции-члены класса                     | 126 |
| Часть 5. Объектно-ориентированное программирование |   | 131 |
| 5.1  | Концепции объектно-ориентированного программирования  | 131 |
| 5.2  | Обзор объектно-ориентированного программирования      | 133 |
| 5.3  | Полиморфизм без наследования                          | 137 |
| 5.4  | Определение базового абстрактного класса              | 139 |
| 5.5  | Определение производного класса                       | 142 |
| 5.6  | Использование иерархии наследования                   | 147 |
| 5.7  | Насколько абстрактным должен быть базовый класс       | 149 |
| 5.8  | Инициализация, деструктор и копирование               | 151 |
| 5.9  | Определение виртуальной функции производного класса   | 152 |
| 5.10   | Идентификация типов при исполнении программы          | 155 |
| Часть 6. Программирование с шаблонами              |   | 158 |
| 6.1  | Параметризованные типы                                | 159 |
| 6.2  | Определение класса шаблона                            | 161 |
| 6.3  | Поддержка параметров типов шаблона                    | 163 |
| 6.4  | Внедрение класса шаблона                              | 164 |
| 6.5  | Функция оператора вывода шаблона                      | 169 |
| 6.6  | Постоянные выражения и параметры по умолчанию         | 170 |
| 6.7  | Параметры шаблонов и стратегия                        | 173 |
| 6.8  | Функции членов шаблона                                | 175 |
| Часть 7. Поддержка исключений                      |   | 177 |
| 7.1  | Вбрасывание исключения                                | 177 |
| 7.2  | Захват исключения                                     | 178 |
| 7.3  | Испытание для исключения                              | 180 |
| 7.4  | Управление локальным ресурсом                         | 183 |
| 7.5  | Стандартные исключения                                | 184 |
| Приложение А. Решение упражнений                   |   | 188 |
|  | Упражнение 1.4  |     |
|  | Упражнение 1.5  |     |
|  | Упражнение 1.6  |     |
|  | Упражнение 1.7  |     |
|  | Упражнение 1.8  |     |
|  | Упражнение 2.1  |     |
|  | Упражнение 2.2  |     |
|  | Упражнение 2.3  |     |

Упражнение 2.4  
Упражнение 2.5  
Упражнение 2.6  
Упражнение 3.1  
Упражнение 3.2  
Упражнение 3.3  
Упражнение 3.4  
Упражнение 4.1  
Упражнение 4.2  
Упражнение 4.3  
Упражнение 4.4  
Упражнение 4.5  
Упражнение 5.1  
Упражнение 5.2  
Упражнение 5.3  
Упражнение 5.4  
Упражнение 6.1  
Упражнение 6.2  
Упражнение 7.1  
Упражнение 7.2  
Упражнение 7.3

Приложение В. Указатель общих алгоритмов

228

accumulate()  
adjacent\_difference()  
adjacent\_find()  
binary\_search()  
copy()  
copy\_backward()  
count()  
count\_if()  
equal()  
fill()  
fill\_n()  
find()  
find\_end()  
find\_first\_of()  
find\_if()  
for\_each()  
generate()  
generate\_n()  
includes()  
inner\_product()  
inplace\_merge()  
iter\_swap()  
lexicographical\_compare()  
max(), min()  
max\_element(), min\_element()

merge()  
nth\_element()  
partial\_sort(), partial\_sort\_copy()  
partial\_sum()  
partition(), stable\_partition()  
random\_shuffle()  
remove(), remove\_copy()  
remove\_if(), remove\_copy\_if()  
replace(), replace\_copy()  
replace\_if(), replace\_copy\_if()  
reverse(), reverse\_copy()  
rotate(), rotate\_copy()  
search()  
search\_n()  
set\_difference()  
set\_intersection()  
set\_symmetric\_difference()  
set\_union()  
sort(), stable\_sort()  
transform()  
unique(), unique\_copy()

## Язык изложения

То, что вы видите, это не перевод книги С. Липпмана “Основы С++”.

В один из моментов работы, я захотел использовать KDevelop в связке с Qt Designer'ом из ASPLinux 9.0, как очень удобное инструментальное средство при создании вспомогательных простеньких программочек. Но, вопреки моим стараниям, ничего не получалось. Вполне резонно рассудив, что язык С, в том объеме, который я когда-то знал, я основательно забыл, а С++ не знал никогда, я заглянул в книжный магазин, поискал в Интернете, и остановил выбор на Essential С++ .

Чтобы как-то продвинуться в чтении дальше первых 50 страниц, на большее в данной ситуации, как правило, терпения не хватает, я решил ее содержательную часть “законспектировать”. То, что вы видите, не более чем конспект, осуществленный для личного пользования.

Как любой конспект, он отражает меру моего понимания (или непонимания) изложенного материала, и меру терпения в воспроизведении оригинала. Завершив записи, я подумал, что есть достаточно много людей, которые знают английский язык хуже, чем я, что не мешает им знать языки программирования значительно лучше, чем я. Может быть им, в сочетании с оригиналом, будет полезен мой конспект.

Из этих соображений я предлагаю свои конспекты всем, кому они могут оказать помощь в освоении языка С++, если качество изложения не помешает этому.

## Copyright

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Addison-Wesley was aware of a trademark claim, the designations have been printed in initial capital letters or all capital letters.

The author and publisher have taken care in preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The programs and applications presented in this book have been included for their instructional value. They have been tested with care, but are not guaranteed for any particular purpose. The authors and publisher do not offer any warranties or representations, nor do they accept any liabilities with respect to the programs or applications.

The publisher offers discounts on this book when ordered in quantity for special sales. For more information please contact:

Corporate, Government, and Special Sales

Addison Wesley Longman, Inc.

One Jacob Way

Reading, Massachusetts 01867

Copyright © 2000 Addison Wesley Longman

*Library of Congress Cataloging-in-Publication Data*

Lippman, Stanley B.  
Essential C++ / Stanley B. Lippman

p. cm.  
Includes bibliographical references and index.

1. C++ (Computer program language) I. Title.

QA76.73.C153 L577 1999

005.13'3--dc21 99-046613  
CIP

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher. Printed in the United States of America. Published simultaneously in Canada.

1 2 3 4 5 6 7 8 9—MA—0302010099

First printing, October 1999

## Dedication

To Beth, who remains essential

---

*To Danny and Anna, hey, kids look, it's done ...*

## Предисловие

Господи, какая тоненькая получилась книжка! Увы, мне, я сдал. Мой «Букварь C++» был в 1237 страниц с указателем, титулом и страницами посвящения. Эта же весит 276 – в боксерской терминологии «в петушином весе».

Первый вопрос, конечно, как это получилось? Об этом, собственно, и рассказ.

Я долгие годы донимал всех в Disney Feature Animation, чтобы мне позволили у них поработать. Я просил директоров, управляющий персонал – особенно досталось Mickey, честно говоря. Отчасти, я полагал, Голливуд - это волшебство. Большой экран. Также, я имел звание в области изобразительного искусства, как и в компьютерной науке, так что работа над фильмом, казалось, обещает некий персональный синтез. То, что я говорил руководству, конечно, касалось необходимости экспериментов с продукцией в порядке производства полезных инструментальных средств. Как системный программист, я всегда был одним из моих главных пользователей. А это трудно - держать оборону, или оставаться безучастным к критике, если ты один из главных жалобщиков на свои программы.

Компьютерные эффекты, наложенные во фрагменте Firebird Фантазии 2000, были интересны тем, что позволили мне присоединиться к работе. На пробу мне предложили написать что-то, что бы считывало необработанную информацию для сцены с Диснеевской камеры, и генерировало бы связку, которую можно вставить в Гудиниевский анимационный пакет. Я написал это на C++, конечно. Оно работало. И им понравилось. Так я был приглашен подняться на борт.

Однажды в работе (спасибо Jinko и Chuuan), меня попросили переписать все на Perl'e. Другой ТД, это было подчеркнуто, не будучи могучим программистом, знал, однако же, Perl, Tcl и т.д. (ТД на кино-жаргоне означало технический директор). Я был ТД программных фрагментов. Был еще также световой ТД (привет, Mira) и ТД моделей (привет, Tim), так же еще собственно аниматоры компьютерных эффектов (привет, Mike, Steve, и Topua). И, кстати, могу я сделать это побыстрее, поскольку, господи, мы должны провести тестирование концепции, чтобы подготовиться, чтобы директора (привет, Paul и Gaetan) и супервизор по эффектам (привет, Dave) дождались завершения тестов для доклада главе Feature Animation (привет, Peter). Не фатально, ты понимаешь, но...

Это ставило меня в затруднительное положение. Я мог достаточно быстро программировать на C++, и достаточно уверенно. Но, к сожалению, я не знаю Perl'a. Я подумал, ну хорошо, я почитаю что-нибудь. Но не слишком большое. Наконец, пусть, не совсем правильное. И хорошо бы, чтоб оно не слишком много мне поведало, хотя я знаю, я должен знать все, но потом, попозже. Как-никак, это шоу бизнес – директорам нужна проверка концепции, артистам вставка для проверки концепции, а продюсеру, ох, ей нужно 48 часов в сутках. Мне не нужна наилучшая книга по Perl'у – только нормальная книжка для начала, и чтобы не сбила с праведного пути.

Я нашел такую книгу – «Изучение Perl'a» Randal Schwartz. Она меня просветила, подтолкнула, и ее было

весело читать. Ну, настолько, насколько любая компьютерная книга может повеселить. Она оставила в стороне массу хороших вещей. В это время мне, однако, было не до них – мне нужно было, чтобы мои тексты на Perl'e работали.

В конечном счете, я с грустью отмечаю, что третье издание «Букваря C++» не выполняет подобной роли для кого-либо, нуждающегося в изучении C++. Она слишком для этого велика. Я думаю, она великолепная книга, конечно, особенно благодаря Josée Lajoie в качестве соавтора третьего издания. Но она слишком всеобъемлюща для такого рода скоростного изучения языка C++. Вот почему я решил написать эту книгу.

Вы, возможно, думаете, что C++ - это не Perl. Да так. И это не учебник по Perl'у. Это об изучении C++. В действительности вопрос в том, как можно выбросить тысячу страниц, и надеяться научить чему-то?

**Уровень детализации.** В компьютерной графике уровень детализации относится к тому, насколько резко передается изображение. Въезжающий верхом Хан в левом переднем углу экрана нуждается в лице с глазами, волосах, пятичасовой тени, одежде, и т.д. Хан заворачивает – нет, не за скалу, дурачок – хорошо, мы не воспроизводим обе картинки с одинаковой степенью детализации. Аналогично, уровень детализации в этой книге сильно снижен. «Букварь C++», с моей точки зрения, имеет наиполнейший и великолепно изложенный раздел по оверлейным операторам (я могу сказать это, поскольку он написан Josée). Однако он имеет 46 страниц описаний и примеров кода. Здесь у меня 2 страницы.

**Сердцевина языка.** Когда я был редактором *C++ Report*, я часто повторял: половина работы редактора журнала в том, чтобы решить, что не публиковать. Это же можно отнести к данной книге. Книга организована вокруг некоторых проблем программирования. Возможности же языка проявляются в способности решить отдельные проблемы. У меня не было проблем, которые решались бы множественным или виртуальным наследованием, я о них и не пишу. Для внедрения класса итератора, однако, я описываю вложенные типы. С операторами класса реструктуризации легко запутаться – их следует полнее раскрыть. Вместе с тем, я предпочел не представлять их. И так далее. Выбор и полнота представления возможностей языка всегда под прицелом критики. Это мой выбор. Это моя ответственность.

**Количество примеров.** В «Букваре C++» сотни страниц кодов, которые детально рассматриваются, включают систему объектно-ориентированных запросов и полдюжины внедренных классов. Хотя эта книга содержит исходные коды, многие из них незамысловаты, не столь значимы, как в «Букваре C++». Для поддержки в [Приложении А](#) приведены решения упражнений. Как сказала мой редактор Deborah Lafferty: «Если ты пытаешься быстро освоить что-нибудь, очень полезно иметь под рукой ответы для закрепления изученного».

## Структура книги

Книга содержит семь частей и два приложения. [Часть 1](#) дает представление об основах языка в контексте написания маленьких интерактивных программ. Она охватывает встроенные типы данных, предопределенные операторы, библиотеку классов векторов и строк, условия и циклы, и библиотеку `iostream` для ввода-вывода. Я включил строковый и векторный класс в эту часть, поскольку я поддерживаю их использование вместо встроенных массивов и строк символов в C-стиле.

[Часть 2](#) объясняет, как создавать и использовать функции, проводя через множество типов функций, поддерживаемых в C++: линейных, перезаписываемых, функций шаблонов, равно как и указателей на функции.

[Часть 3](#) посвящена тому, что обычно относится к стандартной библиотеке шаблонов (STL): подборка классов контейнеров, таких, как векторы, списки, множества, отображения (карты), и общие алгоритмы

для операций с этими контейнерами - `sort()`, `copy()`, `merge()`. Приложение В дает алфавитный список наиболее употребительных общих алгоритмов с примерами их использования.

Как C++ программиста, вас в первую очередь интересует состав классов и их иерархия. Часть 4 описывает легкость создания и использования классов в C++ для открытия типов данных специфических для ваших приложений. Например, в Dreamworks Animation, где я выполнял некоторую работу по консультации, мы разрабатывали классы для 4-канальной композиции образов и т.д.

Часть 5 описывает, как расширить конструкцию класса для поддержки семейства производных классов в объектно-ориентированной иерархии классов. Вместо создания восьми независимых классов композиции образов, например, мы определяем иерархию композиции, используя наследование и динамическое связывание.

Класс шаблонов – тема Части 6. Класс шаблонов – род предписывания для создания класса, в котором один или больше типов или значений параметризованы. Класс векторов, например, может параметризовать тип элементов, который содержит. Класс буфера может параметризовать не только тип элементов, который содержит, но также и размер их буфера. Раздел возник из внедрения класса шаблона двоичного дерева.

И наконец, Часть 7 иллюстрирует, как легко использовать поддержку исключений в C++ и приспособливать их к существующей стандартной библиотеке иерархии классов исключений. Приложение А дает решения упражнений в программировании. Приложение В дает примеры программирования и поясняет наиболее часто используемые общие алгоритмы.

## Заметки по исходным кодам

Полный исходный код программ, разработанных в книге, как и решения всех упражнений, доступны для загрузки в режиме on-line с Web-сайта Addison Wesley Longman ([www.awl.com/cseng/titles/0-201-48518-4](http://www.awl.com/cseng/titles/0-201-48518-4)) и с моей домашней страницы ([www.objectwrite.com](http://www.objectwrite.com)). Все коды проверялись с Visual C++ 5.0, использующей компилятор Intel C++, и с Visual C++ 6.0, использующей компилятор Microsoft C++. Вам может понадобиться модификация кода для компиляции в вашей системе. Если вы модифицируете код, вышлите мне список ([slippman@objectwrite.com](mailto:slippman@objectwrite.com)), а я опубликую их под вашим именем в файле модификации, добавленном к коду решения. (Заметьте, что полный код не отображается на дисплее внутри самого текста).

## Благодарности

Особая благодарность Josée Lajoie, соавтору “Букваря C++”, 3-о издания. Она оказала удивительную поддержку, благодаря ее очень важным замечаниям по различным деталям текста и ее неослабному поощрению. Я также приношу специальные благодарности Dave Slayton за работу, как с текстом, так и с кодами примеров его “острым” зеленым карандашом, и Steve Vinoski за его сочувственные, но твердые замечания по наброскам к этой книге.

Специальные благодарности также редакторскому корпусу Addison-Wesley: Deborah Lafferty, кто как редактор, поддерживала этот проект с самого начала, Betsy Hardinger, кто как выпускающий редактор, много способствовала “читабельности” книги, и John Fuller, кто в качестве менеджера по производству вел нас, свою паству, от рукописи к Книге.

Когда я писал эту книгу, я работал независимым консультантом и разрывался между “Основами C++” и

множеством (умеренным) понимающих клиентов. Я хотел бы поблагодарить Colin Lipworth, Edwin Leonard и Kenneth Meyer за их терпение и добрую веру.

## Где найти дополнительную информацию

From a completely biased point of view, the two best one-volume introductions to C++ are Lippman and Lajoie's *C++ Primer* and Stroustrup's *The C++ Programming Language*, both in their third edition. Throughout the text I refer you to one or both of the texts for more in-depth information. The following books are cited in the text. (A more extensive bibliography can be found in both *C++ Primer* and *The C++ Programming Language*.)

[LIPPMAN98] Lippman, Stanley, and Josйе Lajoie, *C++ Primer, 3rd Edition*, Addison Wesley Longman, Inc., Reading, MA 1998) ISBN 0-201-82470-1.

[LIPPMAN96a] Lippman, Stanley, *Inside the C++ Object Model*, Addison Wesley Longman, Inc., Reading, MA(1996) ISBN 0-201-83454-5.

[LIPPMAN96b] Lippman, Stanley, Editor, *C++ Gems*, a SIGS Books imprint, Cambridge University Press, Cambridge, England(1996) ISBN 0-13570581-9.

[STROUSTRUP97] Stroustrup, Bjarne, *The C++ Programming Language, 3rd Edition*, Addison Wesley Longman, Inc., Reading, MA(1997) ISBN 0-201-88954-4.

[SUTTER99] Sutter, Herb, *Exceptional C++*, Addison Wesley Longman, Inc., Reading, MA(2000) ISBN 0-201-61562-2.

## Типографские соглашения

Текст книги набран в 10.5 pt. Palatino. Текст программы и ключевые слова языка появляются как 8.5 pt. *lucida*. Функции идентифицируются по их именам с последующим оператором вызова функций в C++ (`()`). Таким образом, например, `foo` представляет программный объект, а `bar()` представляет функцию. Имена классов даются в Palatino.

## Часть 1. Основы программирования на C++

В этой части мы разработаем небольшую программу, чтобы познакомиться с основными компонентами языка C++. Эти компоненты состоят из:

Небольшого множества типов данных: Булевские, символьные, целые, с плавающей точкой.

Множества арифметических, логических и операторов отношения для работы с этими типами. Это включает не только обычно предполагаемые сложение, равенство, “меньше, чем” и присваивание, но так же операторы присваивания структуры, условия, соглашения по наращиванию.

Группы условных переходов и создание циклов, как if условия, и while циклы для изменения порядка управления в нашей программе.

Некоторого количества структурных типов, как указатели и массивы. Это позволит нам, соответственно, перейти непосредственно к существующим объектам и определить подборку элементов одинарного типа.

Стандартной библиотеки базовых программных абстракций, как строка и вектор.

### 1.1 Как писать программы на C++

Положим, нам нужно написать простую программу, отправляющую сообщение на терминал пользователя, которое просит ввести имя. Затем мы прочитываем введенное имя, сохраняем, чтобы использовать в дальнейшем, и, наконец, приветствуем пользователя по имени.

Прекрасно, откуда начнем? Начнем там, откуда начинаются все программы на C++ - с функции, называемой `main()`. `main()`, внедряемая пользователем функция в следующей основной форме:

```
int main(){  
    // код нашей программы далее}
```

`int` это ключевое слово языка C++. *Ключевые слова* – предопределенные имена, имеющие особое значение в языке. `int` представляет встроенный целый тип данных. (У меня есть много, что сказать о типах данных в следующих разделах).

*Функция* – независимая последовательность кодов, производящих некие выкладки. Она состоит из частей: возвращаемый тип, имя функции, список параметров, и тело функции. Давайте рассмотрим каждую из частей по очереди.

*Возвращаемый тип* функции обычно представляет результат вычислений (выкладок). `main()` имеет целый возвращаемый тип. Значение, возвращаемое `main()` показывает, было ли выполнение нашей программы успешным. По соглашению, `main()` возвращает 0 при удачном выполнении. Не нулевое значение показывает, что что-то идет не так.

*Имя* функции выбирается программистом из соображений наилучшим образом определить, что функция будет делать. `min()` и `sort()`, например, хорошие имена функций. `f()` и `g()` не так хороши. Почему? Они менее информативны в отношении того, что функция будет делать.

main – не ключевое слово. Система компиляции, выполняющая нашу C++ программу, предполагает, что функция main() предопределена. Если мы забудем включить ее, наша программа не будет работать.

*Список параметров* функции заключен в круглые скобки и размещен за именем функции. Пустой список параметров, как у main(), обозначает, что функции не передаются параметры.

Список параметров, обычно разделяемый запятой, перечень типов данных, которые пользователь может передать функции при ее выполнении. (Мы говорим, что пользователь *вызывает* или *активизирует* функцию). Например, если мы напишем функцию min(), возвращающую наименьшее из двух значений, ее список параметров должен определить типы двух значений, которые мы сравниваем. Функция min() для сравнения двух целых чисел может быть определена следующим образом:

```
int min(int val1, int val2){  
    // код программы следует здесь ...}
```

*Тело* программы заключено в фигурные скобки ( {} ). Оно содержит последовательность кодов, которая осуществляет все действия функции. Двойная прямая косая черта ( // ) означает пояснения, заметки программиста по каким-то аспектам кодов. Они предназначены для читающих исходный текст программы и выбрасываются при компиляции. Все, следующее за прямой двойной косой чертой до конца линии, относится к комментарию.

Наша первая задача, написать вывод сообщения на терминал пользователя. Ввод и вывод не предопределены в языке C++. Вернее, они поддерживаются *объектно-ориентированным классом иерархии*, встроенным в C++ и поддерживаемым, как часть стандартной библиотеки C++.

Класс – тип данных, определяемых пользователем. Механизм работы классов – метод добавления типа данных, распознаваемого нашей программой. Объектно-ориентированная иерархия классов определяет семейство относительных типов класса, таких, как терминал и файловый ввод, терминал и файловый вывод, и так далее. (Позже мы основательнее обсудим классы и объектно-ориентированное программирование в этой книге).

C++ предопределяет небольшое множество основных типов данных: булевы, символьные, целые и с плавающей точкой. Хотя они служат основой для всего нашего программирования, мы на них не будем сосредотачивать внимание. Кинокамера, например, должна иметь положение в пространстве, обычно представляемое тремя числами в формате с плавающей точкой. Камера должна, кроме того, иметь ориентацию в пространстве, описываемую еще тремя числами с плавающей точкой. Есть еще одно отношение, описывающее отношение расстояния до камеры к высоте предмета. Оно представлено единственным числом с плавающей точкой.

На очень примитивном уровне, можно сказать, что камера представлена семью числами с плавающей точкой, шесть из которых образуют две группы координат x, y, z. Программирование на этом низком уровне требует, чтобы мы направляли нашу мысль то назад, то вперед - от манипуляций с абстрактной камерой к манипуляции с семью числами с плавающей точкой, представляющими камеру в нашей программе.

Механизм классов позволяет нам добавить уровни абстракции типов в нашу программу. Например, мы можем определить класс 3-мерных точек для представления локализации и ориентации в пространстве. Подобным же образом мы можем определить класс Камер, состоящий из двух объектов класса 3-мерных точек и числа с плавающей точкой. Мы все еще представляем камеру семью числами с плавающей точкой. Но разница в том, что в нашей программе мы теперь можем напрямую манипулировать с классом Камер, а не с семью числами с плавающей точкой.

Определение класса обычно разбивается на две части, каждая из которых представлена своим файлом: *файл заголовка*, производящий объявление операций, поддерживаемых классом, и *файл текста программы*,

который содержит реализацию этих операций.

Для использования класса мы включаем файл заголовка в нашу программу. Файл заголовка делает класс узнаваемым программой. Стандартные библиотеки ввода/вывода в C++ вызываются из библиотеки *iostream*. Она содержит подборку связанных классов, поддерживающих ввод и вывод на пользовательский терминал и в файлы. Чтобы использовать библиотеку класса *iostream*, мы должны включить ассоциированный с ней файл заголовка:

```
#include <iostream>
```

Для обращения к терминалу пользователя мы используем предопределенный объект класса, названный `cout` (произносится «си аут»). Мы направляем данные, которые мы хотим, чтобы `cout` написал, пользуясь оператором вывода ( `<<` ), следующим образом:

```
cout << "Please enter your first name: ";
```

Это представлено *выражением* программы C++, маленьким независимым кусочком программы на C++. Это аналог предложения в обычном языке. Выражение завершается точкой с запятой. Наше выражение пишет строку символов (отмеченных кавычками) на терминале пользователя. Кавычки идентифицируют строку, они не отображаются на терминале. Пользователь видит

```
Please enter your first name:
```

Наша следующая задача прочитать ввод пользователя. Прежде, чем мы сможем прочитать имя, напечатанное пользователем, мы должны определить объект, в котором сохраним информацию. Мы определим объект, обозначив тип данных объекта, и дав ему имя. Мы уже видели один тип данных `int`. Но это, однако, трудно использовать для хранения чьего-либо имени. Больше подходит тип данных из стандартной библиотеки строкового класса:

```
string user_name;
```

Это определяет `user_name`, как объект строкового класса. Определение, довольно чудно, названо *заявительным предложением*. Это предложение не будет приниматься до тех пор, пока мы не сделаем строковый класс известным программе. Мы осуществляем это включением файла заголовка строкового класса:

```
#include <string>
```

Для чтения ввода с терминала пользователя, мы используем предопределенный объект класса, называемый `cin` (произносится «си ин»). Мы используем оператор ввода ( `>>` ) для направления `cin` на чтение данных с терминала пользователя в объект подходящего типа:

```
cin >> user_name;
```

Последовательности вывода и ввода появятся следующим образом на терминале пользователя. (Ввод пользователя отмечен жирным шрифтом).

```
Please enter your first name: anna
```

Все, что нам еще остается сделать, поприветствовать пользователя по имени. Мы хотим, чтобы наш вывод выглядел, примерно, так:

```
Hello, anna ... and goodbye!
```

Понимаю, это не верх совершенства. Но терпение, мы только начали. Мы научимся более изящным вещам до завершения книги.

Для вывода нашего приветствия, наш первый шаг перенаправить приветствие на следующую строку. Мы сделаем это, написав символ новой строки в cout:

```
cout << '\n';
```

Символы обозначаются двумя апострофами. Существует две разновидности символов: печатные символы, такие как буквы ('a', 'A' и т.д.), числа и знаки препинания (';', '-' и т.д.), и непечатные символы, такие как символ перевода строки ('\n') или табуляция ('\t'). Поскольку нет символьного представления непечатных символов, самый общий пример символ перевода строки и табуляция, они представляются специальной последовательностью из двух символов.

Теперь, перейдя на новую строку, мы хотим воспроизвести наше Hello:

```
cout << "Hello, ";
```

Дальше нам нужно вывести имя пользователя. Оно сохранено в нашем строковом объекте user\_name. Как мы сделаем это? Так же, как и с другими типами:

```
cout << user_name;
```

Наконец, мы завершаем наше приветствие, сказав «прощай» (обратите внимание, что строка символов может содержать оба вида, как печатные, так и непечатные символы):

```
cout << " ... and goodbye!\n";
```

В основном, все встроенные типы выводятся тем же путем – помещаем значение справа от оператора вывода. Например,

```
cout << "3 + 4 = ";  
cout << 3 + 4;  
cout << '\n';
```

дает следующий вывод:

```
3 + 4 = 7
```

Как мы определили новые типы класса для использования в нашем приложении, так же мы проводим привязку оператора вывода для каждого класса. (Мы увидим, как это сделать в Части 4). Это позволяет пользователям нашего класса выводить отдельные объекты класса совершенно одинаковым образом со встроенными типами данных.

Вместо написания отдельных выводов каждый на своей строке мы можем объединить их в одно предложение вывода:

```
cout << '\n'  
<< "Hello, "  
<< user_name  
<< " ... and goodbye!\n";
```

В завершении мы можем явно завершить main(), используя выражение return (возврат):

```
return 0;
```

return – ключевое слово C++. Выражение, следующее за ним, в данном случае 0, представляет результат выполнения функции. Напомню, что возвращаемое функцией значение 0 говорит об успешном завершении

работы.<sup>4</sup>

Соберем кусочки вместе – получим нашу первую завершённую программу на C++:

```
#include <iostream>
#include <string>
using namespace std; // пока не объясняю этого ...

int main(){
string user_name;
cout << "Please enter your first name: ";
cin >> user_name;
cout << '\n'
    << "Hello, "
    << user_name
    << " ... and goodbye!\n";

return 0;}
```

После компиляции при выполнении, этот код произведет следующее (мой ввод подсвечен жирным шрифтом):

```
Please enter your first name: anna

Hello, anna ... and goodbye!
```

Одно выражение я не пояснил:

```
using namespace std;
```

Давайте посмотрим, смогу ли я объяснить это, не напугав вас. ( Сделайте глубокий вдох!). Оба using и namespace – ключевые слова C++. std – имя стандартной библиотеки namespace. Все выбираемое из стандартной библиотеки (как объекты строкового класса, класса iostream - cout и cin ) заключено внутри std namespace. Конечно, ваш следующий вопрос, а что такое namespace?

*namespace* – метод упаковки библиотечных имен таким образом, чтобы они могли быть использованы в окружении пользовательской программы без появления коллизий. (*Коллизия имен* обнаруживается, когда есть два использования одного и того же имени в приложении, так что программа не может различить их. Когда такое случается, программа не может продолжать работу, пока не разрешится конфликт имен). Namespaces – способ ограждающий видимость имен.

Для использования объектов строкового класса и класса iostream – cin и cout – внутри нашей программы мы должны не только включить файлы заголовков string и iostream, но так же сделать имена видимыми с std namespace. *Использование директивы*

```
using namespace std;
```

простейший метод сделать имена внутри namespace (пространство имен) видимыми.

(Более детально namespaces описано в Разделе 8.5 [LIPMAN98] )

---

<sup>4</sup> Если мы не напишем подходящего выражения для return в конце main(), return 0; выражение будет вставлено автоматически. В примерах программ, далее в книге, я не записываю этого выражения.

## Упражнение 1.1

Введите программу `main()`, показанную выше. Напишите ее вручную или загрузите программу – посмотрите в предисловии, как получить исходные программы и решения к упражнениям. Скомпилируйте и выполните программу на вашем компьютере.

## Упражнение 1.2

Закомментируйте включение строкового файла заголовка:

```
// #include <string>
```

Теперь перекомпилируйте программу. Что случилось? Теперь раскомментируйте строковый файл и закомментируйте вывод

```
//using namespace std;
```

Что теперь?

## Упражнение 1.3

Измените имя `main()` на `my_main()` и перекомпилируйте программу. Что произойдет?

## Упражнение 1.4

Попробуйте расширить программу: (1) попросите пользователя ввести имя и фамилию, (2) измените вывод для написания и имени, и фамилии.

## 1.2 Определение и инициализация объектов данных

Теперь, чтобы завладеть вниманием пользователя, давайте предложим короткий тест. Мы отобразим два числа из числовой последовательности и предложим пользователю угадать следующие значения в последовательности. Например,

```
The values 2,3 from two consecutive elements of a numerical sequence.  
What is the next value?
```

Эти значения третий и четвертый элементы из последовательности Фибоначчи: 1, 1, 2, 3, 5, 8, 13 и т.д. Последовательность Фибоначчи начинается с двух элементов – единиц. Каждый следующий элемент это сумма двух предшествующих. (В Части 2 мы напишем функцию для вычисления элементов).

Если пользователь введет 5, мы поздравим его и спросим, хочет ли он попробовать другую числовую последовательность. Любое другое введенное значение – неверное, и мы спросим пользователя, хочет ли он погадать еще?

Чтобы поддержать интерес к программе, мы сохраним текущий счет, основанный на отношении правильных ответов к числу попыток.

Наша программа нуждается, по меньшей мере, в пяти объектах: объекте строкового класса для сохранения имени пользователя, трех объектах класса `int` для запоминания по очереди попыток, числа попыток, числа успешных попыток, и объект класса `float` с плавающей точкой для запоминания счета.

Для определения объектов данных мы должны ввести имена и тип данных. Имена могут быть любыми комбинациями букв, цифр, подчеркиваний. Буквы регистро-зависимые. Каждое из имен `user_name`, `User_name`, `uSeR_nAmE`, и `user_Name` относятся к разным объектам.

Имя не должно начинаться с цифры. Например, `1_name` неправильно, `name_1` – правильно. Так же имя не должно совпадать с ключевыми словами языка. Например, `delete` - ключевое слово языка, так что мы не должны использовать его в нашей программе. (Это объясняет, почему оператор удаления символа из строкового класса это `erase()`, а не `delete()`).

Каждый объект должен быть своего типа данных. Имя объекта позволяет нам обратиться к нему непосредственно. Тип данных определяет область значений, сохраняемых объектом, и количество памяти необходимой для запоминания этого значения.

Мы видели определение `user_name` в предыдущем разделе. Перенесем то же определение в нашу новую программу:

```
#include <string>
string user_name;
```

Класс программно-определенный тип данных. C++ также поддерживает множество встроенных типов данных: булевы, целые, с плавающей точкой и символьные. Ключевые слова, ассоциированные с каждым из них, позволяют нам определить тип данных. Например, для запоминания значения, введенного пользователем, мы определим объект целого типа:

```
int usr_val;
```

`int` - ключевое слово языка определяющее, что объект `usr_val` целого типа. Оба объекта, число попыток, сделанных пользователем и число правильных ответов, объекты целого типа. Разница только в том, что мы бы хотели дать им начальное значение 0. Мы можем вывести каждое на отдельную строку:

```
int num_tries = 0;

int num_right = 0;
```

Или мы можем определить их в одной строке через запятую:

```
int num_tries = 0, num_right = 0;
```

В общем, лучше придерживаться правила инициализировать объект данных, даже если значение только обозначает, что объект не имеет полезного значения вовсе. Я не инициализировал `usr_val` потому, что значение будет получено непосредственно из пользовательского ввода прежде, чем программа как-то использует объект.

Альтернативный синтаксис для инициализации – использовать так называемый *конструкционный синтаксис*

```
int num_tries(0);
```

Понимаю. Почему есть два инициализационных синтаксиса? Хуже того, почему я сейчас об этом говорю?

Что ж, давайте посмотрим, отвечает ли мое объяснение на оба вопроса.

Использование оператора присваивания для инициализации пришло из языка C. Это хорошо работает с объектами данных встроенных типов и класса объектов, которые могут быть инициализированы единственным значением, таким, как строковый класс:

```
string sequence_name = "Fibonacci";
```

Это не работает так же хорошо с классом объектов, которые требуют нескольких значений для инициализации, как, например, класс стандартной библиотеки комплексных чисел, где каждое требует двух значений: первое для действительной части, второе для мнимой. Альтернативный конструкционный синтаксис был введен для поддержки многозначной инициализации:

```
#include <complex>

complex<double> purei(0, 7);
```

Странная нотация скобок, следующая за `complex`, означает, что класс комплексных чисел – класс шаблонов. Мы узнаем больше о классе шаблонов на протяжении книги. Класс шаблонов позволяет нам определять класс без спецификации типа данных какого-либо или всех членов класса.

Класс комплексных чисел, например, состоит из двух членов объекта данных. Один представляет реальную часть числа, второй мнимую. Эти члены должны быть числами типа данных с плавающей точкой, но какого? C++ поддерживает три типа чисел с плавающей точкой: *единичной точности*, представляемых ключевым словом `float`; *двойной точности*, представленной ключевым словом `double`; и расширенной точности, представленной двумя ключевыми словами `long double`.

Механизм класса шаблонов позволяет программисту откладывать определение типа данных, используя класс шаблонов. Это дает ему возможность вставить «заглушку», которую позднее он заполнит реальным типом данных. В предыдущем примере использовался выбор данных типа класса комплексных чисел `double`.

Понимаю, возможно, появляется больше вопросов, чем получается ответов. Это происходит оттого, что шаблоны, поддерживаемые C++, двух инициализационных синтаксисов для встроенных типов данных. Когда встроенные типы данных и программно-определенный класс типов имеют разный инициализационный синтаксис, невозможно написать шаблон, который поддерживает и встроенный, и класс типа данных. Унификация синтаксиса упрощает разработку шаблонов. К сожалению, раскрытие синтаксиса приводит к появлению еще большей путаницы!

Счет пользователя должен быть значением с плавающей точкой, поскольку может быть нецелое отношение. Мы определим его типом `double`:

```
double usr_score = 0.0;
```

Нам так же нужно сохранить место для ответов пользователя *yes/no: Make another try? Try another sequence?*

Мы можем сохранить ответы пользователя в символьном объекте данных:

```
char usr_more;
cout << "Try another sequence? Y/N? ";
cin >> usr_more;
```

Ключевое слово `char` относится к символьному типу. Символ окаймляется апострофами, обозначая `'a'`, `'7'`, `';` и т.д. Некоторые специальные встроенные символьные обозначения приведены ниже (их иногда называют эскейп последовательности):

```
\n' newline (новая строка) '\t' tab (табуляция) '\0' null (нуль) '\'' single quote (апостроф) '\"' double quote (кавычки) '\\\' backslash (обратная косая черта)
```

Например, для создания перевода на новую строку, и затем табуляции перед вводом имени пользователя, мы можем написать

```
cout << '\n' << '\t' << user_name;
```

Альтернативно, мы можем объединить отдельные символы в строку:

```
cout << "\n\t" << user_name;
```

Обычно мы используем эти специальные символы в буквенной строке. Например, для представления пути к файлу в системе Windows мы нуждаемся в обратных косых чертах:

```
"F:\\essential\\programs\\chapter1\\ch1_main.cpp";
```

C++ поддерживает встроенный булев тип данных для представления значений `true/false`. В нашей программе, например, мы можем определить булев объект для контроля над необходимостью вывода на дисплей следующей числовой последовательности:

```
bool go_for_it = true;
```

Булев объект обозначен ключевым словом `bool`. Он может сохранять два буквенных значения `true` или `false`.

Все определенные объекты данных затем модифицируются в ходе нашей программы. `go_for_it`, например, в конечном счете, установится в `false`. `usr_score` обновляется с каждой попыткой пользователя.

Иногда, однако, нам нужен объект для представления постоянных значений: максимальное число попыток, определяемых для пользователя, например, или значение числа “пи”. Объекты, сохраняющие эти значения, не будут меняться по ходу программы. Как мы можем предотвратить случайное изменение этих объектов? Мы можем заручиться поддержкой языка, объявив эти объекты, как `const`:

```
const int max_tries = 3;
```

```
const double pi = 3.14159;
```

Объекты `const` не могут изменяться после инициализации их значения. Любые попытки переустановить значение объекта `const` приведут к ошибке при компиляции. Например:

```
max_tries = 42; // error: объект const
```

## 1.3 Написание выражений

Встроенные типы данных поддерживаются набором операторов: арифметических, логических, отношения, структурообразующих. Арифметические операторы не имеют особенностей, исключая деление целых и получение остатка.

```
// Арифметические операторы
+ сложение a + b
- вычитание a - b
* умножение a * b
/ деление a / b
% остаток a % b
```

Деление двух целых значений выдает целое. Любой остаток отсекается, и это не округление. Остаток получается с помощью оператора %:

```
5 / 3 результат 1 тогда, как 5 % 3 имеет результат 2
5 / 4 результат 1 тогда, как 5 % 4 имеет результат 1
5 / 5 результат 1 тогда, как 5 % 5 имеет результат 0
```

Когда мы нуждаемся в операторе получения остатка? Представьте, что мы хотим печатать не более восьми слов в линии. Если количество слов меньше восьми, мы выводим пробелы после слов. Если строка из восьми или более слов, мы выводим переход на новую строку (newline). Вот как мы это делаем:

```
const int line_size = 8;

int cnt = 1;

// это выражение выполняется много раз со

// строками представленными разными значениями

// и cnt увеличивается на единицу при каждом проходе...

cout << a_string << (cnt % line_size ? ' ' : '\n');
```

Выражение в скобках за оператором вывода, похоже, не имеет значения для вас, пока вы не освоите условный оператор (? :). Результат выражения – выводится либо пробел, либо переход на новую строку, в зависимости от того, нулевое или ненулевое значение принимает оператор остатка. Посмотрим, какой смысл мы можем ему придать.

Выражение

```
cnt % line_size
```

превращается в ноль, когда cnt достигает значения line\_size; иначе оно не равно нулю. Что это означает для нас? Условный оператор имеет основную форму:

```
expr? execute_if_expr_is_true: execute_if_expr_is_false;
```

```
выражение?выполнять_если_выражение_истинно:выполнять_если_выражение_ложно;
```

Если `expr` (выражение) принимает значение `true`, выражение, следующее за знаком вопроса, выполняется. Если `expr` принимает значение `false`, выполняется выражение, следующее за двоеточием. В нашем случае выражение становится либо пробелом, либо переводом на новую строку для оператора вывода.

Условное выражение обрабатывается, как ложное (`false`), если его значение равно нулю. Любое ненулевое значение принимается, как `true`. В примере, пока `cnt` не делится на восемь, и результат не нулевой, выполняется переход оператора по условию `true`, и печатается пробел.

Оператор структурного присваивания дает краткую нотацию при применении арифметической операции для задания объектов. Например, чтобы не писать

```
cnt = cnt + 2;
```

C++ программист обычно пишет

```
cnt += 2; // прибавить 2 к данному значению cnt
```

Структурное присваивание применимо к каждому арифметическому оператору: `+=`, `-+`, `/=`, и `%=`.

Когда к объекту прибавляется или вычитается 1, C++ программист использует операторы увеличения или уменьшения:

```
cnt++; // добавляет 1 к значению cnt (инкремент)
```

```
cnt--; // вычитает 1 из значения cnt (декремент)
```

Существует префиксная (до) и постфиксная (после) разновидность операторов увеличения и уменьшения. В префиксном написании объект увеличивается или уменьшается на 1 до того, как его значение используется:

```
int tries = 0;
```

```
cout << "Are you ready for try #" << ++tries << "?\n";
```

Здесь `tries` увеличивается на 1 до того, как его значение выводится. В постфиксном написании значение объекта вначале используется в выражении, а затем увеличивается (или уменьшается) на 1:

```
int tries = 1;
```

```
cout << "Are you ready for try #" << tries++ << "?\n";
```

Теперь значение `tries` выводится до того, как оно увеличивается на 1. В обоих примерах значение 1 выводится.

Каждый из операторов сравнения принимает значение `true` или `false`. Их всего шесть операторов:

`==` равно `a == b`

`!=` не равно `a != b`

`<` меньше чем `a < b`

> больше чем a > b

<= меньше чем или равно a <= b

>= больше чем или равно a >= b

Вот пример того, как мы можем использовать оператор равенства (эквивалентности) для проверки ответа пользователя:

```
bool usr_more = true;

char usr_rsp;

// спросим пользователя, хочет ли он продолжить
// прочитаем ответ в usr_rsp
if (usr_rsp == 'N') usr_more = false;
```

Условное выражение if выполняет предложение, следующее за ним, если выражение в скобках принимает значение true. В данном примере usr\_more устанавливается в false, если usr\_rsp равно 'N'. Если usr\_rsp не равно 'N', не делается ничего. «От противного», используя оператор неравенства, можно написать так:

```
if (usr_rsp != 'Y')usr_more = false;
```

Проблема с проверкой usr\_rsp только на значение 'N' в том, что пользователь может ввести ответ в нижнем регистре, т.е. 'n'. Мы должны распознать оба значения. Одна из стратегий – добавить else:

```
if (usr_rsp == 'N')
    usr_more = false;
else if (usr_rsp == 'n')
    usr_more = false;
```

Если usr\_rsp равно 'N', то usr\_more устанавливается в false и больше ничего не делается. Если оно не равно 'N', то выполняется else. Если usr\_rsp равно 'n', то usr\_more принимает значение false. Если же значение usr\_rsp не равно ни тому, ни другому, значение usr\_more не определено.

Одна из распространенных ошибок начинающих программистов в использовании оператора присваивания при проверке равенства, как показано ниже:

```
// раз, и присвоили usr_rsp символ 'N'
// а теперь всегда условие принимает значение true
if (usr_rsp = 'N')

// ...
```

Логический оператор ИЛИ (||) предлагает альтернативный путь проверки истинности во множественных выражениях:

```
if (usr_rsp == 'N' || usr_rsp == 'n')usr_more = false;
```

Логический оператор ИЛИ становится истинным (true), если любое из выражений истинно. Левое выражение проверяется первым. Если оно истинно, следующее выражение не проверяется. В нашем примере usr\_rsp проверяется на равенство 'n', только, если оно не равно 'N'.

Логический оператор И (&&) становится истинным, только если оба выражения истинны. Например,

```
if (password &&validate(password) &&
(acct = retrieve_acct_info(password))
// процесс доступа ...
```

Верхнее выражение проверяется первым. Если оно ложно, оператор И принимает значение ложно, а остальное выражение не проверяется. В данном случае информация доступа принимается только тогда, когда введен пароль, и это правильный пароль.

Логический оператор НЕ (!) принимает значение true, если выражение, которому он принадлежит, имеет значение false. Например, вместо записи

```
if (usr_more == false)

cout << "Your score for this session is " << usr_score << " Bye!\n";
```

мы можем написать

```
if (! usr_more) ...
```

## Оператор предшествования

Есть одна «заморочка» в использовании встроенных операторов – при комбинации нескольких операторов в одном выражении, порядок выполнения операций определяется предустановленным уровнем приоритетности для каждого оператора. Например, результат выражения  $5+2*10$  всегда равен 25 и никогда 70, поскольку оператор умножения имеет больший приоритет, чем оператор сложения. В итоге 2 всегда умножается на 10 прежде, чем складывается с 5.

Мы можем переопределить приоритет, взяв в скобки операцию, с выполнения которой мы хотели бы начать.  $(5+2)*10$ , например, принимает значение 70.

Для операторов, о которых я говорил, предопределенные уровни приоритетности написаны ниже. Оператор, который выше, имеет больший приоритет, чем тот, что ниже. Операторы, расположенные в одну линию, имеют порядок определения слева-направо.

логическое NOT  
арифметическое (\*, /, %)  
арифметическое (+, -)  
отношение (<, >, <=, >=)  
отношение (==, !=)  
логическое AND  
логическое OR  
присваивание

Например, для определения четности `ival` мы можем написать

```
! ival % 2 // не совсем хорошо
```

Наше выражение проверяет результат оператора остатка. Если `ival` четно, результат нулевой и логический оператор НЕ становится истинным. Иначе, если результат ненулевой, логический оператор НЕ принимает значение «ложно». Во всяком случае, так нам хотелось бы.

К сожалению, результат выражения совершенно иной. Наше выражение всегда будет ложным, исключая значение `ival` равное нулю!

Более высокий приоритет оператора логического отрицания приводит к тому, что он выполняется первым. Он действует на `ival`. Если `ival` имеет не нулевое значение, результат ложный, иначе истинный. Полученное значение становится левой частью операции получения остатка. «Ложно» становится «0» при использовании в арифметических операциях, а «истинно» принимает значение «1». В результате порядка выполнения операций выражение превращается в `0%2` для любых значений `ival` за исключением нуля.

Хотя это не то, что мы хотели получить, это и не является ошибкой, по крайности языковой ошибкой. Это лишь неправильное представление нашей задуманной программной логики. Компилятор об этом, конечно, не догадывается. Порядок выполнения одно из того, что затрудняет программирование на C++. Для правильного выполнения выражения мы должны изменить порядок выполнения с помощью скобок:

```
! (ival % 2) // ok
```

Чтобы избежать этих проблем, вы должны сойтись поближе с порядком следования операторов в C++. Я не помощник вам, в том смысле, что раздел не содержит ни полного перечня операторов, ни полного представления порядка следования операций – того, что я показал, должно хватить только для начала. Для более полного знакомства отсылаю вас к Части 4 [LIPPMAN98] или Части 6 [STROUSTRUP97].

## 1.4 Написание условий и создание циклов

По определению, выражения выполняются по разу по мере прохождения программы, начиная с первого выражения `main()`. В предыдущих разделах мы кратко говорили о выражении `if`. Выражение `if` позволяет нам выполнять по условию одно или последовательность выражений, основываясь на вычислении истинности условия. Дополнение `else` позволяет проверить несколько условий. Циклические выражения позволяют выполнять одно или последовательность выражений, основываясь на вычислении истинности выражения. Следующий псевдокод программы использует два цикла (№1 и №2), один условный оператор `if` (№5), один условный оператор `if-else` (№3), и второй условный оператор, называемый `switch` (переключатель) (№4).

```
// Pseudo code: Основная логика программы – пока пользователь хочет угадывать
последовательности
{ #1
вывести на дисплей последовательность
пока угадано неверно, и
пользователь хочет угадать еще раз

{ #2
прочитать гипотезу
увеличить счетчик попыток
если угадано правильно

{ #3
увеличить счетчик угадываний
установить got_it в true

} else {
выразить сожаление по поводу неудачной попытки сделать запрос на основании
текущего номера попыток пользователя
```

```

// #4 спросить пользователя, хочет ли он попытаться еще раз и
//прочитать ответ, если пользователь отвечает нет

// #5 установить go_for_it в false
}}}
```

## Условные выражения

Условия выражения if должны быть записаны в круглых скобках. Если они истинны, выражение, непосредственно следующее за if, выполняется:

```

// #5

if (usr_rsp == 'N' || usr_rsp == 'n') go_for_it = false;
```

Если должно выполниться несколько выражений, они должны быть заключены в фигурные скобки, следующие за if (это называется блоком выражений):

```

//#3
if (usr_guess == next_elem)
{
// начало блока выражений
num_right++;

got_it = true;} // окончание блока выражений
```

Общая ошибка начинающих – забыть отметить блок:

```

// p-раз: пропущена отметка блока

// только num_cor++ относится к if,

// а got_it = true; выполняется вне условия

if (usr_guess == next_elem)

num_cor++;

got_it = true;
```

Выполнение got\_it отражает намерения программиста. К сожалению, оно не отражает поведение программы. Нарастивание num\_cor относится к условному оператору if и выполняется только тогда, когда попытка пользователя эквивалентна значению next\_elem. got\_it, однако, не относится к условному оператору, поскольку мы забыли обозначить два выражения, как блок выражений. got\_it всегда устанавливается в значение «истинно» в этом примере, независимо от того, что делает пользователь.

Условный оператор if так же поддерживает расширение else. Else представляет одно или блок выражений, которые будут выполняться, если условие принимает значение «ложно». Например,

```

if (usr_guess == next_elem)
{
// пользователь угадал
```

```

}
else {
// пользователь не угадал

}

```

Другое использование оператора else – объединить два условных выражения. Например, если попытка пользователя неудачна, мы хотим варьировать наш ответ в зависимости от числа попыток. Мы должны написать три проверки, как независимые условные выражения:

```

if (num_tries == 1) cout << "У-ух! Хорошая попытка, но не совсем.\n";

if (num_tries == 2) cout << "М-да. Извини. Опять неправильно.\n";

if (num_tries == 3) cout << "А-а, это труднее, чем могло показаться, не так ли?\n";

```

Однако только одно из трех условий может быть истинным одновременно. Если одно из условий верно, остальные должны быть ложны. Мы можем отобразить зависимость между выражениями if, объединив их вместе с помощью if-else выражений:

```

if (num_tries == 1) cout << "У-ух! Хорошая попытка, но не совсем\n";

else

    if (num_tries == 2) cout << "М-да. Извини. Опять неправильно.\n";

else

if (num_tries == 3) cout << "А-а, это труднее, чем могло показаться, не так ли?\n";

else cout << "Похоже, дальше бесполезно!\n";

```

Первый условное выражение выполняется. Если оно истинно, выражение, следующее за ним, выполняется, а последовательность else-if, нет. Если же первое условное выражение ложно, выполняется следующее, затем следующее, пока одно из условий не становится истинным, или num\_tries не становится больше 3, все условия ложны, и последнее else выполняется.

Один из неприятных аспектов использования вложенных if-else – трудности с правильной их логической организацией. Например, мы хотели бы использовать наше if-else для разделения логики программы на два случая: когда пользователь угадывает, и когда он не угадывает. Первая попытка не работает совсем, как мы распланировали:

```

if (usr_guess == next_elem)
{
// пользователь угадывает

}

else if (num_tries == 1)

else // ... выводим вопрос

if (num_tries == 2)

```

```

else // ... выводим вопрос

if (num_tries == 3)

else // ... выводим вопрос

// ... выводим вопрос

// теперь спросим у пользователя, хочет ли он еще раз угадать

// но только, если он не угадал

// р-раз! и куда же мы воткнем это?

```

Каждая пара else-if непреднамеренно была сделана альтернативной к удачной попытке. В результате – нам некуда поместить вторую часть нашего кода для поддержки неудачных попыток. Вот правильная организация:

```

if (usr_guess == next_elem){
// пользователь угадал} else { // пользователь не угадал
if (num_tries == 1)
else // ...
    if (num_tries == 2)
else // ...

if (num_tries == 3)
// ...
else // ...

cout << "Желаете попробовать еще? (Y/N) ";
char usr_rsp;
cin >> usr_rsp;

if (usr_rsp == 'N' || usr_rsp == 'n') go_for_it = false;}

```

Если значение проверяемого условия имеет интегральный тип, мы можем заменить набор if-else-if выражением switch:

```

// равноценно if-else-if выше

switch (num_tries)

{
case 1:
cout << "У-ух! Хорошая попытка, но не совсем\n"
break;

case 2:
cout << "М-да. Извини. Опять неправильно.\n";
break;

case 3:
cout << "А-а, это труднее, чем могло показаться, не так ли?\n";
break;

```

```

default:
cout << "Похоже, дальше бесполезно!\n";
break;

}

```

Ключевое слово `switch` с последующим выражением в круглых скобках (да, имя объекта работает, как выражение). Выражение должно вычисляться как целое значение. Серия меток `case`, следующая за ключевым словом `switch`, каждая определяет постоянное выражение. Результат выражения сравнивается с каждой из меток `case` по очереди. Если есть совпадение, выражение, следующее за `case`, выполняется. Если же совпадений нет, и метка `default` присутствует, выполняется выражение, следующее за ней. Если же нет ни совпадений, ни метки `default`, не делается ничего.

Почему я поместил выражение `break` в конце каждой метки `case`? Каждая метка `case` проверяется по очереди на совпадение со значением выражения. Каждая несовпадающая метка `case` пропускается по очереди. Когда обнаруживается совпадение, выражение, следующее за меткой, выполняется. Увы, выполнение продолжается со следующими метками, пока не будет достигнут конец меток. Если значение `num_tries`, например, равно 2, и если нет выражений `break`, вывод будет выглядеть, примерно, так:

```

// output if num_tries == 2 and
// we had forgotten the break statements
М-да. Извини. Опять неправильно.
А-а, это труднее, чем могло показаться, не так ли?
Похоже, дальше бесполезно!

```

Вторая метка `case` совпадает, все метки `case`, следующие за совпадающей меткой `case`, так же выполняются, пока не завершается оператор. Это то, что дает выражение `break`. Почему, вы можете спросить, выражение `switch` так устроено? Вот пример этого провала поведения, который придется в самый раз:

```

switch (next_char)
{
case 'a': case 'A':
case 'e': case 'E':
case 'i': case 'I':
case 'o': case 'O':
case 'u': case 'U':
++vowel_cnt;

break;// ...}

```

## Циклы

Циклы выполняют выражения или блоки выражений до тех пор, пока выражение условия не становится истинным. Наша программа требует двух циклов, один вложен в другой:

```

пока пользователь желает угадывать последовательности
{ вывести на дисплей последовательности
  пока догадка неверна, а пользователь желает угадать еще раз}

```

Цикл `while` в C++ вполне подходит для нашей цели:

```

bool next_seq = true; // показать следующую последовательность?

bool go_for_it = true; // пользователь хочет продолжить угадывание?

bool got_it = false; // пользователь угадал?

int num_tries = 0; // количество попыток

int num_right = 0; // количество правильных ответов

while (next_seq == true){
// вывести на дисплей последовательность для пользователя
    while ((got_it == false) &&(go_for_it == true)){
        int usr_guess;
        cin >> usr_guess;
        num_tries++;

        if (usr_guess == next_elem) {
            got_it = true;
            num_cor++;
        }
        else {
            // пользователь не угадал
            // сказать, что ответ неверен
            // спросить, еще угадываем?
            if (usr_rsp == 'N' || usr_rsp == 'n') go_for_it = false;
        }
    } // конец вложенного цикла while

    cout << "Want to try another sequence? (Y/N) "
    char try_again;
    cin >> try_again;

    if (try_again == 'N' || try_again == 'n')next_seq = false;

} // конец while(next_seq == true)

```

Цикл `while` начинается с вычисления выражения условия в круглых скобках. Если оно истинно, выражение или блок выражений, следующий за циклом `while`, выполняется. После выполнения выражения (блока) выражение условия обновляется. Этот цикл обновления/выполнения продолжается, пока условие не станет ложным. Обычно, некоторое условие внутри исполняемого блока устанавливает условное выражение в состояние ложно. Если выражение условия никогда не становится ложным, мы говорим, что ошибочно попали в *бесконечный цикл*.

Наш внешний цикл `while` выполняется, пока пользователь не скажет, что хочет остановиться:

```

bool next_seq = true;
while (next_seq == true){ // ...

    if (try_again == 'N' || try_again == 'n')next_seq = false;

}

```

Где при инициализации `next_seq` в ложно, блок выражений не будет выполняться. Вложенный цикл `while` позволяет нашему пользователю выполнять множество аналогичных попыток.

Цикл может быть прерван внутри выполняемого блока выполнением выражения `break`. В следующем

фрагменте кода, например, цикл `while` выполняется пока `tries_cnt` не станет равно `max_tries`. Однако, если пользователь дает правильный ответ, цикл прерывается выражением `break`:

```
int max_tries = 3;
int tries_cnt = 0;
while (tries_cnt < max_tries){ // прочитываем попытку пользователя
    if (usr_guess == next_elem)
        break; // прерываем цикл

    tries_cnt++; // more stuff
}
```

Программа может иметь быстрое завершение текущего выполнения цикла, выполнив выражение `continue`. Например, рассмотрим следующий фрагмент программы, в котором все слова большие, чем в четыре буквы, отбрасываются:

```
string word;
const int min_size = 4;
while (cin >> word){
    if (word.size() < min_size)
        // прерываем этот цикл
        continue;
    // попадаем сюда, только если слово
    // больше или равно min-size ...
    process_text(word);}
```

Если слово меньше, чем `min_size`, выполняется выражение `continue`. Результатом выполнения выражения `continue` является прерывание текущего прохождения цикла. Остаток тела цикла `while`, в данном случае `process_text()`, не выполняется. Вернее, цикл начинается заново, с новым вычислением условия, которое прочитывает другое значение строки в `word`. Если `word` больше или равно `min_size`, полное тело цикла выполняется. В этом случае все слова, имеющие больше четырех букв, отбрасываются.

## 1.5 Как использовать массивы и векторы

Ниже приведены первые восемь элементов из шести числовых последовательностей:

Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21

Lucas: 1, 3, 4, 7, 11, 18, 29, 47

Pell: 1, 2, 5, 12, 29, 70, 169, 408

Triangular: 1, 3, 6, 10, 15, 21, 28, 36

Square: 1, 4, 9, 16, 25, 36, 49, 64

Pentagonal: 1, 5, 12, 22, 35, 51, 70, 92

Наша программа должна выводить на дисплей пары элементов из последовательности, и позволить пользователю угадать следующий элемент. Если пользователь угадывает и желает продолжить, программа

должна вывести на дисплей следующую пару элементов, затем третью, и так далее. Как мы можем это сделать?

Если следующая пара берется из той же последовательности, пользователь, разгадавший одну пару, угадает их все. Это не интересно. Так что будем брать следующую пару из другой числовой последовательности при каждом проходе основного цикла программы.

Теперь мы будем выводить на дисплей максимум шесть пар элементов за сессию: по одной паре из каждой из шести последовательностей. Мы постараемся сделать это так, чтобы при выводе на дисплей пары элементов, не зная, из какой числовой последовательности будет взята пара при следующем проходе цикла. Каждый проход должен иметь доступ к трем значениям: паре элементов, и элементу, следующему за ней в последовательности.

Решение, которое мы обсудим в этом разделе, использует контейнерный тип, способный поддерживать смежную последовательность целых значений, которые могут быть доступны не по имени, а по позиции в контейнере. Мы запомнили 18 значений в контейнере, как подборку шести групп: первые два в группе представляют пару для вывода на дисплей, третий представляет следующий элемент в последовательности. При каждом проходе цикла мы добавляем три индексных значения, проходя по шести группам по очереди.

В C++ мы можем определить контейнер, либо как встроенный массив, либо как вектор класса стандартной библиотеки. В основном, я рекомендую использовать класс векторов, а не встроенные массивы. Однако есть резон использовать массив, и важно понять, как использовать оба варианта.

Для определения встроенного массива мы должны обозначить тип элементов массива, дать массиву имя, и обозначить размер – количество элементов, которые массив должен содержать. Размер должен быть константой, т.е. выражением, которое не меняется во время выполнения программы. Например, следующий код объявляет `pell_seq` массивом из 18 целых элементов.

```
const int seq_size = 18;

int pell_seq[seq_size];
```

Для определения объектов класса векторов мы должны вначале включить файл заголовка `vector`. Класс векторов – шаблон, поэтому мы определяем тип элементов в угловых скобках, следующих за именем класса. Размер помещается в круглых скобках, он не обязательно должен быть постоянным выражением. Следующий код определяет `pell_seq`, как объект класса векторов, содержащий 18 элементов типа `int`. По определению каждый элемент инициализируется в 0.

```
#include <vector>

vector<int> pell_seq(seq_size);
```

Мы добираемся до элементов - либо массива, либо вектора – определяя его позицию в контейнере. Этот элемент *индексируется* с использованием оператора списка индексов (`[]`). Одна потенциальная «незадача» в том, что первый элемент находится в позиции 0, а не 1. Последний элемент индексируется на 1 меньше, чем размер контейнера. Для `pell_seq` правильные индексы это от 0 до 17, а не от 1 до 18. (Эта ошибка настолько распространена, что заслуживает иметь собственное имя: печально известная *off-by-one* ошибка). Например, для получения первых двух элементов последовательности Pell'a мы пишем

```
pell_seq[0] = 1; // присваиваем 1 первому элементу
```

```
pell_seq[1] = 2; // присваиваем 2 второму элементу
```

Давайте, вычислим следующие десять элементов последовательности Pell'a. Для *прохождения через* элементы вектора или массива мы обычно используем цикл for, другой базовый цикл C++. Например,

```
for (int ix = 2; ix < seq_size; ++ix)

pell_seq[ix] = pell_seq[ix - 2] + 2*pell_seq[ix - 1];
```

Цикл for состоит из следующих элементов:

```
for (начальное значение; условие; индексация) выражение;
```

Начальное значение выполняется единожды перед выполнением цикла. В нашем примере ix инициализируется как 2 перед началом выполнения цикла.

условие служит для контроля над циклом. Оно вычисляется перед каждой итерацией цикла. Так что, сколько итераций при вычисленном условии равно true, столько раз выражение выполняется. Выражение может быть единственным или блочным. Если первое условие не удовлетворяется, выражение не выполняется никогда. В нашем примере, условие проверяет - ix меньше, чем seq\_size?

индексация вычисляется после каждой итерации цикла. Она обычно используется для модификации начального значения объекта и проверяется в условии. Если первое вычисление условия принимает значение false, индексация не выполняется никогда. В нашем случае ix увеличивается с каждой итерацией цикла.

Для вывода элементов мы проходим через следующие операции:

```
cout << "The first " << seq_size << " elements of the Pell Series:\n\t";

for (int ix = 0; ix < seq_size; ++ix) cout << pell_seq[ix] << ' ';

cout << '\n';
```

Если мы предпочтем, мы можем обойтись без начального значения, индексации, или, менее часто, без условия для цикла for. Например, мы можем переписать предыдущий цикл как

```
int ix = 0; // ...

for (; ix < seq_size; ++ix) // ...
```

Точка с запятой необходима, чтобы показать пустое начальное значение.

Наш контейнер содержит второй, третий и четвертый элементы каждой из шести последовательностей. Как мы заполним контейнер подходящими значениями? Встроенный массив может специфицироваться инициализационным списком, содержащим список значений, разделенных запятыми для всех или подмножества его элементов:

```
int elem_seq[seq_size] = {

1, 2, 3, // Fibonacci

3, 4, 7, // Lucas
```

```

2, 5, 12, // Pell

3, 6, 10, //Triangular

4, 9, 16, // Square

5, 12, 22 // Pentagonal

};

```

Количество значений, записанных в список инициализации, не должно превышать размера массива. Если мы предлагаем меньше значений, чем размер массива, недостающие элементы инициализируются как 0. Если мы хотим, мы можем позволить компилятору вычислить размер массива на основании количества значений, которые мы включаем в список:

```

// компилятор рассчитает размер в 18 элементов
int elem_seq[] = {1, 2, 3, 3, 4, 7, 2, 5, 12,3, 6, 10, 4, 9, 16, 5, 12, 22};

```

Класс векторов не поддерживает явно список инициализации. Несколько нудным решением будет присвоение значения каждому элементу отдельно:

```

vector<int> elem_seq(seq_size);
elem_seq[0] =1;
elem_seq[1] =2;
// ...
elem_seq[17] =22;

```

Еще одна альтернатива инициализировать встроенный массив и использовать его для инициализации вектора:

```

int elem_vals[seq_size] = {1, 2, 3, 3, 4, 7, 2, 5, 12,3, 6, 10, 4, 9, 16, 5, 12,
22 };

// инициализация elem_seq значениями elem_vals

vector<int> elem_seq(elem_vals, elem_vals+seq_size);

```

`elem_seq` получает два значения. Эти значения в действительности адресованные. Они отмечают диапазон элементов, с которым вектор будет инициализирован. В этом случае мы отметили 18 элементов, содержащихся в `elem_vals`, копируемые в `elem_seq`. В Части 3 мы увидим в деталях, как это работает.

А теперь давайте посмотрим, как мы можем использовать `elem_seq`. Одно различие между встроенными массивами и классом векторов в том, что вектор знает свой размер. Наш предыдущий цикл `for` проходил через встроенный массив, посмотрим, велика ли разница при использовании вектора:

```

// elem_seq.size() возвращает число элементов
// содержащихся в векторе elem_seq

cout << "The first " << elem_seq.size() << " elements of the Pell Series:\n\t";

for (int ix = 0; ix < elem_seq.size(); ++ix)

cout << pell_seq[ix] << ' ';

```

`cur_tuple` представляет наш индекс в текущей последовательности, выводимой на дисплей. Мы инициализируем его в 0. С каждым проходом цикла мы добавляем 3 в `cur_tuple`, устанавливая его для индексации первого элемента следующей последовательности для вывода на дисплей.

```
int cur_tuple = 0;

while (next_seq == true &&cur_tuple < seq_size) {
    cout << "The first two elements of the sequence are: "
         << elem_seq[cur_tuple] << ", "
         << elem_seq[cur_tuple + 1]
         << "\nWhat is the next element? ";

    // ...
    if (usr_guess == elem_seq[cur_tuple + 2])
        // правильно!

    // ...

    if (usr_rsp == 'N' || usr_rsp == 'n')next_seq = false;

    else cur_tuple += 3;

}
```

Полезно было бы сохранять путь к последовательности, которая в настоящий момент активна. Давайте запомним имя каждой последовательности, как строку:

```
const int max_seq = 6;
string seq_names[max_seq] =
{"Fibonacci", "Lucas", "Pell", "Triangular", "Square", "Pentagonal"};
```

Мы можем использовать `seq_names` следующим образом:

```
if (usr_guess == elem_seq[cur_tuple + 2]) {
    ++num_cor;
    cout << "Very good. Yes, "
         << elem_seq[cur_tuple + 2]
         << " is the next element in the "
         << seq_names[cur_tuple / 3] << "sequence.\n";
}
```

Выражение `cur_tuple/3` получает по очереди 0, 1, 2, 3, 4 и 5 индексируя в массиве строк элементы, которые идентифицируют активную последовательность.

## 1.6 Указатели дают больше гибкости

Наше решение по выводу на дисплей в предыдущей секции имеет два основных недостатка. Во-первых, оно ограничено выводом шести числовых последовательностей – если пользователь угадает все шесть, программа сразу завершается. Во-вторых, она всегда выводит те же самые шесть пар элементов, в той же последовательности. Как мы можем увеличить гибкость программы?

Одно из возможных решений – создать шесть векторов, по одному на каждую последовательность, рассчитанных на одинаковое количество элементов. На каждом проходе цикла мы выбираем наши пары

элементов из разных векторов. При повторном использовании вектора, мы возьмем нашу пару элементов из другой части вектора. Это приблизит нас к устранению обоих отмеченных недостатков.

Как и в предыдущем решении, нам бы хотелось иметь «прозрачный» доступ к разным векторам. В предыдущем разделе мы достигали «прозрачности» за счет доступа по индексам, а не по имени. На каждом проходе цикла, мы увеличивали значение индекса на 3. Сам код оставался неизменным.

В этом разделе мы добьемся «прозрачности» тем, что будем обращаться к вектору косвенно, через указатель, вместо обращения по имени. *Указатель* вносит определенный уровень косвенности в программу. Вместо того чтобы работать с объектом напрямую, мы будем работать с указателем, сохраняющим адрес объекта. В нашей программе мы определим указатель, который будет адресоваться к вектору чисел целого типа. При каждой итерации цикла мы будем модифицировать указатель, адресуясь к разным векторам. Фактический код для манипуляций с указателями не изменится.

Использование указателей дает два преимущества нашей программе. Оно увеличивает гибкость программы и добавляет уровень гибкости, отсутствующий при прямой работе с объектом. Этот раздел убедит вас в правдивости обоих утверждений.

Мы уже знаем, как определять объект. Следующее выражение, например, определяет `ival`, как объект целого типа `int`, инициализированного значением 1024:

```
int ival = 1024;
```

Указатель сохраняет адрес объекта данного типа. Для определения указателя данного типа, мы добавляем к имени типа звездочку:

```
int *pi; // pi указатель на объект типа int
```

`pi` указатель на объект типа `int`. Как мы должны инициализировать его для указания на `ival`? Определением имени объекта, примерно так

```
ival; // определяет значение ival
```

определяет ассоциированное значение, в нашем случае 1024. Для получения адреса объекта, а не его значения, мы добавляем оператор адресации (`&`):

```
&ival; // определяет адрес ival
```

Для инициализации `pi`, как адреса `ival`, мы запишем следующее:

```
int *pi = &ival;
```

Для доступа к объекту, адресуемому указателем, мы должны *разыменовать* указатель, это даст содержимое объекта по адресу, содержащемуся в указателе. Чтобы это сделать, мы добавляем звездочку к указателю, следующим образом:

```
// разыменовываем pi для доступа к объекту им адресуемому  
if (*pi != 1024) // читаем
```

```
*pi = 1024; // пишем
```

Сложность инициализации с использованием указателя, как вы видите, исходит от запутывающего синтаксиса. Виной тому двойственная природа указателя. Мы можем манипулировать адресом, содержащимся в указателе, а можем манипулировать объектом, на который указывает указатель. Когда мы пишем

```
pi; // определяет адрес сохраняемый в pi
```

мы управляем фактически указателем объекта. А когда мы пишем

```
*pi; // определяет значение объекта адресуемого pi
```

мы управляем объектом, адресуемым pi.

Вторая сложность в понимании указателей – возможность отсутствия адресуемого объекта. Например, когда мы пишем \*pi, это может быть, или может не быть причиной краха программы при выполнении! Если pi адресуется к объекту, наше разыменовывание pi работает совершенно правильно. Если pi не адресуется к объекту, наша попытка разыменить pi влечет непредсказуемое поведение программы во время выполнения. Это означает, что когда мы используем указатель, мы должны быть уверены, что он адресуется к объекту, прежде чем мы сделаем попытку разыменить его. Как мы это делаем?

Указатель, который не адресуется к объекту, имеет адресуемое значение 0 (иногда его называют нулевым указателем). Любой тип указателя может быть инициализирован, или определен со значением 0.

```
// инициализация каждого указателя без адресации к объекту
```

```
int *pi = 0;
```

```
double *pd = 0;
```

```
string *ps = 0;
```

Для защиты от разыменовывания нулевого указателя, мы проверяем указатель, чтобы убедиться, что его адресуемое значение не нулевое. Например,

```
if (pi && *pi != 1024) *pi = 1024;
```

Выражение

```
if (pi && ...)
```

становится истинным, только если pi содержит адрес иной, чем 0. Если оно ложно, оператор И не выполняется во втором выражении. Для проверки, что указатель нулевой, мы обычно пользуемся оператором логического отрицания НЕ:

```
if (! pi) // истинно, если pi установлено в 0
```

А вот наши шесть объектов векторов последовательностей:

```
vector<int> fibonacci, lucas, pell, triangular, square, pentagonal;
```

На что похож указатель вектора объектов целого типа? Что ж, в общем, указатель имеет такую форму:

```
(тип_объекта_указывающего_на * имя_указателя_объекта)
```

```
type_of_object_pointed_to * name_of_pointer_object
```

Наш указатель адресует тип `vector<int>`. Назовем его `pv` и инициализируем нулем:

```
vector<int> *pv = 0;
```

`pv` может адресоваться к каждому вектору последовательности по очереди. Конечно, мы можем определить `pv` адресацией к каждой последовательности:

```
pv = &fibonacci; // ...  
  
pv = &lucas;
```

Но этим приносится в жертву «прозрачность» кода. Альтернативное решение – запомнить адреса каждой из последовательностей в векторе. Этот прием позволяет нам добраться до них «прозрачно» через индекс:

```
const int seq_cnt = 6;  
  
// массив seq_cnt указателей на  
// объекты типа vector<int>  
  
vector<int> *seq_addrs[seq_cnt] = {&fibonacci, &lucas, &pell,  
&triangular, &square, &pentagonal};
```

`seq_addrs` это встроенный массив элементов типа `vector<int>*`. `seq_addrs[0]` содержит адрес `fibonacci` вектора, `seq_addrs[1]` адрес `lucas` вектора и т.д. Мы используем это для доступа к различным векторам через индекс, а не по имени:

```
vector<int> *current_vec = 0;  
  
// ...  
  
for (int ix = 0; ix < seq_cnt; ++ix) {  
    current_vec = seq_addrs[ix];  
    // вывод на дисплей всех элементов осуществляется  
    // косвенно через current_vec  
  
}
```

Оставшейся проблемой с задуманной реализацией является полная предсказуемость. Последовательности всегда `Fibonacci`, `Lucas`, `Pell` ... Мы бы хотели сделать вывод на дисплей наших последовательностей случайным. Это возможно с использованием стандартной библиотеки языка C функциями `rand()` или `srand()`:

```
#include <cstdlib>  
  
srand(seq_cnt);  
seq_index = rand() % seq_cnt;  
current_vec = seq_addrs[seq_index];
```

`rand()` и `srand()` функции стандартной библиотеки, которые поддерживают псевдо-случайную генерацию. `srand()` активизирует генератор с его параметрами. Каждый вызов `rand()` возвращает целое значение в

диапазоне от 0 до максимального целого значения, представленного `int`. Мы должны это ограничить числами от 0 до 5, чтобы они были правильными индексами `seq_addrs`. Оператор остатка (%) гарантирует нам индексацию между 0 и 5. Файл заголовка `cstdlib` содержит объявление обеих функций.

Мы сохраняем указатель на класс объекта несколько иначе, чем делали это с указателем на объект встроенного типа. Это потому, что класс объекта имеет связанное с ним множество операций, которые мы можем захотеть вызвать. Например, для проверки, является ли первый элемент вектора `fibonacci` единицей, мы можем написать

```
if (! fibonacci.empty() && (fibonacci[1] == 1))
```

Как мы могли бы осуществить ту же проверку через `pv`? Объединение `fibonacci` и `empty()` через точку называется *оператором выбора члена*. Оно используется для выбора операций класса через объект класса. Для выбора операции класса через указатель используем *оператор-стрелку (->) выбора члена*:

```
! pv->empty()
```

Поскольку указатель может адресоваться к отсутствующему объекту, перед тем, как мы используем `empty()` через `pv`, мы должны проверить, что адресация не нулевая:

```
pv && ! pv->empty()
```

Окончательно для вызова оператора индексов мы должны разыменовать `pv`. (Понадобятся дополнительные скобки вокруг разыменованного `pv` из-за более высокого приоритета индексного оператора).

```
if (pv && ! pv->empty() && ((*pv)[1] == 1))
```

Мы еще раз обратимся к указателям в разделе “Библиотеки стандартных шаблонов” Часть 3, и в Части 6, где мы сконструируем и внедрим класс двоичного дерева. Для более глубокого знакомства с указателями обратитесь к Разделу 3.3 of [LIPPMAN98].

## 1.7 Запись и чтение файлов

Если пользователю случится запустить нашу программу повторно, было бы здорово, если бы счет сохранялся для обеих сессий. Чтобы это стало возможным, мы должны (1) записать имя пользователя и данные сессии в файл в конце сессии (2) прочитать данные предыдущей сессии в программу при ее повторном запуске. Посмотрим, как мы можем это сделать.

Для чтения и записи в файл, мы должны включить файл заголовка `fstream` :

```
#include <fstream>
```

Чтобы открыть файл для вывода, мы определяем объект класса `ofstream` (an output file stream – поток вывода файла), передавая его имя в открываемый файл:

```
// seq_data.txt открыт на вывод  
  
ofstream outfile("seq_data.txt");
```

Что происходит, когда мы объявляем `outfile`? Если он не существует, он создается и открывается на вывод. Если же он существует, он открывается на вывод и все данные, которые в нем содержатся, игнорируются.

Если мы хотим добавить, а не замещать данные в существующем файле, мы должны открыть файл в режиме добавления.

Мы делаем это, передавая второе значение `ios_base::app` объекту `ofstream`. (В этом месте книги, будет лучше, если вы будете использовать это, не вдаваясь слишком глубоко в вопрос, что это такое!)

```
// seq_data.txt открывается в append mode (режим добавления)

// новые данные добавляются в конец файла

ofstream outfile("seq_data.txt", ios_base::app);
```

Файл может не открыться. Прежде, чем записывать в него, мы должны убедиться, что он открылся успешно. Простейший путь для проверки этого, проверить истинность объекта класса:

```
// если outfile определяется, как false,

// файл не может быть открыт

if (! outfile)
```

Если файл не может быть открыт, объект класса `ofstream` становится ложным. В этом примере мы предупредим пользователя, выводом сообщения `cerr`. `cerr` представляет *стандартную ошибку*. `cerr`, подобно `cout`, выводится на терминал пользователя. Разница в том, что вывод `cerr` не буферизуется, оно выводится сразу на терминал.

```
if (! outfile)
// по какой-то причине не открывается ...
cerr << "Бах! Не могу сохранить данные сессии!\n";

else
// ok: outfile открыт, давайте писать данные
outfile << usr_name << ' '
        << num_tries << ' '
        << num_right << endl;
```

Если файл открывается успешно, мы непосредственно выводим в него данные, наподобие того, как делаем это для объектов класса `ostream` `cout` и `cerr`. В этом примере мы пишем три значения в `outfile`, последние два отделены пробелами. `endl`- предопределенный манипулятор, поставляемый библиотекой `iostream`.

Манипулятор выполняет несколько операций с `iostream`, отличных от записи и чтения данных. `endl` вставляет символ перевода на новую строку, и затем сбрасывает на диск выходной буфер. Другие предопределенные манипуляторы включают `hex`, отображающий на дисплее целое число в шестнадцатеричном виде, `oct`, который отображает целое в восьмеричном виде, и `setprecision(n)`, который устанавливает точность отображения чисел с плавающей точкой в `n`. (Для полного знакомства с манипуляторами `iostream` обратитесь к Разделу 20.9 [LIPPMAN98]).

Для того чтобы открыть файл на ввод, мы определяем объект класса `ifstream` (an input file stream – поток ввода в файл), передавая ему имя файла. Если файл не может быть открыт, объект класса `ifstream` определяется как ложный. Иначе, файл позиционируется в начало данных, записанных в него.

```

// infile открыт в output mode
ifstream infile("seq_data.txt");
int num_tries = 0;
int num_cor = 0;

if (!infile){
    // по какой-то причине файл не открывается ...
    // мы будем предполагать, что это новый пользователь ...}
else {
    // ok: читаем каждую линию входного файла
    // смотрим, играл ли пользователь раньше ...
    // формат каждой линиии:
    // name num_tries num_correct
    // nt: количество попыток
    // nc: количество отгадываний

    string name;
    int nt;
    int nc;

    while (infile >> name){
        infile >> nt >> nc;
        if (name == usr_name) {
            // нашли!
            cout << "С возвращением, " << usr_name<< "\nВаш текущий счет " << nc << "
out of " << nt << "\nУдачи!\n";

            num_tries = nt; num_cor = nc;}}}}

```

Каждый проход цикла `while` прочитывает новую линию файла, пока не будет достигнут *конец-файла*. Когда мы пишем

```
infile >> name
```

возвращаемое значение входного выражения – объект класса, из которого мы читаем— `infile` в данном случае. Когда *конец-файла* достигнут, условие “истинно” объекта класса сменяется на “ложно”. Это причина, по которой условное выражение цикла `while` прерывается, когда достигается *конец-файла*:

```
while (infile >> name)
```

Каждая линия файла содержит строку, за которой следуют два целых в форме

```
anna 24 19
```

```
danny 16 12 ...
```

Выражение

```
infile >> nt >> nc;
```

читает по очереди количество попыток пользователя в `nt` и количество угадываний в `nc`.

Если мы хотим и читать из, и писать в тот же самый файл, мы определяем объект класса `fstream`. Для открывания его в режиме дополнения, мы должны передать второе значение в форме

```
ios_base::in|ios_base::app:5

fstream iofile("seq_data.txt",ios_base::in|ios_base::app);

if (! iofile)

// файл не открывается по какой-то причине ... гад!

{ // переходим к началу файла для начала чтения

iofile.seekg(0);

// ok: все остальное без изменений ...}
```

Когда мы открываем файл в режиме добавления, текущая позиция – конец файла. Если мы пытаемся читать файл без перепозиционирования, мы просто получаем *конец-файла*. Оператор `seekg()` возвращает `iofile` к началу файла. Поскольку он открыт в режиме дополнения, любая операция записи добавляет данные в конец файла.

Библиотека `iostream` весьма функциональна, есть много деталей, для которых у меня нет места здесь. Для детального знакомства с библиотекой `iostream` смотрите Часть 20 [LIPPMAN98] или Часть 21 [STROUSTRUP97].

## Упражнение 1.5

Напишите программу, которая спрашивает у пользователя его или ее имя. Прочитайте ответ. Подтвердите, что ввод, по крайней мере, имеет длину в два символа. Если имя представляется правильным, ответьте пользователю. Сделайте два варианта: первый для строки в C-стиле, второй с использованием объекта строкового класса.

## Упражнение 1.6

Напишите программу для чтения в стандартный ввод из последовательности целых чисел. Расположите значения по очереди во встроенный массив и в вектор. Пройдитесь по контейнерам, суммируя значения. Выведите на дисплей сумму и среднее введенных значений через стандартный вывод.

## Упражнение 1.7

Используя ваш любимый редактор, запишите две или более линий текста в файл. Напишите программу, для открытия файла, чтения каждого слова в объект `vector<string>`. Пройдите по вектору, отображая его в `cout`. Когда сделаете, рассортируйте слова, используя общий алгоритм `sort()`,

```
#include <algorithm>

sort(container.begin(), container.end());
```

Потом запишите рассортированные слова в выходной файл.

---

<sup>5</sup> Если я предпочел не объяснять элемент `ios_base::app` ранее, не вижу, почему должен делать это теперь! Смотрите Раздел 20.6 [LIPPMAN98] с полными объяснениями и детальными примерами.

## Упражнение 1.8

Выражение `switch` Раздела 1.4 отображает разные утешающие сообщения, основанные на количестве неправильных ответов. Замените это массивом из четырех строк сообщений, которые индексировались бы, основываясь на количестве неправильных ответов.

## Часть 2. Процедурное программирование

Написание всей программы в разделе `main()`, как мы делали это в Части 1, не слишком практично, исключая маленькие программы, написанные для автономного использования. Обычно мы выделяем общие операции, такие как вычисление элементов последовательности Фибоначчи или генерация случайных чисел, в независимые функции. Это дает три важных преимущества. Во-первых, упрощается понимание программы, поскольку она выглядит, как последовательность вызовов функций, а не последовательность кодов, написанных для каждой операции. Во-вторых, мы можем использовать функции в дальнейшем в других программах. И, наконец, в-третьих, легче распределить работу между несколькими программистами или группами внутри проекта.

Эта часть описывает основные правила для написания независимых функций. Так же кратко обсуждаются прегзаружаемые функции и функции шаблонов, иллюстрируется использование указателей на функции.

### 2.1 Как писать функции

В этом разделе мы напишем функцию, которая возвращает элемент Фибоначчи в позицию, определенную пользователем. Например, если пользователь спрашивает: “Какой восьмой элемент Фибоначчи?”, - наша функция отвечает: “21”. Как мы начнем определение этой функции?

Мы должны определить следующие четыре части нашей функции:

- 1. Возвращаемый тип функции.** Наша функция возвращает значение элемента в определенной пользователем позиции. Значение элемента типа `int`, так что возвращаемое значение также типа `int`. Функция, которая не возвращает значения, имеет возвращаемый тип `void`. Для функции, которая выводит последовательность Фибоначчи на терминал, например, предпочтительнее определить возвращаемый тип как `void`.
- 2. Имя функции.** `foo()` это общее имя. Это не самое лучшее имя, поскольку никак не помогает понять смысл операции, осуществляемой функцией. `fibon_elem()` несколько лучше, хотя, вероятно, вы сможете придумать что-нибудь и получше.
- 3. Список параметров функции.** Параметры функции создаются как место для значений, которые позже наполняются пользователем при каждом обращении к функции. Параметр, по существу, определяет некую переменную при каждом вызове функции. Наша функция, например, определяет один параметр – позицию элемента в последовательности. Пользователь определяет эту позицию всякий раз, когда обращается к функции. С параметром связано и имя, и тип. Для нашей функции мы определим один параметр типа `int`. Функция может иметь пустой список параметров. Например, функция, которая приветствует пользователя и читает имя пользователя, не похоже, чтобы имела параметры.
- 4. Тело функции.** Тело функции вводит логику операции. Обычно оно манипулирует с названными параметрами функции. Тело функции заключается в фигурные скобки и располагается за списком параметров.

Прежде, чем функция может быть вызвана из программы, она должна быть объявлена. Объявление функции позволяет компилятору проверить корректность ее использования – достаточно ли параметров, правильного ли они типа, и т.д. **Объявление** функции определяет возвращаемый тип, имя, и список параметров, но не тело функции. Это называется *прототипом функции*.

```
// объявление нашей функции
int fibon_elem(int pos);
```

**Определение** функции состоит из прототипа плюс тело функции. Чтобы вернуть позицию элемента, `fibon_elem()` должна вычислить это значение. Вот одна из возможных реализаций. (Такая пара `/*,*/` используется в качестве многострочного комментария. Все, что находится за `/*` вплоть до `*/`, рассматривается, как комментарий).

```
/* Вторая форма написания комментария
 * 1й и 2й элементы последовательности Fibonacci
 * это 1; каждый последующий - сумма
 * предыдущих двух
 *
 * elem: сохраняет возвращаемое значение
 * n_2, n_1: сохраняют предыдущие значения
 * pos: позиция элемента, запрошенная пользователем */

int elem = 1; // содержит возвращаемое значение
int n_2 = 1, n_1 = 1;
for (int ix = 3; ix <= pos; ++ix){
    elem = n_2 + n_1;
    n_2 = n_1; n_1 = elem;}

```

Если пользователь запрашивает первый или второй элемент, тело цикла `for` не выполняется. `elem` инициализировано в 1, которая имеет правильное значение для возврата. Если пользователь запрашивает третью позицию, цикл вычисляет каждое значение, пока `ix` не достигает `pos`. `elem` содержит значение элемента в позиции `pos`.

Для возврата значения из функции мы используем выражение `return`. Для нашей функции выражение `return` выглядит, примерно, так:

```
return elem;
```

Если мы желаем быть уверенными, что наш пользователь не сделает ошибок, и, если мы хотим вычислять любые позиции в последовательности Фибоначчи, как бы велика она ни была, тогда мы “приплыли” - к сожалению, если мы игнорируем оба этих случая, наша функция, похоже, в какой-то момент “рухнет”.

Какую ошибку может сделать пользователь? Он может ввести неправильную позицию – возможно, нулевое или отрицательное число. Если пользователь сделает это, `fibon_elem()` возвращает 1, а это неверно. Так что, проверим эту возможность:

```
// проверим неправильную позицию
if (pos <= 0)
    // ок, а что дальше?
```

Что мы должны сделать, если пользователь запрашивает неверную позицию? Самое радикальное – прервать выполнение программы. Функция `exit()` стандартной библиотеки делает как раз это. Мы передаем ей значение, и оно означает статус выхода из программы:

```
// прервать программу со статусом exit равным -1

if (pos <= 0) exit(-1);
```

Для использования `exit()` мы должны включить `cstdlib` файл заголовка:

```
#include <cstdlib>
```

Получив нашу маленькую функцию, прерывающую программу при вводе, возможно, мы не слишком правы. Альтернативой было бы устроить исключение, определяющее, что `fibon_elem()` получила неправильное значение позиции. К сожалению, поддержка исключений не обсуждается до Части 7, так что это решение пока не приемлемо.

Мы можем вернуть 0 и доверить пользователю определить, что ноль – неверное значение в последовательности Фибоначчи. В общем, однако, доверять – это не правило инженерии. Более подходящим выбором станет изменение нашего возвращаемого значения для определения, может ли `fibon_elem()` вычислить значение:

```
// проверяемый прототип функции
bool fibon_elem(int pos, int &elem);
```

Функция может вернуть только одно значение. В этом случае возвращаемое значение может быть либо истинным, либо ложным, в зависимости от того, сможет ли `fibon_elem()` вычислить значение элемента. А это оставляет нам проблему возвращения актуального значения элемента. В нашем пересмотренном прототипе функции мы решаем эту проблему добавлением второго параметра с типом обозначенным, как `int`. Это дает нам возврат из функции двух значений. Я объясню разницу в записи параметров `pos` и `elem` в следующем разделе. Идея несколько путанная, так что лучше разъяснить ее в ее собственном разделе.

Что произойдет, если пользователь запросит значение элемента в позиции 5000? Это большое число. Когда я попробовал вычислить его, то получил следующий ответ:

```
element # 5000 is -1846256875
```

Это неправильно. А что произошло? Я вызвал ее с неподдерживаемыми возможностями компьютера. Каждый арифметический тип может быть представлен только минимальным и максимальным значением из области возможных значений.<sup>6</sup>

`int`, например, знаковый тип. Результат `fibon_elem()` *перекрывает* максимальное положительное значение, которое может быть представлено. Когда я заменил тип `elem` с `int` на `unsigned`, ответ изменился

```
#include <limits>
int max_int = numeric_limits<int>::max();
double min_dbl = numeric_limits<double>::min();
```

```
element # 5000 is 2448710421
```

Что будет, если пользователь запросит позицию 10000? Или 100000? Миллион? Последовательность Фибоначчи бесконечна. Наша же реализация, однако, должна устанавливать конечную точку для значений позиционирования, которую мы хотели бы поддерживать. Какое нам принять решение о конечной точке? Все это зависит от запросов наших пользователей. Для моих целей в книге я установил случайный, но обоснованный верхний предел в 1024 (это позволяет работать `elem` в области `int`).

---

<sup>6</sup> Для определения минимального и максимального значения для обычных типов мы должны обратиться к стандартной библиотеке класса числовых ограничений (более подробно смотрите в [STROUSTRUP97]).

Вот как выглядит `fibon_elem()` в окончательном варианте:

```
bool fibon_elem(int pos, int &elem){
// проверим, правильна ли позиция ...
if (pos <= 0 || pos > 1024)
{ elem = 0; return false; }

// elem равна 1 для позиций 1 и 2
elem = 1; int n_2 = 1, n_1 = 1;

for (int ix = 3; ix <= pos; ++ix) {
elem = n_2 + n_1;
n_2 = n_1; n_1 = elem;}

return true;}
```

Эта маленькая программа выполняет `fibon_elem()`:

```
#include <iostream>
using namespace std;
// следующее объявление fibon_elem()
// делает функцию узнаваемой компилятором ...
bool fibon_elem(int, int&);

int main(){
int pos;
cout << "Пожалуйста, введите позицию: ";
cin >> pos;

int elem;
if (fibon_elem(pos, elem))
cout << "элемент № " << pos
<< " is " << elem << endl;
else cout << "Извините. Не могу вычислить элемент № "
<< pos << endl;}
```

В нашем примере объявления `fibon_elem()` не задано имен для ее двух параметров. Это хорошо. Имя параметра необходимо только, если мы хотим получить доступ к параметру внутри функции. Некоторые авторы рекомендуют всегда обозначать имена параметров, как форму документирования. В такой маленькой программе, как эта, пользы от этого, пожалуй, мало.

Когда программа скомпилирована и выполняется, вывод выглядит, примерно, так (ввод отмечен жирным шрифтом):

```
Пожалуйста, введите позицию: 12

элемент № 12 это 144
```

Функция, которая объявляется, как возвращающая не `void` тип, должна возвращать значение в каждой ее точки выхода. Каждое выражения возврата внутри функции представляет явную точку выхода. Не явная точка выхода следует за последним выражением в теле функции, если нет выражения `return`. Следующее определение `print_sequence()`, например, не будет компилироваться, поскольку его неявная точка выхода не имеет выражения `return`.

```

bool print_sequence(int pos) {
    if (pos <= 0 || pos > 1024){
        cerr << "invalid position: " << pos
            << " -- cannot handle request!\n";
        return false;}
    cout << "The Fibonacci Sequence for "

        << pos << " positions: \n\t";

    // выводит 1 1 для всех значений, исключая pos == 1
    switch (pos){

        default:
        case 2:
            cout << "1 ";
            // no break;

        case 1:
            cout << "1 ";
            break;}

    int elem;
    int n_2 = 1, n_1 = 1;
    for (int ix = 3; ix <= pos; ++ix){
        elem = n_2 + n_1;
        n_2 = n_1; n_1 = elem;

        // print 10 elements to a line
        cout << elem << (!(ix % 10) ? "\n\t" : " ");}
    cout << endl;

    // компилятор генерирует здесь ошибку:

    // нужна точка выхода ... нет выражения return!}

```

`print_sequence()` имеет две точки выхода, но мы определили только одну – выражением `return`. Неявная точка выхода следует за последним выражением. Наше встроенное возвращаемое значение (`return`) предназначено только для случая неправильного позиционирования! Бах! К счастью компилятор отслеживает это и расценивает как ошибку. Мы должны добавить `return true;` в качестве последнего выражения.

Для проверки этого я добавил вызов `print_sequence()`, следующий за вызовом `fibon_elem()`, в предыдущую программу `main()`. При компиляции и выполнении нашей `main()` программы осуществляется следующий вывод:

```

Please enter a position: 12
element # 12 is 144
The Fibonacci Sequence for 12 positions:
1 1 2 3 5 8 13 21 34 55 89 144

```

Вторая форма выражения возврата не возвращает значения. Она используется только с функциями, использующими возвращаемый тип `void`. И используется для преждевременного завершения работы функции.

```

void print_msg(ostream &os, const string &msg){

```

```

if (msg.empty())
// нечего возвращать; прерываем функцию ...
return;

os << msg;}

```

В этом примере последнее выражение возврата излишне, поскольку нет значения, подлежащего возврату. Хватило бы неявной точки возврата.

## Упражнение 2.1

`main()`, представленная ранее, позволяет пользователю ввести только одно значение позиции, а затем завершается. Если пользователь хотел бы запросить две или более позиций, он должен был бы выполнить программу два или более раз. Модифицируйте `main()` так, чтобы разрешить пользователю находится в режиме выбора позиции, пока он не скажет, что хотел бы остановиться.

## 2.2 Вызов функции

В этой секции мы реализуем функцию для сортировки вектора целых значений таким образом, чтобы мы могли поэкспериментировать с поведением передаваемых параметров **по ссылке**, и **по значению**. Алгоритм сортировки – простейшая пузырьковая сортировка, построенная на двух встроженных циклах `for`. Внешний цикл проходит через элементы вектора от `ix`, начинающегося с 0 и до конца `size-1`. Идея в том, что по завершении каждого прохода внешнего цикла, элемент, индексируемый `ix`, находится на своем месте. Когда `ix` равно 0, наименьший элемент найден и помещен в позицию 0, а когда `ix` равно 1, второй из наименьших элементов на его месте и т.д. Перемещение выполняется внутренним циклом `for`. `jx` начинается с `ix+1` и заканчивается `size-1`. Сравнивается значение элемента в `ix` со значением элемента в `jx`. Если значение элемента в `jx` меньше, два значения элемента меняются местами. Наше первое выполнение рухнуло. Цель этого раздела объяснить, почему. Начнем.

```

void display(vector<int> vec){
    for (int ix = 0; ix < vec.size(); ++ix)
        cout << vec[ix] << ' ';
    cout << endl;}

void swap(int val1, int val2){
    int temp = val1; val1 = val2; val2 = temp;}

void bubble_sort(vector<int> vec){
    for (int ix = 0; ix < vec.size(); ++ix)
        for (int jx = ix+1; jx < vec.size(); ++jx)
            if (vec[ix] > vec[jx]) swap(vec[ix], vec[jx]);}

int main(){
    int ia[8] = { 8, 34, 3, 13, 1, 21, 5, 2 };

```

```

vector<int> vec(ia, ia+8);

cout << "vector before sort: ";
display(vec);

bubble_sort(vec);

cout << "vector after sort: ";

display(vec);}

```

Когда эта программа компилируется и выполняется, генерируется следующий вывод, показывающий, что вектор, определенный в `main()` не сортируется:

```

vector before sort: 8 34 3 13 1 21 5 2

vector after sort: 8 34 3 13 1 21 5 2

```

Нет ничего удивительного в программе, которая не работает при первом запуске. Вопрос в том, что мы теперь сделаем?

Если в нашем распоряжении есть отладчик, хорошо бы пройти по шагам выполнение программы, проверяя по ходу значения различных объектов в нашей программе, и наблюдая изменения в управлении циклами `for` и выражениями `if`. Более реалистичный подход в контексте книги - добавить выражения `print` для трассировки управления логикой и вывода на дисплей состояния объектов. Итак, откуда начнем?

Встроенный цикл `for` представляет критическое пространство в нашей программе, в частности, проверка двух элементов и вызов `swap()`. Если алгоритм сортировки не работает, эта часть программы, похоже, рухнет. Я оснащу ее следующим образом:

```

ofstream ofil("text_out1");
void bubble_sort(vector<int> vec){
    for (int ix = 0; ix < vec.size(); ++ix)
        for (int jx = ix+1; jx < vec.size(); ++jx)
            if (vec[ix] > vec[jx]){
                // ошибочный вывод
                ofil << "about to call swap!"
                    << " ix: " << ix << " jx: " << jx << '\t'<< " swapping: "

                    << vec[ix]<< " with " << vec[jx] << endl;

                // ok: фактическая замена кода ...

                swap(vec[ix], vec[jx]);}}

```

После компиляции и выполнения программы, создается следующий отладочный вывод. Две вещи вызывают удивление: `swap()` вызывается в точности, как и должна, а вектор, тем не менее, не изменяется по сравнению с исходным видом. Кто сказал, что программирование забава?

```

vector before sort: 8 34 3 13 1 21 5 2

about to call swap! ix: 0 jx: 2 swapping: 8 with 3
about to call swap! ix: 0 jx: 4 swapping: 8 with 1
about to call swap! ix: 0 jx: 6 swapping: 8 with 5
about to call swap! ix: 0 jx: 7 swapping: 8 with 2
about to call swap! ix: 1 jx: 2 swapping: 34 with 3

```

```

about to call swap! ix: 1 jx: 3 swapping: 34 with 13
about to call swap! ix: 1 jx: 4 swapping: 34 with 1
about to call swap! ix: 1 jx: 5 swapping: 34 with 21
about to call swap! ix: 1 jx: 6 swapping: 34 with 5
about to call swap! ix: 1 jx: 7 swapping: 34 with 2
about to call swap! ix: 2 jx: 4 swapping: 3 with 1
about to call swap! ix: 2 jx: 7 swapping: 3 with 2
about to call swap! ix: 3 jx: 4 swapping: 13 with 1
about to call swap! ix: 3 jx: 6 swapping: 13 with 5
about to call swap! ix: 3 jx: 7 swapping: 13 with 2
about to call swap! ix: 5 jx: 6 swapping: 21 with 5
about to call swap! ix: 5 jx: 7 swapping: 21 with 2
about to call swap! ix: 6 jx: 7 swapping: 5 with 2
vector after sort: 8 34 3 13 1 21 5 2

```

Проверка `swap()` должна убедить нас, что все сделано правильно. Если бы мы начали колебаться в нашем суждении, мы могли бы оснастить `swap()`, чтобы убедиться, что она корректно заменяет значения:

```

void swap(int val1, int val2){
    ofil << "swap(" << val1<< ", " << val2 << ")\n";
    int temp = val1;
    val1 = val2;
    val2 = temp;
    ofil << "after swap(): val1 " << val1<< " val2: " << val2 << "\n";}

```

Дополнительно после вызова `swap()`, я добавил вызов `display()`, чтобы посмотреть состояние вектора. Вывод показал нам, что (1) все работает корректно, и (2) ничто не выводится правильно:

```

vector before sort: 8 34 3 13 1 21 5 2
about to call swap! ix: 0 jx: 2
swapping: 8 with 3
swap(8, 3)
after swap(): val1 3 val2: 8
vector after swap(): 8 34 3 13 1 21 5 2

```

Трассировка вывода показывает, что `bubble_sort()` корректно распознает, что первый и третий элементы, со значениями 8 и 3, должны быть переставлены. `swap()` корректно работает, внутри `swap()` значения правильно переставлены. Вектор, однако, не изменился.

К сожалению, это не что-то такое, что мы в состоянии самостоятельно полностью разгадать. Если вы похожи на меня, то вы склонны «бодаться» с проблемами: падать и думать, что вы должны «протаранить»

препятствия. Иногда, - в чем мы нуждаемся больше, чем в непреклонной воле, - это чтобы кто-нибудь показал нам пропущенный кусочек головоломки.

Проблема имеет место в том, как мы передаем аргументы в `swap()`. Способ подступиться к ней состоит в том, чтобы задать себе вопрос: «Каково взаимоотношение между, с одной стороны, двумя элементами, которые вектор передает `swap()` из `bubble_sort()`, и, с другой стороны, двумя параметрами, с которыми совершаются манипуляции внутри `swap()`?

```
void bubble_sort(vector<int> vec){
    // ...
    if (vec[ix] > vec[jx])
        swap(vec[ix], vec[jx]); // ...}

void swap(int val1, int val2) {
    // каковы взаимоотношения между
    // формальными параметрами val1, val2
    // и реальными аргументами vec[ix] и vec[jx]?}
```

Каковы взаимоотношения между `vec[ix]` и `vec[jx]`, передаваемыми при вызове `swap()`, и двумя параметрами `swap()` - `val1` и `val2`? Если обе пары представляют те же самые два объекта, тогда изменение `val1` и `val2` внутри `swap()` должно изменить значения внутри `vec[ix]` и `vec[jx]`. Но это совсем не то, что происходит в действительности. Получается так, как если бы мы манипулировали двумя разными парами объектов, которые никак не взаимосвязаны между собой, исключая пары, имеющие одинаковые значения.

Именно это получается в действительности. Что и объясняет, почему, хотя мы и переставляем значения, изменения не отражаются на векторе. В сущности, объекты, передаваемые `swap()`, копируются, а взаимосвязи между двумя парами объектов нет.

Когда мы вызываем функцию, специальная область памяти устанавливается под так называемый *программный стек*. Внутри этой специальной области памяти есть пространство для запоминания значений каждого параметра функции. (Она так же содержит память, ассоциированную с каждым объектом, определенным внутри функции - мы называем их *локальными объектами*). Когда функция выполняется, эта область памяти освобождается. (Мы говорим, что она *выталкивается* из программного стека).

По определению, когда мы передаем объект в функцию, как `vec[ix]`, его значение копируется в локальное определение параметра. (Это семантически называется *передача по значению*). Нет взаимосвязи между объектами, с которыми производятся манипуляции внутри `swap()` и объектами, передаваемыми туда из `bubble_sort()`. Вот, почему наша программа не работает.

Для того чтобы она работала, мы должны как-то связать параметры `swap()` с реальными объектами, передаваемыми функции. (Семантически это называется *передача по ссылке*). Самый простой путь для реализации этого - объявить параметры, как ссылки:

```
/*
 * ОК: объявление val1 и val2 ссылками
 * применительно к двум параметрам в swap()
 * отражается на объектах, передаваемых swap()
 */

void swap(int &val1, int &val2){

/*
 * заметьте, что наш код внутри swap()
 * не изменился - только взаимосвязь
```

```

* между параметрами swap() и
* объектами, передаваемыми swap() изменилась
*/
int temp = val1;
val1 = val2;
val2 = temp;}

```

Прежде, чем я объясню ссылки, давайте воочию подтвердим, что это маленькое изменение нашей программы скорректирует ее. Вот частичная трассировка перекомпиляции и выполнения нашей программы:

```

vector before sort: 8 34 3 13 1 21 5 2
about to call swap! ix: 0 jx: 2 swapping: 8 with 33 34 8 13 1 21 5 2
about to call swap! ix: 0 jx: 4 swapping: 3 with 11 34 8 13 3 21 5 2
about to call swap! ix: 1 jx: 2 swapping: 34 with 81 8 34 13 3 21 5 2
about to call swap! ix: 1 jx: 4 swapping: 8 with 3 // ...
about to call swap! ix: 5 jx: 7 swapping: 21 with 131 2 3 5 8 13 34 21 about to
call swap! ix: 6 jx: 7 swapping: 34 with 211 2 3 5 8 13 21 34
vector after sort: 8 34 3 13 1 21 5 2

```

Опять! Все теперь работает, за исключением того, что вектор внутри `main()`, который мы передаем `bubble_sort()`, не меняется. Но теперь у нас есть опыт, и первое, на что мы обратим внимание, на его параметры, передаваемые по значению, а не по ссылке:

```
void bubble_sort(vector<int> vec){ /* ... */ }
```

Изменение `vec` на ссылку – окончательная коррекция нашей программы:

```
void bubble_sort(vector<int> &vec){ /* ... */ }
```

Произведя изменения, перекомпилируем и запустим программу:

```

vector before sort: 8 34 3 13 1 21 5 2

vector after sort: 1 2 3 5 8 13 21 34

```

Во-о! Это было тяжело. Но такова уж доля программистская! Часто решение оказывается крайне простым, но лишь после того, как вы поймете, в чем существо проблемы.

## Семантика передачи по ссылке

Ссылка обслуживает непрямую поддержку объекта. Мы объявляем ссылку, добавляя амперсанд (&) между названием типа и именем ссылки:

```

int ival = 1024; // объект типа int

int *pi = &ival; // указатель на объект типа int

int &rval = ival; // ссылка на объект типа int

```

Когда мы пишем

```
int jval = 4096;

rval = ival;
```

мы передаем `ival`, объекту, на который ссылается `rval`, значение, сохраняемое в `jval`. Мы не говорим, что теперь `rval` ссылается на `jval`. Ссылки не могут быть переадресованы на другой объект. Когда мы пишем

```
pi = &rval;
```

мы передаем `pi` адрес `ival`, на который ссылается `rval`. Мы не делаем `pi` указателем на `rval`. Все манипуляции со ссылками отображаются на объекте, на который указывает ссылка. Это остается верным, и когда ссылка является параметром функции.

Когда мы меняем значения `val1` и `val2` внутри `swap()`,

```
void swap(int &val1, int &val2){
    // действительные аргументы модифицируются...
    int temp = val1; val1 = val2; val2 = temp;}

```

мы в действительности меняем значения `vec[ix]` и `vec[jx]`, на которые два объекта `val1` и `val2` ссылаются при вызове `swap()` внутри `bubble_sort()`:

```
swap(vec[ix], vec[jx]);
```

Проще говоря, когда `val2` присваивается значение `temp` внутри `swap()`, мы в действительности присваиваем `vec[jx]` первоначальное значение `vec[ix]`.

Когда объект передается ссылочным параметром, значение объекта не копируется. Вместо этого копируется адрес передаваемого объекта. Каждый доступ к ссылочному параметру внутри функции – это косвенная манипуляция с передаваемым объектом.

Одно из оснований для объявления параметра, как ссылки, возможность модифицировать непосредственно сам объект, передаваемый функции. Это важно, поскольку мы видели, наша программа без этого может работать неправильно.

Вторая причина для объявления ссылочных параметров – возможность избежать переписывания при копировании больших объектов. Это менее важная причина. Наша программа корректна, она чуть менее эффективна.

Например, мы передаем наш вектор в `display()` по значению. Это означает, что мы копируем полный векторный объект каждый раз, когда хотим увидеть его на дисплее. Это не ошибка. Генерация вывода, как раз то, что мы хотели. Но несколько быстрее передать адрес вектора. И опять, прямой путь для этого – объявить векторный параметр ссылкой:

```
void display(const vector<int> &vec){
    for (int ix = 0; ix < vec.size(); ++ix)
        cout << vec[ix] << ' ';cout << endl;}

```

Мы объявили его `const` вектором, поскольку мы не меняем его внутри тела функции. Но не будет ошибкой опустить `const`. Употребление `const` в данном случае информирует читателей программы, что

мы передаем вектор по ссылке для получения копии, а не для модификации внутри функции.

Если мы хотим, мы можем передать наш вектор, как указатель. Результат тот же самый, мы передаем адрес объекта в функцию, а не копируем целый объект. Одно различие, конечно, есть – в синтаксисе между ссылкой и указателем. Например,

```
void display(const vector<int> *vec){
    if (! vec){
        cout << "display(): the vector pointer is 0\n";
        return;}

    for (int ix = 0; ix < vec->size(); ++ix)

        cout << (*vec)[ix] << ' ';

    cout << endl;}

int main(){

    int ia[8] = { 8, 34, 3, 13, 1, 21, 5, 2 };

    vector<int> vec(ia, ia+8);

    cout << "vector before sort: ";

    display(&vec);

    // теперь передаем адрес

    // ...}
```

Более важное различие между указателем и ссылкой, как параметром, в том, что указатель может адресоваться, а может и нет, к объекту. Прежде, чем мы перешлем указатель, мы должны всегда удостовериться, что он не установлен в 0. Ссылка же всегда относится к самому объекту, так что проверка на 0 не нужна.

В общем, если вы намерены модифицировать параметры внутри функции, как мы делали с `fibon_elem()` в предыдущем разделе,

```
bool fibon_elem(int pos, int &elem);
```

я рекомендую не передавать встроенные типы по ссылке. Механизм ссылок первоначально задуман для поддержки передачи объектов класса, как параметров, в функции.

## Граница и область

Объекты, определенные внутри функции со своими атрибутами, существуют, только пока функция выполняется. Возвращение адреса одного из этих *локальных* объектов – серьезная ошибка при выполнении программы. Причина в том, что функция временно размещается в программном стеке в специальной области памяти на время ее выполнения. Локальные объекты запоминаются в этой области памяти. Когда функция завершает работу, эта область памяти освобождается. Локальные объекты перестают существовать.

Адресация к несуществующим объектам, в основном, плохой стиль программирования. Например, `fibon_seq()` возвращает вектор элементов последовательности Фибоначчи с некоторым, определенном пользователем, размером:

```
vector<int> fibon_seq(int size){
    if (size <= 0 || size > 1024){
        cerr << "Warning: fibon_seq(): "
            << size << " not supported -- resetting to 8\n";

        size = 8;}

    vector<int> elems(size);
    for (int ix = 0; ix < size; ++ix)
        if (ix == 0 || ix == 1) elems[ix] = 1;
        else elems[ix] = elems[ix-1] + elems[ix-2];

    return elems;}
```

Было бы не корректно возвращать `elems` ссылкой или указателем, поскольку `elems` перестает существовать с завершением `fibon_seq()`. Возвращение `elems` по значению – другое дело: копия возвращаемого объекта существует вне функции.<sup>7</sup>

Период времени, на который память отводится под объект, называется его *длительностью хранения* или *областью*. Память для `elems` отводится каждый раз, когда `fibon_seq()` выполняется. Она отбирается каждый раз, когда `fibon_seq()` прерывается. Мы говорим, что это *локальная область*. (Параметры функции, такие как `size`, также из локальной области).

Пространство программы, на котором объект активен, называется его границами. Мы говорим, что `size` и `elems` имеют *локальные границы* внутри функции `fibon_seq()`. Имя объекта, имеющего локальные границы, не видно вне его локальных границ.

Объект, объявленный вне функции имеет *границы файла*. Объект, имеющий границы файла виден от точки его объявления до конца файла, внутри которого он назван. Объект с границами файла имеет *статическую область*. Это означает, что его память выделяется до начала `main()` и остается выделенной пока программа не закончит свою работу.

Объекты встроенного типа, определенные в границах файла, всегда инициализируются в 0. Объект встроенного типа, определенный в локальных границах, однако, остается не инициализированным до момента, пока не получает значения инициализации от программиста.

## Управление динамической памятью

Как локальная, так и файловая область, управляются для нас автоматически. Есть третья форма длительности хранения, называемая *динамической областью*. Эта память приходит из программного свободного хранилища и иногда называется *куча* (*heap*). Эта память должна управляться исключительно программистом. Область памяти создается использованием выражения `new`, тогда как освобождение памяти осуществляется выражением `delete`.

---

<sup>7</sup> Класс объектов возвращаемых по значению оптимизируется большинством C++ компиляторов в дополнительные ссылочные параметры. Для ознакомления с оптимизацией *name return value* смотрите Раздел 14.8 в [LIPPMAN98].

Пишется выражение `new` следующим образом:

```
new Type;
```

Здесь, `Type` может быть любым встроенным типом, или типом класса, известным программе, или выражение `new` может быть записано как

```
new Type(initial_value);
```

Например,

```
int *pi; pi = new int;
```

присваивание `pi` адреса объекта типа `int` располагается в “куче”. По определению, объект, расположенный в куче, не инициализируется. Вторая форма выражения `new` позволяет нам показать начальное значение. Например,

```
pi = new int(1024);
```

также задает для `pi` адрес объекта типа `int`, расположенного в свободной памяти. Этот объект, однако, имеет начальное значение 1024.

Чтобы создать массив в “куче” мы пишем

```
int *pia = new int[24];
```

Этот код создает массив из 24 объектов целого типа в “куче”. `pia` инициализируется адресацией первого элемента массива. Элементы же массива, сами по себе, остаются без начального значения. Нет синтаксиса для инициализации массива элементов, расположенного в “куче”.

Говорят, что объект “кучи” имеет динамическую область, поскольку она определяется при выполнении программой выражения `new`, и продолжает существовать, пока специально не освобождается выражением `delete`. Например, следующее выражение `delete` приводит к тому, что объект, адресуемый `pi`, будет уничтожен:

```
delete pi;
```

Для уничтожения массива объектов мы добавляем оператор с пустым описанием между указателем на массив и выражением удалить:

```
delete [] pia;
```

Нам не нужно проверять, что `pi` не нулевое:

```
if (pi != 0) // не нужно - компилятор проверит это для нас  
  
delete pi;
```

Компилятор делает эту проверку автоматически. Если, из каких-то соображений, программист опустит выражение `delete`, объект “кучи” не будет уничтожен никогда. Это называется *утечка памяти*. В Части 6 мы увидим, какие причины могут заставить нас выбрать использование динамического распределения памяти в разработке нашей программы. Мы рассмотрим пути преодоления утечки памяти при обсуждении поддержки

исключений в Части 7.

## 2.3 Обеспечение параметров значениями по умолчанию

Вывод трассировки нашей программы пузырьковой сортировки в `ofil` требовал, чтобы я сделал `ofil` подходящей для множества функций, которые я хотел бы отлаживать. Поскольку я реагировал на неожиданное и непредвиденное поведение, я выбрал наискорейшее решение проблемы, сделав объект видимым для множества функций: я определил `ofil` в области файла.

Одно из основных правил программирования, однако - применять взаимодействие между функциями через использование параметров, а не задавать объект, определенный в области файла. Одна из причин этого в том, что функцию, зависящую от объекта, объявленного в области файла, труднее применять в разном контексте. Функцию также труднее модифицировать: мы должны понять не только логику работы функции, но и логику поведения объектов, определенных в области файла.

Давайте посмотрим, как мы могли бы переделать `bubble_sort()`, чтобы вывести ее из области файла исключительно в `ofil`:

```
void bubble_sort(vector<int> &vec, ofstream &ofil){
    for (int ix = 0; ix < vec.size(); ++ix)
        for (int jx = ix+1; jx < vec.size(); ++jx)
            if (vec[ix] > vec[jx]) {
                ofil << "about to call swap! ix: " << ix
                    << " jx: " << jx << "\tswapping: "
                    << vec[ix] << " with " << vec[jx] << endl;
                swap(vec[ix], vec[jx], ofil);}}}
```

Хотя этот прием выводит из области файла внутрь `ofil`, он создает ряд потенциально неприятных проблем. Каждый вызов `bubble_sort()` теперь требует от нашего пользователя передачи в объект класса `ofstream`. С другой стороны, мы генерируем эту информацию без возможности для пользователя выключить ее. В общем случае, когда все идет хорошо, никому не интересна эта информация.

Мы предпочли бы не докучать пользователю, ни просьбами назначить поток вывода, ни просьбами что-либо выключить. По определению, мы предпочли бы не выводить информацию. Однако нам хотелось бы позволить тем, кому это интересно, вывести информацию и назначить файл, в котором бы эта информация запоминалась. Как мы можем это сделать?

C++ позволяет нам назначить значения по умолчанию для всех или подмножества параметров. В нашем случае мы передаем в `ofstream` параметр указателя со значением по умолчанию 0:

```
void bubble_sort(vector<int> &vec, ofstream *ofil = 0){
    for (int ix = 0; ix < vec.size(); ++ix)
        for (int jx = ix+1; jx < vec.size(); ++jx)
            if (vec[ix] > vec[jx]){
                if (ofil != 0) (*ofil) << "about to call swap! ix: " << ix
                    << " jx: " << jx << "\tswapping: "
```

```

        << vec[ix] << " with " << vec[jx] <<endl;
    swap(vec[ix], vec[jx], ofil);}}

```

Эта переделанная версия `bubble_sort()` объявляет ее вторым параметром указатель на объект `ofstream`, а не ссылку. Мы должны сделать эти изменения для присвоения значения по умолчанию 0, показывая, что объект `ofstream` адресован. В отличие от указателя ссылка не может быть установлена в 0. Ссылка всегда должна ссылаться на какой-то объект.

Вызов `bubble_sort()` с единственным аргументом не выдает сообщения об ошибке. Вызов же со вторым аргументом, который адресуется к объекту `ofstream`, генерирует информацию об ошибке:

```

int main(){

    int ia[8] = { 8, 34, 3, 13, 1, 21, 5, 2 };

    vector<int> vec(ia, ia+8);

    // нет сообщения об ошибке --
    // как если бы мы вызвали bubble_sort(vec, 0);
    bubble_sort(vec);
    display(vec);

    // ok: информация об ошибке генерируется ...
    ofstream ofil("data.txt");
    bubble_sort(vec, &ofil);
    display(vec, ofil);}

```

Выполнение `display()` представляет другую ситуацию. В настоящем, это аппаратно-кодируемый вывод на `cout`. Вообще, `cout` это хорошо. В конкретных же случаях, однако, пользователи, похоже, предпочитают иметь возможность другой цели – такой, как файл. Наша реализация должна поддерживать оба использования `display()` в `main()`. Примем решение – сделать `cout` параметром `ostream` по определению:

```

void display(const vector<int> &vec, ostream &os = cout){

    for (int ix = 0; ix < vec.size(); ++ix)

        os << vec[ix] << ' ';

    os << endl;}

```

Есть два неочевидных правила, касающихся употребления параметров по умолчанию. Первое правило: значения по умолчанию побуждают позиционно начинать с крайнего справа параметра. Если параметр приходит со значением по умолчанию, все параметры до него должны тоже иметь значения по умолчанию. Следующее, например, неверно:

```

// error: нет значения по умолчанию для vec

void display(ostream &os = cout, const vector<int> &vec);

```

Второе правило: значение по умолчанию может быть определено только один раз, либо при объявлении, либо при определении функции, но не в обоих случаях. Так, где же мы должны обозначить значение по умолчанию?

Как правило, объявление функции располагается в файле заголовка. Затем этот файл заголовка включается в файлы, в которых собираются использовать функцию. (Напомню, что мы включали файл заголовка `cstdlib`, чтобы включить объявление библиотечной функции `exit()`). Определение функции обычно располагается в файле текста программы. Этот файл сразу компилируется и линкуется с нашей программой, когда мы хотим использовать функцию. Файл заголовка, и это так, обеспечивает большую «видимость» функции. (Файлы заголовков обсуждаются более детально в Разделе 2.9).

Из-за большей «видимости», мы располагаем значения по умолчанию в объявление функции, а не в ее определении. Например, объявление и определение `display()` обычно выглядит следующим образом:

```
// объявление файла заголовка задает значение по умолчанию
// давайте вызовем файл заголовка: NumericSeq.h
void display(const vector<int>&, ostream&=cout);
// файл текста программы включает определение файла заголовка
// определение, само по себе, не задает значения по умолчанию
#include "NumericSeq.h"
void display(const vector<int> &vec, ostream &os){
    for (int ix = 0; ix < vec.size(); ++ix)
        os << vec[ix] << ' ';os << endl;}
```

## 2.4 Использование локальных статических объектов

Наша функция `fibon_seq()` из Раздела 2.2 вычисляет последовательность Фибоначчи размера, определенного пользователем, с каждым запуском и возвращает вектор, содержащий элементы. Это немного больше работы, чем необходимо.

В действительности нам нужен только один вектор последовательности Фибоначчи. Элементы, как-никак, остаются инвариантными. Единственное, что меняется от одного вызова `fibon_seq()` к следующему, число элементов, которые пользователь хотел бы видеть. Обдумайте следующие три вызова `fibon_seq()`:

```
fibon_seq(24);
fibon_seq(8);
fibon_seq(18);
```

Первый вызов вычисляет все значения, необходимые для выполнения запроса второго и третьего вызовов. Если четвертый вызов запросит 32 элемента, мы в действительности нуждаемся в вычислении только элементов с 25 по 32, при условии, что сможем сохранять элементы, вычисленные между вызовами. Как бы мы могли это сделать?

Векторные объекты, локализуемые в функции, не дают решения. Локальные объекты создаются при каждом вызове функции и уничтожаются сразу после ее завершения. Заманчивой альтернативой было бы определить векторный объект в области файла. Это всегда соблазнительно, поместить объект в область файла, чтобы решить коммуникационные проблемы между функциями. В основном, однако, объекты области файла усложняют независимость и понятность индивидуальных функций.

Альтернативное решение в этом случае – локальные статические объекты. Например,

```
const vector<int>* fibon_seq(int size){
    static vector<int> elems;
```

```

// дальше заполняем все это ...

return &elems;}

```

`elems` теперь определены, как локальный статический объект `fibon_seq()`. Что это значит? В отличие от нестатического локального объекта, память, связанная со статическим локальным объектом, сохраняется и после выполнения функции. `elems` больше не разрушается и воссоздается с каждым вызовом функции `fibon_seq()`. Благодаря этому мы можем теперь в целости и сохранности вернуть адрес `elems`.

Локальный статический объект позволяет нам определить единственный вектор для удержания элементов последовательности Фибоначчи. С каждым вызовом `fibon_seq()`, нам нужно вычислять только те элементы, которые еще не вставлены в `elems`. Вот одна из возможных реализаций:

```

const vector<int>* fibon_seq(int size){
    const int max_size = 1024;
    static vector<int> elems;

    if (size <= 0 || size > max_size){
        cerr << "fibon_seq(): oops: invalid size: "
             << size << " -- can't fulfill request.\n";
        return 0;}

    // если размер равен или больше, чем elems.size(),
    // никакие вычисления не требуются ...
    for (int ix = elems.size(); ix < size; ++ix){
        if (ix == 0 || ix == 1) elems.push_back(1);
        else elems.push_back(elems[ix-1]+elems[ix-2]);}
    return &elems;}

```

Ранее мы всегда определяли вектор своего размера и заполняли значениями существующих элементов. В этой версии `fibon_seq()`, мы не имеем оснований гадать, насколько большой вектор нам нужен. Вместо этого мы определяем `elems`, как пустой вектор и вставляем элементы по мере надобности. `push_back()` вставляет значения в конец вектора. Память для поддержки этого управляется автоматически самим классом векторов. (В Части 3 мы рассмотрим в деталях векторы и другие классы контейнеров стандартной библиотеки).

## 2.5 Объявление встроенных функций (inline)

Напомню, что `fibon_elem()` возвращает элемент Фибоначчи из обозначенной пользователем позиции внутри последовательности. В нашей первоначальной реализации производились вычисления с последовательностью до нужной позиции с каждым вызовом. Попутно проверялось, является ли запрошенная позиция подходящей. Мы могли бы упростить ее реализацию, осуществляя подзадачи внутри отдельных функций:

```

bool is_size_ok(int size){
    const int max_size = 1024;
    if (size <= 0 || size > max_size){
        cerr << "Oops: requested size is not supported : "<< size
             << " -- can't fulfill request.\n";
        return false;}
    return true;}

```

```

// вычисляется размер последовательности Фибоначчи
// возвращается адрес статического контейнера, сохраняющего элементы

const vector<int> *fibon_seq(int size){
    static vector<int> elems;
    if (! is_size_ok(size))    return 0;

    for (int ix = elems.size(); ix < size; ++ix)

        if (ix == 0 || ix == 1)    elems.push_back(1);

        else    elems.push_back(elems[ix-1]+elems[ix-2]);

    return &elems;}

// возвращается элемент Фибоначчи в позиции pos
// (мы должны уладить с 1, поскольку первый элемент сохраняется в 0)
// возвращается "ложно", если позиция не поддерживается

bool fibon_elem(int pos, int &elem) {
    const vector<int> *pseq = fibon_seq(pos);
    if (! pseq) { elem = 0; return false; }

    elem = (*pseq)[pos-1];
    return true;}

```

Разобравшись с проверкой размера в `is_size_ok()` и вычислив элементы Фибоначчи в `fibon_seq()`, мы упрощаем `fibon_elem()` для использования и понимания. Вдобавок эти две функции теперь применимы для других приложений.

Есть неприятность с тем, что теперь `fibon_elem()` требует трех вызовов функции для завершения операции, тогда как прежде требовался только один. Критичны ли эти дополнительные издержки? Все зависит от контекста их использования. Если их применение оказывается неприемлемым, одно из решений – собрать три функции опять в одну. В C++ альтернативное решение – объявить функции, как `inline` (встроенные).

Функция `inline` передает запрос к компилятору - расширить функцию из каждой точки вызова. С функцией `inline` компилятор замещает вызов функции копией исполняемого кода. В результате мы можем использовать реализацию с функциями, собранными обратно в `fibon_elem()`, все еще поддерживая эти три функции, как самостоятельные операции.

Мы объявляем функцию встроенной, предпосылая ее определение ключевым словом `inline`:

```

// ок: теперь fibon_elem() функция inline
inline bool fibon_elem(int pos, int &elem){
    /* определение, как и выше */
}

```

Спецификация `inline` - это только запрос к компилятору. Притом, что компилятор, фактически удовлетворяющий запрос, является исполнимо-зависимым. (Обсуждение вопроса, почему спецификация `inline` – это только запрос, смотрите в Разделе 7.1.1 [STROUSRUP97]).

В основном, наилучшие функции-кандидаты для `inline`, как `fibon_elem()` и `is_size_ok()` – это маленькие, часто вызываемые, и несложные при выполнении функции.

Определение функции `inline` обычно размещают в файле заголовка. Для того чтобы они были расширены, их определение должно быть доступно до их вызова. (Это обсуждается далее в Разделе 2.9).

## 2.6 Использование перезагружаемых (Overloaded) функций

Прежде, чем каждая из функций выведет ее собственное диагностическое сообщение, давайте определим основную `display_message()` функцию. Это можно сделать так:

```
bool is_size_ok(int size){
    const int max_size = 1024;
    const string msg("Requested size is not supported");

    if (size <= 0 || size > max_size){
        display_message(msg, size);
        return false;}
    return true;}
```

Подобным же образом наша программа в Части 1 могла использоваться для вывода приветствия пользователя

```
const string greeting("Hello. Welcome to Guess the Numeric Sequence");

display_message(greeting);
```

и для вывода двух элементов последовательности:

```
const string seq("The two elements of the sequence are ");

display_message(seq, elem1, elem2);
```

В другом случае мы могли бы использовать `display_message()` для простого вывода символа перехода на новую строку или табуляции:

```
display_message('\n');

display_message('\t');
```

Можем ли мы действительно передать параметры в `display_message()`, которые отличаются и типом, и количеством? Да.

Как? Через функцию перезагрузки (overloading).

Две или более функций могут иметь одинаковое имя, если список параметров каждой функции уникален по типу или числу параметров. Например, следующее объявление представляет четыре перезагружаемых реализации `display_message()`, вызванных ранее:

```
void display_message(char ch);

void display_message(const string&);

void display_message(const string&, int);
```

```
void display_message(const string&, int, int);
```

Как компилятор понимает, какой вариант из четырех перезагружаемых функций вызвать? Он сравнивает действительные аргументы, передаваемые функции, с параметрами каждого перезагружаемого варианта, выбирая наилучшее совпадение. Вот почему список параметров каждой перезагружаемой функции должен быть уникален.

Возвращаемый тип функции, сам по себе, не различает два варианта функции с одинаковым именем. Следующее, например, неправильно. Результатом станет ошибка при компиляции:

```
// error: список параметров невозвращаемого типа должен быть уникален

ostream& display_message(char ch);

bool display_message(char ch);
```

Почему возвращаемого типа, самого по себе, не достаточно для перезагружаемой функции? Потому, что возвращаемый тип не гарантирует достаточности контекста, по которому различаются варианты. Например, в следующем вызове для компьютера нет реального пути для определения, какой из вариантов пользователь запрашивает:

```
display_message('\t'); // which one?
```

Перезагружаемое множество функций, которые имеют уникальную реализацию, но выполняют схожие задачи, упрощает использование этих функций для наших пользователей. Без перезагрузки, мы были бы вынуждены использовать каждую функцию с уникальным именем.

## 2.7 Определение и использование шаблонов функций

Положим, наши коллеги хотели бы три дополнительных варианта для `display_message()`, поддерживающих вектор целых, вектор чисел двойной точности и строчный вектор:

```
void display_message(const string&, const vector<int>&);

void display_message(const string&, const vector<double>&);

void display_message(const string&, const vector<string>&);
```

Выполняя реализацию этих трех вариантов, мы замечаем, что тело функции для каждого варианта то же самое. Разница между ними в типе второго параметра:<sup>8</sup>

```
void display_message(const string&, const vector<string>&,
                    ostream& = cout);

void display_message(const string &msg, const vector<int> &vec){
```

---

<sup>8</sup> Более гибкая реализация имеет третий параметр типа `ostream`, который по определению установлен в `cout`:

```

    cout << msg;

    for (int ix = 0; ix < vec.size(); ++ix) cout << vec[ix] << ' ';}

void display_message(const string &msg, const vector<string> &vec){
    cout << msg;
    for (int ix = 0; ix < vec.size(); ++ix)

        cout << vec[ix] << ' ';cout << '\n';}

void display_message(const string&, const vector<string>&,
                    ostream& = cout);

```

Нет оснований думать, что другой коллега не появится с просьбой о совершенно другой реализации поддержки какого-то дополнительного типа для этих векторов. Это, естественно, наводит нас на мысль о попытке определить единственный вариант тела функции, вместо того, чтобы копировать код множество раз, делая необходимые минимальные изменения в каждом варианте. Однако чтобы сделать это, нам нужна возможность привязки этого единственного варианта к каждому типу вектора, который мы хотим вывести на дисплей. Механизм шаблонов функций предоставляет нам такую возможность.

Шаблон функции факторизует (выносит за скобки) информацию о типе всех или части типов, обозначенных в списке параметров. В случае `display_message()`, мы хотим факторизовать тип элемента, заключенного в векторе. Это позволит нам определить единственный вариант неизменяемой части шаблона функции. Этого недостаточно, однако, поскольку факторизованная информация о типе пропущена. Эта информация о типе предоставляется пользователем через особую разновидность шаблона функции.

Шаблон функции начинается с ключевого слова `template`. За ним следует список из одного или более идентификаторов, которые представляют типы, которые мы хотели бы определить. Список заключен в скобки из знаков меньше-больше (<,>). Пользователь подставляет информацию о реальном типе каждый раз, когда использует реальный вариант функции. Эти идентификаторы в действительности используются, как заполнители для действительного типа данных внутри списка параметров и тела шаблона функции. Например,

```

template <typename elemType>
void display_message(const string &msg,const vector<elemType> &vec) {
    cout << msg;
    for (int ix = 0; ix < vec.size(); ++ix){
        elemType t = vec[ix];  cout << t << ' ';;}

```

Ключевое слово `typename` определяет `elemType`, как тип-заполнитель внутри шаблона функции `display_message()`. `elemType` - произвольное имя. Я легко могу выбрать `foobar` или `T`. Мы должны отличать реальный тип вектора, выводимого на экран. Что мы и делаем, помещая `elemType` внутри пары скобок за вектором.

А как насчет первого параметра? `msg` никогда не меняет своего типа с каждым вызовом `display_message()`. Это постоянная ссылка на объект строкового класса, так что нет необходимости факторизовать его тип. Шаблон функции обычно имеет комбинацию подробных и «отложенных» типов спецификаторов в своем списке параметров.

Как мы используем шаблон функции? На тот же манер, что и обычную функцию. Например, когда мы пишем

```

vector<int> ivec;
string msg;

```

```
// ...
display_message(msg, ivec);
```

компилятор связывает `elemType` с типом `int`. Создается вариант `display_message()`, в котором второй параметр типа `vector<int>`. Внутри тела функции локальный объект `t` также становится типа `int`. Подобным образом, когда мы пишем

```
vector<string> svec;//
...display_message(msg, svec);
```

`elemType` оказывается связан со строковым типом. Создается вариант `display_message()`, в котором второй параметр типа `vector<string>` и т.д.

Шаблон функции используется, как разновидность предписывания генерировать неограниченное число вариантов функции, в которых `elemType` связывается со встроенным или определенным пользователем типом класса.

В общем, мы перезагружаем функцию, когда есть множество реализаций, но каждый вариант осуществляет то же самое основное действие. Мы делаем функцию шаблоном, если код функции остается неизменным при многообразии типов.

Шаблон функции может также быть перезагружаемой функцией. Например, давайте, осуществим два варианта `display_message()`: один со вторым параметром векторного типа, а второй списочного (`list`) типа. (`list` – другой класс контейнера, определенный в стандартной библиотеке C++. Мы рассмотрим этот класс в Части 3).

```
// перезагружаемый образец шаблона функции
template <typename elemType>
void display_message(const string &msg, const vector<elemType> &vec);

template <typename elemType>
void display_message(const string &msg, const list<elemType> &lt);
```

## 2.8 Указатели на функции добавляют гибкости

Мы должны использовать функцию для возвращения вектора элементов подобных `fibon_seq()` для каждой из наших других пяти числовых последовательностей. Полный набор функций может быть определен следующим образом:

```
const vector<int> *fibon_seq(int size);

const vector<int> *lucas_seq(int size);

const vector<int> *pell_seq(int size);

const vector<int> *triang_seq(int size);

const vector<int> *square_seq(int size);

const vector<int> *pent_seq(int size);
```

Что насчет `fibon_elem()`? Должны ли мы также использовать шесть отдельных вариантов по одному на каждую последовательность? Определение `fibon_elem()` следующее:

```
bool fibon_elem(int pos, int &elem){
    const vector<int> *pseq = fibon_seq(pos);
    if (! pseq) { elem = 0; return false; }
    elem = (*pseq)[pos-1];
    return true;}

```

Единственный зависящий от последовательности аспект `fibon_elem()` – вызов ассоциированной с последовательностью функции для получения вектора элементов. Если мы исключим эту зависимость, мы устраним необходимость более чем в одном варианте функции. Мы получим эту независимость благодаря использованию указателя – точнее указателя на функцию.

Определение *указателя на функцию* несколько путано. Оно должно определить тип возврата и список параметров функции, к которой указатель адресуется. В нашем случае список параметров состоит из единственного `int`, а возвращаемый тип - `vector<int>*`. Вдобавок определение должно расположить где-нибудь `*` для обозначения, что данный объект определен, как указатель. Окончательно, конечно, мы должны дать имя указателю. Давайте назовем его `seq_ptr`. Как правило, наша первая проба почти корректна.

```
const vector<int>* *seq_ptr(int); // почти правильно

```

Этот код определяет `seq_ptr`, как функцию, которая имеет в списке параметров только `int` и возвращает тип указателя на указатель на `const` вектор элементов типа `int`! Для распознавания `seq_ptr`, как указателя, мы должны пометить предшествующее по умолчанию `*` круглыми скобками:

```
const vector<int>* (*seq_ptr)(int); // ok

```

`seq_ptr` может адресоваться к любой функции с тем же возвращаемым типом и списком параметров. Это значит, что можно адресоваться к каждой из шести функций числовых последовательностей. Давайте перепишем `fibon_elem()` в общем виде `seq_elem()` следующим образом:

```
bool seq_elem(int pos, int &elem, const vector<int>* (*seq_ptr)(int)){
    // вызов функции, адресуемой seq_ptr
    const vector<int> *pseq = seq_ptr(pos);
    if (! pseq) { elem = 0; return false; }
    elem = (*pseq)[pos-1]; return true;}

```

Функция, адресованная указателем на функцию, вызывается так же, как сама функция. То есть,

```
const vector<int> *pseq = seq_ptr(pos);

```

непрямой вызов функции, адресуемой `seq_ptr`. Мы не знаем (или не заботимся), к какой функции это адресуется. Разумно было бы, однако, подтвердить, что адресуется хотя бы *какая-то* функция:

```
if (!seq_ptr)display_message("Internal Error: seq_ptr is set to null!");

```

Указатель на функцию может быть инициализирован или соотнесен либо с 0, указывая, что не адресуется к функции,

```
const vector<int>* (*seq_ptr)(int) = 0;
```

либо адресоваться к функции. Следующий вопрос – как мы получаем адрес функции? Это одна из наименее туманных операций в C++. Мы только называем его

```
// присваиваем seq_ptr адрес pell_seq()

seq_ptr = pell_seq;
```

Что если бы мы хотели устанавливать `seq_ptr` на другую функцию последовательности с каждым проходом через цикл вывода на дисплей без явного переименования каждой функции? Для решения этой проблемы мы можем вновь использовать индексацию внутри массива. В этом случае мы определяем массив указателей на функции:

```
// seq_array массив указателей на функции
const vector<int>* (*seq_array[])(int) = {fibon_seq, lucas_seq, pell_seq,
triang_seq, square_seq, pent_seq};
```

`seq_array` массив указателей на функции, содержащий шесть элементов. Первый элемент адресуется к `fibon_seq()`, второй к `lucas_seq()` и т.д. Мы можем переустанавливать `seq_ptr` с каждым проходом цикла `while`, как только пользователь захочет угадывать другую последовательность:

```
int seq_index = 0;
while (next_seq == true){
    seq_ptr = seq_array[++seq_index]; // ...}
```

В качестве альтернативы: что если бы мы захотели получить указатель непосредственно на функцию, генерирующую последовательность Pell? Как-то оно «топорно» - запоминать, что вариант Pell адресуется третьим элементом массива. Более наглядный метод индексации – обозначить все множеством мнемонических постоянных значений. Например,

```
enum ns_type {ns_fibon, ns_lucas, ns_pell, ns_triang, ns_square, ns_pent};
```

Нумерованный тип обозначается ключевым словом `enum`, за которым следует не обязательный идентификатор, как `ns_type`. Элементы в списке, разделенные запятыми - значения имен в фигурных скобках, называемые *нумераторами*. По умолчанию, первый нумератор присваивается значению 0. Каждый последующий нумератор получает значение на 1 большее предшествующего. Таким образом, `ns_type: ns_fib` имеет значение 0, `ns_lucas` 1, `ns_pell` 2, и т.д. до `ns_pent`, который имеет значение 5.

Для явного доступа к определенному указателю на функцию мы используем присвоенный нумератор в качестве индекса:

```
seq_ptr = seq_array[ns_pell];
```

## 2.9 Создание файлов заголовков

Прежде, чем использовать `seq_elem()`, я должен вначале определить ее в программе. Если это используется в пяти текстах файлов, должно быть сделано пять определений. Вместо отдельных определений `seq_elem()` в каждом из пяти файлов мы располагаем определение функции в файле заголовка. Файл заголовка затем включается в каждый файла текста программы, в котором предполагается использовать функцию.

Используя это соглашение, мы нуждаемся в создании только единственного определения функции. Если список ее параметров или возвращаемых типов изменится, только в одном месте определения функции понадобится сделать изменения. Все пользователи функции автоматически получают обновленное определение функции.

Файлы заголовков по соглашению получают суффикс `.h`, исключая файлы заголовков стандартной библиотеки, которые не имеют суффикса. Мы называем наш файл заголовка `NumSeq.h` и помещаем в него определение всех функций, относящихся к нашим числовым последовательностям:

```
// NumSeq.h
bool seq_elem(int pos, int &elem);
const vector<int> *fibon_seq(int size);
const vector<int> *lucas_seq(int size);
const vector<int> *pell_seq(int size);
const vector<int> *triang_seq(int size);
const vector<int> *square_seq(int size);
const vector<int> *pent_seq(int size);
// и т.д. ...
```

Теперь есть единственное определение функции в программе. Однако может быть множество объявлений. Мы не включаем объявлений в файл заголовка, поскольку файл заголовка включается во множество файлов текста программы.

Существует одно исключение из этого правила единственного определения – определение функции `inline`. Для расширения `inline` функции определение должно быть доступно компилятору в каждой точке вызова. Это означает, что мы должны постараться поместить определения `inline` функции внутри файла заголовка вместо размещения в отдельных программных файлах.

Объекты, определенные в границах файла, также объявляются в файле заголовка, если множеству файлов нужен доступ к объектам. Это происходит потому, что к объекту не может быть обращения до его объявления в программе. Например, если `seq_array` была определена в границах файла, мы предпочли бы ее объявить в `NumSeq.h`. Непредвиденно, наша первая попытка не вполне корректна:

```
// Это не вполне хорошо ...
const int seq_cnt = 6;
const vector<int>* (*seq_array[seq_cnt])(int);
```

Это неправильно, поскольку интерпретируется как определение `seq_array`, а не как объявление. Как и с функцией, объект может быть определен единственный раз в программе. Определение объекта, так же как определение функции, должно быть размещено в файле текста программы. Мы превращаем определение `seq_array` в объявление, предваряя его ключевым словом `extern`:

```
// ОК: это объявление
```

```
extern const vector<int>* (*seq_array[seq_cnt])(int);
```

Хорошо, разобрались, можете сказать вы. Это аналог распознавания объявления функции, помещенного в файл заголовка, но определение функции сделано в файле текста программы. Но. Но. (В этом месте вы встаете). Но если бы это все было так, тогда почему бы не объявить `seq_cnt`, как `extern`, вместо явного определения?

Очевидно, это только запутает вас. Меня. Всех нас.

Объект `const`, как функция `inline`, рассматривается, как исключение из правила. Определение объекта не видимо вне файла, где он определен. Что означает, мы можем определять его во множестве программных файлов без появления ошибок.

Зачем нам это? Затем, что мы хотим, чтобы компилятор использовал значение объекта `const` в нашем массиве объявлений и других ситуациях, в которых нужны постоянные выражения, возможно во множестве файлов.

Файл, использующий либо `seq_elem()`, либо `seq_array` включает файл заголовка до первого использования любого имени:

```
#include "NumSeq.h"
void test_it() {
    int elem = 0;
    if (seq_elem(1, elem) && elem == 1) // ...}
```

Почему `NumSeq.h` в кавычках, а не в угловых скобках (`<`, `>`)? Вкратце, если файл заголовка в той же директории, что файл программы, включающей его, мы используем кавычки. Если он где-то еще, мы используем угловые скобки. Несколько “более технический” ответ – если имя файла заключено в угловые скобки, файл предполагается принадлежащим проекту или стандартному файлу заголовка. При его поиске проверяется предопределенное множество мест. Если имя файла заключено в кавычки, файл принимается, как определенный пользователем файл заголовка. Поиск файла начинается в директории, в которой расположен включаемый файл.

## Упражнение 2.2

Формула для Pentagonal числовой последовательности  $P_n = n * (3n - 1) / 2$ . Это дает последовательность 1, 5, 12, 22, 35 и т.д. Определите функцию для заполнения вектора элементов, вычисляемых в соответствующей функции до некоторой, определяемой пользователем, позиции. Проведите проверку, что названная позиция верна. Напишите вторую функцию, которая выводит этот вектор на экран. Она должна иметь второй параметр, идентифицирующий тип числовой серии, представляемой вектором. Напишите `main()` функцию для упражнений с этими функциями.

## Упражнение 2.3

Разделите функцию, вычисляющую Pentagonal числовую последовательность в упражнении 2.2, на две функции. Одна функция должна быть `inline`, она проверит правильность позиции. Правильная позиция, которая еще не вычислена, вызывает вторую функцию, которая делает необходимое вычисление.

## Упражнение 2.4

Введите статический локальный вектор для поддержания элементов вашей Pentagonal серии. Эта функция возвращает `const` указатель на вектор. Она допускает позицию, на которую возрастает вектор, если он еще не достиг этого размера. Введите вторую функцию, которая, в данной позиции, возвращает элемент в этой позиции. Напишите `main()` функцию для упражнений с этими функциями.

## Упражнение 2.5

Введите множество перезагружаемых функций `max()`, допускающее (a) два целых, (b) два с плавающей точкой, (c) две строки, (d) вектор целых, (e) вектор значений с плавающей точкой, (f) вектор строк, (g) массив целых и целое для задания размера массива, (h) массив с плавающей точкой целое для задания размера массива, (i) массив строк и целое для размера массива. Опять напишите функцию `main()` для упражнений с этими функциями.

## Упражнение 2.6

Переделайте функцию упражнения 2.5, используя шаблоны. Переделайте `main()` функцию соответственно.

## Часть 3. Общее программирование

Библиотека стандартных шаблонов (STL) состоит из двух первичных компонентов: множества классов контейнеров, включающего классы вектора, списка, множеств и отображений, и множества общих алгоритмов для операций с этими контейнерами, включающее `find()`, `sort()`, `replace()` и `merge()`.

Классы контейнеров вектора и списка представлены последовательными контейнерами. *Последовательный контейнер* содержит первый элемент, второй элемент и т.д. до последнего элемента. Мы вначале рассматривали последовательные контейнеры. Классы отображений и множеств представлены ассоциативными контейнерами. *Ассоциативный контейнер* поддерживает быстрый поиск значений.

*Отображение (map)* – пара ключ/значение. Ключ используется для поиска, а значение представляет данные, которые мы сохраняем и отыскиваем. Телефонный справочник, например, легко представляется отображением. Ключ – индивидуальное имя, значение ассоциировано с номером телефона.

*Множество* содержит только значения ключей. Мы запрашиваем его на присутствие значения. Например, если бы мы создавали указатель слов, содержащихся в статье, то предпочли бы исключить нейтральные слова, такие как *the, an, but (u, a, no, же)* и т.д. Прежде, чем слово попадет в указатель, мы запрашиваем множество `исключенных_слов`. Если слово представлено, мы отбрасываем его, иначе включаем слово в наш указатель.

Общие алгоритмы осуществляют большое количество операций, которые могут быть применены как к классу контейнеров, так и к встроенным массивам. Алгоритмы названы общими, поскольку они не зависят ни от типа элементов, с которыми оперируют (например, целые, двойной точности или строки), ни от типа контейнера, в которых они содержатся (будь то вектор, список или встроенный массив).

Общие алгоритмы получают независимость от типа, будучи реализованы как шаблоны функций. Они получают независимость от контейнера за счет отсутствия прямых операций на контейнере. Точнее, они проходят по парам *итераторов* (первый, последний), отмечая круг элементов по которым будут перемещаться. Пока первый не равен последнему, алгоритм оперирует с адресом элемента первый, увеличивает первый до адреса следующего элемента, и затем вновь сравнивает первый и последний на равенство. Хороший первый вопрос – а что такое итератор? Следующие два раздела постараются дать ответ на это.

### 3.1 Арифметика указателей

Мы получили следующее задание на программирование. Имеем вектор целых чисел и значение целого. Если значение содержится в векторе, мы должны вернуть указатель на него, иначе возвращаем 0, показывающий, что значение не представлено. Вот реализация:

```
int* find(const vector<int> &vec, int value){
    for (int ix = 0; ix < vec.size(); ++ix)
        if (vec[ix] == value) return &vec[ix]; return 0;}

```

Мы проверяем функцию и удостоверяемся, что она работает. Получаем следующее задание – пусть

функция работает не только с целыми, но с любыми типами, для которых операция эквивалентности определена. Если вы прочитали Раздел 2.7, вы узнаете эту задачу, как требование преобразовать `find()` в шаблон функции:

```
template <typename elemType>
elemType* find(const vector<elemType> &vec, const elemType &value){
    for (int ix = 0; ix < vec.size(); ++ix)
        if (vec[ix] == value) return &vec[ix];    return 0;}

```

Вновь мы проверяем функцию и удостоверяемся, что она работает. Наша следующая задача заставить работать `find()` как с векторами, так и с массивами элементов любого типа, для которого определена операция равенства. Наша первая мысль – перезагружаемая функция, поддерживающая вариант, включающий как вектор, так и массив.

Мы предупреждены против применения перезагружаемости в этом случае. Слегка подумав, скажем мы, мы могли бы разработать `find()` так, чтобы один вариант мог поддерживать элементы либо вектора, либо встроенного массива. Однако ж, это кажется нелегко.

Одна из стратегий решения тяжелых проблем – разбить ее на некоторое количество меньших, и будем надеяться, более простых проблемок. В этом примере, наша одна большая проблема разбивается на (1) прохождение по элементам массива для `find()` без спецификации массива и (2) прохождение по элементам вектора для `find()` без спецификации вектора. В идеале решение этих двух проблем содержит общее решение нашей исходной проблемы.

Давайте вначале решим проблему встроенного массива. Как мы можем пройти нашей `find()` по элементам массива без спецификации собственно массива?

Программное решение проблемы безмерно упрощается, когда мы понимаем проблему, которую пытаемся решить. В нашем случае поможет вначале понять, как массив передается в функцию и возвращается ею. Когда я пишу

```
int min(int array[24]) { ... }
```

это представляется, как если бы `min()` допускала только массив из 24 элементов, и что реальный массив передан по значению. Фактически, разве предположения верны: массив не скопирован по значению, и массив любого размера может быть передан в `min()`. Я знаю, вы подумали – «как это?»

Когда массив передается в функцию или возвращается из нее, передается только адрес первого элемента. Вот более аккуратное объявление `min()`:

```
int min(int *array) { ... }
```

`min()` допускает массивы целых любого размера: 1, 32, 1024 и т.д. Альтернатива вызывала бы беспокойство, поскольку требовала бы от нас создавать различные варианты `min()` для каждого массива с его уникальной размерностью.

Указатель на начало массива позволяет `min()` начать чтение массива. Как-нибудь мы должны обозначить

для `min()`, когда чтение следует закончить. Один из путей – добавить параметр, который удерживал бы размер массива. Например, вот наше объявление `find()`, использующее эту стратегию:

```
template <typename elemType>
elemType* find(const elemType *array, int size, const elemType &value);
```

Альтернативное решение – передать адрес, который, когда достигнут, просигналит, что мы закончили чтение элементов массива. (Мы назовем это значение «часовой» (*sentinel*)).

```
template <typename elemType>
elemType* find(const elemType *first, const elemType *sentinel,
               const elemType &value);
```

Один интересный аспект этого решения в том, что мы исключаем объявление массива из списка параметров – решение первой нашей маленькой проблемки.

Теперь давайте посмотрим, как реализована каждая версия `find()`. В первой версии мы получаем доступ к каждому элементу по очереди, начиная с 0 и проходя к `размер-1`. Первый вопрос вот какой: поскольку массив передается в `find()` как указатель на его первый элемент, то, как мы добираемся до элементов по их позиции? Хотя мы и добираемся до массива через указатель, мы еще используем оператор индексирования в точности, как делали это раньше:

```
template <typename elemType>
elemType* find(const elemType *array, int size, const elemType &value){
    if (! array || size < 1) return 0;
    for (int ix = 0; ix < size; ++ix)
        // мы можем использовать оператор индексирования для указателя
        if (array[ix] == value) return &array[ix];
    return 0; // значение не найдено}
```

Не смотря на то, что массив передается в `find()`, как указатель на его первый элемент, мы видим, что индивидуальные элементы могут быть достигнуты посредством оператора индексирования, как если бы массив был объектом. Почему? На практике индексирование выполняется добавлением индекса к начальному адресу массива для получения адреса элемента. Этот адрес затем разыменовывается для возвращения значения элемента. Например,

```
array[2];
```

возвращает значение третьего элемента массива (индексация, напомним, начинается с 0). Следующее также возвращает значение третьего элемента:

```
*(array + 2)
```

Если адрес первого элемента массива `array` это 1000, какой адрес имеет `array+2`? 1002 – разумный, но неверный ответ. 1002 – это арифметика целых. `Array+2`, однако, представляет *арифметику указателей*. Арифметика указателей увеличивает размер типа используемой адресации.

Положим, что `array` содержит целые элементы. `array+2` тогда добавляет размер двух целых элементов к адресу `array`. В арифметике указателей на машине, где целое имеет 4 байта, ответ - 1008.

Когда мы имеем адрес элемента, мы должны затем разыменовать адрес для получения значения элементов. Когда мы пишем `array[2]`, арифметика указателей и разыменовывание адреса выполняются автоматически.

Вот альтернативная реализация `find()`, в которой мы адресуемся к каждому элементу через указатель. Для адресации к каждому элементу `array` по очереди, мы увеличиваем его на 1 с каждой итерацией (с каждым проходом) нашего цикла. Для чтения адресуемого элемента мы должны разыменовать указатель. Что и есть - `array` возвращает адрес элемента, а `*array` возвращает его значение..

```
template <typename elemType>
elemType* find(const elemType *array, int size, const elemType &value){
    if (! array || size < 1) return 0;
    // ++array увеличивает array на один элемент
    for (int ix = 0; ix < size; ++ix, ++array)
        // *array разыменовывает адрес
        if (*array == value) return array; return 0;}

```

В следующей версии мы заменяем параметр размера вторым указателем, который служит в качестве «часового» (sentinel address – сторожевой адрес). Вот версия, которая позволяет нам удалить при объявлении массив из списка параметров:

```
template <typename elemType>
elemType* find(const elemType *first, const elemType *last,
               const elemType &value){
    if (! first || ! last) return 0;
    // пока первый не равен последнему,
    // сравним значение с элементом, адресованным первым
    // если два элемента равны, возвращаем первый
    // иначе увеличиваем первый до адреса следующего элемента
    for (; first != last; ++first)
        if (*first == value) return first;
    return 0;}

```

Это реализует первую из наших программных подзадач: мы устроили для `find()` доступ к каждому элементу нашего массива независимо от того, какие объекты в качестве элементов содержатся в нем. Как вызывается `find()`? Следующий код использует арифметику указателей, проиллюстрированную ранее:

```
int ia[8] = { 1, 1, 2, 3, 5, 8, 13, 21 };
double da[6] = { 1.5, 2.0, 2.5, 3.0, 3.5, 4.0 };
string sa[4] = { "pooh", "piglet", "eeyore", "tigger" };
int *pi = find(ia, ia+8, ia[3]);

```

```
double *pd = find(da, da+6, da[3]);
string *ps = find(sa, sa+4, sa[3]);
```

Второй адрес, помеченный 1, передается последнему элементу массива. Правильно ли это? Да. Если мы когда-либо попытаемся прочитать или написать по этому адресу, готов спорить, все кончено. Но если все, что мы делаем с этим адресом, это сравниваем его с адресом другого элемента, все чудесно. Адрес 1, переданный последнему элементу массива, служит часовым, сигнализирующим, что мы завершили нашу итерацию.

Как можем мы реализовать вторую программную подзадачу, получить доступ к каждому элементу вектора, независимо от того, элементы какого рода в нем? Вектор также сохраняет свои элементы в смежной области памяти, так что мы можем передать `find()` пару начало/конец адресов тем же путем, каким мы все устроили для встроенного массива, за исключением одного случая. В отличие от массива вектор может быть пустым. Например,

```
vector<string> svec;
```

определяет пустой вектор строковых элементов. Следующий вариант `find()`, если `svec` пуст, некорректен, а результатом станет крах программы при выполнении:

```
find(&svec[0], &svec[svec.size()], search_value);
```

Для безопасности следует вначале проверить, что `svec` не пуст.

```
if (! svec.empty()) // ... ok, вызываем find()
```

Хотя это безопасно, оно выглядит для пользователя неуклюже. Более привычный путь доступа к адресу первого элемента «обернуть» операцию в функцию, во что-нибудь похожее на следующее:

```
template <typename elemType>
inline elemType* begin(const vector<elemType> &vec)
{
    return vec.empty() ? 0 : &vec[0];
}
```

Вторая функция `end()` возвращает либо 0, либо указатель на 1, переданный последним элементом вектора. В этом случае мы можем безопасно и привычно вызвать `find()` для любого вектора:

```
find(begin(svec), end(svec), search_value);
```

Более того - это решение нашей оригинальной программной задачи по созданию `find()` так, чтобы единый вариант мог получить доступ, как к вектору, так и к встроенному массиву. Вновь мы проверяем функцию и убеждаемся, что она работает.

Превосходно, говорим мы. Теперь расширим функцию `find()` так, чтобы единый вариант мог также поддерживать класс списков стандартной библиотеки. Вот это трудно.

Класс списков это тоже контейнер. Разница в том, что элементы списка связаны через пару указателей: первый адресует к следующему элементу, второй адресует к предыдущему.

Арифметика указателей не работает со списком. Арифметика указателей подразумевает, что элементы смежные. Добавляя размер одного элемента к текущему указателю, мы переустанавливаем указатель на адресацию следующего элемента. Это базовое положение нашей реализации `find()`. К сожалению, это положение не поддерживается, когда мы добираемся до следующего элемента списка.

Первой мыслью будет вновь использовать перезагрузку второго варианта `find()`, которая допускает

объекты списка. Поведение указателей на встроенный массив, вектор и список будет, мы утверждаем, совершенно различным, как бы ни хотелось нам достичь унификации синтаксиса для доступа к следующему элементу.

Да и нет. Да, поведение базовых указателей слишком различно для унификации синтаксиса. И нет, мы не нуждаемся в обеспечении второго варианта для поддержки класса списков. Фактически, мы не нуждаемся в изменении реализации `find()` совсем, за исключением списка параметров.

Решение в обеспечении слоя абстракции над поведением базовых указателей. Чем программировать базовые указатели непосредственно, мы программируем слой абстракции. Мы помещаем исключительную поддержку базовых указателей в этот слой, защищая их от наших пользователей. Эта техника позволяет нам поддерживать любые классы контейнеров стандартной библиотеки единым вариантом `find()`.

## 3.2 Придадим смысл итераторам

Очевидный вопрос, а как мы реализуем этот слой абстракции? Нам нужна подборка объектов, которая поддерживает то же множество операторов, что и встроенный указатель (`++`, `*`, `==`, `!=`), но которая позволит нам осуществить уникальное выполнение этих операторов. Именно это мы можем сделать с механизмом создания классов `C++`. Мы разработаем множество классов *итераторов*, программируемых с тем же синтаксисом, что и указатели. Например, если `first` и `last` из класса итераторов списка, мы можем написать

```
// first и last объекты класса итераторов
while (first != last) {
    cout << *first << ' ';
    ++first;}

```

так же, как если бы `first` и `last` были обычными указателями. Разница в том, что оператор разыменовывания (`*`), оператор неравенства (`!=`), и оператор увеличения (`++`) представляют вызовы функции `inline`, ассоциированные с классом итераторов. Для класса итераторов списка, например, ассоциированный инкремент (увеличение) функции перемещается к следующему элементу, следуя за указателем списка. Для класса векторов инкремент функции перемещается к следующему элементу, добавляя размер одного элемента к текущему адресу.

В Части 4 мы рассмотрим, как реализовать класс итераторов, и как осуществить варианты функций обычных операторов. В этом же разделе мы рассмотрим, как определить и использовать итераторы, ассоциированные с классами контейнеров стандартной библиотеки.

Где мы возьмем итераторы? Каждый класс контейнера предусматривает оператор `begin()`, который возвращает итератор, адресуемый к первому элементу контейнера, и оператор `end()`, возвращающий итератор, адресуемый к 1 после последнего элемента контейнера. Например, не учитывая, как мы определим объект итератора в данный момент, мы назначим, сравним, увеличим и разыменуем итератор следующим образом:

```
for (iter = svec.begin(); iter != svec.end(); ++iter)

cout << *iter << ' ';

```

Прежде, чем мы разберемся, как определить итератор, давайте задумаемся на минутку об информации, которую это определение должно обеспечивать: тип контейнера, через который осуществляется итерация и который определяет, как осуществляется доступ к следующему элементу; и тип элемента, к которому

адресуются, с определением значения, возвращаемого после разыменовывания итератора.

Один возможный синтаксис для определения итератора – это передать эти два типа, как параметры в классе итератора:

```
// один из возможных синтаксисов итератора

// заметьте: в действительности не используется в STL

iterator< vector, string > iter;
```

Действительный синтаксис выглядит значительно более сложно. Во всяком случае, на первый взгляд. Он также позволяет более элегантные решения, хотя это может быть не очевидно, по крайней мере, до тех пор, пока мы не реализуем и не используем класс итератора в [Части 4](#).

```
vector<string> svec;

// синтаксис итератора стандартной библиотеки

// iter адресуется к элементам вектора строкового типа

// это инициализация первого элемента svec

vector<string>::iterator iter = svec.begin();
```

`iter` определен, как итератор для векторов строковых элементов. Он инициализируется адресацией к первому элементу `svec`. (Двойное двоеточие `[ : ]` означает, что итератор имеет тип, вложенный в определение строкового вектора. Это будет иметь больше смысла после прочтения [Части 4](#) и создания нашего собственного класса итератора. Теперь же, мы только используем итератор). Для `const` вектора, такого как

```
const vector<string> cs_vec;
```

мы просматриваем элементы, используя `const_iterator`:

```
vector<string>::const_iterator = cs_vec.begin();
```

`const_iterator` позволяет нам прочитать элементы вектора, но не записать их.

Для доступа к элементам через итератор мы разыменовываем его, в точности так, как делали со встроенным указателем:

```
cout << "string value of element: " << *iter;
```

Аналогично, чтобы вызвать операцию перехода к следующему строковому элементу через `iter`, мы используем синтаксис “стрелок” для конкретного выбора:

```
cout << "(" << iter->size() << "): " << *iter << endl;
```

Вот переделка `display()`, как функции-шаблона, с использованием итераторов вместо индексного оператора:

```
template <typename elemType>
void display(const vector<elemType> &vec, ostream &os){
```

```

vector<elemType>::const_iterator iter = vec.begin();
vector<elemType>::const_iterator end_it = vec.end();

// если vec пусто, iter и end_it эквивалентны
// и цикл for никогда не выполняется
for (; iter != end_it; ++iter) os << *iter << ' '; os << endl;}

```

Наша переделка `find()` поддерживает либо пару встроенных указателей, либо пару итераторов к контейнеру определенного типа:

```

template <typename IteratorType, typename elemType >
IteratorType find(IteratorType first, IteratorType last,
                 const elemType &value){

    for (; first != last; ++first) if (value == *first) return first;

    return last;}

```

Посмотрите, как мы можем использовать эту переделанную `find()` со встроенным массивом, вектором и списком:

```

const int asize = 8;
int ia[asize] = { 1, 1, 2, 3, 5, 8, 13, 21 };

// инициализируем список и вектор с 8 элементами ia
vector<int> ivec(ia, ia+asize);
list<int> ilist(ia, ia+asize);

int *pia = find(ia, ia+asize, 1024);

if (pia != ia+asize) // found ...

vector<int>::iterator it;
it = find(ivec.begin(), ivec.end(), 1024);
if (it != ivec.end()) // found ...

list<int>::iterator iter;
iter = find(ilist.begin(), ilist.end(), 1024);
if (iter != ilist.end()) // found ...

```

Неплохо. Мы перенесли главное в `find()` достаточно удачно, и гораздо больше, чем мы себе представляли, и чем могли, когда мы начинали предыдущий раздел. Это не конец истории, однако, хотя и превосходное завершение этого раздела.

Реализации `find()` используют оператор равенства нижележащего типа элемента. Если тип нижележащего элемента не поддерживает оператор равенства, или пользователь хотел бы определить равенство элементов отдельно, этот вариант `find()` окажется слишком негибким. Как мы можем добавить гибкости? Одно из решений – заменить использование оператора равенства на функцию, передаваемую, как указатель на функцию. Второе решение – что-нибудь вроде *функционального объекта*, специального класса реализаций. В [Части 4](#) мы рассмотрим, как разработать функциональный объект.

Что мы завершим в наших следующих итерациях `find()` - это превращение ее в общий `find()` алгоритм. (`find_if()` придаст дополнительную гибкость в передаче указателя на функцию или функциональный объект, вместо применения оператора равенства для нижележащего объекта).

Есть более 60 общих алгоритмов. Ниже представлен неполный список (полный список и примеры использования каждого из них можно найти в Приложении В).

- **Search algorithms:** `find()`, `count()`, `adjacent_find()`, `find_if()`, `count_if()`, `binary_search()`, and `find_first_of()`.
- **Sorting and general ordering algorithms:** `merge()`, `partial_sort()`, `partition()`, `random_shuffle()`, `reverse()`, `rotate()`, and `sort()`.
- **Copy, deletion, and substitution algorithms:** `copy()`, `remove()`, `remove_if()`, `replace()`, `replace_if()`, `swap()`, and `unique()`.
- **Relational algorithms:** `equal()`, `includes()`, and `mismatch()`.
- **Generation and mutation algorithms:** `fill()`, `for_each()`, `generate()`, and `transform()`.
- **Numeric algorithms:** `accumulate()`, `adjacent_difference()`, `partial_sum()`, and `inner_product()`.
- **Set algorithms:** `set_union()` and `set_difference()`.

Алгоритмы, заканчивающиеся суффиксом `_if`, берут либо указатель на функцию, либо функциональный объект для определения эквивалентности. Вдобавок есть алгоритмы, модифицирующие контейнер, такие как `replace()` и `unique()`, имеют две разновидности: замещающая версия, которая изменяет оригинальный контейнер, и версия, которая возвращает копию модифицированного контейнера. Это, например, пара `replace()` и `replace_copy()` алгоритмов.

### 3.3 Операции общие для всех контейнеров

Следующие операции общие для всех классов контейнеров (как и для строкового класса):

- Операторы равенства (`==`) и неравенства (`!=`) возвращают истинно или ложно.
- Оператор присваивания (`=`) копирует один контейнер в другой.
- `empty()` возвращает истинно, если контейнер не содержит элементов.
- `size()` возвращает подсчет текущего числа элементов в контейнере.
- `clear()` удаляет все элементы.

Следующая функция выполняет каждый из этих операторов:

```
void comp(vector<int> &v1, vector<int> &v2){
    // два вектора равны?
    if (v1 == v2) return;

    // пуст ли вектор?
    if (v1.empty() || v2.empty()) return;

    // нет точки, определяющей его, пока мы не собираемся его использовать!
    vector<int> t;

    // присваиваем t наибольший вектор
    t = v1.size() > v2.size() ? v1 : v2;

    // ... используем t ...

    // ok. освободим t от его элементов
    // t.empty() теперь вернет true
    // t.size() вернет 0
```

```
t.clear();  
  
// ... ok, наполним t и используем еще раз ...}
```

Каждый контейнер поддерживает `begin()` и `end()` операции для возвращения, соответственно, итератора для первого элемента контейнера и 1 после последнего правильного элемента:

- `begin()` возвращает итератор для первого элемента.
- `end()` возвращает итератор, который адресуется 1 после последнего элемента.

Обычно мы начинаем итерацию через контейнер, стартуя от `begin()` и останавливаясь, когда достигаем `end()`. Все контейнеры поддерживают операцию `insert()` для добавления элементов и операцию `erase()` для удаления элементов.

- `insert()` добавляет один или несколько элементов в контейнер.
- `erase()` удаляет один или несколько элементов из контейнера.

Выполнение `insert()` и `erase()` очень зависит от того, является контейнер последовательным или ассоциативным. Последовательные контейнеры обсуждаются в следующей секции.

### 3.4 Использование последовательных контейнеров

Последовательные контейнеры содержат упорядоченный ряд элементов одного типа. Есть первый элемент, второй элемент и т.д. до последнего элемента. Вектор и список – два первичных последовательных контейнера. Вектор содержит свои элементы в смежной области памяти. Случайный доступ, например, для получения 5 элемента, затем 17, затем 9 – эффективен, поскольку каждый элемент фиксирован сдвигом от начала вектора. Добавление элемента в любую позицию, отличную от конца вектора, однако, неэффективно – каждый элемент справа от добавленного должен быть перекопирован по очереди до конца вектора. Аналогично, удаление любого элемента, кроме последнего, не эффективно.

Список представлен не смежной памятью, с двойной связью, чтобы позволить передвижения и вперед, и назад. Каждый элемент списка содержит три поля: значение, указатель на предыдущий элемент списка и указатель на следующий элемент списка. Добавление и удаление элементов внутри списка эффективно. Список должен просто установить соответствующие указатели назад и вперед. Случайный доступ, с другой стороны, поддержан менее эффективно. Для достижения элемента 5, затем 17, а затем 9 необходимо перемещение промежуточных элементов. (Можете представить себе, как разницу между CD и кассетной лентой при переходе с одной записи на другую).

Для представления элементов числовой последовательности вектор наиболее подходящий контейнер. Почему? Огромная доля использования случайного доступа. `fibon_elem()`, например, укладывается в контейнер, основанный на позиционном доступе к нему пользователя. Сверх того, мы удаляем элементы, а элементы всегда добавляются в конец вектора.

Когда список более удобен? Если бы мы читали тестовые метки из файла, и хотели бы запомнить каждую метку в возрастающем порядке, мы, вероятно, случайно заполняли бы контейнер каждой прочитанной меткой. В этом случае контейнер списка предпочтительнее.

Третий последовательный контейнер – это (*deque*) очередь с двусторонним доступом (произносится дек). Дек работает очень похоже на вектор – элементы запоминаются подряд. Но, в отличие от вектора, дек поддерживает эффективное добавление и удаление его первого элемента (как, впрочем, и последнего).

Если, например, нам нужно вставлять элемент перед контейнером, и удалять элемент из его конца - дек самый подходящий тип контейнера. (Класс дек стандартной библиотеки реализован с использованием дек, для содержания элементов очереди).

Для использования последовательного контейнера мы должны включить ассоциированный с ним файл заголовка, один из следующих:

```
#include <vector>

#include <list>

#include <deque>
```

Есть пять путей определения объекта последовательного контейнера:

1. Создать пустой контейнер:

```
list<string> slist;
vector<int> ivec;
```

2. Создать контейнер некоторого размера. Каждый элемент инициализируется его значением по умолчанию. (Напомню, что значения по умолчанию для встроенных арифметических типов, таких как `int` и `double` - это нуль).

```
list<int>  ilist(1024);
vector<string> svec(32);
```

3. Создать контейнер заданного размера и инициализировать каждый элемент его значением:

```
vector<int> ivec(10, -1);
list<string> slist(16, "unassigned");
```

4. Создать контейнер, снабженный парой итераторов, отмечающих область элементов, которыми инициализировать контейнер:

```
int ia[8] = { 1, 1, 2, 3, 5, 8, 13, 21 };

vector<int> fib(ia, ia+8);
```

5. Создать контейнер, содержащий другой контейнерный объект. Новый контейнер инициализируется копированием второго:

```
list<string> slist; // пустой
// заполняем slist ...
list<string> slist2(slist);

// копируем slist ...
```

Две специальные операции поддерживают добавление и удаление в конце контейнера: `push_back()` и `pop_back()`. `pop_back()` удаляет элементы, а `push_back()` добавляет элементы в конец. Дополнительно, списочный и дек (`deque`) контейнеры (но не вектор) поддерживают `push_front()` и

`pop_front()`, `pop_back()` и `pop_front()` операции не возвращают удаленные значения. Для чтения фронтальных значений мы используем `front()`, а для чтения последних `back()`. Например,

```
#include <deque>
deque<int> a_line;
int ival;
while (cin >> ival){
    // вставляем ival в конец a_line
    a_line.push_back(ival);

    // ok: читаем значение в начале a_line
    int curr_value = a_line.front();

    // ... что-нибудь делаем ...

    // удаляем значение в начале a_line
    a_line.pop_front();}
```

`push_front()` и `push_back()` - специализированные операции добавления. Существует четыре вариации более общей операции `insert()`, поддерживаемой каждым контейнером.

- `iterator insert(iterator position, elemType value)` добавляет `value` до `position`. Она возвращает `iterator`, адресуемый к добавленному элементу. Например, следующее добавляет `ival` в рассортированный ряд внутри `ilist`:

```
list<int> ilist;
// ... заполняем ilist

list<int>::iterator it = ilist.begin();
while (it != ilist.end())
    if (*it >= ival) {
        ilist.insert(it, ival);
        break; // покидаем цикл }

if (it == ilist.end())    ilist.push_back(ival);
```

- `void insert(iterator position, int count, elemType value)` добавляет `count` элементы `value` до `position`. Например,

```
string sval("Part Two");
list<string> slist;
// ... наполняем slist ...

list<string>::iterator it = find(slist.begin(), slist.end(), sval);

slist.insert(it, 8, string("dummy"));
```

- `void insert(iterator1 position, iterator2 first, iterator2 last)` добавляет область элементов, отмеченных `first`, `last` до `position`:

```
int ial[7] = { 1, 1, 2, 3, 5, 55, 89 };
int ia2[4] = { 8, 13, 21, 34 };
list<int> elems(ial, ial+7);
```

```
list<int>::iterator it = find(elems.begin(), elems.end(), 55);
elems.insert(it, ia2, ia2 + 4);
```

- `iterator insert(iterator position)` добавляет элемент до `position`. Элемент инициализируется значением по умолчанию его типа. `pop_front()` и `pop_back()` - специализированные операции удаления элемента. Существует две версии более общей операции `erase()`.
- `iterator erase(iterator posit)` удаляет элемент, адресованный `posit`. Например, используя `slist` определенную ранее, давайте `erase()` первый образец `str`:

```
list<string>::iterator it = find(slist.begin(), slist.end(), str);

slist.erase(it);
```

- `iterator erase(iterator first, iterator last)` удаляет элементы начиная с `first` до, но не включая `last`. Например, опять используя определенную ранее `list`, давайте `erase()` `num_times` образцы `str`:

```
list<string>::iterator
first = slist.begin(),
last = slist.end();

// it1: первый элемент для erase,
// it2: первый элемент после элемента для erase
list<string>::iterator it1 = find(first, last, str);
list<string>::iterator it2 = find(first, last, sval);

slist.erase(it1, it2);
```

Возвращенный итератор в обоих образцах `erase()` адресуется к элементу, следующему за элементом или областью удаления.

Класс списка не поддерживает сдвиговую арифметику его итераторов. Это причина, по которой мы не пишем

```
// error: сдвиговая арифметика не

// поддерживается для списков

classslist.erase (it1, it1+num_tries);
```

но взамен поддерживается `erase()` как для `it1`, так и для `it2`.

### 3.5 Использование общих алгоритмов

Для использования общих алгоритмов мы должны включить ассоциированный с `algorithm` файл заголовка:

```
#include <algorithm>
```

Давайте, выполним общие алгоритмы с нашим вектором числовой последовательности. `is_elem()` должен

вернуть true, если значение является элементом последовательности, иначе вернуть false. Четыре возможных базовых поисковых алгоритма:

1. `find()` ищет в неупорядоченной совокупности, отмеченной парой итераторов `first, last` для некоторых значений. Если значение найдено, `find()` возвращает итератор, адресуемый к значению, иначе возвращает итератор, адресуемый к `last`.
2. `binary_search()` ищет в упорядоченной совокупности. Возвращает true, если значение найдено, иначе false. `binary_search()` более эффективно, чем `find()`.
3. `count()` возвращает количество элементов, соответствующих некоторому значению.
4. `search()` сопоставляет подпоследовательности в контейнере. Например, для данной последовательности {1, 3, 5, 7, 2, 9}, `search` для подпоследовательности {5, 7, 2} возвращает итератор на начало последовательности. Если подпоследовательность не существует, возвращается итератор на конец контейнера.

Поскольку наш вектор гарантированно принадлежит восходящей последовательности, наилучший для нас выбор `binary_search()`:

```
#include <algorithm>
bool is_elem(vector<int> &vec, int elem){
    // если переданный elem это 34, 9й элемент
    // последовательности Фибоначчи, но запомненная последовательность
    // содержит только первые 6 элементов: 1,1,2,3,5,8
    // наш поиск не будет успешен.
    // Прежде, чем мы вызовем binary_search(),
    // мы должны проверить, не нужно ли пополнить последовательность

    return binary_search(vec.begin(), vec.end(), elem);}
```

В качестве комментария: до объявления вызова `binary_search()`, мы должны быть уверены, что числовая серия содержит элемент со значением, заключенном в `elem`, где оно член серии. Один из путей сделать это – сравнить наибольший элемент последовательности с `elem`. Если наибольший элемент меньше `elem`, мы расширяем последовательность, пока ее наибольший элемент становится равен или превышает `elem`.

Одна из стратегий определения наибольшего элемента серии – это использование общего алгоритма `max_element()`. `max_element()` передается итераторной парой, отмечающей область элементов исследования. Возвращает наибольший элемент вектора. Вот наша переделка `is_elem()`:

```
#include <algorithm>

// предваряющее объявление

extern bool grow_vec(vector<int>&, int);

bool is_elem(vector<int> &vec, int elem){
    int max_value = max_element(vec.begin(), vec.end());
    if (max_value < elem) return grow_vec(vec, elem);

    if (max_value == elem) return true;

    return binary_search(vec.begin(), vec.end(), elem);}
```

`grow_vec()` добавляет элементы в вектор, пока элемент в последовательности не станет равен или больше, чем `elem`. Если элемент последовательности равен `elem`, возвращается true, иначе false.

Конечно, поскольку наш вектор содержит восходящую последовательность, мы не нуждаемся в полной мере, чтобы `max_element()` находила наибольший элемент, достаточно гарантии иметь позицию `vec.size()-1` для непустого вектора:

```
int max_value = vec.empty() ? 0 : vec[vec.size()-1];
```

`binary_search()` требует, чтобы контейнер был упорядочен, но оставляет на совести программиста гарантии этого. Если мы не уверены, мы можем `copy()` наш контейнер в другой контейнер:

```
vector<int> temp(vec.size());
copy(vec.begin(), vec.end(), temp.begin());
```

Теперь мы `sort()` наш временный контейнер, перед вызовом `binary_search()`:

```
sort(temp.begin(), temp.end());

return binary_search(temp.begin(), temp.end(), elem);
```

`copy()` берет два итератора, которые отмечают размер элементов для копирования. Третий итератор указывает на первый элемент целевого контейнера. Элементы затем устанавливаются поочередно. На нашей совести уверенность, что целевой контейнер достаточно велик, чтобы были скопированы все элементы. Если мы не уверены, мы можем использовать «заглушку» для перезаписи определенной последовательности со вставкой (смотрите Раздел 3.9 для ознакомления).

Приложение В содержит пример использования каждого общего алгоритма.

## 3.6 Как разрабатывать общий алгоритм

Вот наша задача. Нам дается вектор целых. Нас просят вернуть новый вектор, содержащий все значения меньше 10. Быстрое, но не гибкое решение такое:

```
vector<int> less_than_10(const vector<int> &vec) {
    vector<int> nvec;
    for (int ix = 0; ix < vec.size(); ++ix)
        if (vec[ix] < 10) nvec.push_back(vec[ix]);

    return nvec;}
```

Если пользователь захочет получить все элементы меньше 11, мы должны либо создать новую функцию, либо обобщить готовую, дав возможность пользователю определить значение, с которым нужно сравнивать элементы. Например,

```
vector<int> less_than(const vector<int> &vec, int less_than_val);
```

Но наша следующая задача будет потруднее. Мы должны дать возможность пользователю определить альтернативную операцию, такую как «больше, чем», «меньше, чем» и т.д. Как мы можем параметризовать операцию?

Одно из решений – заменить оператор «меньше, чем» вызовом функции. Мы добавим третий параметр `pred`, определяющий указатель на функцию, имеющую список параметров из двух целых и

возвращающую `bool.less_than()` это теперь неправильное имя, так что давайте назовем ее `filter()`:

```
vector<int> filter(const vector<int> &vec,int filter_value,
                 bool (*pred)(int, int));
```

Для удобства нашего пользователя мы так же определяем некоторое количество родственных функций, которые могут передаваться в `filter()`:

```
bool less_than(int v1, int v2){ return v1 < v2 ? true : false; }
bool greater_than(int v1, int v2){ return v1 > v2 ? true : false; }
```

и т.д. Пользователь может теперь либо передать ту или иную из этих функций в вызов `filter()`, или определить ее собственную родственную функцию. Единственное неудобство в том, что передаваемая функция должна вернуть `bool` и получить два целых в свой список параметров. Вот как `filter()` может быть вызвана:

```
vector<int> big_vec;
int value;
// ... наполним big_vec и value
vector<int> lt_10 = filter(big_vec, value, less_than);
```

Единственное, что нам осталось – фактически реализовать `filter()`:

```
vector<int> filter(const vector<int> &vec,int filter_value,
                 bool (*pred)(int, int)) {
    vector<int> nvec;
    for (int ix = 0; ix < vec.size(); ++ix)
        // вызываем функцию, адресованную pred
        // проверяем элемент vec[ix] с filter_value
        if (pred(vec[ix], filter_value)) nvec.push_back(vec[ix]);
    return nvec;}
```

Эта реализация `filter()` явно проходит по всем элементам в цикле `for`. Давайте, заменим использование цикла `for` на общий алгоритм `find_if()`. Мы повторно применяем `find_if()` к последовательности для идентификации каждого элемента, который встречается с критерием, определенным указателем на функцию, обозначенным пользователем. Как можем мы сделать это?

Начнем с поиска всех элементов равных 10. общий алгоритм `find()` дает три аргумента: два итератора, отмечающие первый и 1 после последнего проверяемого элемента, и значение, которое мы ищем. В следующем коде, `count_occurs()`, иллюстрируется, как использовать `find()` повторно для контейнера без остановки на каждом элементе дважды:

```
int count_occurs(const vector<int> &vec, int val){
    vector<int>::const_iterator iter = vec.begin();
    int occurs_count = 0;
    while ((iter = find(iter, vec.end(), val)) != vec.end()){
        ++occurs_count;
        ++iter; // адресуемся к следующему элементу}
```

```
return occurs_count;}
```

Цикл `while` определяет `iter` на возвращаемое значение `find()`. `find()` возвращает итератор, адресуемый к элементу равному `val`, или, если нет подходящего элемента, итератор равен `vec.end()`. Когда `iter` равен `vec.end()`, цикл прерывается.

Успех цикла `while` зависит от того, насколько успешно наращивание `iter` на 1 приводит к нужному элементу с каждым проходом цикла. Например, положим, что `vec` содержит следующие элементы: `{6, 10, 8, 4, 10, 7, 10}`. Объявление

```
vector<int>::const_iterator iter = vec.begin();
```

инициализирует `iter` для адресации к первому элементу вектора, который имеет значение 6. `find()` возвращает итератор, адресуемый ко второму элементу. Прежде, чем мы вновь вызовем `find()`, мы должны увеличить `iter` на 1. `find()` при следующем вызове будет с `iter`, адресуемым к третьему элементу. `find()` возвращает итератор, адресуемый к пятому элементу, и т.д.

## Объекты функции

Перед тем, как переделать `filter()` для поддержки `find_if()`, давайте рассмотрим predefined объекты функции, поддерживаемые стандартной библиотекой. *Объект функции* – это копия класса, который поддерживает перезагружаемую копию оператора вызова функции. Перезагружаемый вызов оператора позволяет объекту функции быть использованным так же, как, если бы он был функцией.

Реализовав объект функции, с другой стороны, что мы будем определять, как независимую функцию. Почему мы беспокоимся? Первая причина в эффективности. Мы можем вызвать оператор `inline`, попутно выделив заголовок вызова функции, который получим с вызовом оператора через указатель на функцию.

Предопределения стандартной библиотеки – множество объектов отношения, арифметических и логических объектов функции. В следующем списке `type` заменяется встроенным типом или типом класса в реальном использовании объекта функции:

- Six arithmetic function objects: `plus<type>`, `minus<type>`, `negate<type>`, `multiplies<type>`, `divides<type>`, `modulus<type>`
- Six relational function objects: `less<type>`, `less_equal<type>`, `greater<type>`, `greater_equal<type>`, `equal_to<type>`, `not_equal_to<type>`
- Three logical function objects, using the `&&`, `||`, and `!` operators, respectively: `logical_and<type>`, `logical_or<type>`, and `logical_not<type>`

Для использования predefined объектов функции мы должны включить соответствующий файл заголовка:

```
#include <functional>
```

Например, по умолчанию `sort()` упорядочивает свои элементы в восходящем порядке, используя оператор «меньше, чем» нижележащего типа элемента. Если мы передадим объект функции «больше, чем», элементы будут сортироваться в нисходящем порядке:

```
sort(vec.begin(), vec.end(), greater<int>());
```

Синтаксис

```
greater<int>()
```

создает безымянный объект шаблона класса «больше» и передает его в `sort()`.

`binary_search()` ожидает элементы, которые он отыскивает для сортировки оператором «меньше, чем». Чтобы поиск в нашем векторе проходил корректно, мы должны теперь передать его в вызов объекта функции, используемый для сортировки нашего вектора:

```
binary_search(vec.begin(), vec.end(), elem, greater<int>());
```

Давайте, выведем на дисплей серию Фибоначчи в виде возрастающей непроницаемой маскировки – каждый элемент прибавляется к себе, каждый элемент умножается на себя, каждый элемент добавляется к связанному с ним в серии Pell, и т.д. Один из вариантов в использовании `transform()` общего алгоритма и объектов функций `plus<int>` и `multiplies<int>`.

`transform()` должна передать (1) пару итераторов для маркировки границ элементов для преобразования, (2) итератор на точку начала контейнера, из которого извлекаются значения, используемые для преобразования, (3) итератор на точку начала контейнера, где мы разместим результаты каждого преобразования, и (4) объект функции (или указатель на функцию), представляющий используемую операцию. Например, вот наше добавление к элементам последовательности Pell соответствующих из серии Фибоначчи:

```
transform(fib.begin(), fib.end(), // (1)
pell.begin(), // (2)
fib_plus_pell.begin(), // (3)
plus<int>()); // (4)
```

В этом примере целевой вектор `pell` должен быть, по меньшей мере, таким же большим, как `fib`, или в противном случае алгоритм `transform()` переполнит `pell`.

В случае следующего вызова `transform()` мы должны умножить каждый элемент на себя и сохранить результат поверх оригинального элемента:

```
transform(fib.begin(), fib.end(), // (1)
fib.begin(), fib.begin(), // (2), (3)
multiplies<int>()); // (4)
```

## Адаптеры объекта функции

Эти объекты функции совсем не работают в том, что нам нужно сделать с `find_if()`. Объект функции `less<type>`, например, ожидает двух значений. Он становится истинным, если первое значение меньше второго. В нашем случае каждый элемент должен сравниваться со значением, заданным пользователем. В идеале, нам бы подошло вернуть `less<type>` в унарный оператор, связав второе значение с тем, которое определил пользователь. Таким путем `less<type>` сравнит каждый элемент с этим значением. Можем ли мы так сделать? Да. Стандартная библиотека поддерживает механизм *адаптера* именно для этой цели.

Адаптер объекта функции модифицирует объект функции. *Связующий* адаптер конвертирует бинарный объект функции в унарный объект, связывая один из аргументов с частным значением. Это именно то, что нам нужно. Есть два связывающих адаптера: `bind1st`, который связывает значение с первым операндом, и `bind2nd`, который связывает значение со вторым. Вот возможная модификация `filter()`, использующая адаптер `bind2nd`:

```
vector<int> filter(const vector<int> &vec,
```

```

        int val, less<int> &lt;){
vector<int> nvec;
vector<int>::const_iterator iter = vec.begin();

// bind2nd(less<int>, val)
// связываем val со вторым значением less<int>
// less<int> теперь сравнивает каждое значение с val

while((iter = find_if(iter, vec.end(),bind2nd(less<int>, val))) != vec.end()){
    // каждый раз, когда iter != vec.end(),
    // iter адресует к элементу меньшему, чем val
    nvec.push_back(*iter);
    iter++;}
return nvec;}

```

Как можем мы обобщить `filter()`, чтобы в дальнейшем исключить зависимость и от типа элемента вектора, и от собственно контейнера вектора? Чтобы исключить зависимость от типа элемента, мы обратим `filter()` в функцию шаблона и добавим тип в наше объявление шаблона. Для исключения зависимости от контейнера вектора мы передаем в `first, last` пару итераторов. Вместо этого мы добавим другой итератор в список параметров, обозначив, где именно мы собираемся начать копировать элементы. Вот такая получается реализация:

```

template <typename InputIterator, typename OutputIterator,
         typename ElemType, typename Comp>
OutputIterator filter(InputIterator first, InputIterator last,
                    OutputIterator at, const ElemType &val, Comp pred){
    while ((first = find_if(first, last, bind2nd(pred, val))) != last){
        // только посмотрим, что происходит ...
        cout << "found value: " << *first << endl;

        // присвоим значение, затем передвинем оба итератора вперед
        *at++ = *first++;}
return at;}

```

Вы видите, как можно реально вызвать `filter()`? Давайте напишем маленькую программу для тестирования этого и со встроенным массивом, и с вектором. Нам нужно два типа контейнеров: один для содержания значений, которые должны фильтроваться, и второй для содержания элементов, которые будут отобраны. Пока мы определим целевой контейнер того же размера, что и оригинал. В Секции 3.9 мы увидим альтернативное решение, использующее вставку итераторных адаптеров.

```

int main(){
    const int elem_size = 8;

    int ia[elem_size] = { 12, 8, 43, 0, 6, 21, 3, 7 };
    vector<int> ivec(ia, ia+elem_size);

    // контейнер для содержания результатов нашей filter()
    int ia2[elem_size];
    vector<int> ivec2(elem_size);

    cout << "filtering integer array for values less than 8\n";
    filter(ia, ia+elem_size, ia2, elem_size, less<int>());

    cout << "filtering integer vector for values greater than 8\n";
    filter(ivec.begin(), ivec.end(), ivec2.begin(),
          elem_size, greater<int>());}

```

После компиляции, при выполнении программа генерирует следующее:

```
filtering integer array for values less than 8
found value: 0 found value: 6 found value: 3 found value: 7
filtering integer vector for values greater than 8
found value: 12 found value: 43 found value: 21
```

*Инвертирующий* адаптер обращает значение истинности объекта функции. `not1` обращает значение истинности унарного объекта функции. `not2` обращает значение истинности бинарного объекта функции. Например, для идентификации элементов больших или равных 10 мы можем инвертировать результат объекта функции `less<int>()`:

```
while((iter =find_if(iter,vec.end(),not1(bind2nd(less<int>,10))))!= vec.end())
```

В основном, нет единого решения проблемы. Наш подход к поиску всех элементов меньших, чем некоторое значение, например, использует поочередное рассмотрение каждого элемента и копирование каждого значения, если оно меньше, чем заданное. Это решает нашу проблему, но это не единственный подход, который мы могли бы применить.

Альтернативный подход следующий: вначале мы сортируем вектор. Затем, используя `find_if()`, мы находим первый элемент, больший заданного. И, наконец, мы удаляем все элементы от этого найденного до конца вектора. В действительности мы сортируем локальную копию вектора. Пользователи могут не оценить наши изменения в последовательности элементов их вектора. Вот версия этого решения без использования шаблона:

```
vector<int> sub_vec(const vector<int> &vec, int val) {
    vector<int> local_vec(vec);sort(local_vec.begin(), local_vec.end());
    vector<int>::iterator iter=find_if(local_vec.begin(),local_vec.end(),
                                     bind2nd(greater<int>, val));
    local_vec.erase(iter, local_vec.end());
    return local_vec;}

```

Ладно. Это важный раздел. Для более глубокого понимания, возможно, потребуется его перечитать и написать несколько программ. Хорошим упражнением было бы попытаться превратить `sub_vec()` в функцию шаблона по линии `filter()`. Давайте я подведу итог сделанного нами.

Мы начали с функции для поиска элементов в векторе целых, имеющих значение меньше, чем 10. Мы решили, что прямое кодирование элементов слишком частное решение.

Вначале мы добавили параметр, позволяющий пользователю, назначить значение, с которым будут сравниваться элементы вектора.

Затем мы добавили указатель на параметр функции, чтобы пользователь мог обозначить, какой сравнивающий фильтр применить.

Позже мы ввели объекты функции, которые поддерживают альтернативный, более эффективный метод передачи операции в функцию. Мы кратко остановились на встроенных объектах функции, предоставляемых стандартной библиотекой. (В [Части 4](#) мы рассмотрим, как создать наши собственные объекты функции).

В завершение мы перестроили функцию, как функцию шаблона. Для поддержки множественных типов контейнеров, мы передали итераторную пару, отмечающую первый и 1 после последнего элемента прохода. Для поддержки множественных типов элементов, мы параметризовали тип элемента. Для поддержки как указателей на функции, так и объектов функции, мы параметризовали операцию сравнения для применения к элементам.

Наша функция теперь не зависит от типа элементов, операции сравнения, и контейнера. Вкратце, мы преобразовали нашу исходную функцию в общий алгоритм.

### 3.7 Использование отображений

*Отображение* (*map*) определяется, как пара значений: ключ, обычно строка, используемая в качестве индекса, и значение, ассоциированное с этим ключом. Словарь – один из примеров отображений. Программа анализирует подсчет обнаруженных слов в тексте, хранящем отображение со строковым ключом и целым значением, представляющим местоположение:

```
#include <map>

#include <string>

map<string,int> words;
```

Простейший путь ввести пару ключ/значение

```
words["vermeer"] = 1;
```

Для нашей программы, отыскивающей слова, мы можем написать следующее:

```
string tword;

while (cin >> tword) words[tword]++;
```

Выражение

```
words[tword]
```

отыскивает значение, ассоциированное с содержимым строки `tword`. Если `tword` не присутствует в отображении, оно вставляется в отображение со значением по умолчанию 0. Оператор увеличения изменяет это значение на 1.

Следующий цикл `for` печатает слова и вычисленное местонахождение:

```
map<string,int>::iterator it = words.begin();
```

```

for (; it != words.end(); ++it)

    cout << "key: " << it->first << "value: " << it->second << endl;

```

Член, названный `first` получает доступ к ключу отображения, и в этом случае строка представляет слово. Член, названный `second` получает доступ к значению, и это местонахождение слова.

Есть три пути обращения к отображению с целью выяснить присутствие ключа. Очевидный путь – использовать ключ в качестве индекса:

```

int count = 0;

if (!(count = words[ "vermeer"])) // vermeer не присутствует

```

Неудобство в том, что индексация отображения *вставляет* ключ в отображение, если он еще не присутствует. Его значению придается значение по умолчанию, ассоциированное с его типом. Если `"vermeer"` не присутствует, например, эта форма поиска вставляет в отображение нулевое местоположение.

Второй путь запросов – использовать операцию `find()`, ассоциированную с отображением (это не общий алгоритм `find()`). `find()` вызывается со значением ключа:

```

words.find("vermeer");

```

Если значение ключа присутствует, `find()` возвращает итератор на пару ключ/значение. Иначе она возвращает `end()`:

```

int count = 0;

map<string,int>::iterator it;

it = words.find("vermeer");

if (it != words.end())    count = it->second;

```

Третья альтернатива – запросить отображение с использованием операции `count()`, ассоциированной с отображением. `count()` возвращает число местонахождений предмета в отображении.

```

int count = 0;

string search_word("vermeer");

if (words.count(search_word)) // ok: присутствует ...

count = words[search_word];

```

Отображение может иметь только одно местонахождение для каждого ключа. Если мы хотим сохранить множество вставок ключа, мы должны использовать *множественное отображение* (*multimap*). (Я не обсуждаю множественные отображения в этой книге. Посмотрите [[LIPPMAN98](#)], Раздел 6.15 для ознакомления, там есть и примеры использования).

## 3.8 Использование множества

Множество – это набор ключевых значений. Множество используется тогда, когда нам нужно знать,

присутствует ли значение. В алгоритме графа, например, мы можем использовать множество для содержания каждого пройденного узла. Прежде, чем мы перейдем к следующему узлу, мы запрашиваем множество – не был ли узел уже пройден?

Программа поиска слов в предыдущем разделе, например, могла бы выбирать не учтенные слова. Чтобы это сделать, мы определяем множество, строкового типа, слов-исключений:

```
#include <set>
#include <string>
set<string> word_exclusion;
```

Перед вводом слова в наше отображение, мы проверяем, не присутствует ли оно во множестве `word_exclusion`:

```
while (cin >> tword){
    if (word_exclusion.count(tword))
        // присутствует во множестве исключаемых слов?
        // тогда пропустим остаток этой итерации
        continue;

    // ok: если здесь, слов-исключений нет

    words[tword]++;}
```

Выражение `continue` предлагает циклу пропустить оставшиеся выражения текущего прохода цикла. В этом случае, если `tword` принадлежит множеству `word_exclusion`

```
words[tword]++;
```

выражение никогда не будет выполнено. Цикл `while` вместо начала следующей итерации оценивает

```
while (cin >> tword)
```

Множество содержит только один вариант для каждого значения ключа. (Для сохранения множества значений ключа, мы должны использовать *мультимножество*. Вновь отсылаю к [LIPPMAN98], Раздел 6.15 для ознакомления, и примеров использования).

По умолчанию элементы упорядочиваются с использованием оператора «меньше, чем» соответствующего типа элементов. Например, возьмем

```
int ia[10] = { 1, 3, 5, 8, 5, 3, 1, 5, 8, 1 };

vector<int> vec(ia, ia+10);
set<int> iset(vec.begin(), vec.end());
```

элементы множества `iset` это `{1, 3, 5, 8}`.

Отдельные элементы множества добавляются с помощью единственного аргумента

```
insert():iset.insert(ival);
```

Группа элементов добавляется оператором `insert()`, требующим двух итераторов:

```
iset.insert(vec.begin(), vec.end());
```

Прохождение через множество, как вы и могли ожидать:

```
set<int>::iterator it = iset.begin();  
for (; it != iset.end(); ++it)  
    cout << *it << ' '  
  
cout << endl;
```

Общие алгоритмы содержат несколько алгоритмов множеств: `set_intersection()`, `set_union()`, `set_difference()`, и `set_symmetric_difference()`.

### 3.9 Как использовать итераторные вставки

В нашей реализации `filter()`, выше в Разделе 3.6, мы назначили каждому элементу исходного контейнера, который подбирается, объявленный тест в целевом контейнере:

```
while ((first = find_if(first, last, bind2nd(pred, val))) != last)  
    *at++ = *first++;
```

Это потребовало, чтобы целевой контейнер был достаточно велик для содержания каждого заданного значения. У `filter()` нет способа узнать, будет ли каждый инкремент (увеличение) в `continue` адресоваться к правильному контейнерному слоту (сегменту). Задача программиста обеспечить, чтобы указанный целевой контейнер в этом случае оказался достаточно велик. В нашей тестовой программе в Разделе 3.6 мы обеспечили это определением целевого контейнера такого же размера, как и исходный:

```
int ia[elem_size] = { 12, 8, 43, 0, 6, 21, 3, 7 };  
vector<int> ivec(ia, ia+elem_size);  
  
int ia2[elem_size];  
  
vector<int> ivec2(elem_size);
```

Проблема этого решения в том, что в большинстве случаев целевой контейнер оказывается слишком велик. Альтернативный подход – определить пустой контейнер и расширять его по мере надобности через вставку элементов. К сожалению `filter()`, в том виде, как есть, разработана для назначения существующего контейнерного слота. Если мы переделаем `filter()` для вставок, то, что случится с нашими действующими программами, использующими вариант присваивания в `filter()`? Более того, какую программную вставку мы должны осуществить?

Общие алгоритмы, которые копируют элементы, такие как `copy()`, `copy_backwards()`, `remove_copy()`, `replace_copy()`, `unique_copy()` и т.д., сходны в разработке с `filter()`. Каждый передает итератор, маркирующий позицию в контейнере, с которой начинается копирование. Каждой копии элемента присваивается значение, и итератор инкрементируется. Каждая копия требует, чтобы мы гарантировали, что целевой контейнер имеет соответствующий размер для содержания множества присвоенных элементов. Эти алгоритмы не дают нам возможности для переделок.

Означает ли это, что мы должны всегда передавать контейнер фиксированного размера для этих алгоритмов? Это едва ли в духе STL. Вернее, стандартная библиотека поддерживает три адаптера вставок. Эти адаптеры позволяют нам пренебречь оператором присваивания контейнера.

- `back_inserter()` действует так, что контейнерный оператор `push_back()` будет вызван вместо оператора присваивания. Это предпочтительная вставка для вектора. Аргумент `back_inserter - контейнер`:

```
vector<int> result_vec;
unique_copy(ivec.begin(), ivec.end(), back_inserter(result_vec));
```

- `inserter()` действует так, что будет вызван контейнерный оператор `insert()`. `Inserter()` берет два аргумента: контейнер и итератор в контейнер, указывающий на позицию, с которой начнется вставка. Для вектора мы должны написать следующее:

```
vector<string> svec_res;
unique_copy(svec.begin(), svec.end(), inserter(svec_res, svec_res.end()));
```

- `front_inserter()` действует так, что будет вызван контейнерный оператор `push_front()`. Этот (вставочник) `inserter` может быть использован только со списком и дековыми (с двухсторонней очередью) контейнерами:

```
list<int> ilist_clone;

copy(ilist.begin(), ilist.end(), front_inserter(ilist_clone));
```

Для использования этого адаптера мы должны включить файл заголовка `iterator`:

```
#include <iterator>
```

Эти адаптеры, однако, не могут использоваться со встроенным массивом. Встроенный массив не поддерживает вставок элементов. Вот переделка программы Раздела 3.6, использующей `back_inserter` для вектора в `filter()`:

```
int main(){ const int elem_size = 8;

int ia[elem_size] = { 12, 8, 43, 0, 6, 21, 3, 7 };
vector<int> ivec(ia, ia+elem_size);

// встроенный массив не поддерживает вставок ...
int ia2[elem_size];
vector<int> ivec2;

cout << "filtering integer array for values less than 8\n";
```

```

filter(ia, ia+elem_size, ia2,elem_size, less<int>());

cout << "filtering integer vector for values greater than 8\n";
filter(ivec.begin(),ivec.end(),back_inserter(ivec2),elem_size,greater<int>());}

```

`filter()` присваивает каждый элемент по очереди целевому вектору, в нашем случае `ivec2`. В этом примере мы не инициализируем `ivec2` размером элемента, так что результатом присваивания будет сбой при выполнении. Осуществляя передачу вектору `ivec2` через адаптер вставок, мы возвращаем присваивание элементов во вставку. Поскольку вставка в вектор результативна только при возврате, мы выбираем `back_inserter`.

### 3.10 Использование итераторов `istream`

Вообразите, что мы перед нами поставлена задача, прочитать последовательность строковых элементов из стандартного входа; затем их необходимо сохранить в векторе, а затем отправить слова на стандартный вывод. Типичное решение выглядит так:

```

#include <istream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

int main(){
    string word;
    vector<string> text;

    // ok: давайте читать каждое слово по очереди до завершения
    while (cin >> word) text.push_back(word);

    // ok: рассортируем их
    sort(text.begin(), text.end());

    // ok: отправляем их назад

    for (int ix = 0; ix < text.size(); ++ix)    cout << text[ix] << ' ';}

```

Стандартная библиотека определяет оба класса итераторов `istream` ввода и вывода, называемые `istream_iterator` и `ostream_iterator`, которые поддерживают стенографический метод чтения и записи элементов одного типа. Для использования класса итераторов мы должны включить файл заголовка `iterator`:

```

#include <iterator>

```

Например, давайте посмотрим, как мы можем использовать класс итераторов входного потока (`istream_iterator`) для чтения нашей строковой последовательности из стандартного ввода. Как и для всех итераторов, мы нуждаемся в паре `first, last` для `istream_iterators`, чтобы отметить границы элементов.

Определение

```
istream_iterator<string> is(cin);
```

дает нам `first` итератор. Он определяет `is`, как `istream_iterator` границу для стандартного ввода, который прочитывает элементы типа строка. Нам также нужен итератор `last`, который представляет 1 после последнего элемента для чтения. Для стандартного ввода, end-of-file (конец-файла, EOF) представляет 1 после последнего элемента для чтения. Как мы обозначим это? Определение `istream_iterator` без объекта `istream`, такого, как `eof`,

```
istream_iterator<string> eof;
```

представляет end-of-file. Как мы реально можем использовать эту пару? В следующем примере мы передадим их общему алгоритму `copy()` вместе с вектором для хранения строчных элементов. Поскольку мы не знаем размер будущего вектора, мы адаптируем его с помощью `back_inserter`:

```
copy(is, eof, back_inserter(text));
```

Теперь нам нужен `ostream_iterator` для отметки места записи каждого строкового элемента. Мы остановимся, когда не останется элементов для вывода. Следующее определение `os` будет `ostream_iterator`, связанный со стандартным выводом, который содержит элементы типа строка.

```
ostream_iterator<string> os(cout, " ");
```

Второй аргумент – это либо строка символов C-стиля, либо буквенная строка, которая определяет разделитель между элементами для вывода. По умолчанию, элементы записаны без разделителей между ними. В этом примере я выбираю для вывода каждый элемент, отделенный пробелом. Вот, как мы должны это осуществлять:

```
copy(text.begin(), text.end(), os);
```

`copy()` записывает каждый элемент сохраненный в `text` в `ostream` обозначенный `os`. Каждый элемент отделен пробелом. Вот полная программа:

```
#include <iostream>
#include <iterator>
#include <algorithm>
#include <vector>
#include <string>

using namespace std;

int main(){

    istream_iterator<string> is(cin);

    istream_iterator<string> eof;

    vector<string> text;
    copy(is, eof, back_inserter(text));

    sort(text.begin(), text.end());
```

```

ostream_iterator<string> os(cout, " ");

copy(text.begin(), text.end(), os);}

```

Часто вместо чтения из стандартного ввода или записи в стандартный вывод, мы читаем из файла и записываем в файл. Как мы можем это сделать? Мы просто связываем `istream_iterator` с объектом класса `ifstream` и `ostream_iterator` с объектом класса `ofstream`:

```

#include <iostream>
#include <fstream>
#include <iterator>
#include <algorithm>
#include <vector>
#include <string>
using namespace std;
int main(){
    ifstream in_file("as_you_like_it.txt");
    ofstream out_file("as_you_like_it_sorted.txt");
    if (! in_file || ! out_file) {
        cerr << "!!не могу открыть необходимые файлы.\n";
        return -1;}

    istream_iterator<string> is(in_file);
    istream_iterator<string> eof;

    vector<string> text;
    copy(is, eof, back_inserter(text));

    sort(text.begin(), text.end());

    ostream_iterator<string> os(out_file, " ");
    copy(text.begin(), text.end(), os); }

```

## Упражнение 3.1

Напишите программу для чтения текстового файла. Запомните каждое слово во множестве. Значение ключа множества – подсчитанное число появления слова в тексте. Определите множество слов-исключений, содержащих такие слова, как и, или, но, а. Прежде, чем слово попадает во множество, удостоверьтесь, что оно не присутствует во множестве слов-исключений. Выведите на дисплей список слов и их ассоциированный подсчет, когда чтение текста будет завершено. Как расширение, перед отображением текста, позвольте пользователю выяснить присутствие слова в тексте.

## Упражнение 3.2

Прочитайте текстовый файл, он может быть тем же, что и в [Упражнение 3.1](#), запомните его в векторе. Рассортируйте вектор по длине строк. Определите объект функции для передачи в `sort()`; он должен принимать две строки и возвращать `true`, если первая строка короче второй. Выведите на дисплей рассортированный вектор.

### Упражнение 3.3

Определите отображение, для которого индекс – фамилия, а ключ – вектор имен детей. Заполните отображение шестью последними вводами. Проверьте его на поддержку пользовательских запросов, основанных на фамилии, и выведите на дисплей все вводы отображения.

### Упражнение 3.4

Напишите программу для чтения последовательности целых чисел из стандартного ввода, используя `istream_iterator`. Напишите нечетное число в один файл, используя `ostream_iterator`. Каждое значение должно быть отделено пробелом. Напишите четные числа во второй файл, также используя `ostream_iterator`. Каждое из этих значений должно быть расположено на отдельной линии.

## Часть 4. Объектно-базируемое программирование

Хотя мы еще только собираемся создать свой собственный класс, используем мы классы интенсивно, начиная с *Части 1*: строковый и векторный классы, классы, поддерживающие ввод и вывод, и т.д. В этой части мы разработаем и реализуем наш собственный класс.

Что мы знаем о классах из опыта их использования? Прежде, чем мы можем использовать класс, мы должны сделать его известным программе, поскольку класс не встроен в язык. Обычно мы делаем это, включая файл заголовка:

```
#include <string>

string pooh[4] = { "winnie", "robin", "eeyore", "piglet" };
```

Имена классов обслуживаются, как имена типов - на тот же манер, что встроенный тип имен, как `int` и `double`. Зачастую существует множество путей к инициализации объекта класса:

```
#include <vector>

string dummy("dummy");

vector<string> svec1(4);

vector<string> svec2(4, dummy);

vector<string> svec3(pooh, pooh+4);
```

Каждый класс поддерживает множество операций, которые мы можем применить к объектам класса. Эти операции обычно состоят из именованных функций, таких как `size()` и `empty()`, и перезагружаемых образцов предопределенных операторов, как неэквивалентность и присвоение:

```
if (svec2 != svec3 && ! svec3.empty())    svec2 = svec3;

if (svec2.size() == 4) // все в порядке ...
```

Что мы вообще не знаем – это, как реализуется класс. Вычисляет ли строковый класс свой размер при каждом запросе, или он хранит размер внутри каждого объекта класса? Элементы вектора, запоминаются ли они внутри объекта вектора, или где-то еще, а в объекте вектора к ним производится адресация через указатель?

В основном, класс состоит из двух частей: общего (`public`) множества операций и частной (`private`) реализации. Эти операции и операторы называются *функциями-членами* класса и представляют *общий интерфейс* класса. Как пользователи класса, мы можем иметь доступ только к общему интерфейсу (`public interface`). Фактически это то, как мы используем строковый класс, класс векторов и т.д. Например, все, что мы знаем о `size()` строковой функции-члене – это прототип: он имеет список параметров `void` и возвращает целое значение.

Частная реализация класса состоит из определений функций-членов и каких-либо данных, ассоциированных с классом. Например, если объект строкового класса вычисляет длину своей строки с каждым вызовом `size()`, не требуется ассоциированных данных, а определение `size()` похоже на вызов цикла `for` проходящего по длине всей строки. Если объект строкового класса запоминает длину своей строки, частные *данные-члены*

(*private data member*) должны быть определены внутри каждого объекта класса. Это определение `size()` возвращает значение соответствующего члена. Каждый раз, когда изменится длина строки, данные-члены должны быть обновлены. Подобный род деталей реализации обычно не является заботой пользователя класса. Как пользователи, мы просто программируем общий интерфейс. Таким образом, до тех пор, пока интерфейс не меняется, наш код использует этот интерфейс, также не требующий изменений, даже если произойдет внутренняя модификация.

В этой части мы обратимся от простого использования классов к поддержке классов для себя и использования другими.

## 4.1 Как реализуется класс

Хорошо, с чего начнем? В общем, мы начнем с абстракций. Рассмотрим стек. Стек – фундаментальная абстракция в компьютерной науке. Он позволяет вложения и восстановление значений в последовательности «последним вошел, первым вышел». Мы вкладываем значения, *вталкивая* (*pushing*) новое значение в стек, и извлекаем их, *выталкивая* (*poping*) последнее значение, пополнив стек. Другие операции, которые пользователям часто требуются – это запросы *полон* ли стек или *пуст*, и определение *размера* стека. Стек может также поддерживать считывание (*peeking*) последнего положенного в стек значения.

В описании стека подчеркнуты слова, представляющие операции, которые пользователям понадобятся для использования в объектах нашего класса стека.

Какой тип элементов мы будем хранить? В основном стек хранит все типы. Мы получим это, определяя стек, как шаблон класса. Поскольку шаблоны класса это предмет рассмотрения в [Части 6](#), а мы добрались только до [Части 4](#), мы не будем определять шаблон класс стека для хранения объектов строкового класса.

Объявление класса начинается с ключевого слова `class`, сопровождаемого определенным пользователем именем класса:

```
class Stack;
```

Это выражение служит опережающим объявлением для стекового класса. Оно вводит имя класса в компилятор, но не содержит деталей поддерживаемых операций или данных-членов, которые содержит. Опережающее объявление позволяет нам определить указатели класса и использовать класс, как тип, определенный для объявлений класса:

```
// ok: здесь нужно дать опережающее объявление класса
Stack *pt = 0;
void process(const Stack&);
```

Определение класса необходимо сделать до того, как мы сможем определить конкретный объект стекового класса, или обратиться к какому-нибудь члену класса. Скелет определения класса выглядит, примерно, так:

```
class Stack {public:
// ... общий интерфейс
private:
// ... частная реализация};
```

Определение класса состоит из объявления класса, сопровождающееся телом класса, заключенным в

фигурные скобки, и завершающееся точкой с запятой. Ключевые слова `public` и `private` внутри тела класса управляют доступом к членам, объявленным внутри каждой секции. Общие члены могут достигаться из любого места программы. Частные члены доступны только для функций-членов и *друзей* (*friends*) класса – позднее я поясню, что это за друзья (или в крайности, что за друзья есть в языке C++). Вот начало определения нашего класса Стек:

```
class Stack {
public:
    // каждая операция возвращает true, если может быть выполнена
    // pop и peek помещают строковое значение внутри elem
    bool push(const string&);
    bool pop(string &elem);
    bool peek(string &elem);

    bool empty();
    bool full();

    // определение size() помещено внутри class
    // другие члены просто объявлены ...
    int size() { return _stack.size(); }
private:
    vector<string> _stack;};
```

Наше определение класса Стек поддерживает шесть операций, которые мы задали в начале этой секции. Элементы сами по себе сохраняются в векторе строк, названном нами `_stack`. (Мое правило написания кодов – снабжать данные-члены подчеркиванием). Вот как мы можем определить и использовать объект класса Стек:

```
void fill_stack(Stack &stack, istream &is = cin){
    string str;
    while (is >> str && ! stack.full())    stack.push(str);

    cout << "Read in " << stack.size() << " elements\n";}
```

Все функции-члены должны быть объявлены внутри определения класса. Дополнительно функция-член может также быть определена внутри определения класса. Если определение сделано внутри тела класса, функция-член автоматически обрабатывается, как если бы была `inline`. `size()`, например, это `inline` член Стекa.

Для определения функции-члена вне определения класса мы используем специальный синтаксис объявления. Его цель – идентифицировать функцию, как члена частного класса. Если функция планируется, как `inline`, должно присутствовать ключевое слово `inline`:

```
inline bool Stack::empty(){
    return _stack.empty();}

bool Stack::pop(string &elem){
    if (empty())    return false;

    elem = _stack.back();
    _stack.pop_back();
    return true;}
```

Синтаксис

```
Stack::empty()
```

говорит компилятору (и читателям), что мы ссылаемся на члена `empty()` класса `Стек` – в противоположность, скажем, аналогичному в классах вектора или строчном. Имя класса, сопровождаемое двойным двоеточием (`Stack::`) называется *оператором границ класса (class scope operator)*.

Безразлично в обработке функции `inline`, определена она внутри или вне определения класса. Как с определением функции `inline` не членом, так и `inline` функция-член должны быть помещены в файл заголовка. Определение класса и `inline` функций-членов обычно помещают в файл заголовка с именем класса. Например, определение класса `Стек` и определение `empty()` должны быть помещены внутри файла заголовка, названного `Stack.h`. Это то, что пользователь включает, когда собирается использовать наш класс.

Функции-члены не `inline` (`non-inline`) определяются внутри текста файла программы, обычно называемого по имени класса и сопровождаемого суффиксами: `.C`, `.cc`, `.cpp` и `.cxx` (`x` представляет заготовку +). Microsoft Visual C++, например, использует `.cpp` по умолчанию. Соглашение в Disney Feature Animation – использует `.C`. Соглашение в Dreamworks Animation – использует `.cc`.

Вот остаток определения функций-членов `Стек`. `full()` сравнивает текущий размер нижележащего вектора с `max_size()` – максимально возможным размером вектора. `push()` вставляет элемент при условии, что `_stack` не полон.

```
inline bool Stack::full(){ return _stack.size() == _stack.max_size(); }

bool Stack::peek(string &elem){

    if (empty())return false;

    elem = _stack.back();
    return true;}

bool Stack::push(const string &elem){

    if (full())return false;

    _stack.push_back(elem);
    return true;}
```

Хотя мы поддержали определение для полного множества пользовательских операций, это еще не завершает определение класса `Стек`. В следующем разделе мы рассмотрим, как поддерживать специальные функции инициализации и деинициализации, называемые *конструктором (constructor)* и *деструктором (destructor)* класса.

## Упражнение 4.1

Создайте `Stack.h` и `Stack.suffix`, где `suffix` зависит от соглашения, которое поддерживается вашим компилятором или следует из проекта. Напишите функцию `main()` для упражнений в полном общем интерфейсе (`full public interface`), откомпилируйте ее и выполните. Оба, текст программы и `main()` должны включать `Stack.h`:

```
#include "Stack.h"
```

## Упражнение 4.2

Расширьте класс `Стек` для поддержки операций `find()` и `count()`. `find()` возвращает `true` или

`false` в зависимости от того, найдено ли значение. `count()` возвращает число включений строки. Переделайте `main()` из [Упражнения 4.1](#) для вызова обеих функций.

## 4.2 Что такое конструктор и деструктор класса?

Каждая из наших числовых последовательностей хороший кандидат на звание класса. Объект класса числовой последовательности представляет группу элементов внутри ассоциированной последовательности. По определению начальная позиция 1. Например,

```
Fibonacci fib1(7, 3);
```

определяет объект класса Фибоначчи из 7 элементов, начинающихся с позиции 3, а

```
Pell pel(10);
```

определяет объект класса Pell из 10 элементов, начинающихся с позиции по умолчанию – 1. Окончательно,

```
Fibonacci fib2(fib1);
```

инициализирует `fib2` копируемую из `fib1`.

Каждый класс должен хранить след и его длины, определяющей, как много элементов серии представлено, и начальной позиции. 0 или отрицательная начальная позиция, или длина недопустимы. Мы храним обе величины, длину и начальную позицию, как целые. Сейчас мы определим третий член, `_next`, который хранит путь к следующему элементу для следующей итерации:

```
class Triangular {
public:
    // ...
private:
    int _length;
    // число элементов
    int _beg_pos;
    // начальная позиция группы
    int _next;

    // следующий элемент для итерации};
```

Данные-члены запоминаются внутри каждого объекта класса `Triangular`. Когда я пишу

```
Triangular tri(8, 3);
```

`tri` содержит образец `_length` (инициализированной в 8), `_beg_pos` (инициализированной в 3), и `_next` (инициализированной в 2 поскольку третий элемент индексирован внутри вектора в позиции 2). Заметьте, что нет образца реального вектора, содержащего элементы треугольной последовательности. Почему? Потому, что мы не хотим копировать вектор в каждый объект класса, одного образца достаточно для объектов класса. (В [Разделе 4.5](#) мы рассмотрим, как это поддерживается).

Как инициализируются эти данные-члены? Без волшебства. Компилятор не делает этого для нас. Однако если мы создадим одну или несколько специальных функций инициализаторов, компилятор создаст соответствующий вызов каждый раз, когда определяется объект класса. Эти специальные функции

инициализаторы называются *конструкторы (constructors)*.

Мы идентифицируем конструктор, давая ему то же имя, что и классу. Синтаксические правила таковы – конструктор не должен точно определять ни возвращаемого типа, ни возвращаемого значения. Они должны перезагружаться. Например, вот три возможных конструктора для класса `Triangular`:

```
class Triangular {
    public:// перезагружаемое множество конструкторов

    Triangular(); // конструктор по умолчанию

    Triangular(int len);

    Triangular(int len, int beg_pos);

    // ... };
```

Конструктор вызывается автоматически, основываясь на значениях, включенных в объект класса при определении. Например,

```
Triangular t;
```

показывает, что конструктор по умолчанию принадлежит к `t`. Аналогично,

```
Triangular t2(10, 3);
```

показывает использование двух-аргументного конструктора. Значения в скобках обрабатываются, как значения, передаваемые в конструктор. Подобным же образом

```
Triangular t3 = 8;
```

показывает использование одно-аргументного конструктора целого.

На удивление, следующее НЕ определяет объект класса `Triangular`:

```
Triangular t5(); // не то, чем кажется :-)
```

Вернее, это определяет функцию `t5` с пустым списком параметров, возвращающую объект `Triangular`. Очевидно, это непонятная интерпретация. Почему оно интерпретируется так? Потому, что C++ по необходимости должен быть совместим языком C, а в C скобки, следующие за `t5`, в этом случае, идентифицируют ее, как функцию. Правильное объявление `t5` будет таким же, как `t`, показанное ранее:

```
Triangular t5; // ok
```

Простейшим конструктором является *конструктор по умолчанию (default constructor)*. Конструктор по умолчанию не требует аргументов. Это означает одно из двух. Либо он не имеет аргументов:

```
Triangular::Triangular() {

    // default

    constructor_length = 1; _beg_pos = 1; _next = 0;}
```

либо, в общем виде, он поддерживает значения по умолчанию для каждого параметра:

```

class Triangular {
public:// так же конструктор по умолчанию
    Triangular(int len = 1, int bp = 1);
    // ...};
Triangular::Triangular(int len, int bp){
    // _length и _beg_pos оба должны быть хотя бы 1
    // лучше не полагаться на то, что пользователь всегда будет
    // правильно себя вести
    _length = len > 0 ? len : 1;
    _beg_pos = bp > 0 ? bp : 1;
    _next = _beg_pos-1;}

```

Поскольку мы поддерживаем значения по умолчанию для обоих целых параметров, единственный образец конструктора по умолчанию поддерживает три оригинальных конструктора:

```

Triangular tri1; // Triangular::Triangular(1, 1);

Triangular tri2(12); // Triangular::Triangular(12, 1);

Triangular tri3(8, 3); // Triangular::Triangular(8, 3);

```

## Список инициализации членов

Второй синтаксис инициализации внутри определения конструктора использует *список инициализации членов* (*member initialization list*):

```

Triangular::Triangular(const Triangular &rhs): _length
(rhs._length), _beg_pos(rhs._beg_pos), _next(rhs._beg_pos-1) {} // да, пусто!

```

Список инициализации членов отделен от списка параметров двоеточием. Это разделенный запятыми список, в котором значения, присваиваемые членам, размещены в скобках, следующих за именами членов, что напоминает вызов конструктора.

В этом примере два альтернативных определения конструктора эквивалентны. Здесь нет преимущественного указания на выбор одной или другой формы.

Список инициализации членов используется непосредственно для передачи аргументов в конструкторы объекта члена класса. Например, давайте переопределим класс `Triangular` для поддержки членов строкового класса:

```

class Triangular {
public:
    // ...
private:
    string _name;
    int _next, _length, _beg_pos;};

```

Для передачи значения в строковый конструктор, которым инициализируется `_name`, мы используем список инициализации членов. Например,

```

Triangular::Triangular(int len, int bp): _name("Triangular"){
    _length = len > 0 ? len : 1;
    _beg_pos = bp > 0 ? bp : 1;
    _next = _beg_pos-1;}

```

Дополнением механизма конструктора служит деструктор. Деструктор – определяемый пользователем класс функций-членов, который, если присутствует, используется автоматически для объекта класса до окончания его действия. Первая польза от деструктора – освобождение ресурсов, занятых в конструкторе или при работе объекта.

Деструктору дается имя класса, предваряемое тильдой (~). Оно не должно определять возвращаемое значение и должно объявлять пустой список параметров. Поскольку список параметров пуст, деструктор класса не может быть перезагружен.

Рассмотрим следующий класс `Matrix`. Внутри конструктора, выражение `new` используется для локализации массива чисел двойной точности (`doubles`) из кучи. Деструктор используется для освобождения этой памяти:

```
class Matrix {
public:
    Matrix(int row, int col): _row(row), _col(col){
        // конструктор локализует ресурсы
        // note: не показана проверка ошибок
        _pmat = new double[row * col];}

    ~Matrix(){
        // деструктор освобождает эти ресурсы
        delete [] _pmat;}

    // ...

private:
    int _row, _col;
    double *_pmat;};
```

В результате мы имеем автоматическое обслуживание всей кучи внутри класса `Matrix` через определение его конструктора и деструктора. Например, рассмотрим следующий блок выражений:

```
{ Matrix mat(4, 4);
// конструктор используется здесь

// ...
// здесь используется деструктор}
```

Компилятор использует конструктор класса `Matrix` неявно следующий за определением `mat`. Внутренне, `_pmat` инициализируется адресами массива из 16 `doubles` локализованных в свободном от программы резерве. Только перед закрытием фигурной скобки блока выражения деструктор `Matrix` неявно используется компилятором. Внутренне массив из 16 `doubles`, адресуемый `_pmat`, освобождается через выражение `delete`. Пользователи класса `Matrix` не нуждаются в знании каких-либо деталей обслуживания памяти. Это соглашение неточно подражает конструкции класса контейнера стандартной библиотеки.

Нет необходимости определять деструктор. В нашем классе `Triangular`, например, три данных-члена сохраняются по значению. Они возникают, когда объект класса определяется, и исчезают автоматически, когда деятельность объекта класса завершается. Нет реальной работы для деструктора класса `Triangular`. Мы не обязаны поддерживать деструктор. Самое трудное – понять, когда это нужно, а когда нет.

## Инициализация в стиле членства

По определению, когда мы инициализируем объект одного класса другим, как

```
Triangular tri1(8);Triangular tri2
= tri1;
```

члены-данные класса копируются поочередно. В нашем примере `_length`, `_beg_pos` и `_next` копируются по очереди из `tri1` в `tri2`. Это называется *инициализацией по умолчанию в стиле членства (default memberwise initialization)*.

В случае класса `Triangular` инициализация в стиле членства корректно копирует данные-члены класса, и нам не нужно что-то специально предпринимать. В случае класса `Matrix`, приведенного ранее, выполнение инициализации по умолчанию в стиле членства выполняется не адекватно. Например, рассмотрим следующее:

```
{ Matrix mat(4, 4); // здесь используется конструктор
  { Matrix mat2 = mat;
    // используется копирование default memberwise
    // ... здесь используем mat2
    // здесь используется деструктор для mat2  }
  // ... здесь используем mat
  // здесь используем деструктор для mat  }
```

Инициализация по умолчанию `_pmat` членов для `mat2` через `mat`,

```
mat2._pmat = mat._pmat;
```

имеет следствием два образца `_pmat` в адресации к тому же массиву в общей памяти. Когда деструктор `Matrix` используется для `mat2`, массив исчезает. К сожалению, `_pmat` член `mat` продолжает адресоваться и манипулировать с исчезнувшим массивом. Это серьезная программная ошибка.

Как можно поправить ее? В данном случае мы должны подменить исполнение `default memberwise`. Мы добьемся этого, обеспечив конструктор класса `Matrix` явным образцом копирования. (Под «мы» я подразумеваю разработчика класса `Matrix`. Пользователь класса `Matrix` только может надеяться, что мы сделали все правильно).

Если разработчик класса обеспечит конструктор класса явным образцом копирования, этот образец используется вместо инициализации в стиле членства по определению. Исходный код для пользователя не понадобится менять, хотя перекомпиляция потребуется.

На что наш конструктор копирования будет похож? Его единственным аргументом будет ссылка `const` на объект класса `Matrix`:

```
Matrix::Matrix(const Matrix &rhs){ // что здесь будет?}
```

Как это может быть реализовано? Давайте создадим отдельную копию массива так, чтобы деструктор одного объекта класса не пересекался с выполнением другого:

```
Matrix::Matrix(const Matrix &rhs): _row(rhs._row), _col(rhs._col){
    // создаем "deep copy" массива, адресованного rhs._pmat
```

```

int elem_cnt = _row * _col;

_pmat = new double[elem_cnt];

for (int ix = 0; ix < elem_cnt; ++ix) _pmat[ix] = rhs._pmat[ix];}

```

Когда мы разрабатываем класс, мы должны спросить себя, будет ли стиль членства по умолчанию работать адекватно для класса. Если да, нам не нужно обеспечивать его специальным конструктором копирования. Если нет, мы *должны* определить специальный образец, а внутри реализации правильную семантику инициализации.

В нашем классе требуется конструктор копирования, и требуется оператор присваивания копирования (смотрите Раздел 4.8). Для более детального ознакомления с классом конструкторов и деструкторов смотрите [LIPMAN98], Часть 14 и [LIPMAN96a] Часть 2 и 5.

### 4.3 Что такое mutable и const?

Рассмотрим следующую маленькую функцию:

```

int sum(const Triangular &trian){
    int beg_pos = trian.beg_pos();
    int length = trian.length();
    int sum = 0;
    for (int ix = 0; ix < length; ++ix)
        sum += trian.elem(beg_pos+ix); return sum;}

```

`trian` это ссылка на `const` параметр. Компилятор, однако, должен гарантировать, что `trian` не модифицируется внутри `sum()`. Потенциально, `trian` модифицируется внутри любой функции-члена, которая вызывается. Чтобы быть уверенным, что `trian` не модифицируется, компилятор должен быть уверен, что `beg_pos()`, `length()` и `elem()` не меняют объект класса, который вызывает их. Как компилятор может узнать это? Разработчик класса должен сказать это компилятору, маркируя как `const`, каждую функцию-член, что не приводит к модификации объекта класса:

```

class Triangular {
public:// const члены-функции

    int length() const { return _length; }

    int beg_pos() const { return _beg_pos; }

    int elem(int pos) const;

    // не-const члены-функции
    bool next(int &val);
    void next_reset() { _next = _beg_pos - 1; }

    // ...
private:
    int _length; // число элементов
    int _beg_pos; // начальная позиция группы
    int _next; // следующий элемент для итерации
    // статические data members описаны в Разделе 4.5

    static vector<int> _elems;};

```

Модификатор `const` следует за списком параметров функции. `const` функция-член, определенная вне тела класса, должна определяться модификатором `const` и в ее объявлении, и в определении. Например,

```
int Triangular::elem(int pos) const { return _elems[pos-1]; }
```

Хотя компилятор не анализирует каждую функцию для определения, `const` она или `non-const` функция-член, он проверяет каждую `const` функцию-член, чтобы быть уверенным, что она не модифицирует объект класса. Например, если мы объявили следующий образец `next()`, как `const` функцию-член, это объявление будет отмечено, как ошибка, поскольку он, несомненно, модифицирует объект класса при вызове.

```
bool Triangular::next(int &value) const {
    if (_next < _beg_pos + _length - 1) {
        // error: модификация _next
        value = _elems[_next++];
        return true; }

    return false;}
```

В следующем классе, функция-член `val()` не модифицирует непосредственно `_val` данные-член, но она возвращает `non-const` ссылку на `_val`. Может объявить еще `val()`, как `const`?

```
class val_class {

public:

    val_class(const BigClass &v): _val(v){}

    // хорошо ли это?
    BigClass& val() const { return _val;}

private:

    BigClass _val;};
```

Нет. Возвращение `non-const` ссылки на `_val` имеет следствием открывания его для модификации в основной программе. Поскольку функция-член может быть перезагружена на основе ее констант, одним из решений проблемы может стать создание двух определений: `const` и `non-const` версий, как показано ниже.

```
class val_class {
public:
    const BigClass& val() const { return _val; }
    BigClass& val(){ return _val; }// ... };
```

Для `non-const` объекта класса, вызывается `non-const` образец `val()`. Для `const` объекта класса, вызывается `const` образец. Например,

```
void example(const BigClass *pbc, BigClass &rbc) {
    pbc->val(); // вызывается const вариант
    rbc.val(); // вызывается non-const вариант
    // ... }
```

Когда мы разрабатываем класс, очень важно идентифицировать `const` функции-члены. Если вы забудете, каждая `const` ссылка параметра класса будет не способна вызвать `non-const` часть интерфейса класса.

Пользователи могут обидеться. Подгонка `const` под класс многообещающая, особенно, если имеет место расширение вызова одной функции-члена другой.

## Изменчивые (Mutable) данные-члены

Вот альтернативная реализация `sum()` с использованием `next()` и `next_reset()` функций-членов для прохождения через элементы `trian`.

```
int sum(const Triangular &trian){
    if (! trian.length())return 0;

    int val, sum = 0;
    trian.next_reset();
    while (trian.next(val))
        sum += val;

    return sum;}

```

Будет ли код компилироваться? Нет, абсолютно. `trian` - это `const` объект класса. `next_reset()` и `next()` - не `const` функции-члены, поскольку обе функции модифицируют `_next`. Их вызов из `trian`, следовательно, вызовет ошибку.

Если мы собираемся использовать эту реализацию `sum()`, обе `next()` и `next_reset()` должны стать `const` функциями-членами. Мы можем сделать это, используя изящный маневр.

`_length` и `_beg_pos` поддерживают атрибуты абстрактной числовой серии. Если мы изменим длину или начальную позицию `trian`, в результате мы изменим их идентичность. Они больше не эквивалентны тому, чем были до изменения. `_next`, однако, поддерживает нашу реализацию механизмом итераторов. Она не поддерживает сама по себе абстрактную числовую последовательность. Изменение значения `_next` семантически не меняет объекта класса и, одно заявление, не нарушает свойства `const` объекта. (Я сказал, мы сделаем изящный маневр). Ключевое слово `mutable` предоставит нам возможность осуществить это заявление. Идентифицируя `_next`, как изменяющийся (`mutable`), мы говорим, что изменения в нем не изменяют `const`-свойство объекта класса.

```
class Triangular {
public:
    bool next(int &val) const;
    void next_reset() const { _next = _beg_pos - 1; }
    // ...

private:
    mutable int _next,int _beg_pos;
    int _length; };

```

`next()` и `next_reset()` могут теперь модифицировать `_next` и все еще быть объявлены, как `const` функции-члены, осуществляя нашу альтернативную реализацию `sum()`. Вот небольшая программа, которая выполняет `sum()` на трех объектах класса `Triangular`:

```
int main(){
    Triangular tri(4);
    cout << tri << " -- sum of elements: "
         << sum(tri) << endl;
}

```

```
Triangular tri2(4, 3);
cout << tri2 << " -- sum of elements: "
    << sum(tri2) << endl;
```

```
Triangular tri3(4, 8);
cout << tri3 << " -- sum of elements: "
    << sum(tri3) << endl; }
```

После компиляции, при выполнении, она выводит следующее:

```
(1 , 4) 1 3 6 10 -- sum of elements: 20
(3 , 4) 6 10 15 21 -- sum of elements: 52
(8 , 4) 36 45 55 66 -- sum of elements: 202
```

## 4.4 Что такое this указатель?

Мы должны реализовать функцию-член `copy()`, которая инициализирует один объект класса `Triangular` другим. Например, в данной

```
Triangular tr1(8);

Triangular tr2(8, 9);
```

вызов

```
tr1.copy(tr2);
```

присваивает `tr1` длину и начальную позицию `tr2`. Вдобавок `copy()` должна вернуть объект класса, который является целевым для копирования. В нашем примере `tr1` одновременно и цель, и объект, который должен быть возвращен. Как мы можем осуществить это? Например, вот реализация `copy()`:

```
Triangular& Triangular::copy(const Triangular &rhs){
    _length = rhs._length;
    _beg_pos = rhs._beg_pos;
    _next = rhs._beg_pos-1;

    return ??? что именно ???};
```

`rhs` ограничивает `tr2` в нашем примере. В присвоении

```
_length = rhs._length;
```

`_length` относится к члену, ассоциированному с `tr1`. Нам нужен путь для ссылки на объект класса `tr1` в целом. Указатель `this` осуществляет для нас эту поддержку.

Внутри функции-члена `this` указатель адресуется к объекту класса, который вызывает функцию-член. В нашем примере `tr1` адресуется `this` указателем. Как это осуществляется? Внутренне, компилятор добавляет `this` указатель, как аргумент к каждой функции-члену класса `copy()`, например, трансформируя следующим образом:

```
// Псевдокод: Внутренняя трансформация функции-члена

Triangular& Triangular::copy(Triangular *this, const Triangular &rhs) {
```

```
this->_length = rhs._length;
this->_beg_pos = rhs._beg_pos;
this->_next = rhs._beg_pos-1;};
```

Эта трансформация требует второй трансформации: каждый вызов `copy()` должен теперь поддерживать два аргумента. Для выполнения этого, наш оригинальный вызов

```
tr1.copy(tr2);
```

внутренне трансформируется в

```
// внутреннее преобразование кода:
// tr1 становится объектом класса, адресуемым this указателем
copy(&tr1, tr2);
```

Внутри функции-члена `this` указатель поддерживает доступ к объекту класса через вызов конкретной функции-члена. Для возвращения `tr1` из `copy()`, мы просто разыменовываем `this` указатель, как показано:

```
// возвращает объект класса, адресуемый this указателем

return *this;
```

Когда мы копируем один объект класса другим, хорошим правилом было бы вначале устроить проверку, чтобы быть уверенным в том, что два объекта класса не являются одним и тем же. Для этого мы опять используем `this` указатель:

```
Triangular& Triangular::copy(const Triangular &rhs){
    // проверяем, что два объекта не являются тем же самым
    if (this != &rhs){
        _length = rhs._length;
        _beg_pos = rhs._beg_pos;
        _next = rhs._beg_pos-1;}
    return *this;}
```

## 4.5 Статические члены класса

В нашей процедурной реализации [Части 2](#) мы поддерживали один экземпляр контейнера для содержания элементов последовательности Фибоначчи с помощью локального статического (`static`) вектора. Наша реализация класса так же нуждается только в одном экземпляре контейнера для поддержки всех элементов каждой числовой последовательности. Ключевое слово `static` вновь предлагает решение, хотя означает нечто иное, когда используется внутри класса.

Статические данные-члены представлены единственным, общим экземпляром этих членов, который доступен для всех объектов класса. В следующем определении класса, например, мы объявляем `_elems`, как статический член-данные класса `Triangular`:

```
class Triangular {
public:
```

```

// ...
private:
    static vector<int> _elems;};

```

Поскольку содержится только один экземпляр статического члена-данных, мы должны обеспечить подробное определение этого экземпляра внутри файла текста программы. Определение похоже на глобальное определение исключения объекта, так как его имя квалифицируется оператором границ класса:

```

// располагается в файле текста программы, таком как Triangular.cpp

vector<int> Triangular::_elems;

```

При желании можно провести инициализацию:

```

int Triangular::_initial_size = 8;

```

Функции-члены класса могут получить доступ к статическим данным-членам так же, как если бы они были обычными данными-членами:

```

Triangular::Triangular(int len, int beg_pos): _length(len > 0 ? len : 1),
    _beg_pos(beg_pos > 0 ? beg_pos : 1){
    _next = _beg_pos-1;
    int elem_cnt = _beg_pos + _length;

    if (_elems.size() < elem_cnt) gen_elements(elem_cnt);}

```

const static int данные-члены, такие как buf\_size и next, один образец, в котором члены класса могут явно инициализироваться внутри объявления:

```

class intBuffer {
public:
    // ...
private:
    static const int _buf_size = 1024; // ok

    int _buffer[_buf_size]; // ok};

```

## Статические функции-члены

Рассмотрим следующую реализацию is\_elem(). Получая значение, она возвращает true или false в зависимости от принадлежности элемента к последовательности Triangular:

```

bool Triangular::is_elem(int value){
    if (!_elems.size() || _elems[_elems.size()-1] < value)
        gen_elems_to_value(value);

    vector<int>::iterator found_it;
    vector<int>::iterator end_it = _elems.end();

    found_it = find(_elems.begin(), end_it, value);

    return found_it != end_it;}

```

Обычно члены-функции должны вызываться через объект класса. Объект ограничивается `this` указателем функции-члена. Именно через `this` указатель функция-член получает доступ к нестатическим данным-членам, хранящимся в каждом объекте класса.

`is_elem()`, однако, не получает доступ к каким-либо данным-членам. Ее операции не зависят от особых объектов класса, и она будет удобна для вызова ее, как свободной функции. Мы не можем написать:

```
if (is_elem(8)) ...
```

поскольку нет способа у компилятора или читателя узнать, какую `is_elem()` мы хотим вызвать. Использование оператора границ класса устраняет эту неоднозначность:

```
if (Triangular::is_elem(8)) ...
```

Статические функции-члены могут быть вызваны независимо от объекта класса в точности также. Функция-член может быть объявлена, как статическая, только если она не обращается к каким-либо нестатическим членам класса. Мы делаем ее статической, предваряя ее объявление внутри определения класса ключевым словом `static`:

```
class Triangular {
public:
    static bool is_elem(int);
    static void gen_elements(int length);
    static void gen_elems_to_value(int value);
    static void display(int len, int beg_pos, ostream &os = cout); // ...
private:
    static const int _max_elems = 1024; static vector<int> _elems; // ...
};
```

При определении вне тела класса, ключевое слово `static` не повторяется (это справедливо и для статических данных-членов):

```
void Triangular::gen_elems_to_value(int value){
    int ix = _elems.size();
    if (! ix){

        _elems.push_back(1);
        ix = 1; }

    while (_elems[ix-1] < value && ix < _max_elems){
        //
        cout << "elems to value: " << ix*(ix+1)/Artifact << endl;
        _elems.push_back(ix*(ix+1)/2);
        ++ix; }

    if (ix == _max_elems)
        cerr << "Triangular Sequence: oops: value too large "<< value

        << " --exceeds max size of " << _max_elems << endl;}
```

Вот пример вызова независимой `is_elem()`:

```
#include <iostream>
```

```

#include "Triangular.h"

using namespace std;

int main(){

    char ch;

    bool more = true;

    while (more) {
        cout << "Enter value: ";
        int ival;
        cin >> ival;

        bool is_elem = Triangular::is_elem(ival);
        cout << ival << (is_elem ? " is " : " is not ")

            << "an element in the Triangular series.\n"

            << "Another value? (y/n) ";

        cin >> ch;

        if (ch == 'n' || ch == 'N')more = false;}}

```

После компиляции и при выполнении она генерирует следующее (мой ввод выделен жирным шрифтом):

```

Enter value: 1024
1024 is not an element in the Triangular series.
Another value? (y/n) y
Enter value: 0
0 is not an element in the Triangular series.
Another value? (y/n) y
Enter value: 36
36 is an element in the Triangular series.
Another value? (y/n) y
Enter value: 55
55 is an element in the Triangular series.
Another value? (y/n) n

```

Для полноты картины, вот определение `gen_elements()`:

```

void Triangular::gen_elements(int length){
    if (length < 0 || length > _max_elems){
        // выдает сообщение об ошибке и возвращается }

    if (_elems.size() < length){
        int ix = _elems.size() ? _elems.size()+1 : 1;
        for (; ix <= length-1; ++ix)

            _elems.push_back(ix*(ix+1)/2);}}

```

## 4.6 Построение класса итераторов

Для иллюстрации того, как перезагружать множество операторов класса, давайте рассмотрим реализацию класса итераторов. Мы должны поддерживать следующее:

```
Triangular trian(1, 8);
Triangular::iterator it = trian.begin(), end_it = trian.end();

while (it != end_it) { cout << *it << ' '; ++it; }
```

Чтобы это работало, операторы `!=`, `*` и `++`, конечно, должны быть определены для объектов класса итераторов. Как мы сделаем это? Мы определим их, как операторы функций-членов. Оператор функции выглядит, как обычная функция, исключая то, что вместо представления функции именем, мы добавляем ключевое слово `operator`, предшествующее оператору, который мы хотим перезагрузить. Например,

```
class Triangular_iterator{
public:
    // множество _index для index-1, соответственно, не вычитание 1
    // с доступом к каждому элементу ...
    Triangular_iterator(int index) : _index(index-1){}
    bool operator==(const Triangular_iterator&) const;
    bool operator!=(const Triangular_iterator&) const;
    int operator*() const;
    int& operator++(); // префиксный вариант
    int operator++(int); // постфиксный вариант

private:

    void check_integrity() const; int _index;};
```

Класс `Triangular_iterator` поддерживает индексацию в классе `Triangular` статических данных-членов `_elems`, которые содержат элементы последовательности. (Чтобы это работало, класс `Triangular` должен предоставлять специальное разрешение на доступ к функциям-членам класса `Triangular_iterator`. В [Разделе 4.7](#) мы увидим, как предоставить привилегии доступа через механизм `friend` (друг)). Два объекта класса `Triangular_iterator` эквивалентны, если два `_index` члена эквивалентны:

```
inline bool Triangular_iterator::operator==(const Triangular_iterator &rhs) const{
return _index == rhs._index; }
```

Оператор применяется прямо к объекту(ам) класса:

```
if (trian1 == trian2) ...
```

Если мы хотим применить оператор к объекту класса, адресуемому указателем, мы должны вначале разыменовать указатель для получения доступа к объекту класса:

```
if (*ptri1 == *ptri2) ...
```

Для полноты - оператор обычно реализуется в смысле его ассоциированного оператора. Например,

```
inline bool Triangular_iterator::operator!=(const Triangular_iterator
&rhs) const{ return !(*this == rhs); }
```

Правила, управляющие перезагрузкой оператора, следующие:

- Мы не можем вставлять новые операторы. Перезагружены могут быть все predefined операторы, исключая четыре оператора `., .*`, `::` и `?:`.
- Predefined *-арность (arity)* существующих операторов не может быть перезаписана. Каждый бинарный оператор должен быть поддержан двумя операндами, а каждый унарный оператор только одним. Мы не можем определить бинарный оператор эквивалентности, например, для получения ни большего, ни иного количества параметров, чем его predefined два.
- Predefined приоритет существующих операторов не может быть переопределен. Например, оператор деления всегда имеет приоритет перед сложением.
- Оператор функции должен иметь хотя бы один тип класса в качестве аргумента. Это причина, по которой мы не можем переопределить существующие операторы или вставить операторы вне классического типа, как указатели.

Оператор может быть определен либо как оператор функции-члена,

```
inline int Triangular_iterator::operator*() const {
    check_integrity();
    return Triangular::_elems[_index]; }
```

либо как оператор функции-не-члена,

```
inline int operator*(const Triangular_iterator &rhs){
    rhs.check_integrity();
    // заметьте: вариант не-член не имеет
    // привелегий доступа к не общим членам
    return Triangular::_elems[rhs.index()]; }
```

Список параметров оператора-не-члена, всегда определяет на один параметр больше, чем у оператора-члена, его двойника. У оператора-члена, `this` указатель неявно представляет левый операнд.

`check_integrity()` функция-член гарантирует, что `_index` не более, чем `_max_elems`, и что `_elems` содержит необходимые элементы.

```
inline void Triangular_iterator::check_integrity() const{
    // мы рассмотрим throw выражение в Части 7 ...
    if (_index > Triangular::_max_elems) throw iterator_overflow();

    // увеличим вектор, если нужно ...

    if (_index > Triangular::_elems.size())

        Triangular::gen_elements(_index); }
```

Мы должны поддерживать префиксный (`++trian`) и постфиксный (`trian++`) варианты оператора увеличения. Префиксный образец определяется с пустым списком параметров:

```
inline int& Triangular_iterator::operator++(){
    // префиксный вариант
    ++_index;
    check_integrity();
    return Triangular::_elems[_index]; }
```

Обычно постфиксный вариант тоже должен быть определен с пустым списком параметров. Однако каждый перезагружаемый оператор должен иметь уникальный список параметров. Языковое решение – потребовать,

чтобы мы определили постфиксный вариант с одним целым параметром:

```
inline int Triangular_iterator::operator++(int){
    // образец postfix
    check_integrity();
    return Triangular::_elems[_index++]; }
```

Как префиксный, так и постфиксный образец оператора инкремента (и декремента) может быть применен непосредственно к объекту класса:

```
++it; // prefix

it++; // postfix
```

Где единственный аргумент для постфиксного варианта? Компилятор генерирует это для нас автоматически для обязательного вызова постфиксного варианта оператора. К счастью, пользователь не должен беспокоиться об этом.

Наша следующая задача – поддержать `begin()` и `end()` пару членов-функций для класса `Triangular` также удачно, как поддержано определение `iterator`. Поддержка итератора потребовала, чтобы я ввел понятие *вложенных типов* (*nested types*), которое мы вскоре обсудим.

Вначале, однако, необходимые изменения определения класса `Triangular`:

```
#include "Triangular_iterator.h"

class Triangular {
    public:// это защищает пользователя от необходимости знать

        // реальное имя класса итератора ...

    typedef Triangular_iterator iterator;

    Triangular_iterator begin() const {
        return Triangular_iterator(_beg_pos); }

    Triangular_iterator end() const {
        return Triangular_iterator(_beg_pos+_length); }
    // ...

private:
    int _beg_pos;int _length;
    // ... };
```

## Вложенные типы

`typedef` вводит альтернативное имя для типа и имеет следующую основную форму:

```
typedef existing_type new_name;
```

Здесь `existing_type` может быть встроенным, смешанным или типом класса. В нашем примере мы ввели `iterator` как синоним класса `Triangular_iterator` в порядке простоты его использования. Синтаксис для определения объекта типа `iterator`

```
Triangular::iterator it = trian.begin();
```

использует оператор границ класса для указания компилятору найти в определении класса `Triangular` объявления `iterator`. Если мы просто напишем

```
iterator it = trian.begin(); // error
```

компилятор не узнает, что нужно искать в классе `Triangular` объявление `iterator`, так что определение `it` будет вызывать ошибку.

Вкладывая `iterator` внутрь каждого класса, который поддерживает абстракцию итератора, мы можем поддерживать множество определений с одинаковым именем, но за счет более сложного синтаксиса объявления.

```
Fibonacci::iterator fit = fib.begin();
Pell::iterator pit = pel.begin();
vector<int>::iterator vit = _elems.begin();
string::iterator sit = file_name.begin();
```

## 4.7 Сотрудничество иногда требует дружбы

Вариант «не член» `operator*()` непосредственно получает доступ к частному члену `_elems` класса `Triangular` и частной функции-члену `check_integrity()` класса `Triangular_iterator`.

```
inline int operator*(const Triangular_iterator &rhs) {
    rhs.check_integrity();
    return Triangular::_elems[rhs.index()];}
```

Почему это компилируется? Класс может определить функции и классы, как друзей (*friends*). *Друзья* класса получают доступ к частным членам этого класса так же, как функции-члены этого класса. Для компиляции этого варианта `operator*()`, он должен быть другом как `Triangular`, так и `Triangular_iterator` классов:

```
class Triangular {
    friend int operator*(const Triangular_iterator &rhs);
    // ...};

class Triangular_iterator {
    friend int operator*(const Triangular_iterator &rhs);
    // ...};
```

Мы объявляем функцию другом класса, предваряя ее прототип ключевым словом `friend`. Объявление может появиться в любом месте определения класса. Не имеет значения в `private` или в `public` уровне доступа класса. (Для множественного применения перезагружаемых друзей функции класса, мы должны явно выписать каждый вариант).

Вариант `Triangular_iterator operator*()`, так же, как его функция-член `check_integrity()`, оба имеют прямой доступ к частным членам класса `Triangular`. Мы можем объявить оба варианта, как друзей класса `Triangular`:

```
class Triangular {
```

```

friend int Triangular_iterator::operator*();
friend void Triangular_iterator::check_integrity();
// ...};

```

Для успешной компиляции определение класса `Triangular_iterator` должно состояться до объявления его двух функций, как друзей. Иначе компилятору недостаточно информации для определения, это правильный прототип двух функций-членов или одна, или обе - это функции-члены класса.

Альтернативно мы можем гарантировать дружбу класса к самому себе, по очереди добавляя дружбу ко всем функциям-членам этого класса. Например,

```

class Triangular {
    // добавляем дружбу всем
    // членам-функциям Triangular_iterator
    friend class Triangular_iterator;
    // ...};

```

Эта форма дружбы класса не требует, чтобы определение класса было сделано до объявления друзей.

Дружба, однако, не всегда требуется. Например, рассмотрим определение

```

check_integrity():inline void Triangular_iterator::check_integrity() {
    if (_index < Triangular::_max_elems) throw iterator_overflow();

    if (_index > Triangular::_elems.size())Triangular::gen_elements(_index);}

```

Класс `Triangular` поддерживает общий доступ функций к `_max_elems` и частный для функций к возврату текущего размера `_elems`, так что `check_integrity()` не нуждается в предложении дружбы. Например,

```

class Triangular {
public:
    static int elem_size() { return _elems.size(); }
    static int max_elems() { return _max_elems; }// ... };

    // не нужно больше быть другом
    inline void Triangular_iterator::check_integrity() {
        if (_index < Triangular::max_elems()) throw iterator_overflow();
        // увеличиваем вектор, если нужно ...

        if (_index > Triangular::elems_size())Triangular::gen_elements(_index);}

```

Дружба, в общем-то, требуется в таких случаях, как умножение `Point` и `Matrix` – это операторы не члена-функции. А для простого чтения или записи данных-членов, `inline` общий доступ функций обычно предоставляет адекватную альтернативу дружбе.

Вот маленькая программа для упражнений с нашим классом итераторов:

```

int main(){

    Triangular tri(20);

    Triangular::iterator it = tri.begin());

```

```

Triangular::iterator end_it = tri.end();

cout << "Triangular Series of " << tri.length() << " elements\n";
while (it != end_it){

    cout << *it << ' ';
    ++it; }

cout << endl;}

```

После компиляции при выполнении эта программа генерирует следующее:

```

Triangular Series of 20 elements
3 6 10 15 21 28 36 45 55 66 78 91 105 120 136 153 171 190 210 231

```

## 4.8 Реализация оператора присваивания копии

По умолчанию, когда мы присваиваем один объект класса другому, как в

```

Triangular tri1(8), tri2(8, 9);

tri1 = tri2;

```

данные-члены класса копируются поочередно. В нашем примере `_length`, `_beg_pos` и `_next` копируются из `tri2` в `tri1`. Это называется копированием методом членства по умолчанию.

В случае класса `Triangular` семантики копирования методом членства по умолчанию достаточно. Нам нет нужды что-либо добавлять. В случае класса `Matrix` из [Раздела 4.2](#), однако, выполнение методом членства не адекватно. Причина этого обсуждается в подразделе “Инициализация в стиле членства” [Раздела 4.2](#).

Класс `Matrix` требует как конструктора копирования, так и оператора присваивания копии. Вот как мы можем определить `Matrix` оператор присваивания копии:

```

Matrix& Matrix::operator=(const Matrix &rhs){

    if (this != &rhs){
        _row = rhs._row; _col = rhs._col;
        int elem_cnt = _row * _col;

        delete [] _pmat;
        _pmat = new double[elem_cnt];
        for (int ix = 0; ix < elem_cnt; ++ix) _pmat[ix] = rhs._pmat[ix]; }

    return *this;};

```

Если разработчик класса поддерживает подробный вариант оператора присваивания класса, этот вариант используется вместо инициализации методом членства по умолчанию. Исходный код пользователю не нужно менять, хотя он должен выполнить перекомпиляцию.

(Точнее говоря, эта реализация не сохраняет исключение. Смотрите [SUTTER99] для уточнения).

## 4.9 Реализация объекта функции

В Разделе 3.6 мы рассматривали predefined объекты функции стандартной библиотеки. В этом разделе мы рассмотрим, как реализовать наши собственные объекты функции. Объект функции – это класс, который поддерживает перезагружаемые образцы оператора вызова функции.

Когда компилятор высчитывает, какое появление функции вызывается, как ниже

```
lt(ival);
```

lt может быть именем функции, указателем на функцию, или объектом класса, который поддерживает вариации оператора вызова функции. Если lt объект класса, компилятор внутренне преобразует выражение следующим образом:

```
lt.operator()(ival); // внутреннее преобразование
```

Оператор вызова функции может принять некоторое количество параметров: ни одного, один, два и т.д. Он используется, например, для поддержки многомерной индексации класса Matrix, поскольку реальный оператор индексации ограничен использованием одного параметра.

Давайте, реализуем перезагружаемый вариант оператора вызова для проверки, будет ли передаваемое ему значение меньше, чем какое-то другое. Мы будем вызывать класс LessThan. Каждый объект должен быть инициализирован со значением для сравнения. Вдобавок, мы поддержим доступ на чтение и запись для этого значения. Вот наша реализация:

```
class LessThan {
public:
    LessThan(int val) : _val(val){}
    int comp_val() const { return _val; }
    void comp_val(int nval){ _val = nval; }

    bool operator()(int value) const;

private:
    int _val;};
```

Реализация оператора вызова функции выглядит, примерно, так:

```
inline bool LessThan::
operator()(int value) const { return value < _val; }
```

Мы можем явно определить объект класса LessThan таким же образом, как любой другой объект класса:

```
LessThan lt10(10);
```

Мы вызываем перезагружаемый оператор вызова функции, добавлением оператора вызова (()) к объекту класса. Например,

```
int count_less_than(const vector<int> &vec, int comp){
    LessThan lt(comp);

    int count = 0;
```

```

    for (int ix = 0; ix < vec.size(); ++ix)
        if (lt(vec[ix])) ++count;

    return count;}

```

Более типична передача объекта функции, как аргумента в общий алгоритм:

```

void print_less_than(const vector<int> &vec, int comp, ostream &os = cout {
    LessThan lt(comp);

    vector<int>::const_iterator iter = vec.begin();

    vector<int>::const_iterator it_end = vec.end();

    os << "elements less than " << lt.comp_val() << endl;
    while ((iter = find_if(iter, it_end, lt)) != it_end) {
        os << *iter << ' '; ++iter;}}

```

Вот маленькая программа для упражнений с этими двумя функциями:

```

int main(){
    int ia[16] = { 17, 12, 44, 9, 18, 45, 6, 14,
                  23, 67, 9, 0, 27, 55, 8, 16 };
    vector<int> vec(ia, ia+16);
    int comp_val = 20;

    cout << "Number of elements less than "
          << comp_val << " are "
          << count_less_than(vec, comp_val) << endl;
    print_less_than(vec, comp_val);}

```

После компиляции при выполнении программа генерирует следующее:

```

Number of elements less than 20 are 10

elements less than 20 17 12 9 18 6 14 9 0 8 16

```

[Приложение В](#) содержит дополнительные примеры определения объектов функций.

## 4.10 Представление класса исключений операторов `iostream`

Часто мы хотим как читать, так и записывать в объекты класса. Например, для вывода на дисплей нашего объекта класса `trian`, мы хотим иметь возможность записи

```

cout << trian << endl;

```

Для поддержки этого мы должны поддержать перезагружаемые образцы оператора вывода:

```

ostream& operator<<(ostream &os, const Triangular &rhs){
    os << "(" << rhs.beg_pos() << ", " << rhs.length() << ") ";
    rhs.display(rhs.length(), rhs.beg_pos(), os);
}

```

```
return os;}
```

Мы возвращаем тот же объект `ostream`, переданный в функцию. Это позволяет множеству операторов вывода соединиться. Оба объекта передаются по ссылке. Операнд `ostream` не объявлен как `const`, поскольку каждый оператор вывода изменяет внутреннее состояние объекта `ostream`. Объект класса для вывода, `rhs`, объявлен как `const`, поскольку мы передаем его по ссылке для иного использования, чем модификация самого объекта. Например, возьмем объект

```
Triangular tri(6, 3);
```

выражение

```
cout << tri << '\n';
```

генерирует

```
(3 , 6) 6 10 15 21 28 36
```

Почему оператор вывода не функция-член? Функция-член требует, чтобы ее левый операнд был объектом класса. Если оператор вывода член-функция, тогда объект класса `tri` будет вынужден располагаться слева от оператора вывода:

```
tri << cout << '\n';
```

Это несколько неудобное использование класса!

Следующий оператор ввода читает только первые четыре элемента, представленные классом `Triangular`. Начальная позиция и его длина, только уникальные аспекты объекта класса `Triangular`. Реальные значения элементов инвариантны, они не сохраняются внутри особых объектов класса.

```
istream& operator>>(istream &is, Triangular &rhs) {  
    char ch1, ch2;  
    int bp, len;  
  
    // возьмем ввод: (3 , 6) 6 10 15 21 28 36  
    // ch1 == '(', bp == 3, ch2 == ',', len == 6  
    is >> ch1 >> bp  
        >> ch2 >> len;  
    // множество из трех данных-членов rhs ...  
    rhs.begin_pos(bp);  
    rhs.length(len);  
    rhs.next_reset();  
  
    return is;}
```

Вот маленькая программа для проверки наших операторов ввода и вывода:

```
int main(){  
  
    Triangular tri(6, 3);  
  
    cout << tri << '\n';  
  
    Triangular tri2;  
    cin >> tri2;
```

```
// посмотрим, что мы получили ...

cout << tri2;}
```

После компиляции при выполнении она генерирует следующий вывод (мой ввод отмечен жирным шрифтом):

```
(3,6) 6 10 15 21 28 36
(4,10)
(4,10) 10 15 21 28 36 45 55 66 78 91
```

Операторы ввода обычно более сложны для реализации, поскольку есть возможность считывания неправильных данных. Например, что если левая скобка была пропущена? Для этого примера я не рассматриваю возможности неправильного ввода. Более реальные примеры оператора ввода и обсуждение возможных ошибок `istream` смотрите в [LIPMAN98], Часть 20.

## 4.11 Указатели на функции-члены класса

Классы, поддерживающие Fibonacci, Pell, Lucas, Square и Pentagonal последовательности те же, что и класс `Triangular`, исключая алгоритм генерации элементов последовательности. В Части 5 мы организуем эти классы в объектно-ориентированный иерархический класс. В этом разделе мы реализуем основной класс последовательности, `num_sequence`, для поддержки всех шести последовательностей единым объектом класса. Вот наша `main()` программа:

```
int main() {
    num_sequence ns;
    const int pos = 8;
    for (int ix = 1; ix < num_sequence::num_of_sequences(); ++ix) {
        ns.set_sequence(num_sequence::nstype(ix));
        int elem_val = ns.elem(pos);
        display(cout, ns, pos, elem_val); }}
```

`ns` объект нашего основного класса последовательности. С каждым проходом цикла `for` мы сбрасываем `ns` для представления другой числовой последовательности, используя `set_sequence()` и `nstype()` возвращаемые значения. `num_of_sequences()` возвращает число числовых последовательностей уже поддерживаемых. Обе `num_of_sequences()` и `nstype()` `inline` статические функции-члены. `elem()` возвращает элемент в запрашиваемой позиции. После компиляции при выполнении программа генерирует следующее:

```
The element at position 8 for the fibonacci sequence is 21

The element at position 8 for the pell sequence is 408

The element at position 8 for the lucas sequence is 47

The element at position 8 for the triangular sequence is 36

The element at position 8 for the square sequence is 64

The element at position 8 for the pentagonal sequence is 92
```

Ключом к разработке класса `num_sequence` служат возможности указателя на функции-члены. Указатель на функцию-член выглядит так же, как указатель на функцию-не-член (представленную в Разделе 2.8). Оба

обозначаются возвращаемым типом и списком параметров. Указатель на функцию-член, однако, должен так же обозначать класс, членом которого является. Например,

```
void (num_sequence::*pm) (int) = 0;
```

объявляет `pm` указателем на функцию-член класса `num_sequence`. Адресуемая `pm` функция-член должна иметь возвращаемый тип `void` и должна принимать единственный параметр типа `int`. `pm` инициализируется 0, обозначающим, что он не адресуется в настоящий момент к функции-члену.

Если синтаксис выглядит слишком сложно, мы можем скрыть его за `typedef`. Например,

```
typedef void (num_sequence::*PtrType) (int);
```

```
PtrType pm = 0;
```

объявляя `PtrType`, как альтернативное `typedef` имя для указателя на функцию-член класса `num_sequence`, возвращающую `void` и принимающую единственный параметр типа `int`. Оба объявления `pm` эквивалентны.

Шесть числовых последовательностей одинаковы, исключая алгоритм, генерирующий элементы последовательности. Класс `num_sequence` поддерживает следующие шесть функций-членов, каждая из которых адресуется нашим `PtrType` указателем на функцию-член:

```
class num_sequence {
public:
    typedef void (num_sequence::*PtrType) (int);
    // _pmf адресуется к одной из
    void fibonacci(int);
    void pell(int);
    void lucas(int);
    void triangular(int);
    void square(int);
    void pentagonal(int);
    // ...

private:
    PtrType _pmf;};
```

Для получения адреса функции-члена класса мы используем оператор взятия адреса (`&`) для имен членов-функций определенных их оператором границ класса. Возвращаемый тип и список параметров функции не определены. Например, для определения и инициализации указателя на функцию-член последовательности `fibonacci()`, мы пишем

```
PtrType pm = &num_sequence::fibonacci;
```

Аналогично, присваивая `pm`, мы пишем

```
pm = &num_sequence::triangular;
```

Каждый вызов `set_sequence()` присваивает `_pmf` адрес одной из шести функций. Для упрощения мы можем хранить адреса шести функций-членов в статическом массиве. Для предотвращения пересчета элементов каждой последовательности, мы так же сохраняем статический вектор шести векторов элементов:

```
class num_sequence {
public:
    typedef void (num_sequence::*PtrType) (int);
    // ...
private:
    vector<int>* _elem;
    // указатели на текущий вектор
    PtrType _pmf;
    // указатели на текущий алгоритм
    static const int num_seq = 7;
    //
    !static PtrType func_tbl[num_seq];
    static vector<vector<int> > seq;};
```

Наиболее сложное определение членства здесь в `seq`:

```
static vector<vector<int> > seq;
```

Это говорит о том, что `seq` – вектор, в котором каждый элемент это вектор целых элементов. Например, он может содержать векторы в качестве элементов каждой из наших шести числовых последовательностей. Если мы забудем пробелы между двумя «больше, чем» символами,

```
// не будет компилироваться

!static vector<vector<int>> seq;
```

определение не компилируется. Это происходит из-за правила *максимального прослеживания* (*maximal munch*) компиляции. Это правило требует, чтобы символ следования всегда интерпретировался, как максимально допустимый в последовательности символ. Поскольку `>>` допустимый оператор последовательности, оба символа всегда группируются вместе в отсутствие пробела. Аналогично, когда мы пишем `a+++p` правило максимального прослеживания интерпретирует это как

```
a++ + p
```

Мы должны поддерживать определение каждого статического члена-данных. Поскольку `PtrType` – вложенный тип, любые ссылки на него вне класса `num_sequence` должны быть определены оператором границ класса. Значение `num_seq` определяется внутри определения класса, так что нет нужды нам повторять его здесь.

```
const int num_sequence::num_seq;

vector<vector<int> > num_sequence::seq(num_seq);

num_sequence::PtrType num_sequence::func_tbl[num_seq] ={
    0,
    &num_sequence::fibonacci,
```

```

&num_sequence::pell,
&num_sequence::lucas,
&num_sequence::triangular,
&num_sequence::square,
&num_sequence::pentagonal};

```

Если вы найдете синтаксис вложенного типа путанным, вы можете скрыть его в typedef:

```

typedef num_sequence::PtrType PtrType;

PtrType num_sequence::func_tbl[num_seq] = ...

```

`_elem` и `_pmf` объединены внутри `set_sequence()`. `_elem` адресуется к вектору, который содержит элементы числовой последовательности. `_pmf`, конечно, адресуется к функции-члену для генерации дополнительных элементов последовательности. (Действительная реализация `set_sequence()` изменена в [Разделе 5.2](#)).

Подобно указателю на функцию, указатель на функцию-член должен вызываться через объект класса функции-члена. Объект становится `this` указателем функции-члена, которая вызывается.

Например, посмотрите, как мы получаем следующие определения:

```

num_sequence ns;
num_sequence *pns = &ns;
PtrType pm = &num_sequence::fibonacci;

```

Для вызова `pmf` через `ns`, мы пишем

```

// эквивалентно ns.fibonacci(pos)

(ns.*pm)(pos)

```

Пара символов `.*` – это указатель на оператор выбора члена для объекта класса. Скобки необходимы для того, чтобы он выполнялся корректно. Указатель на оператор выбора члена для указателя на оператор класса – это пара символов `->*`:

```

// эквивалентно pns->fibonacci(pos)

(pns->*pm)(pos)

```

Следующее – это реализация `elem()`. Если позиция, запрашиваемая пользователем, правильна, и если число элементов, в настоящий момент сохраненных, не включает эту позицию, необходимые элементы генерируются через вызов функции, адресуемой `_pmf`.

```

int num_sequence::elem(int pos){

    if (! check_integrity(pos))return 0;

    if (pos > _elem->size())
        (this->*_pmf)(pos-1);

    return (*_elem)[pos-1];}

```

Вот вся реализация, которая нужна нам для обсуждения в этой части. В [Части 5](#) мы рассмотрим программные детали, которые позволят узнать каждому объекту класса `num_sequence` его тип числовой последовательности в любой момент его существования. Затем мы рассмотрим простые пути для поддержки этого рода манипуляций с множеством типов через объектно-ориентированное программирование.

## Упражнение 4.3

Рассмотрим следующие глобальные данные:

```
string program_name;
string version_stamp;
int version_number;
int tests_run;
int tests_passed;
```

Напишите класс для упаковки этих данных.

## Упражнение 4.4

Профиль пользователя состоит из логина, действительного имени пользователя, количества подключений, количества сделанных попыток, количество удачных попыток, текущего уровня – начинающий, средний, опытный или специалист – и процента исправлений (это позже может быть вычислено или сохранение). Создайте класс `UserProfile`. Поддержите ввод и вывод, эквивалентность и неэквивалентность. Конструкторы должны допускать определение пользователем уровня и поддерживать логин «гость» (`guest`). Насколько вы можете гарантировать уникальность гостевого логина для каждой отдельной сессии?

## Упражнение 4.5

Реализуйте класс `Matrix 4x4`, поддерживающий, по меньшей мере, следующий основной интерфейс: сложение и умножение двух объектов `Matrix`, функцию-член `print()`, смешанный оператор `+=` и индексацию, поддержанную парой перезагружаемых операторов вызова функции, как следующие:

```
float& operator()(int row, int column);

float operator()(int row, int column) const;
```

Создайте конструктор по умолчанию, принимающий дополнительные 16 значений данных, и конструктор, принимающий массив из 16 элементов. Вам не нужен конструктор копирования, оператор присваивания копирования, или деструктор для этого класса (это потребуется в [Части 6](#), где мы переделаем класс `Matrix` для поддержки произвольного количества строк и столбцов).

## Часть 5. Объектно-ориентированное программирование

Как вы видели в [Части 4](#) при первом использовании класса – это вводит новый тип, более явно представляющий сущность нашего приложения. В приложении для проверки библиотеки, например, гораздо легче программировать классы Книга, Читатели, СрокВозврата непосредственно, чем переводить все в программную логику для символьных, арифметических и Булевых типов данных.

Модель объектно-базируемого программирования становится громоздкой, когда наше приложение начинает наполняться типами классов, представляющими экземпляры типов *нечто-в-роде* (*is-a-kind-of*). Например, представьте, что по прошествии времени наше приложение проверки библиотеки должно пополниться классом КнигаНапрокат, классом АудиоКнига и классом ИнтерактивныеКниги в дополнение к оригинальному классу Книга. Каждый класс, похоже, расширяет члены-данные для представления его статуса. Каждый класс может (или нет) иметь отдельный алгоритм проверки и штрафов по срокам возврата, хотя каждый класс разделяет тот же интерфейс.

Механизм объектно-базируемого класса [Части 4](#) не может с легкостью моделировать общность и различия этих четырех *нечто-в-роде* класса Книга. Почему? А потому, что эта модель не предоставляет поддержки для спецификации отношений между классами. Для такого рода поддержки мы нуждаемся в модели *объектно-ориентированного программирования*.

### 5.1 Концепции объектно-ориентированного программирования

Двумя первичными характеристиками объектно-ориентированного программирования являются наследование и полиморфизм. *Наследование* позволяет нам группировать классы в фамилии связанных типов, давая возможность разделять общие операции и данные. *Полиморфизм* позволяет нам программировать эти фамилии, как объединения несколько иные, чем индивидуальные классы, предоставляя нам больше гибкости в добавлении или удалении отдельных классов.

Наследование определяет взаимосвязь предок/потомок. *Предок* определяет общий интерфейс и частную реализацию, общие для всех его потомков. Каждый *потомок* добавляет или изменяет, что ему наследовать для реализации его собственного уникального выполнения. Класс потомка АудиоКнига, например, в дополнение к названию и автору, наследованными от предка класса Книга, проводит поддержку для громкоговорителей и подсчета количества кассет. Вдобавок он изменяет наследованную член-функцию `check_out()` своего предка.

В C++ предок называется *базовым* классом, а потомок – производным классом. Взаимосвязь между родительским или базовым классом и его потомками называется *иерархия наследования*. На обзорном совещании по разработке мы, например, могли бы сказать: «Мы намерены разработать производный класс АудиоКнига. Он изменит метод `check_out()` его базового класса Книга. Однако он снова будет использовать наследованные члены-данные класса Книга и члены-функции для обслуживания местоположения на полках, имени автора и названия».

[Рисунок 5.1](#) показывает часть возможной иерархии класса выдаваемого материала библиотеки. Корнем иерархии класса служит абстрактный базовый класс LibMat. LibMat определяет все операции, общие для всех различных типов выдаваемого библиотечного материала: `check_in()`, `check_out()`, `due_date()`, `fine()`, `location()` и т.д. LibMat не представляет реальных объектов выдаваемого материала. Скорее, он - артефакт нашей разработки. И фактически ключевой артефакт. Мы назовем его абстрактным базовым классом.

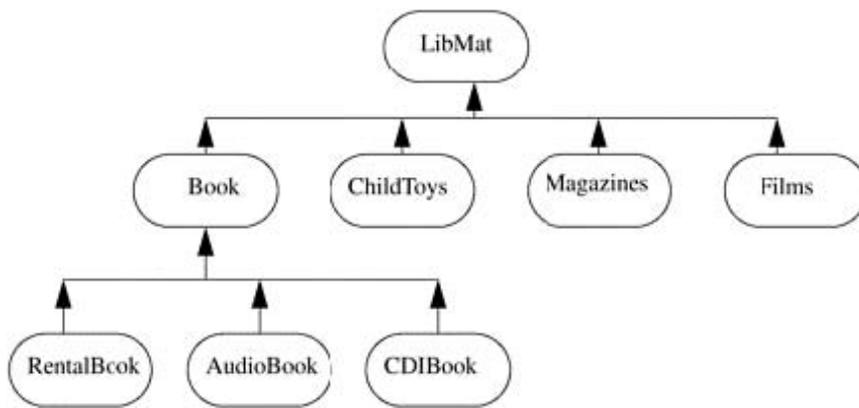


Рисунок 5.1. Составляющие иерархии класса библиотечных выдаваемых материалов

В объектно-ориентированном программировании мы косвенно манипулируем объектами класса нашего приложения через указатель или ссылку на абстрактный базовый класс вместо прямого управления объектами реального производного класса нашего приложения. Это позволяет нам добавлять или удалять производный класс без необходимости каких-либо переделок нашей существующей программы. Например, рассмотрим следующую маленькую функцию:

```

void loan_check_in(LibMat &mat) {
    // mat сейчас ссылается на объект производного класса
    // как Книга, КнигаНапрокат, Журналы и т.д. ...
    mat.check_in();

    if (mat.is_late()) mat.assess_fine();

    if (mat.waiting_list());

    mat.notify_available();}
  
```

В нашем приложении нет объектов LibMat, только Книга, КнигаНапрокат, АудиоCD и т.д. Как эта функция реально работает? Что случится, например, когда операция `check_in()` вызывается через `mat`? Чтобы эта функция имела смысл, `mat` должна как-то ссылаться на один из объектов реального класса нашего приложения каждый раз, как выполняется `loan_check_in()`. Вдобавок функция-член `check_in()`, которая вызывается, должна как-то превращаться в `check_in()` образец объекта реального класса, на который ссылается `mat`. Вот что происходит. Вопрос в том, как оно работает?

Второй уникальный аспект объектно-ориентированного программирования – это полиморфизм: возможность указателю на базовый класс или ссылке явно ссылаться на любой из объектов его производного класса. В нашей функции `loan_check_in()`, например, `mat` всегда адресуется к некоему объекту одного из классов производных от LibMat. К какому? Это невозможно определить вне реального выполнения программы, и, похоже, оно варьируется с каждым вызовом `loan_check_in()`.

*Динамическое связывание* – это третий аспект объектно-ориентированного программирования. В не объектно-ориентированном программировании, когда мы пишем

```
mat.check_in();
```

образец `check-in()`, который будет выполняться, определяется при компиляции, базирясь на типе класса `mat`. Поскольку функция, которая будет вызвана, трансформируется до начала выполнения программы, это

называется *статическим связыванием*.

В объектно-ориентированном программировании компилятор не может знать, какой образец `check_in()` вызывать. Это может быть определено только при выполнении программы, основываясь на объекте реального производного класса, к которому адресуется `mat` каждый раз при вызове `loan_check_in()`. Реальное превращение в некоторый образец `check_in()` производного класса при вызове, каждый раз должно быть отложено до выполнения. Вот почему мы говорим о динамическом связывании.

Наследование позволяет нам определить фамилии классов, которые разделяют общий интерфейс, как наш выдаваемый материал библиотеки. Полиморфизм позволяет нам манипулировать объектами этих классов независимым от типа способом. Мы программируем общий интерфейс через указатель или ссылку на абстрактный базовый класс. Реальная вызываемая операция не определена до выполнения, осуществляющегося на типе объекта реально адресуемого. (Да, полиморфизм и динамическое связывание поддерживаются только, когда мы используем указатель или ссылку. Я скажу об этом позже).

Если библиотека решит больше не выдавать интерактивные книги, мы просто удалим этот класс из иерархии наследования. Реализацию `loan_check_in()` менять не понадобится. Аналогично, если библиотека решит начислить оплату на аудио книги, мы можем реализовать производную `АудиоПлатнаяКнига`, `loan_check_in()` все еще не нужно менять. Если библиотека решит ссужать ноутбуками или игровыми приставками и картриджами, наша иерархия наследования может приспособиться и к этому.

## 5.2 Экскурсия по объектно-ориентированному программированию

Давайте, реализуем простую трехуровневую иерархию класса для представления о конструкциях и идиомах программирования языка C++, которые поддерживают объектно-ориентированное программирование. Мы начнем нашу иерархию класса с абстрактного базового класса `LibMat`. Мы получим класс `Book` из `LibMat` и, в свою очередь, получим класс `AudioBook` из `Book`. Мы ограничим наш интерфейс единственной функцией `print()` вместе с конструктором и деструктором. Я оборудую каждую функцию-член выводом ее содержимого, так что мы сможем отследить прохождение выполнения программы.

По умолчанию функция-член создается статически во время компиляции. Чтобы иметь функцию-член, динамически создающуюся при выполнении программы, мы предположим ее объявлению ключевое слово `virtual`. Класс `LibMat` объявляет свой деструктор и `print()` виртуальными:

```
class LibMat {public:LibMat(){
    cout << "LibMat::LibMat() default constructor!\n"; }
    virtual ~LibMat(){ cout << "LibMat::~LibMat() destructor!\n"; }
    virtual void print() const{
    cout << "LibMat::print() -- I am a LibMat object!\n"; }};
```

Теперь определим функцию-не-член `print()`, которая принимает единственный параметр `const` ссылку на `LibMat`:

```
void print(const LibMat &mat){
    cout << "in global print(): about to print mat.print()\n";
```

```

// это превращение в print(), член-функцию
// основанную на реальном объекте mat, ссылающемся на ...
mat.print();}

```

Внутри нашей программы `main()` мы повторно вызываем `print()`, передавая ее, в свою очередь, объекту класса `LibMat`, объекту класса `Book` и объекту класса `AudioBook`, как их параметр. Базируясь на реальном объекте `mat`, ссылающемся внутри каждого вызова `print()` на соответствующую `LibMat`, `Book` или `AudioBook` функцию-член `print()`, которая вызывается. Наш первый вызов может быть похож на это:

```

cout << "\n" << "Creating a LibMat object to print()\n";
LibMat libmat;
print(libmat);

```

Вот трассировка выполнения:

```

Creating a LibMat object to print()

// конструкция Libmat libmat
LibMat::LibMat() default constructor!

// поддержка print(libmat)
in global print(): about to print mat.print()
LibMat::print() -- I am a LibMat object!

// деструкция Libmat libmat
LibMat::~LibMat() destructor!

```

Надеюсь, здесь нет ничего удивительного. Определение `libmat` сопровождается вызовом конструктора по умолчанию. Внутри `print()`, `mat.print()` превращаются в `LibMat::print()`. Это сопровождается вызовом деструктора `LibMat`. Все становится несколько более удивительным, когда мы передаем `print()` объекту `Book`:

```

cout << "\n" << "Creating a Book object to print()\n";
Book b("The Castle", "Franz Kafka");
print(b);

```

А это трассировка с комментариями:

```

Creating a Book object to print()

// конструкция Book b
LibMat::LibMat() default constructor!
Book::Book(The Castle, Franz Kafka) constructor

// поддержка print(b)
in global print(): about to print mat.print()
Book::print() -- I am a Book object!
My title is: The Castle
My author is: Franz Kafka

// деструкция Book b
Book::~Book() destructor!

```

```
LibMat::~~LibMat() destructor!
```

По первому впечатлению виртуальный вызов через `mat.print()` реально работает! Вызываемая функция `Book::print()`, а не `LibMat::print()`. Вторая интересная особенность в том, что оба, базовый и производный конструкторы класса, вызываются, когда определяется объект производного класса. (Когда объект производного класса уничтожается, вызываются оба, производный и базовый деструкторы класса). Как реально реализовать производный класс `Book`? Чтобы обозначить, что наш новый класс наследуется от существующего класса, мы сопровождаем его имя двоеточием (:), сопровождаемым ключевым словом<sup>9</sup> `public` и именем базового класса:

```
class Book : public LibMat {
public:
    Book(const string &title, const string &author)
        : _title(title), _author(author) {
        cout << "Book::Book(" << _title
            << ", " << _author << ") constructor\n"; }

    virtual ~Book(){
        cout << "Book::~~Book() destructor!\n"; }

    virtual void print() const {
        cout << "Book::print() -- I am a Book object!\n"
            << "My title is: " << _title << '\n'
            << "My author is: " << _author << endl; }

    const string& title() const { return _title; }
    const string& author() const { return _author; }

protected: string _title;string _author;};
```

Образец `print()` внутри `Book` *подменяет* `LibMat` образец. Это функция, вызываемая `mat.print()`. Две доступные функции `title()` и `author()` – это не виртуальные `inline` члены-функции. Мы не видели ключевого слова `protected` прежде. Член, объявленный как защищенный, может непосредственно быть доступен производным классам, но не может быть непосредственно доступен из основной программы.

Давайте, определим следующий `AudioBook` производный класс от класса `Book`. `AudioBook` в дополнение к названию и автору имеет диктора. Прежде, чем мы рассмотрим его реализацию, давайте вначале передадим `print()` объекту класса `AudioBook`:

```
cout << "\n" << "Creating an AudioBook object to print()\n";

AudioBook ab("Man Without Qualities","Robert Musil", "Kenneth Meyer");

print(ab);
```

Что мы ожидаем от трассировки при выполнении? Мы ожидаем (1) что `AudioBook::print()` вызывается

---

<sup>9</sup> Базовый класс может быть обозначен как общий (`public`), защищенный (`protected`) и частный (`private`). Общее наследование единственная форма наследования охваченная этой книгой. Для знакомства с защищенным и частным наследованием загляните в Раздел 18.3 [LIPMAN98].

через `mat.print()` и (2) что `ab` конструируется, в свою очередь, `LibMat`, `Book` и `AudioBook` конструкторами. Вот, что показывает трассировка:

```
Creating an AudioBook object to print()

// the construction of AudioBook ab
LibMat::LibMat() default constructor!
Book::Book(Man Without Qualities, Robert Musil) constructor

AudioBook::AudioBook(Man Without Qualities, Robert Musil, Kenneth Meyer)
constructor

// the resolution of print(ab)
in global print(): about to print mat.print()
// oops: need to handle a Book and an AudioBook!
AudioBook::print() -- I am a AudioBook object!
My title is: Man Without Qualities
My author is: Robert Musil
My narrator is: Kenneth Meyer

// the destruction of AudioBook ab
AudioBook::~AudioBook() destructor!
Book::~Book() destructor!
LibMat::~LibMat() destructor!
```

Как мы реализуем производный класс `AudioBook`? Мы должны только запрограммировать те аспекты `AudioBook`, которые отличают его от базового класса `Book`: `print()`, функция, конечно, должна поддерживать имя диктора в `AudioBook`, а также конструктор и деструктор класса. Члены-данные и члены-функции класса `Book`, которые поддерживают автора и название, могут использоваться непосредственно внутри класса `AudioBook` так же, как если бы они были определены иначе, чем через наследование.

```
class AudioBook : public Book {
public:
    AudioBook(const string &title, const string &author,
              const string &narrator): Book(title, author), _narrator(narrator) {
        cout << "AudioBook::AudioBook(" << _title
              << ", " << _author
              << ", " << _narrator
              << ") constructor\n"; }

    ~AudioBook() {

        cout << "AudioBook::~AudioBook() destructor!\n"; }
    virtual void print() const {

        cout << "AudioBook::print() -- I am an AudioBook object!\n"
              << // note the direct access of the inherited
              << // data members _title and _author
              << "My title is: " << _title << '\n'
              << "My author is: " << _author << '\n'
              << "My narrator is: " << _narrator << endl; }

    const string& narrator() const { return _narrator; }
};
```

```
protected:string _narrator;};
```

Пользователи производного класса не нуждаются в различении наследованных членов и членов, действительно определенных внутри производного класса. Использование и тех, и других прозрачно:

```
int main(){
    AudioBook ab("Mason and Dixon","Thomas Pynchon", "Edwin Leonard");
    cout << "The title is " << ab.title() << '\n'
         << "The author is " << ab.author() << '\n'
         << "The narrator is " << ab.narrator() << endl; }
```

Я надеюсь, что этот раздел дал вам представление о том, как объектно-ориентированное программирование поддерживается в C++. Буквально, все это дано без деталей, которым мы отведем место в оставшейся части. Хорошим упражнением в этом месте было бы следующее: (1) Загрузите исходный текст с сайта Addison Wesley Longman. (2) Рассмотрите в [Части 5](#) указатель на иерархию класса LibMat и программу `main()`, которая ее реализует. (3) Сделайте производный класс Magazine от класса LibMat и добавьте вызов `print()`, передаваемый в объект класса Magazine.

## 5.3 Полиморфизм без наследования

Класс `num_sequence` из [Раздела 4.10](#) симулирует полиморфизм. Каждый объект класса может быть воссоздан в любой из шести числовых последовательностей в любой точке программы через `set_sequence()` функцию-член:

```
for (int ix = 1; ix < num_sequence::num_of_sequences(); ++ix) {
    ns.set_sequence(num_sequence::nstype(ix));
    int elem_val = ns.elem(pos); // ... }
```

Возможность изменять тип последовательности `ns` поддерживается скорее через программирование, чем через прямые возможности языка. Каждый объект класса содержит `_isa` данные-член, идентифицирующие текущую числовую последовательность, которую представляют:

```
class num_sequence {
public:// ...

private:
    vector<int> *_elem; // адресуется к текущему элементу вектора
    PtrType _pmf; // адресуется к текущему элементу генератора
    ns_type _isa; // идентифицирует текущий тип последовательности
    // ... };
```

`_isa` установлена в именованное постоянное значение, которое представляет один из поддерживаемых типов числовой последовательности. Постоянные значения сгруппированы в перенумерованный тип, который я назвал `ns_type`:

```
class num_sequence {
public:
    enum ns_type {
        ns_unset, ns_fibonacci, ns_pell, ns_lucas,
        ns_triangular, ns_square, ns_pentagonal };
```

```
// ...};
```

`nstype()` проверяет, что ее целый параметр представляет правильное значение числовой последовательности. Если так, она возвращает ассоциированный счетчик, иначе возвращает `ns_unset`:

```
class num_sequence {  
  
    public:// ...  
  
        static ns_type nstype(int num) {  
            return num <= 0 || num >= num_seq  
                ? ns_unset // invalid value  
                : static_cast< ns_type >(num);  }};
```

`static_cast` это специальная конверсионная запись. Она преобразует целое `num` в ассоциированный с ним счетчик `ns_type`. Результат `nstype()` передается в `set_sequence()`:

```
ns.set_sequence(num_sequence::nstype(ix));
```

`set_sequence()` выполняет работу по установке `_pmf`, `_isa` и `_elem` данных-членов в корректную числовую последовательность:

```
void num_sequence::set_sequence(ns_type nst){  
    switch (nst){  
        default:  
            cerr << "invalid type: setting to 0\n";  
            // предумышленный сбой  
  
            case ns_unset:  
                _pmf = 0;  
                _elem = 0;  
                _isa = ns_unset;  
                break;  
  
            case ns_fibonacci: case ns_pell: case ns_lucas:  
            case ns_triangular: case ns_square: case ns_pentagonal:  
            // func_tbl: таблица указателей на функции-члены  
            // seq: вектор векторов, содержащих элементы последовательности  
                _pmf = func_tbl[nst];  
                _elem = &seq[nst];  
                _isa = nst;break;  }}
```

Для поддержки запросов к числовой последовательности, которой в текущий момент стал объект, я включил операцию `what_am_i()`. Она возвращает строку символов, идентифицирующую текущую числовую последовательность. Например,

```
inline void display(ostream &os, const num_sequence &ns, int pos){  
    os << "The element at position "  
        << pos << " for the "  
        << ns.what_am_i() << " sequence is "  
        << ns.elem(pos) << endl;}
```

`what_am_i()` индексирует `_isa` в статическом массиве символьной строки, который инвентаризует поддерживаемые имена числовых последовательностей в порядке установленным счетчиком `ns_type`:

```

const char* num_sequence::what_am_i() const {
    static char *names[num_seq] = {
        "notSet",
        "fibonacci", "pell",
        "lucas", "triangular",
        "square", "pentagonal" };

    return names[_isa];}

```

Это куча работы, особенно в плане поддержки. Каждый раз, когда мы хотим добавить или удалить тип числовой последовательности, все вышенаписанное должно быть корректно обновлено: вектор векторов элементов, массив указателей на функции-члены, `what_am_i()` строковый массив, `switch` выражение `set_sequence()`, значение `num_seq` и т.д. В модели объектно-ориентированного программирования, такого рода подробные программные издержки устранены, что делает наш код проще и много эластичнее.

## 5.4 Определение абстрактного базового класса

В этом разделе мы реконструируем класс `num_sequence` предыдущего раздела в абстрактный базовый класс, из которого мы будем наследовать каждый из классов числовых последовательностей. Каким же образом можно это осуществить?

Первый шаг в определении абстрактного базового класса – определить множество операций общих для его потомков. Например, какие операции являются общими для всех классов числовых последовательностей? Эти операции представляют общий интерфейс базового класса `num_sequence`. Вот первая итерация:

```

class num_sequence {
public:
    // elem(pos): возвращает элемент в pos
    // gen_elems(pos): генерирует элементы до pos
    // what_am_i() : определяет реальную последовательность
    // print(os) : записывает элементы в os
    //check_integrity(pos) : правильное ли значение pos?
    // max_elems() : возвращает максимальную поддерживаемую позицию
    int elem(int pos);
    void gen_elems(int pos);
    const char* what_am_i() const;
    ostream& print(ostream &os = cout) const;
    bool check_integrity(int pos);
    static int max_elems();// ... };

```

`elem()` возвращает элемент в запрошенную пользователем позицию. `max_elems()` возвращает максимальное число элементов, поддерживаемое нашей реализацией. `check_integrity()` определяет, является ли `pos` правильной позицией. `print()` выводит на дисплей элементы. `gen_elems()` генерирует элементы последовательности. `what_am_i()` возвращает строку символов идентифицирующий последовательность.

Следующий шаг в разработке абстрактного базового класса – определить, какие операции типозависимые, т.е. какие операции требуют отдельной реализации, базирующейся на типе производного класса. Эти операции станут виртуальными функциями в иерархии класса. Например, класс каждой числовой последовательности должен поддерживать свою реализацию `gen_elems()`. `check_integrity()`, с другой стороны, типозависима. Она должна определить, является ли `pos` правильной позицией элемента. Ее алгоритм не зависит от конкретной числовой последовательности. Аналогично, `max_elems()` типозависима.

инвариантна. Все числовые последовательности содержат одинаковое максимальное число элементов.

Не каждая из этих функций легко определяема. `what_am_i()` может быть, а может не быть, типозависимой, смотря по тому, какую реализацию нашей иерархии наследования мы выберем. Это же справедливо для `elem()` и `print()`. Сейчас положим, что они типозависимы. Позже, мы рассмотрим альтернативную разработку, которая превратит их в типозависимые функции. Статические члены-функции не могут объявляться, как виртуальные.

Третий шаг в разработке абстрактного базового класса – определить уровень доступа к каждой операции. Если операция должна быть доступна из основной программы, мы объявляем ее как `public`. Например, `elem()`, `max_elems()` и `what_am_i()` общие (`public`) операции.

Если предполагается, что операция не будет вызываться вне базового класса, мы объявляем ее `private`. Частный член базового класса не может быть доступен из классов, которые наследуют от базового класса. В этом примере все операции должны быть доступны для наследующих классов, поэтому нет нужды объявлять их, как `private`.

Третий уровень доступа, `protected`, определяет операции, которые доступны для наследующих классов, но не для основной программы. `check_integrity()` и `gen_elems()`, например, операции, которые должны вызываться классами потомками, но не основной программой. Вот наше переделанное определение класса `num_sequence`:

```
class num_sequence {
public:
    virtual ~num_sequence(){};

    virtual int elem(int pos) const = 0;
    virtual const char* what_am_i() const = 0;
    static int max_elems(){ return _max_elems; }
    virtual ostream& print(ostream &os = cout) const = 0;

protected:
    virtual void gen_elems(int pos) const = 0;
    bool check_integrity(int pos) const;

    const static int _max_elems = 1024;};
```

Каждая виртуальная функция должна быть определена для каждого класса, который ее объявляет, или, если нет ясно выраженного внедрения этой функции в класс (как `gen_elems()`), должна быть объявлена, как чисто виртуальная функция. Присваивание 0 обозначает, что виртуальная функция – это чисто виртуальная:

```
virtual void gen_elems(int pos) = 0;
```

Поскольку ее интерфейс не полон, класс, который объявляет ее или несколько чисто виртуальных функций, не может иметь независимых объектов класса, определенных в программе. Он может обслуживаться только, как суб-объект производного класса, который, в результате, пополняется поддержкой за счет конкретной разработки для каждой его чисто виртуальной функции. Какие данные, если они есть, должен объявить класс `num_sequence`? На этот счет нет твердых и быстрых правил. В этой разработке класса `num_sequence` не объявляет никаких данных-членов класса. Разработка поддерживает интерфейс для иерархии числовой последовательности, но различает реализации ее производных классов.

А что насчет конструкторов и деструкторов? Поскольку нет инициализации нестатических данных-членов внутри класса, нет реальной пользы от поддержки конструктора. Мы будем, однако, поддерживать

деструктор. Есть такое правило – базовый класс, который определяет одну или несколько виртуальных функций, всегда должен определять виртуальный деструктор. Например,

```
class num_sequence {
public:
    virtual ~num_sequence();
    // ...};
```

Почему? Рассмотрим следующие коды последовательности:

```
num_sequence *ps = new Fibonacci(12);
// ... используем последовательность
delete ps;
```

`ps` указатель базового класса `num_sequence`, но адресуется он к объекту `Fibonacci` производного класса. Когда выражение `delete` обращается к указателю на объект класса, деструктор первым обращается к объекту, адресуемому указателем, затем память, ассоциированная с объектом класса, возвращается в свободный программный запас. Невиртуальная функция улаживает это при компиляции, основываясь на типе объекта, через который она вызвана.

В таком случае деструктор, вызываемый через `ps`, должен быть деструктором класса `Fibonacci`, а не деструктором класса `num_sequence`. И какой деструктор вызывать, должно быть решено при выполнении программы, основываясь на конкретной адресации к объекту указателем базового класса. Чтобы иметь такую возможность мы должны объявить деструктор виртуальным.

Однако я не рекомендую объявлять деструктор как чисто виртуальную функцию в базовом классе – даже если нет разработки в явном виде. Для деструктора лучше осуществить пустое определение, как показано ниже:<sup>10</sup>

```
inline num_sequence::~~num_sequence() {}
```

Для полноты картины, вот реализация в `num_sequence` экземпляра оператора вывода и

```
check_integrity():bool num_sequence::check_integrity(int pos) const{
    if (pos <= 0 || pos > _max_elems) {
        cerr << "!! invalid position: " << pos
            << " Cannot honor request\n";
        return false; }

    return true;}
ostream& operator<<(ostream &os, const num_sequence &ns){

    return ns.print(os); }
```

Хотя это завершает определение абстрактного базового класса `num_sequence`, класс сам по себе не полон. Он имеет интерфейс для производных классов подпоследовательностей. Каждый производный класс имеет реализацию, которая пополняет определение базового класса `num_sequence`.

---

<sup>10</sup> Смотрите введение в Часть 5 [LIPPMAN96a], где объясняется, почему виртуальный деструктор лучше не объявлять, как чисто виртуальную функцию.

## 5.5 Определение производного класса

Производный класс состоит из двух частей: суб-объекта базового класса (состоит из нестатических данных-членов базового класса, если они есть), и из доли, наследованной производным классом (состоит из нестатических данных-членов производного класса). (Как синяя часть эмблемы Lego, прихваченная вместе с красной). Эта смешанная природа производного класса отражена в его декларационном синтаксисе:

```
// файл заголовка содержит определение базового класса

#include "num_sequence.h"

class Fibonacci : public num_sequence {
public:
    // ...};
```

Имя производного класса сопровождается двоеточием, ключевым словом `public` и именем базового класса.<sup>11</sup> Единственное правило – определение базового класса должно присутствовать до того, как класс сможет наследовать от него (вот почему файл заголовка содержит включенным определение класса `num_sequence`).

Класс `Fibonacci` должен поддерживать реализацию каждой из чисто виртуальных функций, наследуемых от базового класса. Вдобавок, он должен объявить те члены, которые уникальны для класса `Fibonacci`. Вот определение класса:

```
class Fibonacci : public num_sequence {

public:

    Fibonacci(int len=1,int beg_pos = 1): _length(len), _beg_pos(beg_pos){}

    virtual int elem(int pos) const;
    virtual const char* what_am_i() const { return "Fibonacci"; }
    virtual ostream& print(ostream &os = cout) const;
    int length() const { return _length; }
    int beg_pos() const { return _beg_pos; }
protected:
    virtual void gen_elems(int pos) const;
    int _length;
    int _beg_pos;
    static vector<int> _elems;};
```

В этой конструкции длина и начальная позиция – данные-члены каждого производного класса. Доступ на чтение функций `length()` и `beg_pos()` объявлен, как не виртуальный, поскольку нет образцов базового класса, которые будут подменяться. По причине того, что они не являются частью интерфейса базового класса, к ним не будет доступа, когда мы программируем с использованием указателя или ссылки на базовый класс. Например,

```
num_sequence *ps = new Fibonacci(12, 8);
```

---

<sup>11</sup> Как я упоминал в Разделе 5.1, есть также поддержка частного (`private`) и защищенного (`protected`) наследования, столь же удачного, как множественное и виртуальное наследование. Полное и широкое освещение этого в книге не представлено, поэтому загляните в Часть 18 [LIPMAN98], где все изложено более полно

```

// ok: вызываем Fibonacci::what_am_i() через виртуальный механизм
ps->what_am_i();

// ok: вызываем наследованную num_sequence::max_elems();
ps->max_elems();

// ошибка: length() это не часть интерфейса num_sequence
ps->length();

// ok: вызываем деструктор Fibonacci через виртуальный механизм
delete ps;

```

Если недоступность `length()` и `beg_pos()` из-за отказа интерфейса базового класса, создает проблемы для пользователей, нам нужно вернуться и модифицировать интерфейс базового класса. Одна из возможных реконструкций – ввести `length()` и `beg_pos()`, как чисто виртуальные функции внутри базового класса `num_sequence`, что автоматически включает образцы `beg_pos()` и `length()` производного класса в виртуальные функции. Это одна из причин, по которым образцы виртуальных функций производного класса требуют введения ключевого слова `virtual`. Если бы ключевое слово требовалось, модификацию виртуальной функции базового класса, как `beg_pos()`, было бы трудно сделать хорошо: каждый образец производного класса нуждался бы в предекларации.

Альтернативной реконструкцией может служить вынос хранилища длины и начальной позиции из производного класса в базовый класс. За счет этого `length()` и `beg_pos()` становятся наследуемыми не виртуальными функциями. (Мы рассмотрим вариации этой конструкции в [Разделе 5.6](#)).

Моя точка зрения такова – характерным для объектно-ориентированного конструирования является не столько программирование, сколько распределение по базовому и производному классам, и определение интерфейса и членов, которые принадлежат каждому из них. В целом это итеративный процесс, который проходит через практику и обратную связь с пользователями.

Вот реализация `elem()`. Виртуальная функция производного класса должна в точности соответствовать функции-прототипу базового класса. Ключевое слово `virtual` не специфицировано в определении, а производится за пределами класса.

```

int Fibonacci::elem(int pos) const{
    if (! check_integrity(pos)) return 0;

    if (pos > _elems.size()) Fibonacci::gen_elems(pos);

    return _elems[pos-1];}

```

Заметьте, что `elem()` вызывает наследованный член `check_integrity()` в точности так, как если бы функция была членом этого класса. В основном наследованные общие и защищенные члены базового класса, благодаря глубине иерархии наследования, достигаются так, как если бы они были членами производного класса. Общие члены базового класса также будут общими в производном классе, и доступны пользователям производного класса. Защищенные члены базового класса защищены внутри производного класса. Они доступны для классов, наследующих производному классу, но не пользователям класса. Производный класс, однако, не имеет привилегии доступа к частным членам базового класса. Заметьте, что до возврата элемента в `pos`, сделана проверка, содержится ли соответствующий элемент в `_elems`. Если нет, `elem()` вызывает `gen_elems()` для пополнения `_elems` до `pos`. Вызов написан как `Fibonacci::gen_elems(pos)`, а не

простым `gen_elems(pos)`. Хороший вопрос, почему?

Внутри `elem()`, как мы знаем, именно этот образец `gen_elems()` мы хотели бы вызвать. В образце `Fibonacci::elem()` мы хотим вызвать `Fibonacci` образец `gen_elems()`. Откладывать вопрос с `gen_elems()` до момента выполнения этого вызова нет необходимости. В результате для нас предпочтительнее заменить виртуальный механизм и получить функцию, приготовленную при компиляции, а не при выполнении. Это то, почему сделан явный вызов `gen_elems()`. Определяя вызов виртуальной функции оператором границ класса, мы говорим компилятору, какой образец нужно вызвать. Виртуальный механизм при выполнении теперь переделан.

Вот реализация `gen_elems()` и `print()`:

```
void Fibonacci:: gen_elems(int pos) const{
    if (_elems.empty()) { _elems.push_back(1); _elems.push_back(1); }

    if (_elems.size() < pos) {

        int ix = _elems.size();
        int n_2 = _elems[ix-2];
        int n_1 = _elems[ix-1];

        for (; ix < pos; ++ix) {
            int elem = n_2 + n_1;
            _elems.push_back(elem);
            n_2 = n_1; n_1 = elem;  }}}

ostream& Fibonacci:: print(ostream &os) const{
    int elem_pos = _beg_pos-1;
    int end_pos = elem_pos + _length;

    if (end_pos > _elems.size()) Fibonacci::gen_elems(end_pos);

    while (elem_pos < end_pos)
        os << _elems[elem_pos++] << ' ';

    return os;}
```

Отметим, что обе `elem()` и `print()` проверяют, содержится ли соответствующий элемент в `_elems`, и, если нет, они вызывают `gen_elems()`. Как можем мы модифицировать `check_integrity()`, чтобы сделать ее проверку столь же удачной, что и `pos`? Одна возможность – поддержать класс `Fibonacci` функцией-членом `check_integrity()`:

```
class Fibonacci : public num_sequence {
public:
    // ...
protected:
    bool check_integrity(int pos) const;
    // ...};
```

Внутри класса `Fibonacci` каждая ссылка на `check_integrity()` теперь превращается в ссылку на образец функции производного класса. Внутри `elem()`, например, вызов `check_integrity()` теперь вызывает члена `Fibonacci`.

```
int Fibonacci:: elem(int pos) const{
    // теперь превратилось в образец Fibonacci
    if (! check_integrity(pos)) return 0; // ...}
```

Когда бы член производного класса ни повторял имени члена наследованного базового класса, член базового класса становится лексически скрыт внутри производного класса. Это происходит при каждом использовании имени внутри производного класса при превращении в член производного класса. Для доступа к члену базового класса, внутри производного, мы должны оценить его ссылку оператором границ базового класса. Например,

```
inline bool Fibonacci::check_integrity(int pos) const{
    // необходим оператор границ класса ...
    // неквалифицированное имя превращается в этот образец!
    if (! num_sequence::check_integrity(pos)) return false;

    if (pos > _elems.size()) Fibonacci::gen_elems(pos);

    return true;}

```

Проблема с этим решением в том, что внутри базового класса `check_integrity()` не идентифицируется, как виртуальная. Это означает, что каждый вызов `check_integrity()` через указатель базового класса или ссылку превращается в образец `num_sequence`. Нет основания для выбора, к какому реальному объекту производить адресацию. Например,

```
void Fibonacci::example(){

    num_sequence *ps = new Fibonacci(12, 8);

    // ok: превращаем в Fibonacci::elem() через виртуальный механизм
    ps->elem(1024);
    // шлеп: превратились в статическую num_sequence::check_integrity()
    // базирующуюся на типе ps
    ps->check_integrity(pos);}

```

Из этих соображений, - и даже не так, в основном, - хорошая практика, придерживаться одинаковых имен неvirtуальных членов-функций и в базовом, и в производном классе. Одним из следствий этого может быть то, что все функции внутри класса должны объявляться как виртуальные. Я не верю, что это корректный вывод, но он создает непосредственную дилемму для нашей разработки.

Скрытая причина дилеммы в том, что образец базового класса был реализован без адекватного понимания того, что производные классы потребуют проверки чистоты его состояния. Любая реализация, происходящая при недостатке знания, похоже, оказывается неполной. Но это отличается от претензий на то, что реализация типозависима, а потому должна быть виртуальной.

Вновь, я думаю, что наша разработка итеративна, и должна проходить проверку практикой и обратной связью с пользователями. В данном случае наилучшим конструктивным решением будет переопределение `check_integrity()`, чтобы она принимала два параметра:

```
bool num_sequence::check_integrity(int pos, int size){
    if (pos <= 0 || pos > max_seq){
        // как и раньше ... }

    if (pos > size)
        // gen_elems() вызывается через виртуальный механизм
        gen_elems(pos);

    return true;}

```

В этом определении `check_integrity()`, `gen_elems()` вызываются через виртуальный механизм. Если

`check_integrity()` вызывается объектом класса `Fibonacci`, вызывается `Fibonacci` образец `gen_elems()`. Если `check_integrity()` вызывается объектом класса `Triangular`, вызывается `Triangular` образец `gen_elems()`. Новый образец может быть вызван следующим образом:

```
int Fibonacci::elem(int pos){
    if (! check_integrity(pos, _elems.size())) return 0; // ...}
```

Всегда хорошая идея, сделать проверку реализации упреждающей, а не ждать пока весь базовый код будет введен полностью, чтобы посмотреть удержится ли заплатка. Не только это позволяет нам провести проверку готовности к продолжению работы, но и поддержка базиса для набора регрессионных тестов, которые мы можем запускать каждый раз, когда впоследствии развиваем проект. Вот небольшая тестовая программа для упражнений с нашей реализацией в дальнейшем. `gen_elems()` была приспособлена для вывода на дисплей элементов, которые она генерирует, в количестве большем, чем первые два:

```
int main(){

    Fibonacci fib;

    cout << "fib: beginning at element 1 for 1 element: "
         << fib << endl;

    Fibonacci fib2(16);
    cout << "fib2: beginning at element 1 for 16 elements: "
         << fib2 << endl;

    Fibonacci fib3(8, 12);

    cout << "fib3: beginning at element 12 for 8 elements: "
         << fib3 << endl;}
```

После компиляции, при выполнении получим следующее:

```
fib1: beginning at element 1 for 1 element: (1 , 1)
fib2: beginning at element 1 for 16 elements:
gen_elems: 2
gen_elems: 3
gen_elems: 5
gen_elems: 8
gen_elems: 13
gen_elems: 21
gen_elems: 34
gen_elems: 55
gen_elems: 89
gen_elems: 144
gen_elems: 233
gen_elems: 377
gen_elems: 610
gen_elems: 987
(1 , 16) 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
fib3: beginning at element 12 for 8 elements:
gen_elems: 1597
gen_elems: 2584
```

```
gen_elems: 4181
(12 , 8) 144 233 377 610 987 1597 2584 4181
```

## 5.6 Использование иерархии наследования

Давайте предположим, что мы определили пять других классов числовых последовательностей (Pell, Lucas, Square, Triangular и Pentagonal) таким же образом, как и класс Fibonacci. Теперь мы имеем двухуровневую иерархию наследования: абстрактный базовый класс `num_sequence` и шесть наследующих производных классов. Как мы можем использовать их?

Вот простая функция `display()`, чей второй параметр `ns`, а `const` ссылается на объект `num_sequence`.

```
inline void display(ostream &os, const num_sequence &ns, int pos){
    os << "The element at position "
      << pos << " for the "
      << ns.what_am_i() << " sequence is "
      << ns.elem(pos) << endl; }
```

Внутри `display()`, мы вызываем две виртуальные функции `what_am_i()` и `elem()`. Какой образец этих функций вызывается? Мы не можем сказать ничего определенного. Мы знаем, что `ns` не ссылается на реальный объект класса `num_sequence`, если только, на объект производного от `num_sequence` класса. Вызов двух виртуальных функций производится при выполнении программы, основываясь на типе класса, на который ссылается `ns`. Например, в следующей небольшой программе я определяю объект каждого производного класса по очереди и передаю в `display()`:

```
int main(){

    const int pos = 8;

    Fibonacci fib;
    display(cout, fib, pos);

    Pell pell;
    display(cout, pell, pos);

    Lucas lucas;
    display(cout, lucas, pos);

    Triangular trian;
    display(cout, trian, pos);

    Square square;
    display(cout, square, pos);

    Pentagonal penta;
    display(cout, penta, pos); }
```

После компиляции, при выполнении мы увидим:

```
The element at position 8 for the Fibonacci sequence is 21

The element at position 8 for the Pell sequence is 408
```

The element at position 8 for the Lucas sequence is 47

The element at position 8 for the Triangular sequence is 36

The element at position 8 for the Square sequence is 64

The element at position 8 for the Pentagonal sequence is 92

Обратите внимание, что эта программа повторяет вывод нашей ранней программы из [Раздела 4.10](#). Конструкция класса `num_sequence`, однако, изменилась существенно. Механическая установка, поддержание трассировки и сброс типа числовой последовательности в ранней разработке были удалены. Язык, осуществляет эту поддержку непосредственно через механизмы наследования и виртуальных функций. Этот объектно-ориентированный вариант также значительно упрощает и модификацию, и расширение разработки. В отличие от нашей ранней версии, добавление или удаление существующего класса числовой последовательности, не требует больших изменений.

Напомню, что в нашей разработке базового класса `num_sequence` мы также осуществляем перезагрузку образца оператора вывода:

```
ostream& operator<<(ostream &os, const num_sequence &ns){  
  
    return ns.print(os); }
```

Поскольку `print()` – виртуальная функция, оператор вывода работает с каждым образцом производного класса. Например, вот небольшая программа, в которой я определяю объект каждой числовой последовательности, и затем направляю каждую по очереди в оператор вывода:

```
int main() {  
    Fibonacci fib(8);  
    Pell pell(6, 4);  
    Lucas lucas(10, 7);  
    Triangular trian(12);  
    Square square(6, 6);  
    Pentagonal penta(8);  
    cout << "fib: " << fib << '\n'  
        << "pell: " << pell << '\n'  
        << "lucas: " << lucas << '\n'  
        << "trian: " << trian << '\n'  
        << "square: " << square << '\n'  
        << "penta: " << penta << endl;};
```

После компиляции при выполнении программа выведет следующее:

```
fib: (1 , 8) 1 1 2 3 5 8 13 21  
pell: (4 , 6) 12 29 70 169 408 985  
lucas: (7 , 10) 29 47 76 123 199 322 521 843 1364 2207  
trian: (1 , 12) 1 3 6 10 15 21 28 36 45 55 66 78  
square: (6 , 6) 36 49 64 81 100 121  
penta: (1 , 8) 1 5 12 22 35 51 70 92
```

## 5.7 Насколько абстрактным должен быть базовый класс?

В нашей текущей разработке абстрактный базовый класс поддерживает интерфейс, но не поддерживает реализацию. Каждый производный класс должен не только поддерживать уникальный алгоритм для генерации его элементов, но так же осуществлять поддержку поиска элемента, вывода элемента, определение длины и начальной позиции объекта последовательности и т.д. Это что, плохая разработка?

Если разработчик абстрактного базового класса также поддерживает производные классы числовых последовательностей, и если они не рассчитаны на очень частые добавления, тогда эта конструкция работает достаточно хорошо. Если, однако, основная активность направлена на порождение новых классов числовых последовательностей, и если эта активность была возвращена индивидуализмом того, кому уютнее в математике, чем в программировании, эта разработка усложняет порождение каждого производного класса.

Следующая альтернативная разработка базового класса осуществляет реализацию поддержки выделенных данных производного класса в базовом классе. Интерфейс не меняется. Программу следующих разделов нет нужды менять, хотя перекомпиляция потребует. Конструкция упростит работу, необходимую для поддержки производных классов.

Вот определение переделанного определения базового класса `num_sequence`. Два члена-данных - `_length` и `_beg_pos` - теперь члены-данные `num_sequence`. Мы объявили их, как `protected`, чтобы разрешить к ним прямой доступ производных классов. Поддерживающие доступ члены-функции `length()` и `beg_pos()` теперь тоже члены класса `num_sequence`. Мы объявили их как `public`, чтобы разрешить основной программе доступ на чтение к их значениям.

Новые данные-члены также были добавлены в класс `num_sequence`. `_relems` - ссылка на вектор целых, ссылается на статический вектор производного класса. Почему объявлена ссылка, а не указатель? Как мы говорили в [Разделе 2.3](#), ссылка может никогда не ссылаться на нулевой объект, тогда как указатель может быть, а может и не быть нулевым. Тем, что выбрана ссылка, мы избавили себя от проверки указателя на нуль.

Ссылка на данные-члены должна быть инициализирована внутри списка инициализации членов конструктора, и, однажды инициализированная, может никогда не менять ссылки на другой объект. Указатель на данные-члены имеет некоторые ограничения: мы можем либо инициализировать его внутри конструктора, либо инициализировать его нулем и присвоить ему правильный адрес позже. Мы выбираем между ссылкой на данные-члены и указателем, основываясь на этих соображениях.

Базовый класс теперь имеет всю информацию, необходимую для поиска и вывода на дисплей элементов числовой последовательности. Это также позволяет нам переопределить `elem()` и `print()`, как общие члены `num_sequence`.

```
class num_sequence {
public:
    virtual ~num_sequence() {}
    virtual const char* what_am_i() const = 0;
    int elem(int pos) const;
    ostream& print(ostream &os = cout) const;
    int length() const { return _length; }
    int beg_pos() const { return _beg_pos; }
    static int max_elems() { return 64; }
protected:
    virtual void gen_elems(int pos) const = 0;
    bool check_integrity(int pos, int size) const;
    num_sequence(int len, int bp, vector<int> &re)
```

```

: _length(len), _beg_pos(bp), _relems(re){}
int _length;
int _beg_pos;
vector<int> & _relems; };

```

Каждый производный класс числовой последовательности теперь должен программироваться только в том, что специфично для него: `gen_elems()`, которая вычисляет элементы последовательности; `what_am_i()`, которая идентифицирует последовательность; статический вектор для содержания элементов последовательности и конструктор. Производный класс наследует члены для поиска элементов, вывода элементов, и определения длины и начальной позиции. Например, вот наша ревизия определения класса `Fibonacci`:

```

class Fibonacci : public num_sequence {
public:
    Fibonacci(int len = 1, int beg_pos = 1);
    virtual const char* what_am_i() const { return "Fibonacci"; }

protected:
    virtual void gen_elems(int pos) const;
    static vector<int> _elems; };

```

Хотя `num_sequence` остается абстрактным базовым классом, он теперь поддерживает долю реализации, которая наследуется каждым производным классом.

## 5.8 Инициализация, деструкция и копирование

Теперь, когда `num_sequence` объявляет реальные данные-члены, нужно поддержать их инициализацию. Мы должны оставить инициализацию данных-членов для каждого производного класса, но это питательная среда для ошибок. Лучшей конструкцией было бы переложить на конструктор базового класса задачу инициализации всех членов базового класса.

Хочу напомнить, что `num_sequence` – это абстрактный базовый класс. Мы не должны определять независимые объекты его класса; вернее, `num_sequence` обслуживает, как суб-объекты, объекты каждого производного класса. По этой причине мы объявляем конструктор базового класса `protected`, а не `public` членом.

Инициализация объектов производного класса состоит из вызова конструктора базового класса, сопровождаемого соответствующим конструктором производного класса. Полезно думать об объекте производного класса, как состоящем из множества суб-объектов: суб-объект базового класса, инициализированный конструктором базового класса, и суб-объект производного класса, инициализированный конструктором производного класса. В трехуровневой иерархии классов, такой как класс `AudioBook` [Раздела 5.1](#), производный класс состоит из трех суб-объектов, каждый из которых инициализируется своим соответствующим конструктором.

Требования разработки к конструктору производного класса двойкие: нужно не только инициализировать данные-члены производного класса, но нужно также предоставить ожидаемые значения для конструктора базового класса. В нашем примере базовый класс `num_sequence` требует трех значений, которые передаются ему для использования в списке инициализации членов. Например,

```

inline Fibonacci::Fibonacci(int len, int beg_pos): num_sequence(len, beg_pos,
    &_elems) {}

```

Если мы посмотрим вызовы конструктора `num_sequence`, определение конструктора `Fibonacci` покажется ошибочным. Почему? Базовый класс `num_sequence` требует недвусмысленного вызова его трех-аргументного конструктора. В нашей разработке, это именно то, что мы хотим.

Альтернативно, мы должны поддержать определенный конструктор `num_sequence`. Мы должны изменить `_relems` на указатель и добавить код для проверки, что он не нулевой, перед каждым обращением к вектору:

```
num_sequence::num_sequence(int len=1, int bp=1, vector<int> *pe=0)
: _length(len), _beg_pos(bp), _pelems(re){}
```

Теперь, если конструктор производного класса неявно вызывает конструктор базового класса, конструктор по умолчанию базового класса вызывается автоматически.

Что происходит, когда мы инициализируем один объект класса `Fibonacci` другим?

```
Fibonacci fib1(12);

Fibonacci fib2 = fib1;
```

Если определен явный конструктор копирования, этот образец вызывается. Например, мы можем определить конструктор копирования `Fibonacci` следующим образом:

```
Fibonacci::Fibonacci(const Fibonacci &rhs) : num_sequence(rhs){}
```

`rhs`, правосторонний объект производного класса передается в конструктор копирования базового класса, используя список инициализации членов. Что если базовый класс не определяет явно конструктор копирования? Ничего плохого не произойдет. Выполнится инициализация по умолчанию образа члена. Если явно определен конструктор копирования базового класса, вызывается он.

В этом случае явный конструктор копирования `Fibonacci` не нужен, поскольку выполнение по умолчанию дает такой же результат: первый суб-объект базового класса – инициализированный образ члена, сопровождаемый инициализацией образа из производного класса членов.

То же справедливо по отношению к оператору присваивания копии. Если мы присваиваем один объект класса `Fibonacci` другому, и если есть явное определение оператора присваивания копии, оно вызывается для выполнения присваивания. Например, вот как мы можем определить оператор. Только “трюковая” часть служит для явного вызова оператора базового класса.

```
Fibonacci& Fibonacci::operator=(const Fibonacci &rhs){ if (this != &rhs)
// явный вызов оператора базового класса
num_sequence::operator=(rhs);

return *this;}
```

Вновь, в этом случае, явный `Fibonacci` оператор присваивания копии не нужен, поскольку выполнение по умолчанию дает тот же результат. (Отсылаю вас к [Разделу 4.2](#) и [Разделу 4.8](#), где обсуждается, когда конструктор копирования и оператор присваивания копии обязательны).

Деструктор базового класса вызывается автоматически, следуя за вызовом деструктора производного класса. Нам не нужно вызывать его явным образом внутри деструктора производного класса.

## 5.9 Определение виртуальной функции производного класса

Когда мы определяем производный класс, мы должны решить, будем или нет наследовать каждую виртуальную функцию базового класса. Если мы наследуем чисто виртуальную функцию, тогда производный класс соотносится с абстрактным классом, и никакие независимые объекты класса не могут быть определены.

Если мы отвергаем образец базового класса, прототип образца производного класса должен сочетаться с прототипом базового класса явно: список параметров и тип возвращаемого значения, будет ли виртуальная функция `const` или `не-const`. Например, следующее определение `Fibonacci::what_am_i()` не слишком хорошо:

```
class Fibonacci : public num_sequence {
public:
    virtual const char* what_am_i()
    // не слишком удачно ...
    { return "Fibonacci"; } // ... };
```

Хотя `Fibonacci` образец `what_am_i()` не слишком хорош, он отнюдь не неправилен, но вот, где появляются неудобства. Когда я компилирую Интеловским C++ компилятором, то получаю следующее предупреждение:

```
warning #653: "const char *Fibonacci::what_am_i()"
does not match "num_sequence::what_am_i"
-- virtual function override intended?

//"const char *Fibonacci::what_am_i()"
// не связано с "num_sequence::what_am_i"
// -- намерены отказаться от виртуальной функции?
```

О чем нам это говорит? Это говорит нам, что объявление `what_am_i()` производного класса явно не связано с объявлением базового класса. Образец базового класса объявлен, как `const` член-функция. Образец производного класса - `не-const` член-функция. Имеет ли это непреложное значение? К сожалению, да. И вот простая иллюстрация:

```
class num_sequence {
public:
    virtual const char* what_am_i() const{ return "num_sequence\n"; };
class Fibonacci : public num_sequence {
public:
    virtual const char *what_am_i(){ return "Fibonacci\n"; };
int main() {
    Fibonacci b;
    num_sequence p;
```

```

// рассчитываем, что это генерирует: Fibonacci
num_sequence *pp = &b;
cout << pp->what_am_i();

cout << b.what_am_i();

return 0;}

```

Ожидаемый вывод этой программы – две строки последовательности `Fibonacci`. После компиляции, при выполнении программа, однако, выдает неожиданное:

```
num_sequence Fibonacci
```

Предупреждение сообщало нам, что образец производного класса не обрабатывается, как отвергнутый образец базового класса, поскольку эти два образца не связаны явно. Это общая ошибка всех начинающих, и обычно сбивает с толку. Именно поэтому я столько времени посвятил ее разъяснению. Исправить просто, тогда, как понять, почему необходимо исправление, отнюдь не столь просто.

Вот второе неправильное переобъявление `what_am_i()`. В этом случае наш возвращаемый тип не согласован явно с возвращаемым типом образца `num_sequence` :

```

class Fibonacci : public num_sequence {
public:// неправильное объявление!
// образец базового класса возвращает const char*, не char*virtual
char* what_am_i(){ return "Fibonacci"; } // ... };

```

Образец `what_am_i()` из `num_sequence` возвращает `const char*`. Образец `Fibonacci` возвращает `char*`. Это простая ошибка, которая проявляется при компиляции.

Вот одно исключение из правила явно возвращаемого типа: если виртуальная функция базового класса возвращает некоторый тип базового класса, как правило, указатель или ссылку,

```

class num_sequence {
public:
// образец производного класса clone() может вернуть
// указатель на любой класс, производный от num_sequence
virtual num_sequence *clone()=0; // ... };

```

образец производного класса может вернуть тип, производный от возвращаемого типа базового класса:

```

class Fibonacci : public num_sequence {
public:
// ok: Fibonacci производный от num_sequence
// ключевое слово virtual в производном классе не обязательно
Fibonacci *clone(){ return new Fibonacci(*this); } // ... };

```

Когда мы отказываемся от виртуальной функции базового класса в производном классе, ключевое слово `virtual` не обязательно. Функция идентифицируется как отвергнутый образец базового класса, основанный на сравнении двух прототипов функции.

## Статическая резолуция виртуальной функции

Есть два обстоятельства, из-за которых виртуальный механизм может не работать ожидаемым образом: (1)

внутри конструктора и деструктора базового класса и (2) когда мы используем объект базового класса вместо указателя или ссылки на базовый класс.

При создании объекта производного класса, первым вызывается конструктор базового класса. Что произойдет, если внутри конструктора базового класса вызывается виртуальная функция? Будет ли вызываться образец производного класса?

Проблема в том, что данные-члены производного класса еще не инициализированы. Если вызывается образец виртуальной функции производного класса, она, похоже, имеет доступ к данным-членам, которые не инициализированы, а это совсем не здорово.

Из этих соображений внутри конструктора базового класса виртуальная функция производного класса не вызывается никогда!

Внутри конструктора `num_sequence`, например, вызов `what_am_i()` разрешает появление образца `num_sequence`, даже если определен объект класса `Fibonacci`. Все то же справедливо для вызова виртуальной функции внутри деструктора базового класса.

Рассмотрим следующий программный фрагмент, который использует `LibMat` и `AudioBook` классы из [Раздела 5.1](#). Напомню, что `print()` - виртуальная функция этой иерархии класса.

```
void print(LibMat object, const LibMat *pointer, const LibMat &reference) {
    // всегда вызывается LibMat::print()
    object.print();

    // всегда реализуется через виртуальный механизм
    // мы не знаем, какой образец print() это вызовет
    pointer->print();
    reference.print(); }
```

Полиморфизм требует не направленности в порядке возможности представления множества типов внутри единственного объекта. В C++ только указатели и ссылки базового класса поддерживают объектно-ориентированное программирование.

Когда мы объявляем реальный объект базового класса, такой как первый параметр `print()`, мы отводим достаточно памяти для представления реального объекта базового класса. Если мы позже осуществляем передачу в объект производного класса, просто не хватает памяти для сохранения дополнительных данных-членов производного класса. Например, когда мы передаем объект `AudioBook` в `print()`, как в следующем,

```
int main() {
    AudioBook iWish("Her Pride of 10", "Stanley Lippman", "Jeremy Irons");
    gen_elems(iWish, &iWish, iWish);
    // ...}
```

только часть базового `LibMat` класса `iWish` может быть скопирована в память, зарезервированную для `object`. Суб-объекты `Book` и `AudioBook` *вырезаются*. `pointer` и `reference` инициализируются просто адресами оригинального объекта `iWish`. Это то, почему они могут адресоваться к полному объекту `AudioBook`. (Более полное обсуждение вы найдете в Разделе 1.3 [LIPPMAN96a]).

## 5.10 Идентификация типа при выполнении

Наша реализация `what_am_i()` имеет поддержку виртуального образца каждого класса, который возвращает строку, идентифицирующую класс:

```
class Fibonacci : public num_sequence {
public:
    virtual const char* what_am_i() const { return "Fibonacci"; }
    // ...};
```

Альтернативная конструкция поддерживает единственный образец `what_am_i()` из `num_sequence`, переиспользуемый каждым производным классом через наследование. Эта конструкция освобождает каждый производный класс от необходимости поддерживать свой собственный образец `what_am_i()`.

Один из путей реализации этого может быть в добавлении строкового члена в `num_sequence`. Каждый конструктор производного класса будет передавать теперь свое имя класса, как аргумент, в конструктор `num_sequence`. Например,

```
inline Fibonacci::Fibonacci(int len, int beg_pos)
: num_sequence(len, beg_pos, &elems, "Fibonacci"){}
```

Альтернативная реализация в использовании оператора `typeid`. Оператор `typeid` – часть идентификации типа при выполнении (RTTI) поддержки языка. Она позволяет нам запрашивать указатель или ссылку реального типа объекта полиморфного класса, на который ссылается.

```
#include <typeinfo>

inline const char* num_sequence::what_am_i() const {

    return typeid(*this).name(); }
```

Для использования оператора `typeid` мы должны включить файл заголовка `typeinfo`. Оператор `typeid` возвращает объект класса `type_info`. Этот объект запоминает информацию о типе. Есть один объект, ассоциированный с каждым полиморфным классом, таким как производные классы `Fibonacci` и `Pell`. `Name()` возвращает `const char*` представление имени класса. Выражение

```
typeid(*this)
```

возвращает объект класса `type_info`, ассоциированный с типом реального класса, адресуемый указателем `this` внутри `what_am_i()`. Член-функция `name()` из `type_info` вызывается через этот объект, возвращая имя типа реального класса.

Класс `type_info` также поддерживает сравнения: эквивалентности и неэквивалентности. Например, следующий код определяет, будет ли `ps` адресоваться к объекту класса `Fibonacci`:

```
num_sequence *ps = &fib;
// ...
if (typeid(*ps) == typeid(Fibonacci))
// ok, ps адресуется к объекту класса Fibonacci
```

Если мы пишем

```
ps->gen_elems(64);
```

мы знаем, что будет вызван образец `gen_elems()` `Fibonacci`. Однако хотя мы знаем из этого теста, что `ps` адресуется к объекту класса `Fibonacci`, попытка вызова `gen_elems()` образца `Fibonacci` непосредственно через `ps` приведет к ошибке при компиляции:

```
// error: ps не указатель на Fibonacci
// хотя мы знаем, что он сейчас адресуется
// к объекту класса Fibonacci
ps->Fibonacci::gen_elems(64);
```

`ps` не “знает” тип объекта, к которому адресуется, даже если мы, и `typeid`, и механизм виртуальной функции – все это знаем.

Для вызова `Fibonacci` образца `gen_elems()` мы должны проинструктировать компилятор, преобразовать `ps` в указатель типа `Fibonacci`. Оператор `static_cast` производит безусловную конверсию.

```
if (typeid(*ps) == typeid(Fibonacci)) {
    Fibonacci *pf = static_cast<Fibonacci*>(ps);
    pf->gen_elems(64); }
```

`static_cast` потенциально опасно, поскольку компилятор не проверяет корректности нашего преобразования. Это та причина, по которой я отложил его использование при проверке оператора `typeid`. Условное преобразование поддерживается оператором `dynamic_cast`:

```
if (Fibonacci *pf = dynamic_cast<Fibonacci*>(ps))
    pf->gen_elems(64);
```

Оператор `dynamic_cast` – это другой оператор RTTI. Он производит проверку при выполнении программы, является ли объект, адресуемый `ps`, реальным типом класса `Fibonacci`. Если да, преобразование выполняется. `pf` теперь адресуется к объекту `Fibonacci`. Если нет, оператор `dynamic_cast` возвращает 0. Выражение условия `if` не выполняется и статический вызов `Fibonacci` образца `gen_elems()` не выполняется.

Более детально C++ механизм идентификации типа при выполнении рассматривается в Разделе 19.1 [LIPPMAN98].

## Упражнение 5.1

Реализуйте двухуровневую иерархию стека. Базовый класс – чисто абстрактный стековый класс, который минимально поддерживает следующий интерфейс: `pop()`, `push()`, `size()`, `empty()`, `full()`, `peek()` и `print()`. Два конкретных производных класса – это `LIFO_Stack` и `Peekback_Stack`. `Peekback_Stack` позволяет пользователю запросить значение любого элемента стека без модификации самого стека.

## Упражнение 5.2

Переделайте иерархию класса из [Упражнения 5.1](#) так, чтобы базовый класс `Stack` реализовывал разделяемые, не зависящие от типа, члены.

## Упражнение 5.3

Тип/суб-тип наследование отношений в основном отображается “как есть”-отношением (*is-a* relationship): ArrayRC с проверкой размера - разновидность Array, Book – разновидность LibraryRentalMaterial, AudioBook – разновидность Book, и т.д. Какая из следующих пар отображается, “как есть”- отношением?

- (a) member function isA\_kindOf function
- (b) member function isA\_kindOf class
- (c) constructor isA\_kindOf member function
- (d) airplane isA\_kindOf vehicle
- (e) motor isA\_kindOf truck
- (f) circle isA\_kindOf geometry
- (g) square isA\_kindOf rectangle
- (h) automobile isA\_kindOf airplane
- (i) borrower isA\_kindOf library

## Упражнение 5.4

Библиотека поддерживает следующие категории сдаваемых материалов, каждый с его собственной политикой выдачи и приемки. Организуйте это в иерархию наследования:

```
book audio book record children's puppetvideo Sega  
video gamental book Sony Playstation video gameCD-  
ROM book Nintendo video game
```

## Часть 6. Программирование с шаблонами

Когда Bjarne Stroustrup (Бьярн Страуструп) работал над конструкцией шаблонов оригинального языка C++, он ссылался на них, как на *параметризованные типы*. “Параметризованные”, поскольку они производились вне определения шаблона, а “типы”, поскольку каждый класс или функция шаблона, как правило, варьируется одним или множеством типов через операции или содержимое. Реальный их тип позже специфицируется пользователем.

Впоследствии Stroustrup заменил название более общим – *шаблон (template)*. Единственное определение шаблона работает, как предписание для автоматической генерации уникальных образцов функций или классов, основанных на заданных пользователем значениях или типах.

Хотя мы уже широко использовали классы шаблонов, такие как векторный или строковый класс, мы совершенно не занимались разработкой своего собственного. Сделаем это сейчас, разрабатывая шаблон класса бинарных деревьев.

Если вы не накоротке с понятием бинарного дерева, вот небольшой обзор для вас. Дерево состоит из узлов и ветвей, или связей, соединяющих узлы. Бинарное дерево состоит из двух связей между узлами, обычно называемых *левая* и *правая ветка (child)*. Дерево имеет первый узел, называемый *корнем (root)*. Каждая левая и правая ветка может сама стать корнем для суб-дерева. Узел без ветвей называется *листом (leaf)*.

Наша реализация бинарного дерева состоит из двух классов: класса `BinaryTree`, который содержит указатель на корень, и вспомогательного класса `BTnode`, который содержит и реальное значение, и левый и правый побег. Этот тот тип узла, который мы параметризуем.

Какие операции наш `BinaryTree` будет поддерживать? Пользователи должны иметь возможность *вставить* и *удалить* элемент, как и *найти*, если элемент присутствует, *очистить* все элементы дерева, и *вывести (print)* дерево в одном из трех алгоритмов следования: *в порядке (inorder)*, *предпорядке (preorder)* или *постпорядке (preorder)*.

В нашей реализации первое значение вставляется в пустое дерево, которое становится корнем. Каждое последующее значение вставляется так, что все значения меньше, чем корень, располагаются в левом суб-дереве корня. Все значения, большие чем корень, располагаются в правом суб-дереве. Значения повторяются только единожды в дереве. Счетчик обнаружения сохраняет след множества вставок того же значения. Например, дадим последовательность кодов

```
BinaryTree<string> bt;  
  
bt.insert("Piglet");
```

`Piglet` становится корнем нашего двоичного дерева. Посмотрите, как мы далее вставим `Eeyore`:

```
bt.insert("Eeyore");
```

Поскольку `Eeyore` меньше (по алфавиту), чем `Piglet`, `Eeyore` становится левой веткой `Piglet`. Посмотрите на нашу следующую вставку `Roo`:

```
bt.insert("Roo");
```

`Roo` больше `Piglet`, и `Roo` становится правой веткой, и т.д. Давайте закончим наше дерево вставкой

следующих элементов:

```
bt.insert("Tigger");bt.insert("Chris");bt.insert("Pooh");bt.insert("Kanga");
```

Результирующее бинарное дерево изображено на рисунке 6.1. В этом примере Chris, Kanga, Pooh и Tigger – листовые узлы.

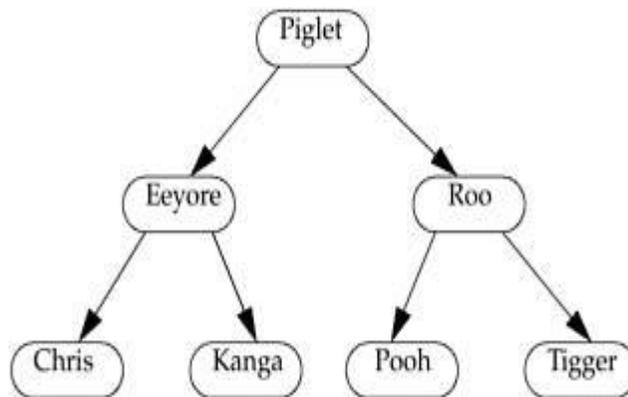


Рисунок 6.1

Каждый алгоритм следования начинается с корневого узла. В следовании *предпорядка* отображается узел, затем посещается левая ветка, затем правая. В следовании *в порядке* вначале посещается левая ветка, затем отображается узел, затем посещается правая ветка. В *постпорядке* посещается левая ветка, затем правая, затем отображается узел. Для рисунка 6.1 три алгоритма следования отображают узлы следующим образом:

```
// следование предпорядком на рисунке 6.1 двоичного дерева  
Piglet, Eeyore, Chris, Kanga, Roo, Pooh, Tigger  
  
// следование в порядке на рисунке 6.1 двоичного дерева  
Chris, Eeyore, Kanga, Piglet, Pooh, Roo, Tigger  
  
// следование постпорядка на рисунке 6.1 двоичного дерева  
Chris, Kanga, Eeyore, Pooh, Tigger, Roo, Piglet
```

## 6.1 Параметризованные типы

Вот нешаблонное объявление класса `BTnode`, в котором сохраняемые значения – объекты строкового класса. Я назвал их `string_BTnode`, поскольку мы должны определить другие образцы типов `int`, `double` и т.д.

```
class string_BTnode {  
public:  
    // ...  
private:  
    string _val;  
    int _cnt;
```

```
int_BTnode *_lchild;
int_BTnode *_rchild; };
```

Без механизма шаблонов каждый тип нуждается в своей собственной отдельной реализации класса `Btnode`, и каждый нуждается в уникальном имени.

Механизм шаблонов позволяет нам разделить типо-зависимые и инвариантные части нашего определения класса. Код следования по дереву, вставки и удаления узлов, и поддержания соответствующего счетчика одинаков, благодаря типу значения. Этот код переиспользуется с каждым образцом шаблона класса. Тип значения сохраняется внутри каждого узла, изменяясь с каждым образцом шаблона класса, в одном случае представляя строковый объект, в другом `int` или `double` и т.д. В шаблоне класса зависимость типа превращается в один или более параметров. В нашем классе `Btnode` тип данных-членов `_val` параметризован:

```
// предварительное объявление шаблона класса Btnode
template <typename valType>
class Btnode;
```

`valType` используется внутри определения шаблона класса, как “заглушка”. Мы можем дать ей любое имя, какое захотим. Она используется в качестве пустой карточки, пока мы не специализировали какой-либо реальный тип. Параметр типа может использоваться где угодно в отличие от реального типа, как `int` или `string`. В классе `Btnode` он используется для объявления типа `_val`:

```
template < typename valType >
class Btnode {
public:
    // ...
private:
    valType _val;
    int _cnt;
    Btnode *_lchild;
    Btnode *_rchild; };
```

Шаблон класса `Btnode` сотрудничает с шаблоном класса `BinaryTree` в реализации нашей абстракции. Класс `Btnode` содержит реальное значение, соответствующий счетчик, и указатели на левую и правую ветки. Напомню, что отношения класса сотрудничества обычно требуют дружбы. Для каждого реального типа класса `Btnode` мы хотим, чтобы соответствующий образец `BinaryTree` был его другом. Вот, как мы объявляем это:

```
template <typename Type>
class BinaryTree;

// предварительное объявление

template <typename valType>
class Btnode {
    friend class BinaryTree<valType>;
    // ... };
```

Чтобы создать образец шаблона класса, мы сопровождаем его именем шаблона класса с реальным типом, заключенным в угловые скобки, которым мы хотим заменить `valType`. Например, для придания `valType` типа `int` мы пишем

```
BTnode< int > bti;
```

Аналогично для придания `valType` типа класса `string` мы пишем

```
BTnode< string > bts;
```

`bti` и `bts` представляют два определения `BTnode`: в одном из которых `_val` определено как `int`, в другом, как `string`. Соответствующий образец строки `BinaryTree` – друг для образца строки `BTnode`, но не для образца `int BTnode`.

Класс `BinaryTree` объявляет один член-данных и `BTnode` указатель на корневой узел дерева:

```
template <typename elemType>
class BinaryTree {
public:
    // ...
private:

    // BTnode должен быть квалифицирован со своим

    //шаблонным списком параметров

    BTnode<elemType> *_root; };
```

Как нам узнать, когда квалифицировать имя шаблона класса его списком параметров? Основное правило – внутри определения шаблона класса и его членов имя шаблона класса квалифицировать нет нужды. Имя шаблона класса должно, с другой стороны, быть квалифицировано со своим списком параметров.

Когда мы специфицируем реальный тип для параметра `BinaryTree`, как

```
BinaryTree< string > st;
```

`_root` становится указателем на объект `BTnode`, содержащий значение строкового типа. Аналогично, когда мы специфицируем `int` тип,

```
BinaryTree< int > it;
```

`_root` становится указателем на объект `BTnode`, содержащий тип `int`.

## 6.2 Определение шаблона класса

Вот часть определения нашего шаблона класса `BinaryTree`:

```
template <typename elemType>
class BinaryTree {
public:
    BinaryTree();
    BinaryTree(const BinaryTree&);
    ~BinaryTree();
    BinaryTree& operator=(const BinaryTree&);
```

```

    bool empty() {
        return _root == 0; }void clear();
private:
    BTreeNode<elemType> *_root;
    // копирует суб-дерево, адресованное src в tar
    void copy(BTreeNode<elemType>*tar, BTreeNode<elemType>*src); };

```

Определение `inline` члена-функции для шаблона класса такое же, как и для не классового шаблона, что проиллюстрировано в определении `empty()`. Синтаксис определения члена-функции шаблона класса вне тела класса, однако, отличается очень сильно, по крайней мере, на первый взгляд:

```

template <typename elemType>
inline BinaryTree<elemType>::BinaryTree() : _root(0) {}

```

Определение функции-члена начинается с ключевого слова `template` и списка параметров класса. За этим следует определение функции вместе с квалификатором `inline` и оператором границ класса. Квалификатор `inline` должен идти после ключевого слова `template` и списка параметров.

Почему второе применение имени `BinaryTree` не квалифицировано? После оператора границ класса, как видно,

```
BinaryTree< elemType >::
```

все следующее за этим обрабатывается, как встретившееся внутри класса определение. Когда мы пишем

```
BinaryTree< elemType >::
```

```
// вне определения класса
```

```
BinaryTree() // внутри определения класса
```

второе появление `BinaryTree` рассматривается внутри определения класса, и, таким образом, не нуждается в квалификаторе. Например, вот определение конструктора копирования, оператора присваивания копии и деструктора:

```

template <typename elemType>
inline BinaryTree<elemType>::BinaryTree(const BinaryTree &rhs){
    copy(_root, rhs._root); }
template <typename elemType>
inline BinaryTree<elemType>::~BinaryTree(){ clear(); }
template <typename elemType>
inline BinaryTree<elemType>&BinaryTree<elemType>::
operator=(const BinaryTree &rhs){
    if (this != &rhs){ clear(); copy(_root, rhs._root); }

```

```
return *this; }
```

Вы, возможно, не поверите мне, но после того, как вы напишите соответствующее количество определений такого рода, они начнут выглядеть вполне натурально.

## 6.3 Поддержка параметра типа шаблона

Поддержка параметра типа шаблона нечто более сложное, чем поддержка явного параметра типа. Например, для явного объявления `int` параметра для функции мы пишем

```
bool find(int val);
```

передавая его по значению. Однако для объявления явного параметра класса `Matrix` для функции мы вместо этого пишем

```
bool find(const Matrix &val);
```

передавая его по ссылке с тем, чтобы не генерировать ненужную копию объекта класса `Matrix`. Наша программа остается верна, если мы объявим `find()`, как показано:

```
// не неверно, но неэффективно
```

```
bool find(Matrix val);
```

Это, просто, займет больше времени для достижения того же результата, и, похоже, подвергнется критике читателей нашего кода, исключительно, если `find()` часто вызывается из программы.

Когда мы манипулируем параметром типа шаблона, мы не должны реально знать, будет ли реальный тип, примененный пользователем, встроенным типом:

```
BinaryTree<int> bti;
```

В этом случае передача по значению списка параметров `find()` предпочтительнее. Если ее тип класса

```
BinaryTree<Matrix> btm;
```

передается по ссылке, список параметров `find()` предпочтительнее.

На практике оба и встроенные, и типы класса, похоже, специфицируются, как реальные типы для шаблона класса. Рекомендуемая стратегия программирования - обрабатывать параметр типа шаблона, *как если бы* он был типом класса. Для параметра функции, например, это означает, что мы объявляем его как `const` ссылку вместо передачи его по значению.

Внутри определения конструктора мы инициализируем каждый параметр типа внутри инициализационного списка членов

```
// предпочтительный метод инициализации для
// параметра типа, передаваемый в конструктор
template <typename valueType>
inline BTreeNode<valType>::BTreeNode(const valueType &val)
```

```
// только для случая, когда valType тип класса
: _val(val){ _cnt = 1;
_lchild = _rchild = 0;}
```

вместо того, чтобы делать это внутри тела конструктора:

```
template <typename valType>
inline BTreeNode<valType>::BTreeNode(const valType &val){
    // не рекомендуется; должен быть тип класса
    _val = val;

    // ok: эти типы инвариантны ...
    _cnt = 1;
    _lchild = _rchild = 0;}
```

Это гарантирует оптимальное выполнение, если пользователь специфицирует тип класса как реальный тип valType. Например, если я пишу

```
BTreeNode<int> btni(42);
```

нет разницы в исполнении между двумя формами. Но, если я пишу

```
BTreeNode<Matrix> btnm(transform_matrix);
```

разница в выполнении появляется. Присваивание \_val в теле конструктора требует двух шагов: (1) конструктор Matrix по умолчанию применяется к \_val перед выполнением тела конструктора и (2) оператор присваивания копии \_val с val применяется внутри тела. Инициализация \_val внутри списка инициализации членов конструктора требует только одного шага: копирования конструкции \_val с val.

Вновь, это не то, чтобы наша программа была не верна в передаче valType по значению, либо в присваивании valType данных-членов внутри тела конструктора. Но они выполняются много дольше и считаются признаком неопытности программиста в C++.

В этом месте изучения C++ вы не должны слишком сосредотачиваться на эффективности. Однако полезно было рассмотреть эти два случая, поскольку эти ошибки очень характерны для начинающих, а сказанного достаточно для их исправления. Достаточно.

## 6.4 Реализация класса шаблона

Каждый раз, когда мы вставляем новое значение, мы должны открывать объект BTreeNode, инициализировать его, и соединять где-нибудь внутри дерева. Мы обслуживаем локализацию и делокализацию каждого узла, явно используя выражения new и delete.

insert(), например, локализует новый BTreeNode в свободной программной области, если \_root не установлен, иначе она вызывает BTreeNode метод insert\_value() для вставки нового значения в дерево:

```
template <typename elemType>
inline void BinaryTreeNode<elemType>::insert(const elemType &elem){
    if (! _root)_root = new BTreeNode<elemType>(elem);

    else _root->insert_value(elem); }
```

Необходимы два шага для выполнения выражения `new`. (1) Запрашивается память из свободной программной области. Если соответствующая память доступна, возвращается указатель на объект. (Если соответствующая память не доступна, возникает исключение `bad_alloc`. Мы обсудим возможную поддержку исключений C++ в Части 7). (2) Если первый шаг успешен, и начальное значение специфицировано, объект инициализируется. Для типа класса, как в

```
_root = new BTreeNode<elemType>(elem);
```

`elem` передается конструктору узла `BTreeNode`. Если шаги по локализации прерываются, инициализации не происходит.

`insert_value()` вызывается, только если корневой узел присутствует. Левое корневое суб-дерево хранит значения, которые меньше, чем значение корня, а правое суб-дерево хранит большие значения.

`insert_value()` рекурсивно вызывает себя либо через левую, либо через правую ветвь пока найдет и отсоединит ветку, и присоединит себя, или найдет уже введенное значение. Только один образец каждого значения сохраняется в дереве. `BTreeNode _cnt`, данные-член, хранит счетчик вставок. Вот реализация:

```
template <typename valType>
void BTreeNode<valType>::insert_value(const valType &val) {
    if (val == _val) { _cnt++; return; }

    if (val < _val) {
        if (! _lchild) _lchild = new BTreeNode(val);

        else _lchild->insert_value(val); }
    else {
        if (! _rchild) _rchild = new BTreeNode(val);

        else _rchild->insert_value(val); }}
```

При необходимости сохранения порядка в дереве выполняется удаление значения. Основной алгоритм – замена узла его правой веткой. Левая затем перекрепляется, как листовенный узел правой ветви левого суб-дерева. Если нет правой ветки, узел заменяется левой веткой. Для упрощения реализации, я выделил удаление корневого узла в специальный случай.

```
template <typename elemType>
inline void BinaryTree<elemType>::remove(const elemType &elem) {
    if (_root) {
        if (_root->_val == elem)
            else remove_root();

        _root->remove_value(elem, _root); }}
```

Оба `remove_root()` и `remove_value()` перекрепляют левую ветку, как лист правой ветви левого суб-дерева. Я вывел эту операцию из `lchild_leaf()`, статического члена-функции класса `BTreeNode`:

```
template <typename valType>
void BTreeNode<valType>::lchild_leaf(BTreeNode *leaf, BTreeNode *subtree) {
    while (subtree->_lchild)
        subtree = subtree->_lchild;
    subtree->_lchild = leaf;}
```

`remove_root()` сбрасывает корневой узел в одну из его веток, если узел веток присутствует. Если присутствует правая ветка, она становится новым корневым узлом. Левая ветка, если есть, перекрепляется

либо непосредственно, либо через вызов `lchild_leaf()`. Если правая ветка нулевая, `_root` устанавливается на левую ветку.

```
template <typename elemType>
void BinaryTree<elemType>::remove_root() {
    if (! _root) return;

    BTreeNode<elemType> *tmp = _root;
    if (_root->_rchild){
        _root = _root->_rchild;

        // ok, теперь мы должны прикрепить левую ветку к
        // низу правой ветки левого суб-дерева
        if (tmp->_lchild){ // делаем только для возможности читать
            BTreeNode<elemType> *lc = tmp->_lchild;
            BTreeNode<elemType> *newlc = _root->_lchild;
            if (! newlc)

                // суб-дерева нет, давайте прикреплять непосредственно

                it _root->_lchild = lc;

                // lchild_leaf() будет давать левое суб-дерево
                // проверим на нуль левую ветку для прикрепления ...
                // lchild_leaf статический член-функция
            else BTreeNode<elemType>::lchild_leaf(lc, newlc);}}
        else _root = _root->_lchild;

    delete tmp; // ok: теперь мы удаляем узел, прежде бывший корнем}
```

`remove_value()` принимает два параметра: значение для удаления, если оно есть, и указатель на родительский узел или узел, в настоящий момент проверяемый.

```
template <typename valType>
void BTreeNode<valType>::remove_value(const valType &val, BTreeNode *& prev);
```

Список параметров `remove_value()` иллюстрирует два использования ссылочного параметра. `val` передается, как ссылка для предотвращения потенциально большого копирования по значению, если `valType` специфицирован, как тип класса. Поскольку мы не собирались менять `val`, мы передаем его как `const`.

Второй ссылочный параметр не так интуитивно понятен. Почему мы передаем `prev`, как ссылку на указатель? Не достаточно ли указателя? Нет. Передача указателя, как параметра позволяет нам изменить объект, адресуемый указателем, но не адрес, на который указатель установлен. Для изменения значения текущего адреса указателя мы должны добавить другой уровень ненаправленности. Объявляя `prev` как ссылку на указатель, мы можем изменить как значение его адреса, так и значение адресуемого объекта.

```
template <typename valType>
void BTreeNode<valType>::remove_value(const valType &val, BTreeNode *& prev){
    if (val < _val){
        if (! _lchild) return; // не присутствует

        else _lchild->remove_value(val, _lchild);}
    else
```

```

if (val > _val){
    if (! _rchild) return; // не присутствует
    else _rchild->remove_value(val, _rchild);}
else {
    // ok: нашли;
    // сбрасываем дерево, затем удаляем этот узел

    if (_rchild) {
        prev = _rchild;
        if (_lchild)
            if (! prev->_lchild) prev->_lchild = _lchild;
            else Btnode<valType>::_lchild_leaf(_lchild,prev->_lchild);}
        else prev = _lchild;
        delete this; }}

```

Нам так же нужна функция для удаления всего дерева. `clear()` реализована, как пара функций: `inline` общая функция и перезагружаемый частный образец, который делает реальную работу.

```

template <typename elemType>
class BinaryTree {
public:
    void clear(){
        if (_root){ clear(_root); _root = 0; }}
    // ...

private:
void clear(BTnode<elemType>*); // ... };

template <typename elemType>
void BinaryTree<elemType>::_clear(BTnode<elemType> *pt){
    if (pt){
        clear(pt->_lchild);
        clear(pt->_rchild);
        delete pt; }}

```

Следующая программа строит бинарное дерево, показанное на [рисунке 6.1](#). Предпорядок дерева, отображаемый на дисплее, генерируется в трех образцах: после построения дерева, после удаления корневого узла и после удаления внутреннего узла.

```

#include "BinaryTree.h"
#include <iostream>
#include <string>
using namespace std;
int main(){

    BinaryTree<string> bt;

    bt.insert("Piglet");
    bt.insert("Eeyore");
    bt.insert("Roo");
    bt.insert("Tigger");
    bt.insert("Chris");
    bt.insert("Pooh");
    bt.insert("Kanga");

```

```

cout << "Preorder traversal: \n";
bt.preorder();

bt.remove("Piglet");
cout << "\n\nPreorder traversal after Piglet removal: \n";
bt.preorder();

bt.remove("Eeyore");

cout << "\n\nPreorder traversal after Eeyore removal: \n";
bt.preorder();

return 0;}

```

После компиляции и при выполнении программа показывает:

```

Preorder traversal:
Piglet Eeyore Chris Kanga Roo Pooh Tigger

Preorder traversal after Piglet removal:
Roo Pooh Eeyore Chris Kanga Tigger

Preorder traversal after Eeyore removal:
Roo Pooh Kanga Chris Tigger

```

Каждый из трех алгоритмов следования – `preorder()`, `inorder()` и `postorder()` - выполняют операцию на текущем узле (в нашем случае, отображается `_val`) и рекурсивно вызывают себя на левой и правой ветке. Единственная разница между тремя алгоритмами – порядок, в котором эти действия совершаются:

```

template <typename valType>
void BTreeNode<valType>::preorder(BTreeNode *pt, ostream &os) const{
    if (pt){
        display_val(pt, os);
        if (pt->_lchild) preorder(pt->_lchild, os);
        if (pt->_rchild) preorder(pt->_rchild, os); }}

template <typename valType>
void BTreeNode<valType>::inorder(BTreeNode *pt, ostream &os) const{
    if (pt) {
        if (pt->_lchild) inorder(pt->_lchild, os);
        display_val(pt, os);
        if (pt->_rchild) inorder(pt->_rchild, os); }}

template <typename valType>
void BTreeNode<valType>::postorder(BTreeNode *pt, ostream &os) const{
    if (pt){
        if (pt->_lchild) postorder(pt->_lchild, os);
        if (pt->_rchild) postorder(pt->_rchild, os);
        display_val(pt, os); }}

```

## 6.5 Функция оператора вывода шаблона

Мы бы хотели поддержать оператор вывода для нашего шаблона класса `BinaryTree`. Для не шаблонного класса мы пишем

```
ostream& operator<<(ostream&, const int_BinaryTree&);
```

Для шаблона класса мы должны поддержать явный образец для каждого определения сгенерированного класса:

```
ostream& operator<<(ostream&, const BinaryTree<int>&);
```

Но это Сизифов труд, утомительный и бесконечный. Лучшим решением было бы определить оператор вывода, как функцию шаблона:

```
template <typename elemType>
inline ostream& operator<<(ostream &os, const BinaryTree<elemType> &bt){
    os << "Tree: " << endl;
    bt.print(os);
    return os; }
```

Когда мы пишем

```
BinaryTree< string > bts;

cout << bts << endl;
```

образец оператора вывода генерируется для поддержки второго параметра `BinaryTree<string>`. Когда мы пишем

```
BinaryTree< int > bti;
cout << bti << endl;
```

образец оператора вывода генерируется для поддержки второго параметра `BinaryTree<int>` и т.д.

`print()` – это частная функция-член шаблона класса `BinaryTree` (сошлюсь на листинг кода на сайте, где она определяется). Чтобы оператор вывода получил доступ к `print()`, он должен стать другом `BinaryTree`:

```
template <typename elemType>
class BinaryTree {
    friend ostream& operator<<(ostream&,const BinaryTree<elemType>&); // ...};
```

## 6.6 Постоянные выражения и параметры по умолчанию

Параметры шаблона не ограничены только по типам, хотя пока я ограничил их обсуждение. Мы можем также объявить постоянные выражения, как параметры. Например, иерархия нашего класса числовой последовательности может быть переопределена, как класс шаблона, в котором число элементов, содержащихся в объекте, параметризовано:

```
template <int len>
class num_sequence {
public:
```

```

        num_sequence(int beg_pos=1); // ...};

template <int len>
class Fibonacci : public NumericSeries<len> {
public:
    Fibonacci(int beg_pos=1): num_sequence<len>(beg_pos){} // ...};

```

Когда создается объект `Fibonacci`, как в

```

Fibonacci< 16 > fib1;

Fibonacci< 16 > fib2(17);

```

образцы обоих, производного класса `Fibonacci` и базового класса `num_sequence`, генерируются с `len` ограниченной 16. Альтернативно, мы можем параметризовать и длину, и начальную позицию:

```

template < int len, int beg_pos > class NumericSeries;

```

Однако, поскольку большинство объектов класса начинают свои границы с позиции 1, это удобно только тогда, когда мы поддерживаем значение по умолчанию для позиции:

```

template <int len, int beg_pos> class num_sequence { ... };
template <int len, int beg_pos=1>
class Fibonacci : public num_sequence<len,beg_pos> { ... };

```

Вот как объект таких классов может быть определен:

```

// расширено, как:
// num_sequence<32,1> *pns1to32 = new Fibonacci<32,1>;
num_sequence<32> *pns1to32 = new Fibonacci<32>;

// подменяется значение по умолчанию параметра выражения
num_sequence<32,33> *pns33to64 = new Fibonacci<32,33>;

```

Подобно значениям параметра функции по умолчанию значения параметра по умолчанию разрешаются позиционно справа налево. Чтобы проиллюстрировать, как мы можем реально реализовать это, я переопределил базу нашей `num_sequence` и `Fibonacci`, и производные классы [Части 5](#) для использования параметров выражения:

```

// определение класса num_sequence
// нам больше не нужно сохранять, как данные-члены
// длину и начальную позицию

template <int len, int beg_pos>
class num_sequence {
public:
    virtual ~num_sequence(){};
    int elem(int pos) const;
    const char* what_am_i() const;
    static int max_elems(){ return _max_elems; }
    ostream& print(ostream &os = cout) const;

protected:

```

```

    virtual void gen_elems(int pos) const = 0;

    bool check_integrity(int pos, int size) const;

    num_sequence(vector<int> *pe) : _pelems(pe){}
    static const int _max_elems = 1024;
    vector<int> *_pelems;};

// определение оператора вывода функцией шаблона
template <int len, int beg_pos>
ostream& operator<<(ostream &os, const num_sequence<len,beg_pos> &ns){

    return ns.print(os); }

// num_sequence члены-функции ...
template <int len, int beg_pos>
int num_sequence<len,beg_pos>::elem(int pos) const{
    if (! check_integrity(pos, _pelems->size())) return 0;

    return (*_pelems)[pos-1];}

template <int length, int beg_pos>
const char* num_sequence<length, beg_pos>::what_am_i() const {

    return typeid(*this).name(); }

template <int length, int beg_pos>
bool num_sequence<length, beg_pos>::check_integrity(int pos, int size) const{
    if (pos <= 0 || pos > max_elems()){
        cerr << "!! invalid position: "

            << pos<< " Cannot honor request\n";
        return false;}

    if (pos > size) gen_elems(pos);

    return true;}

template <int length, int beg_pos>
ostream& num_sequence<length, beg_pos>::print(ostream &os) const{
    int elem_pos = beg_pos-1;
    int end_pos = elem_pos + length;

    if (! check_integrity(end_pos, _pelems->size())) return os;

    os << "("
        << beg_pos << " , "
        << length << ") ";

    while (elem_pos < end_pos)
        os << (*_pelems)[elem_pos++] << ' ';

    return os;}

// ok: шаблон класса Fibonacci со значением параметра по умолчанию
template <int length, int beg_pos=1>
class Fibonacci : public num_sequence<length, beg_pos> {

```

```

public:
    Fibonacci() : num_sequence<length,beg_pos>(&_elems){}
protected:
    virtual void gen_elems(int pos) const;static vector<int> _elems; };

// объявляем статические данные-члены шаблона для Fibonacci
template <int length, int beg_pos>
vector<int> Fibonacci<length,beg_pos>::_elems;

// члены-функции шаблона класса Fibonacci
template <int length, int beg_pos>
void Fibonacci<length,beg_pos>::gen_elems(int pos) const {
    if (pos <= 0 || pos > max_elems()) return;

    if (_elems.empty()) {
        _elems.push_back(1);
        _elems.push_back(1); }
    if (_elems.size() < pos) {
        int ix = _elems.size();
        int n_2 = _elems[ix-2], n_1 = _elems[ix-1];

        int elem;
        for (; ix < pos; ++ix) {
            elem = n_2 + n_1;
            _elems.push_back(elem);
            n_2 = n_1;
            n_1 = elem; }}}

```

Вот небольшая программа для упражнений с нашей реализацией. fib1, fib2 и fib3 – каждая представляет различные образцы шаблона класса Fibonacci. fib1 имеет длину 8 и начальную позицию по умолчанию 1. fib2 также имеет длину 8, но начальную позицию 8. fib3 имеет длину 12 и также начальную позицию 8.

```

int main() {

    Fibonacci<8> fib1;

    Fibonacci<8,8> fib2;

    Fibonacci<12,8> fib3;

    cout << "fib1: " << fib1 << '\n'
         << "fib2: " << fib2 << '\n'
         << "fib3: " << fib3 << endl; };

```

После компиляции при выполнении мы увидим:

```

fib1: (1 , 8) 1 1 2 3 5 8 13 21

fib2: (8 , 8) 21 34 55 89 144 233 377 610

fib3: (8 , 12) 21 34 55 89 144 233 377 610 987 1597 2584 4181

```

Адресация к функциям и объектам в общих границах также постоянные выражения, и могут представляться параметрами значения. Например, вот класс числовой последовательности, который принимает указатель на

функцию, как его параметр:

```
template <void (*pf)(int pos, vector<int> &seq)>
class numeric_sequence{
public:
    numeric_sequence(int len, int beg_pos = 1){
        // тщательно проверяем, что pf не нулевой ...
        if (! pf)
            // выдает сообщение об ошибке и выходит

        _len = len > 0 ? len : 1;
        _beg_pos = beg_pos > 0 ? beg_pos : 1;

        pf(beg_pos+len, _elems);} // ...
private:
    int _len;
    int _beg_pos;
    vector<int> _elems; };
```

В этом примере `pf` адресуется к функции, которая генерирует элементы `pos` специального типа последовательности внутри вектора `seq`. Это может быть использовано для следующего:

```
void fibonacci(int pos, vector<int> &seq);
void pell(int pos, vector<int> &seq);
// ...
numeric_sequence<fibonacci> ns_fib(12);
numeric_sequence<pell> ns_pell(18, 8);
```

## 6.7 Параметры шаблона как стратегия

Наш объект функции класса `LessThan` из [Раздела 4.9](#) естественный кандидат для преобразования в класс шаблона:

```
template <typename elemType>
class LessThan {
public:
    LessThan(const elemType &val) : _val(val){}
    bool operator()(const elemType &val) const { return val < _val; }

    void val(const elemType &newval) { _val = newval; }

    elemType val() const { return _val; }

private:
    elemType _val; };

LessThan<int> lti(1024);

LessThan<string> lts("Pooh");
```

Потенциальной проблемой реализации может стать то, что программа рухнет, если тип, введенный

пользователем, не определен для оператора “меньше, чем”. Одна из возможных стратегий – поддержать второй класс с оператором сравнения, выведенным из определения класса. Даже если этот второй класс поддерживает те же основные семантики, как `LessThan`, мы должны поддержать уникальные имена для них, поскольку шаблон класса не может перезагрузиться, основываясь на его списке параметров. Давайте назовем этот класс `LessThanPred`, поскольку объект функции класса “меньше, чем” специфицирован, как параметр по умолчанию:

```
template <typename elemType, typename Comp = less<elemType> >
class LessThanPred {
public:
    LessThanPred(const elemType &val) : _val(val){}
    bool operator()(const elemType &val) const {

        return Comp(val, _val); }

    void val(const elemType &newval) { _val = newval; }

    elemType val() const { return _val; }

private:

    elemType _val;};

// альтернативный объект функции сравнения
class StringLen {
public:
    bool operator()(const string &s1, const string &s2) {

        return s1.size() < s2.size(); }};

LessThanPred<int> ltpi(1024);

LessThanPred<string, StringLen> ltps("Pooh");
```

Альтернативно мы можем поддержать более общее имя для нашего объекта функции, чтобы обозначить, что он поддерживает любой оператор сравнения. В этом случае, не будет больше иметь значения поддержка объекта функции по умолчанию:

```
template <typename elemType, typename BinaryComp > class Compare;
```

`Compare` применяется к любой операции `BinaryComp` с двумя теми же произвольными объектами `elemType`.

В **Части 5** мы разработали объектно-ориентированную иерархию числовой последовательности. Рассмотрим следующую альтернативную конструкцию, в которой мы определяем шаблон класса числовой последовательности с реальным классом последовательности, выделенной, как параметр:

```
template <typename num_seq>
class NumericSequence {
public:
    NumericSequence(int len = 1, int bpos = 1) : _ns(len, bpos){}

    // это вызов неизвестных членов-функций
    // числовой последовательности через порядок именования:
    // каждый параметр класса num_seq должен поддерживать
```

```

    // именованные функции calc_elems(), is_elem(), и т.д. ...
    void calc_elems(int sz) const { _ns.calc_elems(sz); }
    bool is_elem(int elem) const { return _ns.is_elem(elem); }

    // ...

private:
    num_seq _ns;};

```

Эта разработка шаблона навязывает порядок наименования в классах, используемый как параметр: каждый должен поддерживать именованную функцию, относящуюся к таким вызовам внутри шаблона класса NumericSequence, как `calc_elems()`, `is_elem()` и т.д.

Хотя эта конструкционная идиома есть нечто продвинутое, я думаю, она хуже, поскольку показана бегло, и вы еще не впадаете в размышления о параметрах типов шаблонов класса, как представляющих только типы элементов, подобные показанным в реализации бинарного дерева, и вектора, и списка контейнера класса стандартной библиотеки.

## 6.8 Функции-члены шаблона

Мы можем также определить функции-члены шаблона. Посмотрите на пример и побродите по нему:

```

class PrintIt {

public:

    PrintIt(ostream &os): _os(os){}

    // функции-члены шаблона
    template <typename elemType>
    void print(const elemType &elem, char delimiter = '\n'){

        _os << elem << delimiter; }

private:

    ostream& _os;};

```

`PrintIt` – это не класс шаблона, который инициализирует поток вывода. Он поддерживает члена-функцию шаблона `print()`, которая пишет объект произвольного типа в поток вывода. Сделав `print()` членом-функцией шаблона, мы можем поддержать единственный образец, поддерживающий любой тип, для которого образец оператора вывода может быть использован. Когда мы будем параметризовать `PrintIt` типом элемента, который мы хотим вывести, мы будем создавать новый шаблон класса для каждого особого типа. Под этой реализацией есть единственный класс `PrintIt`. Вот как мы можем использовать это:

```

int main(){

    PrintIt to_standard_out(cout);

    to_standard_out.print("hello");

    to_standard_out.print(1024);

    string my_string("i am a string");

```

```
to_standard_out.print(my_string);}
```

Выполнение после компиляции даст:

```
hello
```

```
i am a string
```

Шаблон класса может также определить член-функцию шаблона. Например, мы можем параметризовать PrintIt ее типом ostream, когда создаем print(), как член-функцию шаблона:

```
template <typename OutStream>
class PrintIt {
public:
    PrintIt(OutStream &os): _os(os){}

template <typename elemType>

void print(const elemType &elem, char delimiter = '\n'){

    _os << elem << delimiter; }

private:

    ostream& _os;};
```

Вот наша модифицированная программа:

```
int main(){
    PrintIt<ostream> to_standard_out(cout);
    to_standard_out.print("hello");
    to_standard_out.print(1024);

    string my_string("i am a string");

    to_standard_out.print(my_string);}
```

Можно сказать значительно больше о шаблонах, чем я сделал здесь. Более детально ознакомится с возможностями C++, можно в [LIPPMAN98] Часть 10 (Шаблоны функции) и Часть 16 (Шаблоны класса) и в [STROUSTRUP97] Часть 13 .

## Упражнение 6.1

Перепишите следующее определение класса, чтобы сделать его шаблоном класса:

```
class example {
public:
    example(double min, double max);
    example(const double *array, int size);
    double& operator[](int index);
    bool operator==(const example&) const;
    bool insert(const double*, int);
    bool insert(double);
    double min() const { return _min; }
    double max() const { return _max; }
    void min(double);
```

```
void max(double);  
int count(double value) const;  
private:  
  
int size;  
  
double *parray;  
  
double _min;double _max;};
```

## Упражнение 6.2

Переделайте реализацию класса `Matrix` из [Упражнения 4.3](#) в шаблон. В дополнение расширьте его для поддержки строк и столбцов произвольного размера, используя свободную память. Локализируйте память в конструкторе и освободите в деструкторе.

## Часть 7. Поддержка исключений

При реализации нашего класса `Triangular_iterator` в [Разделе 4.6](#) мы подчеркивали, что итератор может потенциально ввести программу в состояние ошибки. Член-данные `_index` могут получить значение большее максимально возможного числа элементов, сохраняющихся в статическом векторе класса `Triangular`. Выглядит это так, что не похоже на реально происходящее. Это плохо, если случается, поскольку программа, похоже, при попытке использовать значение рухнет. Это плохо еще и потому, что программист, использующий класс итератора, не имеет простых способов для распознавания или разрешения проблемы.

Как разработчики класса итератора мы можем распознать проблему: итератор оказался не в правильном состоянии и не должен дальше использоваться в программе. Но мы не знаем, насколько серьезна проблема, для всей программы. Только пользователь итератора может это знать. Наша задача – предупредить пользователя. Чтобы это сделать, мы используем возможность поддержки исключений в C++, тему этой части.

### 7.1 Вбрасывание исключения

Средства поддержки исключения состоят из двух базовых составляющих: распознавания и срабатывания исключения, и завершения поддержки исключения. Обычно, исключение срабатывает и поддерживается внутри разных членов или не членов-функций. После срабатывания исключения нормальное продолжение программы приостанавливается. Средства поддержки исключения ищут часть программы, которая может поддержать сработавшее исключение. После того, как исключение было поддержано, выполнение программы покидает ту часть программы, которая осуществляла поддержку исключения.

В C++ мы заставляем сработать исключение, используя выражение `throw`:

```
inline void Triangular_iterator::check_integrity() {
    if (_index > Triangular::_max_elems)

        throw iterator_overflow(_index, Triangular::_max_elems);

    if (_index > Triangular::_elems.size())

        Triangular::gen_elements(_index);}
```

Выражение `throw` несколько похоже на вызов функции. В этом примере, если `_index` больше, чем `_max_elems` вбрасывается объект исключения типа `iterator_overflow`. Второе условие `if` никогда не выполняется, а управление не возвращается в эту точку программы после поддержки исключения. Если `_index` “меньше, чем” или равно `_max_elems`, исключение не срабатывает, а программа выполняется, как мы и ожидаем.

Что это за исключение, которое вбрасывается? Исключение – объект того же типа. Простой объект исключения может быть целым или строковым:

```
throw 42;
throw "panic: no buffer!";
```

Чаще вброшенные исключения – это класс объектов, поддерживающих класс исключения, или иерархию

классов исключения. Например, вот определение класса `iterator_overflow`:

```
class iterator_overflow {
public:
    iterator_overflow(int index, int max): _index(index), _max(max) {}

    int index() { return _index; }
    int max() { return _max; }

void what_happened(ostream &os = cerr) {
    os << "Internal error: current index " << _index
    << " exceeds maximum bound: " << _max; }

private:
    int _index;int _max;};
```

Нет ничего исключительного в этом классе. Мы просто определяем его для сохранения данных о случившемся, когда хотим покинуть русло программы в поисках точки, связанной с природой сработавшего исключения.

Выражение `throw` нашего примера непосредственно вызывает двухпараметрический конструктор. Альтернативно мы можем вбросить явный объект класса:

```
if (_index > Triangular::_max_elems){
    iterator_overflow ex(_index, Triangular::_max_elems);
    throw ex;}
```

## 7.2 Захват исключения

Мы захватываем объекты исключения по их типу, используя один или серию пунктов захвата. Пункты захвата состоят из трех частей: ключевого слова `catch`, объявления единственного типа или единственного объекта внутри круглых скобок, и множества выражений внутри фигурных скобок, которые выполняют реальную поддержку исключения. Например, рассмотрим следующее множество пунктов захвата:

```
// определено где-то еще ...
extern void log_message(const char*);
extern string err_messages[];
extern ostream log_file;

bool some_function(){ bool status = true;

// ... мы перейдем в эту часть!

catch(int errno){
    log_message(err_messages[errno]);
    status = false; }
catch(const char *str){
    log_message(str);
    status = false;}
catch(iterator_overflow &iof){
    iof.what_happened(log_file);
```

```

    status = false; }

// последняя линия функции

    return status;}

```

Эти пункты захвата поддерживают три объекта исключения, которые мы вбрасывали в предыдущем разделе:

```

throw 42;
throw "panic: no buffer!";
throw iterator_overflow(_index, Triangular::_max_elems);

```

Тип объекта исключения сравнивается с объявленным исключением каждого пункта захвата по очереди. Если типы совпадают, тело пункта захвата выполняется. Например, когда мы вбрасываем наш объект `iterator_overflow`, три пункта захвата проверяются по очереди. Объявление исключения третьего пункта совпадает по типу с объектом исключения, и выполняются ассоциированные с ним выражения. Член-функция `what_happened()` нашего класса исключения вызывается через `iof` объекта исключения. `const char*` возвращает значение, переданное какой-нибудь внешней функцией `log_message()`. Следом за этим `status` устанавливается в `false`.

Таково представление полной поддержки исключения. Нормальное выполнение программы прекращается на первом выражении программы, следующим за множеством пунктов захвата. В нашем примере нормальное выполнение программы начинается вновь с возвратом `status`.

Что случится с вбрасыванием буквенной строки? Подходящей к этому пункт захвата – второй образец. Вызывается `log_message()` с объектом исключения `str`, в качестве ее аргумента. `status` устанавливается в `false`. Исключение теперь поддержано. Управление передается трем пунктам захвата, а нормальное выполнение программы начинается вновь с возвращения `status`.

Может быть так, что мы не сможем полностью поддержать исключение. После получения соответствующего сообщения нам может понадобиться опять вбросить исключение для дальнейшей поддержки где-нибудь в другом пункте захвата:

```

catch(iterator_overflow &iof){ log_message(iof.what_happened());

// перевбрасываем в другой пункт захвата для поддержки
throw; }

```

Выражение перевбрасывания содержит только ключевое слово `throw`. Такое может случиться только внутри пункта захвата, что перевбрасывает объект исключения, и поиск подходящего пункта захвата продолжается.

Пункт всеобщей поддержки позволяет нам соответствовать каждому типу исключения. В месте объявления исключения мы специфицируем *ellipsis* (многоточие в скобках). Например,

```

// соответствует всем исключениям
catch(...) {
    log_message("exception of unknown type");
    // очищаем и выходим ...}

```

## 7.3 Испытания для исключения

Пункты захвата ассоциируются с блоками испытаний (*try blocks*). Блок испытаний начинается с ключевого слова `try`, сопровождаемого последовательностью программных выражений, заключенных в скобки. Пункты захвата расположены в конце блока испытаний, и они представляют исключения, которые поддерживаются, если исключение вброшено при выполнении выражений внутри блока испытаний.

Например, следующая функция просматривает `elem` в диапазоне элементов, отмеченных `first`, `last`. Итерация за пределы может потенциально вызвать вбрасывание исключения `iterator_overflow`, так что мы размещаем этот код внутри блока испытаний, сопровождаемого пунктом захвата, который содержит объявление исключения `iterator_overflow`:

```
bool has_elem(Triangular_iterator first, Triangular_iterator last, int elem) {
    bool status = true;

    try {
        while (first != last) {
            if (*first == elem)
                return status;
            ++first; } }
    // только исключение типа iterator_overflow
    // захватывается, если вброшено при выполнении
    // последовательности кода внутри блока try
    catch(iterator_overflow &iof) {

        log_message(iof.what_happened());

        log_message("check if iterators address same container");

        status = false;

        return status;}
}
```

Выражение

```
*first
```

вызывает оператор разыменовывания переполнения:

```
inline int Triangular_iterator::operator*() {
    check_integrity();

    return Triangular::_elems[_index];}
}
```

Это, в свою очередь, вызывает `check_integrity()`:

```
inline void Triangular_iterator::check_integrity() {
    if (_index > Triangular::_max_elems

        throw iterator_overflow(_index, Triangular::_max_elems); // ...}
}
```

Положим, что каким-то образом значение `_index last` больше, чем `_max_elems`, что в некоторой точке теста внутри `check_integrity()` проявляется, как `true`, и исключение вбрасывается. Что происходит?

Механизм исключения просматривает место выражения `throw` и спрашивает, обнаружилось ли это внутри блока испытаний? Если так, блоки захвата, ассоциированные с блоком испытаний, проверяются на возможность поддержки исключения. Если возможность есть, исключение поддерживается, и нормальное выполнение программы восстанавливается.

В нашем примере выражение `throw` не обнаруживается внутри блока испытаний. Нет попытки поддержать исключение. Следующее выражение в функции не выполняется. Механизм поддержки исключения прерывает `check_integrity()`. Это приводит к поиску пункта захвата внутри функции, которая вызвала `check_integrity()`.

Вопрос задается опять внутри перезагруженного разыменованного оператора: обнаруживается ли вызов `check_integrity()` внутри блока испытаний? Нет. Разыменованный оператор прерывается, а механизм исключений продолжает поиск внутри функции, которая вызвала оператор разыменования. Обнаруживается ли вызов

```
*first
```

внутри блока испытаний? В данном случае ответ – да. Проверяется ассоциированный блок захвата. Объявление исключения соответствует типу объекта исключения, и выполняется тело блока захвата. Этим завершается поддержка исключения. Нормальное выполнение программы возобновляется с первого выражения, следующего за пунктом захвата:

```
// выполняется, если элемент не найден

// или если исключение iterator_overflow захватывается

status = false;

return status;
```

Что если цепочка вызовов функций не известна `main()`, и не найден подходящий пункт захвата? Язык требует, чтобы каждое исключение было поддержано. Если не найдена поддержка, следует проверка `main()`, вызывается функция `terminate()` стандартной библиотеки. По умолчанию это прерывает программу.

Остается заботой программиста решить, как много выражений из тела функции расположить внутри или вне блока испытаний. Если выражение может быть потенциально подвержено вбросу исключения, не расположение его вне блока испытаний гарантирует, что оно не поддерживается внутри функции. Это может быть, а может не быть ОК. Не каждая функция должна поддерживать каждое потенциальное исключение.

Например, оператор разыменовывания не располагает вызов `check_integrity()` внутри блока испытаний, хотя его вызов может привести к появлению исключения. Почему? Потому что оператор разыменовывания не подготовлен для поддержки исключения и может быть аккуратно прерван, если будет вброшено исключение.

Как мы можем узнать, будет ли функция аккуратно игнорировать потенциальный вброс исключения? Посмотрим еще раз на определение оператора разыменовывания:

```
inline int Triangular_iterator:: operator*(){
    check_integrity();
    return Triangular::_elems[_index];}
```

Если `check_integrity()` проваливается, значение `_index` должно быть неправильным. Выполнение

выражения возврата в этом случае плохая идея. Можем мы добавить блок испытаний для выявления результата вызова `check_integrity()`?

Если `check_integrity()` была реализована для возвращения `true` или `false`, определение оператора разыменовывания будет нуждаться в гарантии от возврата значения `false`:

```
return check_integrity()? Triangular::_elems[_index]: 0;
```

Пользователь в свою очередь нуждается в гарантии, что оператор разыменовывания не вернет 0.

Поскольку `check_integrity()` вбрасывает исключение, здесь защита не нужна. Возвращение выражения оператора разыменовывания гарантировано будет выполнено, только если не вброшено исключение – в этом случае оно подстраховано выполнением возврата выражения. Иначе функция прерывается до того, как выражение даже достигнуто.

Почему функция `has_elem()` в предыдущем разделе захватывает ее разыменовывание от `first` в блоке испытаний? Это может просто позволить исключению `iterator_overflow` подняться к вызвавшей его функции. Альтернативно, почему `has_elem()` не беспокоится о других потенциальных исключениях? Например, возможно добавление “захватов всего” для поддержки любых вброшенных исключений при ее выполнении. Два решения – две стороны одной медали.

`has_elem()` поддерживает специфическую функциональность: говорить истинно или ложно, в зависимости от присутствия `elem` внутри области элементов, отмеченной `first`, `last`. Для выполнения этого она проходит по элементам, увеличивая `first` пока либо элемент не найдется, либо каждый элемент не будет проверен. Разыменовывание и инкремент `first` – детали реализации `has_elem()`. Исключение `iterator_overflow` один из аспектов этой реализации, и я выбрал ее локализацию внутри `has_elem()`, поскольку `has_elem()` дает больше знания о значимости этого исключения внутри выполняемой программы.

Функция, вызывающая `has_elem()` должна знать, будет ли `elem` присутствовать внутри области, отмеченной `first`, `last`. Знание, что эта область сама неверна, возможно, важно для проекта, и это причина, по которой мы сопровождаем ее. Однако ничего не значит, что функция, вызывающая `has_elem()`, похоже, подходит для поддержки. Поэтому я выбрал – защитить ее от `iterator_overflow` исключения.

На оборотной стороне, реализация `has_elem()` настолько сфокусирована на определении, присутствует ли `elem`, что не будет иметь возможности для поддержки всех потенциальных исключений. Например, если свободная программная память истощится, это слишком катастрофично для реализации `has_elem()`, чтобы перетерпеть.

Когда исключение обнаруживается внутри блока испытаний функции, которая не поддержана ассоциированными пунктами захвата, эта функция обрывается так, как если бы блока испытаний не было. Поиск пункта поддержки продолжается вверх по вызывающей цепочке к вызвавшей функции. Внутри `has_elem()` исключение `iterator_overflow` поддержано. Есть ли какие-либо потенциально не захватываемые исключения, которые могут быть вброшены? Наша проверка цепочки вызовов внутри `has_elem()` убедила нас, что нет. Общая ошибка начинающих – путать исключения C++ с аппаратными исключениями, такими как дефект сегментации (`segmentation fault`) или ошибка шины (`bus error`). Чтобы исключение C++ было вброшено, для этого где-нибудь в программном коде есть выражение `throw`, которое пользователь способен отыскать.

## 7.4 Управление локальным ресурсом

Рассмотрим следующую функцию, в которой ресурсы забираются при старте функции, имеет место некоторое

продолжение процесса, а затем ресурсы освобождаются в конце функции. Направим наше обсуждение в русло того, что неправильно с этой функцией?

```
extern Mutex m;
void f() {
    // получение ресурсов
    int *p = new int;
    m.acquire();

    process(p);

    // освобождение ресурсов
    m.release();
    delete p;}

```

Проблема в том, что мы не можем гарантировать, что полученные ресурсы при старте функции теперь освобождены. Если `process()` или функция, вызываемая внутри `process()`, вбрасывают исключение, два ресурсо-освобождающих выражения, сопровождающих вызов `process()`, никогда не выполняются. Это неприемлемая реализация в присутствии поддержки исключения.

Одно из решений – ввести блок испытания и ассоциированный пункт захвата. Мы захватим все исключения, освободим наши ресурсы, а затем перебросим исключение:

```
void f() {
    try {
        // чтонибудь ниже }
    catch(...) {
        m.release();
        delete p;
        throw; }}

```

Хотя это решает нашу проблему, оно не становится полностью удовлетворительным решением. Мы дублируем код для освобождения наших ресурсов. Мы продлеваем поиск поддержки, пока захватываем исключение, освобождаем наши ресурсы, а затем вновь вбрасываем исключение. Более того, код сам по себе наполняется значительными усложнениями. Мы бы предпочли менее вычурное, более автономное решение. В C++ это обычно означает определение класса.

Vjarne Stroustrup (Бьярн Страуструп), создатель C++, ввел идиому для управления ресурсами, которую он описал во фразе “получение ресурсов – это инициализация”. Для объекта класса инициализация происходит внутри конструктора класса. Получение ресурса выполняется внутри конструктора класса. Ресурс освобождается внутри деструктора класса. Это не только автоматизирует управление ресурсами, но также упрощает наши программы:

```
#include <memory>
void f() {
    auto_ptr<int> p(new int);
    MutexLock ml(m);
    process(p);
    // деструкторы для p и ml
    // подразумевается, вызваны здесь ... }

```

`p` и `ml` локальные объекты класса. Если `process()` выполняется корректно, ассоциированный деструктор класса автоматически выполняется для `p` и `ml` до завершения функции. Но что, если вброшено исключение при выполнении `process()`?

Все активные локальные объекты класса для функции гарантировано имеют их деструкторы, выполняемые до прерывания функции механизмом поддержки исключения. В нашем примере деструктор для `p` и `m1` гарантированно вызывается, вне зависимости от вброса исключения.

Класс `MutexLock` может быть реализован следующим образом:<sup>12</sup>

```
class MutexLock {
public:
    MutexLock(Mutex m) : _lock(m) { lock.acquire(); }
    ~MutexLock() { lock.release(); }
private:
    Mutex &_lock;};
```

`auto_ptr` – класс шаблона, поддержанный стандартной библиотекой. Она автоматизирует удаление объектов локализованных через выражение `new`, таких как `p`. Для ее использования мы должны включить ассоциированный файл заголовка, `memory`:

```
#include <memory>
```

Класс `auto_ptr` перезагружает разыменованный и стрелочный операторы указателя тем же образом, что мы делали с нашим классом итератора в [Разделе 4.6](#). Это позволяет нам использовать объект `auto_ptr` так же, как мы используем указатель. Например,

```
auto_ptr< string > aps(new string("vermeer"));
string *ps = new string("vermeer");
if ((aps->size() == ps->size()) &&(*aps == *ps))// равны ...
```

Для более подробного знакомства с идиомой – получение ресурса это инициализация, обратитесь к [Разделу 14.4 \[STROUSTRUP\]](#). Для детального изучения класса `auto_ptr` загляните в [Раздел 8.4.2 \[LIPPMAN98\]](#).

## 7.5 Стандартные исключения

Если выражение `new` не может получить память из свободной программной области, оно вбрасывает объект исключения `bad_alloc`. Например,

```
vector<string>*init_text_vector(istream &infile){
vector<string> *ptext = 0;
try {
    ptext = new vector<string>;
    // открываем файл и файл вектора}
```

---

<sup>12</sup> Это основано на великолепной статье "A Case Study of C++ Design Evolution" Douglas C.Schmidt в [\[LIPPMAN96b\]](#).

```

catch (bad_alloc) {
    cerr << "ouch. heap memory exhausted!\n";
    // ... очищаем и выходим
    return ptext;}

```

Обычно выражение присваивания

```
ptext = new vector<string>;
```

локализует необходимую память, прилагая по умолчанию конструктор `vector<string>` на свободном объекте, а затем присваивает адрес этого объекта `ptext`.

Если память для предоставления объекту `vector<string>` не доступна, конструктор по умолчанию не вызывается, а `ptext` не присваивается. Объект исключения `bad_alloc` вбрасывается, а управление передается ассоциированному пункту захвата, следующему за блоком испытаний. Объявление исключения

```
catch (bad_alloc)
```

не объявляет объект исключения, поскольку мы интересуемся только в захвате типа исключения, а не в реальных манипуляциях с объектом внутри пункта захвата.<sup>13</sup>

Если бы мы хотели манипулировать объектом исключения `bad_alloc`, какие операции должно оно поддерживать?

Стандартная библиотека определяет иерархию класса исключения, основанную на базовом абстрактном классе, названном `exception`. Класс `exception` объявляет виртуальную функцию, названную `what()`, которая возвращает `const char*`. Ее цель - поддерживать текстовое описание вбрасывания исключения.

Класс `bad_alloc` производный от базового класса `exception`. Он поддерживает свой собственный образец `what()`. В Visual C++ образец `bad_alloc` генерирует сообщение *bad allocation*.

Мы можем тоже произвести наш класс `iterator_overflow` из базового класса `exception`. Чтобы это сделать мы должны включить стандартный файл заголовка `exception` и поддержать образец `what()`:

```

#include <exception>
class iterator_overflow : public exception {
public:
    iterator_overflow(int index, int max): _index(index), _max(max){}

    int index() { return _index; }
    int max() { return _max; }

    // overrides exception::what()
    const char* what() const;

private:

```

---

<sup>13</sup> Для выполнения вбрасывания исключения `bad_alloc` мы можем написать `ptext = new (nothrow) vector<string>;` Если выражение `new` рухнет, оно вернет 0. Любое использование `ptext` должно вначале проверяться, что оно не нулевое.

```
int _index;int _max;};
```

Польза от наследования `iterator_overflow` от иерархии класса исключения стандартной библиотеки в том, что он может теперь быть захвачен всеми кодами, которые захватываются абстрактной базой `exception`, включая коды, написанные до введения класса `iterator_overflow`. Это означает, что мы не должны переделывать код, чтобы выяснить это, или что новый тип класса исключения или захвата исключения анонимно использует “захватить все”. Пункт захвата

```
catch(const exception &ex){ cerr << ex.what() << endl;}
```

соответствует любому классу, производному от `exception`. Он печатает *bad allocation*, когда тип исключения `bad_alloc` вбрасывается. Когда тип исключения `iterator_overflow` вбрасывается, он печатает *Internal error: current index 65 exceeds maximum bound: 64*.

Вот возможная реализация образца `what()` `iterator_overflow`. Он использует объект класса `ostringstream` для форматирования своего сообщения:

```
#include <sstream>
#include <string>

const char* iterator_overflow:: what() const{
    ostringstream ex_msg;
    static string msg;

    // записывает вывод в объект в памяти
    // класса ostringstream,
    // конвертирует целые значения в
    // строковое представление ...

    ex_msg << "Internal error: current index "
           << _index << " exceeds maximum bound: "
           << _max;

    // выделяет объект string
    msg = ex_msg.str();

    // выделяет const char* представление
    return msg.c_str(); }
```

Класс `ostringstream` поддерживает вывод “в-памяти” операций на строковых объектах. Он исключительно полезен, когда мы нуждаемся в форматировании множества типов данных в строковое представление. Например, он автоматически конвертирует арифметические объекты, такие как `_index` и `_max`, в их соответствующее строковое представление без необходимости нам сосредотачиваться на организации необходимых хранилищ, или алгоритме эффективного конвертирования. Член-функция `str()` возвращает строковый объект, ассоциированный с классом `ostringstream`.

Выбор стандартной библиотеки для получения `what()` возвращает строку `const char*` представления C-стиля, а не объект строкового класса. Этот оставляет нас в очевидном недоумении: как мы можем конвертировать объект строкового класса в строковое представление в C-стиле? Строковый класс поддерживает нас решением: функция конвертации `c_str()`, которая возвращает `const char*` представляющий строку, в точности, как нам нужно.

Для использования класса `ostringstream` мы должны включить стандартный файл заголовка `sstream`:

```
#include <sstream>
```

Библиотека `iostream` также поддерживает класс ввода `istream`. Он особенно полезен, если мы нуждаемся в конвертации строкового представления в нестроковые данные, такие как целые значения и адреса в реальные множественные типы данных. Смотрите Раздел 20.8 [LIPPMAN98] для уточнения и иллюстраций использования потоковых строчных классов.

Для более полного знакомства с возможностями поддержки исключений в C++ будет полезно заглянуть в Часть 11 и Раздел 19.2 [LIPPMAN98] и Часть 14 [STROUSTRUP97]. Для хорошего обсуждения исключения-сохранных разработок и основных положений разработки классов в присутствии поддержки исключений отсылаю вас к [SUTTER99].

## Упражнение 7.1

Следующая функция абсолютно не поддерживает проверку ни возможных плохих данных, ни возможности для операций вызвать сбой. Идентифицируйте все то, что может идти неправильно внутри функции (в этом упражнении мы совершенно не беспокоимся о возможности возникновения исключений).

```
int *alloc_and_init(string file_name) {
    ifstream infile(file_name);
    int elem_cnt;
    infile >> elem_cnt;
    int *pi = allocate_array(elem_cnt);

    int elem;
    int index = 0;
    while (infile >> elem)
        pi[index++] = elem;

    sort_array(pi, elem_cnt);
    register_data(pi);

    return pi;}

```

## Упражнение 7.2

Следующая функция, вызываемая в `alloc_and_init()` возрождает следующие типы исключений, если они рухнут:

```
allocate_array() noMemsort_array() int
register_data() string
```

Включите один или более `try` блоков и ассоциированных `catch` пунктов захвата с соответствующей поддержкой этих исключений. Просто выведите обнаруженные ошибки внутри `catch` пункта захвата.

## Упражнение 7.3

Добавьте пару исключений в иерархию класса `Stack` из [Упражнения 5.2](#) для поддержки случаев попыток `pop` для пустого стека и попыток `push` для полного. Покажите модифицированные члены функции `pop()` и `push()`.

## Приложение А. Решение упражнений

### Упражнение 1.4

**Try to extend the program: (1) Ask the user to enter both a first and last name, and (2) modify the output to write out both names.**

We need two strings for our extended program: one to hold the user's first name and a second to hold the user's last name. By the end of [Chapter 1](#), we know three ways to support this. We can define two individual string objects:

```
string first_name, last_name;
```

We can define an array of two string objects:

```
string usr_name[2];
```

Or we can define a vector of two string objects:

```
vector<string> usr_name(2);
```

At this point in the text, arrays and vectors have not yet been introduced, so I've chosen to use the two string objects:

```
#include <iostream> #include
<string>using namespace std;

int main()
{ string first_name, last_name;cout << "Please enter your first name: ";cin >>
  first_name;

  cout << "hi, " << first_name
    << " Please enter your last name: ";

  cin >> last_name;
  cout << '\n';
    << "Hello, "
    << first_name << ' ' << last_name
    << " ... and goodbye!\n";

}
```

When compiled and executed, this program generates the following output (my responses are highlighted in bold):

```
Please enter your first name: stan hi, stan Please enter your last name: lippman
```

```
Hello, stan lippman ... and goodbye!
```

## Упражнение 1.5

**Write a program to ask the user his or her name. Read the response. Confirm that the input is at least two characters in length. If the name seems valid, respond to the user. Provide two implementations: one using a C-style character string, and the other using a string class object.**

The two primary differences between a string class object and a C-style character string are that (1) the string class object grows dynamically to accommodate its character string, whereas the C-style character string must be given a fixed size that is (hopefully) large enough to contain the assigned string, and (2) the C-style character string does not know its size. To determine the size of the C-style character string, we must iterate across its elements, counting each one up to but not including the terminating null. The `strlen()` standard library routine provides this service for us:

```
int strlen(const char*);
```

To use `strlen()`, we must include the `cstring` header file.

However, before we get to that, let's look at the string class implementation. Particularly for beginners, I recommend that the string class be used in favor of the C-style character string.

```
#include <iostream> #include
<string>using namespace std;

int main(){ string user_name;

    cout << "Please enter your name: ";
    cin >> user_name;

    switch (user_name.size()){
        case 0:
            cout << "Ah, the user with no name. "
                << "Well, ok, hi, user with no name\n";
            break;

        case 1: cout << "A 1-character name? Hmm, have you
            read Kafka?: " << "hello, " << user_name <<
            endl;break;

        default:
            // any string longer than 1 character
            cout << "Hello, " << user_name
                << " -- happy to make your acquaintance!\n";
            break;
    }
    return 0;}
```

The C-style character string implementation differs in two ways. First, we must decide on a fixed size to declare `user_name`; I've arbitrarily chosen 128, which seems more than adequate. Second, we use the the standard library `strlen()` function to discover the size of `user_name`. The `cstring` header file holds the declaration of `strlen()`. If the user enters a string longer than 127 characters, there will be no room for the terminating null character. To prevent that, I use the `setw()` iostream manipulator to guarantee that we do not read in more than 127 characters. To use the `setw()` manipulator, we must include the `iomanip` header file.

```
#include <iostream> #include
<iomanip>#include <cstring>using
namespace std;
```

```

int main(){ // must allocate a fixed size
    const int nm_size = 128;
    char user_name[nm_size];
    cout << "Please enter your name: ";
    cin >> setw(nm_size) >> user_name;

    switch (strlen(user_name))
    {
        // same case labels for 0, 1
        case 127:

            // maybe string was truncated by setw()
            cout << "That is a very big name, indeed -- "<< "we may
                have needed to shorten it!\n"<< "In any case,\n";

            // no break -- we fall through ...
            default: // the 127 case drops through to here -- no breakout << "Hello, "
                << user_name
                << " -- happy to make your acquaintance!\n";
    }
    break;
}

return 0;}

```

## Упражнение 1.6

**Write a program to read in a sequence of integers from standard input. Place the values, in turn, in a built-in array and a vector. Iterate over the containers to sum the values. Display the sum and average of the entered values to standard output.**

The built-in array and the vector class differ in primarily the same ways as the C-style character string (which is implemented as an array of `char` elements) and the string class: (1) The built-in array must be of a fixed size, whereas the vector can grow dynamically as elements are inserted, and (2) the built-in array does not know its size. The fixed-size nature of the built-in array means that we must be concerned with potentially overflowing its boundary. Unlike the C-style string, the built-in array has no sentinel value (the null) to indicate its end. Particularly for beginners, I recommend that the vector class be used in favor of the built-in array. Here is the program using the vector class:

```

#include <iostream> #include <vector>
using namespace std;

int main()
{ vector<int> ivec;int ival;while (cin >> ival)ivec.push_back(ival);

    // we could have calculated the sum as we entered the// values, but the
    idea is to iterate over the vector ...for (int sum = 0, ix = 0; ix <
    ivec.size(); ++ix)sum += ivec[ix];

    int average = sum / ivec.size();
    cout << "Sum of " << ivec.size()
        << " elements: " << sum
        << ". Average: " << average << endl;}

```

The primary difference in the following built-in array implementation is the need to monitor the number of elements being read to ensure that we don't overflow the array boundary:

```
#include <iostream> using
namespace std;

int main()
{
    const int array_size = 128;
    int ia[array_size];
    int ival, icnt = 0;

    while (cin >> ival &&
           icnt < array_size)
        ia[icnt++] = ival;

    for (int sum = 0, ix = 0; ix < icnt; ++ix)
        sum += ia[ix];

    int average = sum / icnt;
    cout << "Sum of " << icnt
         << " elements: " << sum
         << ". Average: " << average << endl;}
```

## Упражнение 1.7

**Using your favorite editor, type two or more lines of text into a file. Write a program to open the file, reading each word into a `vector<string>` object. Iterate over the vector, displaying it to `cout`. That done, sort the words using the `sort()` generic algorithm.**

```
#include <algorithm>sort(container.begin(),
container.end());
```

**Then print the sorted words to an output file.**

I open both the input and the output file before reading in and sorting the text. I could wait to open the output file, but what would happen if for some reason the output file failed to open? Then all the computations would have been for nothing. (The file paths are hard-coded and reflect Windows conventions. The `algorithm` header file contains the forward declaration of the `sort()` generic algorithm.)

```
#include <iostream>
#include <fstream>
#include <algorithm>
#include <string>
#include <vector>

using namespace std;
int main()
{
    ifstream in_file("C:\\My Documents\\text.txt");if
```

```

(! in_file){ cerr << "oops! unable to open input
file\n"; return -1; }

ofstream out_file("C:\\My Documents\\text.sort");if
(! out_file){ cerr << "oops! unable to open input
file\n"; return -2; }

string word;
vector< string > text;
while (in_file >> word)

    text.push_back(word);

int ix;
cout << "unsorted text: \n";

for (ix = 0; ix < text.size(); ++ix)
    cout << text[ix] << ' ';
    cout << endl;

    sort(text.begin(), text.end());

out_file << "sorted text: \n";
for (ix = 0; ix < text.size(); ++ix)
out_file << text[ix] << ' ';
out_file << endl;

return 0;}

```

The input text file consists of the following three lines:

```

we were her pride of ten she named us:Phoenix, the
Prodigal, Benjamin,and perspicacious, pacific
Suzanne.

```

When compiled and executed, the program generates the following output (I've inserted line breaks to display it here on the page):

```

Benjamin, Phoenix, Prodigal, Suzanne.and her named of
pacific perspicacious,pride she ten the us: we were

```

## Упражнение 1.8

The `switch` statement of [Section 1.4](#) displays a different consolation message based on the number of wrong guesses. Replace this with an array of four string messages that can be indexed based on the number of wrong guesses.

The first step is to define the array of string messages in which to index. One strategy is to encapsulate them in a display function that, passed the number of incorrect user guesses, returns the appropriate consolation message. Here is a first implementation. Unfortunately, it is not correct. Do you see the problems?

```

const char* msg_to_usr(int num_tries){ static const char*
usr_msgs[] = {
    "Oops! Nice guess but not quite it.",
    "Hmm. Sorry. Wrong again.",
    "Ah, this is harder than it looks, isn't it?",
    "It must be getting pretty frustrating by now!"

};return usr_msgs[num_tries];}

```

The index is off by one. If you flip back to the [Section 1.4](#) `switch` statement, you'll see that the number of incorrect tries begins with 1 because, after all, we are responding to wrong guesses on the user's part. Our array of responses, however, begins at position 0. So our responses are always one guess more severe than called for.

There are other problems as well. The user can potentially try more than four times and be wrong with each try, although I capped the number of unique messages at 4. If we unconditionally index into the array, a value of 4 or greater will overflow the array boundary. Moreover, we must guard against other potential invalid values such as a negative number.

Here is a second iteration. I've added a new first message in case the user somehow has not yet guessed. I don't expect we'll actually return it, but in this way, the other messages at least are in their "natural" position. I defined a `const` object to hold a count of the number of entries in the array.

```

const char* msg_to_usr(int num_tries)
{ const int rsp_cnt = 5;static const char* usr_msgs[rsp_cnt] = {
    "Go on, make a guess. ",
    "Oops! Nice guess but not quite it.",
    "Hmm. Sorry. Wrong again.",
    "Ah, this is harder than it looks, no?",
    "It must be getting pretty frustrating by now!"

};
if (num_tries < 0)

    num_tries = 0;
else
if (num_tries >= rsp_cnt)

    num_tries = rsp_cnt-1;return usr_msgs[num_tries];}

```

## Упражнение 2.1

`main()` [in [Section 2.1](#)] allows the user to enter only one position value and then terminates. If a user wishes to ask for two or more positions, he must execute the program two or more times. Modify `main()` [in [Section 2.1](#)] to allow the user to keep entering positions until he indicates he wishes to stop.

We use a `while` loop to execute the "solicit position, return value" code sequence. After each iteration, we ask the user whether he wishes to continue. The loop terminates when he answers no. We'll jump-start the first iteration by setting the `bool` object `more` to `true`.

```

#include <iostream> using
namespace std;

extern bool fibon_elem(int, int&); int main()

```

```

{ int pos, elem;char ch;bool more = true;

  while (more)
  {
    cout << "Please enter a position: ";
    cin >> pos;

    if (fibon_elem(pos, elem))cout << "element # " << pos<<
      " is " << elem << endl;else cout << "Sorry. Could
      not calculate element # "<< pos << endl;

    cout << "would you like to try again? (y/n) ";
    cin >> ch;
    if (ch != 'y' || ch != 'Y')

      more = false;
  }
}

```

When compiled and executed, the program generates the following output (my input is highlighted in bold):

```

Please enter a position: 4 element # 4 is 3 would you like to try again? (y/n)
yPlease enter a position: 8 element # 8 is 21 would you like to try again? (y/n)
yPlease enter a position: 12 element # 12 is 144 would you like to try again? (y/n)
n

```

## Упражнение 2.2

The formula for the Pentagonal numeric sequence is  $P_n = n * (3n - 1) / 2$ . This yields the sequence 1, 5, 12, 22, 35, and so on. Define a function to fill a vector of elements passed in to the function calculated to some user-specified position. Be sure to verify that the position specified is valid. Write a second function that, given a vector, displays its elements. It should take a second parameter identifying the type of numeric series the vector represents. Write a `main()` function to Упражнение these functions.

```

#include <vector> #include
<string>#include <iostream> using
namespace std;

    bool calc_elements(vector<int> &vec, int pos);void
    display_elems(vector<int> &vec,const string &title,
    ostream &os=cout);

int main(){
  vector<int> pent;
  const string title("Pentagonal Numeric Series");

  if (calc_elements(pent, 0))
    display_elems(pent, title);

  if (calc_elements(pent, 8))
    display_elems(pent, title);

```

```

    if (calc_elements(pent, 14))
        display_elems(pent, title);

    if (calc_elements(pent, 138))display_elems(pent, title);}

bool calc_elements(vector<int> &vec, int pos){
    if (pos <= 0 || pos > 64){
        cerr << "Sorry. Invalid position: " << pos << endl;
        return false;

    }
    for (int ix = vec.size()+1; ix <= pos; ++ix)
        vec.push_back((ix*(3*ix-1))/2);

    return true;}

void display_elems(vector<int> &vec,const string &title, ostream &os)
{ os << '\n' << title << "\n\t";for (int ix = 0; ix < vec.size(); ++ix)
    os<< vec[ix] << ' ';os << endl;}

```

When compiled and executed, this program generates the following output:

```

Sorry. Invalid position: 0

Pentagonal Numeric Series1 5 12 22 35 51
70 92

Pentagonal Numeric Series1 5 12 22 35 51 70 92 117 145 176 210
247 287 Sorry. Invalid position: 138

```

## Упражнение 2.3

**Separate the function to calculate the Pentagonal numeric sequence implemented in [Упражнение 2.2](#) into two functions. One function should be inline; it checks the validity of the position. A valid position not as yet calculated causes the function to invoke a second function that does the actual calculation.**

I factored `calc_elements()` into the inline `calc_elems()` that if necessary calls the second function, `really_calc_elems()`. To test this reimplementaion, I substituted the call of `calc_elements()` in the [Упражнение 2.2](#) function with that of `calc_elems()`.

```

extern void really_calc_elems(vector<int>&, int);inline bool
calc_elems(vector<int> &vec, int pos){
    if (pos <= 0 || pos > 64){
        cerr << "Sorry. Invalid position: " << pos << endl;
        return false;

    }

    if (vec.size() < pos)
        really_calc_elems(vec, pos);

```

```

return true;}void really_calc_elems(vector<int> &vec, int pos){
    for (int ix = vec.size()+1; ix <= pos; ++ix)vec.push_back((ix*(3*ix-
        1))/2);}

```

## Упражнение 2.4

**Introduce a static local vector to hold the elements of your Pentagonal series. This function returns a const pointer to the vector. It accepts a position by which to grow the vector if the vector is not that size as yet. Implement a second function that, given a position, returns the element at that position. Write a main() function to Упражнение these functions.**

```

#include <vector> #include <iostream>
using namespace std;

inline bool check_validity(int pos){ return (pos <= 0 || pos > 64) ? false : true;
}

bool pentagonal_elem(int pos, int &elem){
    if (! check_validity(pos)){
        cout << "Sorry. Invalid position: " << pos << endl;
        elem = 0;
        return false;
    }
    const vector<int> *pent = pentagonal_series(pos);
    elem = (*pent)[pos-1];
    return true;
}

const vector<int>* pentagonal_series(int
pos){
    static vector<int> _elems;if (check_validity(pos) && (pos >
        _elems.size()))for (int ix = _elems.size()+1; ix <= pos;
        ++ix)_elems.push_back((ix*(3*ix-1))/2);return &_elems;}
int main(){int elem;if (pentagonal_elem(8, elem))
    cout << "element 8 is " << elem << '\n';
    if (pentagonal_elem(88, elem))
        cout << "element 88 is " << elem << '\n';
    if (pentagonal_elem(12, elem))
        cout << "element 12 is " << elem << '\n';

    if (pentagonal_elem(64, elem))
        cout << "element 64 is " << elem << '\n';
}

```

When compiled and executed, this program generates the following:

```

element 8 is 92 Sorry. Invalid position:
88element 12 is 210 element 64 is 6112

```

## Упражнение 2.5

**Implement an overloaded set of `max()` functions to accept (a) two integers, (b) two floats, (c) two strings, (d) a vector of integers, (e) a vector of floats, (f) a vector of strings, (g) an array of integers and an integer indicating the size of the array, (h) an array of floats and an integer indicating the size of the array, and (i) an array of strings and an integer indicating the size of the array. Again, write a `main()` function to Упражнение these functions.**

```
#include <string>#include <vector>
#include <algorithm>

using namespace std;

inline int max(int t1, int t2){ return t1 > t2 ?
    t1 : t2; }

inline float max(float t1, float t2){ return t1 >
    t2 ? t1 : t2; }

inline string max(const string& t1, const string& t2){ return t1 >
    t2 ? t1 : t2; }

inline int max(const vector<int> &vec){ return *max_element(vec.begin(),
    vec.end()); }

inline float max(const vector<float> &vec){ return *max_element(vec.begin(),
    vec.end()); }

inline string max(const vector<string> &vec){ return *max_element(vec.begin(),
    vec.end()); }

inline int max(const int *parray, int size){ return
    *max_element(parray, parray+size); }

inline float max(const float *parray, int size){ return
    *max_element(parray, parray+size); }
inline string max(const string *parray, int size){ return
    *max_element(parray, parray+size); }

int main() {string sarray[]={ "we", "were", "her", "pride", "of", "ten"
    };vector<string> svec(sarray, sarray+6);

    int iarray[]={ 12, 70, 2, 169, 1, 5, 29 };
    vector<int> ivec(iarray, iarray+7);

    float farray[]={ 2.5, 24.8, 18.7, 4.1, 23.9 };
    vector<float> fvec(farray, farray+5);

    int imax = max(max(ivec), max(iarray, 7));
    float fmax = max(max(fvec), max(farray, 5));
    string smax = max(max(svec), max(sarray, 6));

    cout << "imax should be 169 -- found: " << imax << '\n' << "fmax should
        be 24.8 -- found: " << fmax << '\n' << "smax should be were --
        found: " << smax << '\n';}
```

When compiled and executed, this program generates the following:

```
imax should be 169 -- found: 169 fmax should be
24.8 -- found: 24.8 smax should be were --
found: were
```

## Упражнение 2.6

Reimplement the functions of [Упражнение 2.5](#) using templates. Modify the `main()` function accordingly.

The nine nontemplate `max()` functions are replaced by three `max()` function templates. `main()` does not require any changes.

```
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

template <typename Type>
inline Type max(Type t1, Type t2){ return t1 > t2 ? t1 : t2; }

template <typename elemType>inline elemType max(const
vector<elemType> &vec){ return *max_element(vec.begin(),
vec.end()); }

template <typename arrayType>inline arrayType max(const arrayType
*parray, int size){ return *max_element(parray, parray+size); }

// note: no changes required of main()!int main() { // same as in
Упражнение 2.4}
```

When compiled and executed, this program generates the same output as the program in [Упражнение 2.5](#).

## Упражнение 3.1

Write a program to read a text file. Store each word in a map. The key value of the map is the count of the number of times the word appears in the text. Define a word exclusion set containing words such as *a*, *an*, *or*, *the*, *and*, and *but*. Before entering a word in the map, make sure it is not present in the word exclusion set. Display the list of words and their associated count when the reading of the text is complete. As an extension, before displaying the text, allow the user to query the text for the presence of a word.

```
#include <map>#include <set> #include
<string>#include <iostream> #include
<fstream> using namespace std;

void initialize_exclusion_set(set<string>&);void process_file(map<string,int>&,
const set<string>&, ifstream&);void user_query(const map<string,int>&);void
display_word_count(const map<string,int>&, ofstream&);
```

```

int main()
{ ifstream ifile("C:\\My Documents\\column.txt");ofstream ofile("C:\\My
  Documents\\column.map");if (! ifile || ! ofile){
  cerr << "Unable to open file -- bailing out!\n";
  return -1;
  }

  set<string> exclude_set;
  initialize_exclusion_set(exclude_set);

  map<string,int> word_count;
  process_file(word_count, exclude_set, ifile);
  user_query(word_count);
  display_word_count(word_count, ofile);
}

void initialize_exclusion_set(set<string> &exs){
  static string _excluded_words[25] = {
    "the","and","but","that","then","are","been",
    "can","a","could","did","for","of",
    "had","have","him","his","her","its","is",
    "were","which","when","with","would"
  };

  exs.insert(_excluded_words, _excluded_words+25);}

  void process_file(map<string,int>
    &word_count,const
    set<string> &exclude_set,
    ifstream &ifile){ string
    word;
    while (ifile >> word){ if
    (exclude_set.count(word))continue;word_count[word]++;}}

void user_query(const map<string,int> &word_map)
{ string search_word;cout << "Please enter a word to search: q to quit";cin >>
  search_word;while (search_word.size() && search_word != "q"){
  map<string,int>::const_iterator it;if ((it = word_map.find(search_word))!=
  word_map.end())
    cout << "Found! " << it->first << " occurs " <<
    it->second << " times.\n";
  else cout << search_word
    << " was not found in text.\n";cout <<
    "\nAnother search? (q to quit) ";cin >> search_word;
  }}

void
display_word_count(const map<string,int> &word_map, ofstream &os)
{
  map<string,int>::const_iterator
  iter = word_map.begin(),
  end_it = word_map.end();

```

```

        while (iter != end_it){
            os << iter->first << " ("
            << iter->second << ")" << endl;

            ++iter;
        }
    os << endl;
}

```

Here is a small piece of text processed by the program. I removed the punctuation from the text because our program does not handle punctuation:

```

MooCat is a long-haired white kitten with largeblack patches
Like a cow looks only he is a kittypoor kitty Alice says
cradling MooCat in her armspretending he is not struggling to
break free

```

Here is a snapshot of the interactive session initiated by `user_query()`. Notice that although *a* occurs two times the text, it is an entry in the excluded word set and is not entered in the map of words found in the text.

```

Please enter a word to search: q to quit AliceFound! Alice
occurs 1 times.

```

```

Another search? (q to quit) MooCatFound! MooCat
occurs 2 times.

```

```

Another search? (q to quit) aa was not found
in text.

```

```

Another search? (q to quit) q

```

## Упражнение 3.2

**Read in a text file — it can be the same one as in Упражнение 3.1 — storing it in a vector. Sort the vector by the length of the string. Define a function object to pass to `sort()`; it should accept two strings and return `true` if the first string is shorter than the second. Print the sorted vector.**

Let's begin by defining the function object to pass to `sort()`:

```

class LessThan {public:bool operator()(const string & s1,const
string & s2){ return s1.size() < s2.size(); }};

```

The call to `sort` looks like this:

```

sort(text.begin(), text.end(), LessThan());

```

The main program looks like this:

```

int main()
{ ifstream ifile("C:\\My Documents\\MooCat.txt");ofstream ofile("C:\\My
Documents\\MooCat.sort");

    if (! ifile || ! ofile){
        cerr << "Unable to open file -- bailing out!\n";
    }
}

```

```

        return -1;
    }

    vector<string> text;
    string word;

    while (ifile >> word)
        text.push_back(word);

    sort(text.begin(), text.end(), LessThan()); display_vector(text, ofile);}

```

`display_vector()` is a function template parameterized on the element type of the vector passed to it to display:

```

template <typename elemType>void display_vector(const
vector<elemType> &vec,ostream &os=cout, int len= 8){
vector<elemType>::const_iterator iter = vec.begin(),
    end_it = vec.end();

int elem_cnt = 1;while (iter != end_it)os <<
*iter++ << (!(elem_cnt++ % len) ? '\n' : ' ');os <<
endl;}

```

The input file is the same text used in [Упражнение 3.1](#). The output of this program looks like this:

```

a a a is to in is he is he not cow her says poor onlyLike arms with free break
Alice kitty kittylooks black large white MooCat kitten MooCat patchescradling
pretending struggling long-haired

```

If we want to sort the words within each length alphabetically, we would first invoke `sort()` with the default less-than operator and then invoke `stable_sort()`, passing it the `LessThan` function object. `stable_sort()` maintains the relative order of elements meeting the same sorting criteria.

## Упражнение 3.3

**Define a map for which the index is the family surname and the key is a vector of the children's names. Populate the map with at least six entries. Test it by supporting user queries based on a surname and printing all the map entries.**

The map uses a string as an index and a vector of strings for the children's names. This is declared as follows:

```
map< string, vector<string> > families;
```

To simplify the declaration of the map, I've defined a typedef to alias `vstring` as an alternative name for a vector that contains string elements. (You likely wouldn't have thought of this because the typedef mechanism is not introduced until [Section 4.6](#). The typedef mechanism allows us to provide an alternative name for a type. It is generally used to simplify the declaration of a complex type.)

```
#include <map>typedef vector<string>
vstring;map< string, vstring > families;
```

We get our family information from a file. Each line of the file stores the family name and the name of each child:

```
surname child1 child2 child3 ... childN
```

Reading the file and populating the map are accomplished in `populate_map()`:

```
void populate_map(istream &nameFile, map<string,vstring> &families)
{
    string textline;
    while (getline(nameFile, textline))
    {
        string fam_name;
        vector<string> child;
        string::size_type
            pos = 0, prev_pos = 0,
            text_size = textline.size();

        // ok: find each word separated by a space
        while ((pos = textline.find_first_of(' ', pos))
            != string::npos)
        {
            // figure out end points of the substring
            string::size_type end_pos = pos - prev_pos;

            // if prev_pos not set, this is the family name// otherwise,
            we are reading the children ...if (! prev_pos)
                fam_name = textline.substr(prev_pos, end_pos);else
                child.push_back(textline.substr(prev_pos,end_pos));prev_pos = ++pos;}

        // now handle last childif (prev_pos <
        text_size)child.push_back(textline.substr(pr
        ev_pos,pos-prev_pos));

        if (! families.count(fam_name))families[fam_name] =
            child;else cerr << "Oops! We already have a "<<
            fam_name << " family in our map!\n";}}
```

`getline()` is a standard library function. It reads a line of the file up to the delimiter specified by its third parameter. The default delimiter is the newline character, which is what we use. The line is placed in the second parameter string.

The next portion of the function pulls out, in turn, the family surname and the name of each child. `substr()` is a string class operation that returns a new string object from the substring delimited by the two position arguments. Finally, the family is entered into the map if one is not already present.

To display the map, we define a `display_map()` function. It prints entries in this general format:

```
The lippman family has 2 children: danny anna
```

Here is the `display_map()` implementation:

```
void display_map(const map<string,vstring> &families, ostream &os){
    map<string,vstring>::const_iterator
        it = families.begin(),
        end_it = families.end();

    while (it != end_it)
    {
        os << "The " << it->first << " family ";
        if (it->second.empty())

            os << "has no children\n"; else
        { // print out vector of childrens << "has " << it->second.size() << "
            children: ";vector<string>::const_iterator
            iter = it->second.begin(),end_iter = it->second.end();while (iter !=
            end_iter){ os << *iter << " "; ++iter; }
            os << endl;
        }
        ++it;
    }
}
```

We must also allow users to query the map as to the presence of a family. If the family is present, we display the family name and number of children in the same way that `display_map()` does for the entire map. (As an Упражнение, you might factor out the common code between the two functions.) I've named this function `query_map()`.

```
void query_map(const string &family,const
    map<string,vstring> &families){
    map<string,vstring>::const_iterator it =
    families.find(family);

    if (it == families.end()){
        cout << "Sorry. The " << family
        << " is not currently entered.\n";
        return;
    }

    cout << "The " << family;
    if (! it->second.size())
        cout << " has no children\n";

    else { // print out vector of childrens cout << " has " << it->second.size() << "
        children: ";vector<string>::const_iterator
        iter = it->second.begin(),end_iter = it-
        >second.end();while (iter != end_iter){ cout << *iter <<
        " "; ++iter; }cout << endl;}}
```

The `main()` program is implemented as follows:

```
int main()
{ map< string, vstring > families;ifstream nameFile("C:\\My
    Documents\\families.txt");

    if (! nameFile) {cerr << "Unable to find families.txt file. Bailing
```

```

        Out!\n";return;
    }
    populate_map(nameFile, families);

    string family_name;
    while (1){ //!! loop until user says to quit ...

        cout << "Please enter a family name or q to quit ";
        cin >> family_name;
        if (family_name == "q")

            break;
        query_map(family_name, families);
    }
    display_map(families);
}

```

The `families.txt` file contains the following six entries:

```

lippman danny annasmith john henry friedamailer tommy june Franz orlen orley ranier
alphonse lou robert brodie

```

When compiled and executed, the program generates the following results. My responses are highlighted in bold.

```

Please enter a family name or q to quit ranier The ranier family has 4 children:
alphonse lou robert brodiePlease enter a family name or q to quit franz The franz
family has no children Please enter a family name or q to quit kafka Sorry. The
kafka family is not currently entered.Please enter a family name or q to quit qThe
franz family has no childrenThe lippman family has 2 children: danny annaThe mailer
family has 2 children: tommy juneThe orlen family has 1 children: orleyThe ranier
family has 4 children: alphonse lou robert brodieThe smith family has 3 children:
john henry Frieda

```

## Упражнение 3.4

**Write a program to read a sequence of integer numbers from standard input using an `istream_iterator`. Write the odd numbers into one file using an `ostream_iterator`. Each value should be separated by a space. Write the even numbers into a second file, also using an `ostream_iterator`. Each of these values should be placed on a separate line.**

To read a sequence of integers from standard input, we define two `istream_iterator`s: one bound to `cin`, and the second representing end-of-file.

```

istream_iterator<int> in(cin), eos;

```

Next, we define a vector to hold the elements read:

```

vector< int > input;

```

To perform the reading, we use the `copy()` generic algorithm:

```

#include <iterator> #include <vector>
#include <iostream> #include
<algorithm>using namespace std;

int main()
{
    vector< int > input;
    istream_iterator in(cin), eos;

    copy(in, eos, back_inserter(input));

    // ...}

```

The `back_inserter()` is necessary because `copy()` uses the assignment operator to copy each element. Because `input` is empty, the first element assignment would cause an overflow error. `back_inserter()` overrides the assignment operator. The elements are now inserted using `push_back()`.

We partition the elements into even and odd using the `partition()` generic algorithm and an `even_elem` function object that evaluates to `true` if the value is even:

```

class even_elem {public:bool operator()(int elem){ return
elem%2 ? false : true; }};

vector<int>::iterator division = partition(input.begin(), input.end(),
    even_elem());

```

We need two `ostream_iterator`s: one for the even number file and one for the odd number file. We first open two files for output using the `ofstream` class:

```

#include <fstream>
ofstream even_file("C:\\My Documents\\even_file"),
    odd_file("C:\\My Documents\\odd_file");

if (! even_file || ! odd_file)
{
    cerr << "arghh!! unable to open the output files. bailing out!";
    return -1;}

```

We bind our two `ostream_iterator`s to the respective `ofstream` objects. The second string argument indicates the delimiter to output following the output of each element.

```

ostream_iterator<int> even_iter(even_file,
    "\n"),odd_iter(odd_file, " ");

```

Finally, we use the `copy()` generic algorithm to output the partitioned elements:

```

copy(input.begin(), division, even_iter);copy(division,
input.end(), odd_iter);

```

For example, I entered the following sequence of numbers:

```
2 4 5 3 9 5 2 6 8 1 8 4 5 7 3
```

At that point, `even_file` contained the following values: 2 4 4 8 8 6 2, whereas `odd_file` contained these values: 5 9 1 3 5 5 7 3. The `partition()` algorithm does not preserve the order of the values. If

preserving the order of the values is important, we would instead use the `stable_partition()` generic algorithm. Both the `stable_sort()` and `stable_partition()` algorithms maintain the elements' relative order.

## Упражнение 4.1

**Create a `Stack.h` and a `Stack.suffix`, where `suffix` is whatever convention your compiler or project follows. Write a `main()` function to Упражнение the full public interface, and compile and execute it. Both the program text file and `main()` must include `Stack.h`:**

```
#include "Stack.h"
```

The header file for our Stack class contains the necessary header file inclusions and the actual class declaration:

```
#include <string>#include <vector> using
namespace std;

class Stack {
public:bool push(const string&);bool pop (string &elem);bool peek(string
    &elem);bool empty() const { return _stack.empty(); }bool full() const { return
    _stack.size() == _stack.max_size(); }int size() const { return _stack.size(); }

private:vector<string> _stack;};
```

The Stack program text file contains the definition of the `push()`, `pop()`, and `peek()` member functions. Under Visual C++, the file is named `Stack.cpp`. It must include the Stack class header file.

```
#include "Stack.h"
bool Stack::pop(string &elem){if (empty())
    return false;elem =
    _stack.back();_stack.pop_back();return
    true;}

bool Stack::peek(string &elem){if (empty())
    return false;elem =
    _stack.back();return true;}

bool Stack::push(const string &elem){if (full())
    return false;_stack.push_back(elem);
    return true;}
```

Here is a small program to Упражнение the Stack class interface. It reads in a sequence of strings from standard input, pushing each one onto the stack until either end-of-file occurs or the stack is full:

```
int main() {Stack st;string str;

    while (cin >> str && ! st.full())
        st.push(str);

    if (st.empty()) {cout << '\n' << "Oops: no strings
        were read -- bailing out\n";
```

```

        return 0;
    }
    st.peek(str);
    if (st.size() == 1 && str.empty()) {

        cout << '\n' << "Oops: no
strings were read -- bailing out\n";
return 0;}cout << '\n' << "Read in " <<
st.size() << " strings!\n"<< "The
strings, in reverse order: \n";

    while (st.size())
        if (st.pop(str))
            cout << str << ' ';

    cout << '\n' << "There are now " << st.size()<< " elements in the
stack!\n";}

```

To test the program, I typed in the last sentence of the James Joyce novel, *Finnegans Wake*. The following is the output generated by the program (my input is in bold):

```

A way a lone a last a loved a long the
Read in 11 strings!
The strings, in reverse order:
the long a loved a last a lone a way A
There are now 0 elements in the stack!

```

## Упражнение 4.2

**Extend the Stack class to support both a `find()` and a `count()` operation. `find()` returns true or false depending on whether the value is found. `count()` returns the number of occurrences of the string. Reimplement the `main()` of Упражнение 4.1 to invoke both functions.**

We implement these two functions simply by using the corresponding generic algorithms of the same names:

```

#include <algorithm>
bool Stack::find(const string &elem) const
    {vector<string>::const_iterator end_it = _stack.end();return
    ::find(_stack.begin(), end_it, elem) != end_it; }

int Stack::count(const string &elem) const{ return ::count(_stack.begin(),
    _stack.end(), elem); }

```

The global scope operator is necessary for the invocation of the two generic algorithms. Without the global scope operator, for example, the unqualified invocation of `find()` within `find()` recursively invokes the member instance of `find()`! The Stack class declaration is extended to include the declarations of these two functions:

```

class Stack {
public:bool find(const string &elem) const;int count(const string &elem) const;

    // ... everything else the same ...};

```

The program now inquires of the user which word she would like to search for and reports whether it is within the stack and, if so, how many times it occurs:

```
int main()
{ Stack st;string str;while (cin >> str && ! st.full())
    st.push(str);
    // check for empty stack as before ...

    cout << '\n' << "Read in " << st.size() << " strings!\n";cin.clear();
    // clear end-of-file set ...

    cout << "what word to search for? ";
    cin >> str;

    bool found = st.find(str);
    int count = found ? st.count(str) : 0;

    cout << str << (found ? " is " : " isn't ") << "in the
    stack. ";if (found)cout << "It occurs " << count << "
    times\n";}
```

Here is an interactive execution of the program. The items highlighted in bold are what I entered:

```
A way a lone a last a loved a long the
Read in 11 strings!
what word to search for? a
a is in the stack. It occurs 4 times
```

## Упражнение 4.3

**Consider the following global data:**

```
string program_name;string
version_stamp;int version_number;
int tests_run;int tests_passed;
```

**Write a class to wrap around this data.**

Why might we wish to do this? By wrapping these global objects within a class, we encapsulate their direct access within a small set of functions. Moreover, the names of the objects are now hidden behind the scope of the class and cannot clash with other global entities. Because we wish only a single instance of each global object, we declare each one to be a static class member as well as the member functions that access them.

```
#include <string>using
std::string;

class globalWrapper {
public:
    static int tests_passed() { return _tests_passed; }
```

```

static int tests_run() { return _tests_run; }
static int version_number() { return _version_number; }
static string version_stamp() { return _version_stamp; }
static string program_name() { return _program_name; }

static void tests_passed(int nval) { _tests_passed = nval; }
static void tests_run(int nval) { _tests_run = nval; }

static void version_number(int nval)
    { _version_number = nval; }

static void version_stamp(const string& nstamp)
    { _version_stamp = nstamp; }

static void program_name(const string& npn)
    { _program_name = npn; }

private:
    static string _program_name;
    static string _version_stamp;
    static int _version_number;
    static int _tests_run;
    static int _tests_passed;};

string globalWrapper::_program_name;string
globalWrapper::_version_stamp;int
globalWrapper::_version_number;int
globalWrapper::_tests_run;int
globalWrapper::_tests_passed;

```

## Упражнение 4.4

**A user profile consists of a login, the actual user name, the number of times logged on, the number of guesses made, the number of correct guesses, the current level — one of beginner, intermediate, advanced, guru — and the percentage correct (this latter may be computed or stored). Provide a UserProfile class. Support input and output, equality and inequality. The constructors should allow for a default user level and default login name of "guest." How might you guarantee that each guest login for a particular session is unique?**

```

class UserProfile {public:enum
uLevel { Beginner, Intermediate,
Advanced, Guru };

    UserProfile(string login, uLevel = Beginner);
    UserProfile();

    // default memberwise initialization and copy sufficient// no explicit copy
    constructor or copy assignment operator// no destructor necessary ...

    bool operator==(const UserProfile&);
    bool operator!=(const UserProfile &rhs);

    // read access functions

```

```

string login() const { return _login; }
string user_name() const { return _user_name; }
int login_count() const { return _times_logged; }
int guess_count() const { return _guesses; }
int guess_correct() const { return _correct_guesses; }
double guess_average() const;
string level() const;

// write access functions
void reset_login(const string &val){ _login = val; }
void user_name(const string &val){ _user_name = val; }

void reset_level(const string&);
void reset_level(uLevel newlevel) { _user_level = newlevel; }

void reset_login_count(int val){ _times_logged = val; }
void reset_guess_count(int val){ _guesses = val; }
void reset_guess_correct(int val){ _correct_guesses = val; }

void bump_login_count(int cnt=1){ _times_logged += cnt; }
void bump_guess_count(int cnt=1){ _guesses += cnt; }
void bump_guess_correct(int cnt=1){ _correct_guesses += cnt;}

private:
    string _login;
    string _user_name;
    int _times_logged;
    int _guesses;
    int _correct_guesses;
    uLevel _user_level;

    static map<string,uLevel> _level_map;
    static void init_level_map();
    static string guest_login();
};

inline double UserProfile::guess_average() const{
    return _guesses? double(_correct_guesses) / double(_guesses) * 100: 0.0;
}

    inline UserProfile::UserProfile(string login, uLevel level):
        _login(login), _user_level(level),_times_logged(1), _guesses(0),
        _correct_guesses(0){}
#include <cstdlib>

    inline UserProfile::UserProfile(): _login("guest"),
        _user_level(Beginner),_times_logged(1), _guesses(0),
        _correct_guesses(0)
{ static int id = 0;char buffer[16];

// _itoa() is a Standard C library function

```

```

// turns an integer into an ascii representation
_itoa(id++, buffer, 10);

// add a unique id during session to guest login_login += buffer;}

inline bool UserProfile::operator==(const UserProfile
&rhs){
    if (_login == rhs._login &&_user_name == rhs._user_name)return
        true;return false;}

inline bool UserProfile::
operator !=(const UserProfile &rhs){ return ! (*this == rhs); }

    inline string UserProfile::level() const {static string
        _level_table[] = {"Beginner", "Intermediate", "Advanced",
            "Guru" };return _level_table[_user_level];}

ostream& operator<<(ostream &os, const UserProfile &rhs){ // output of the
form: stanl Beginner 12 100 10 10%
    os << rhs.login() << ' '
        << rhs.level() << ' '
        << rhs.login_count() << ' '
        << rhs.guess_count() << ' '
        << rhs.guess_correct() << ' '
        << rhs.guess_average() << endl;

    return os;}

// overkill ... but it allows a demonstration
...map<string,UserProfile::uLevel> UserProfile::_level_map;

void UserProfile::init_level_map(){_level_map["Beginner"] =
    Beginner;_level_map["Intermediate"] =
    Intermediate;_level_map["Advanced"] =
    Advanced;_level_map["Guru"] = Guru;}

inline void UserProfile::reset_level(const string
    &level){map<string,uLevel>::iterator it;if (_level_map.empty())
        init_level_map();
        // confirm level is a recognized user level
        ..._user_level = ((it = _level_map.find(level)) !=
            _level_map.end())? it->second : Beginner;}

istream& operator>>(istream &is, UserProfile &rhs)
{ // yes, this assumes the input is valid ...
string login, level;
is >> login >> level;

    int lcount, gcount, gcorrect;
    is >> lcount >> gcount >> gcorrect;
    rhs.reset_login(login);
    rhs.reset_level(level);
    rhs.reset_login_count(lcount);
    rhs.reset_guess_count(gcount);
    rhs.reset_guess_correct(gcorrect);

```

```
return is;}
```

Here is a small program to Упражнение our UserProfile class:

```
int main()
{ UserProfile anon;cout << anon; // test out output operator UserProfile
anon_too; // to see if we get unique id
cout << anon_too;

UserProfile anna("AnnaL", UserProfile::Guru);
cout << anna;

anna.bump_guess_count(27);
anna.bump_guess_correct(25);
anna.bump_login_count();
cout << anna;

cin >> anon; // test out input operatorcout << anon;}
```

When compiled and executed, this program generates the following output (my responses are highlighted in bold):

```
quest0 Beginner 1 0 0 0quest1 Beginner 1 0 0 0AnnaL Guru 1 0 0 0 AnnaL Guru 2 27 25
92.5926
robin Intermediate 1 8 3
robin Intermediate 1 8 3 37.5
```

## Упражнение 4.5

**Implement a 4x4 Matrix class supporting at least the following general interface: addition and multiplication of two Matrix objects, a print() member function, a compound += operator, and subscripting supported through a pair of overloaded function call operators, as follows:**

```
float& operator()(int row, int column);float operator()(int
row, int column) const;
```

**Provide a default constructor taking an optional 16 data values and a constructor taking an array of 16 elements. You do not need a copy constructor, copy assignment operator, or destructor for this class (these are required in Chapter 6 when we reimplement the Matrix class to support arbitrary rows and columns).**

```
#include <iostream>
typedef float elemType; // for transition into a template

class Matrix

{ // friends are not affected by the access level they are// declared in. I like to
place them at class beginningfriend Matrix operator+(const Matrix&, const
```

```

Matrix&);friend Matrix operator*(const Matrix&, const Matrix&);

public:Matrix(const
    elemType*);Matrix(elemType=0.,elemType=0.,elemType=0.,elemType=0.,
        elemType=0.,elemType=0.,elemType=0.,elemType=0.,elemType=0.,ele
            mType=0.,elemType=0.,elemType=0.,elemType=0.,elemType=0.,elemTy
                pe=0.,elemType=0.);

    // don't need copy constructor, destructor,
    // or copy assignment operator for the Matrix class

    // simplifies transition to general matrix
    int rows() const { return 4; }
    int cols() const { return 4; }

    ostream& print(ostream&) const;
    void operator+=(const Matrix&);

    elemType operator()(int row, int column) const
        { return _matrix[row][column]; }

    elemType& operator()(int row, int column){ return
        _matrix[row][column]; }private:elemType _matrix[4][4];};

inline ostream& operator<<(ostream& os, const Matrix &m){ return
    m.print(os); }

Matrix operator+(const Matrix &m1, const Matrix &m2){Matrix result(m1);result +=
    m2;return result;}
Matrix operator*(const Matrix &m1, const Matrix &m2){Matrix result;for (int ix = 0;
    ix < m1.rows(); ix++)
    for (int jx = 0; jx < m1.cols(); jx++){result(ix, jx) = 0;for (int kx = 0;
        kx < m1.cols(); kx++)
            result(ix, jx) += m1(ix, kx) * m2(kx,
                jx);}return result;}

    void Matrix::operator+=(const Matrix &m){for (int ix = 0; ix < 4;
        ++ix)for (int jx = 0; jx < 4; ++jx)_matrix[ix][jx] +=
            m._matrix[ix][jx];}

ostream& Matrix::print(ostream &os) const {int cnt = 0;for (int ix = 0; ix < 4;
    ++ix)
    for (int jx = 0; jx < 4; ++jx, ++cnt){if (cnt && !(cnt %
        8)) os << endl;os << _matrix[ix][jx] << ' ';
    }
    os << endl;
    return os;
}

Matrix::Matrix(const elemType *array){int array_index = 0;for (int ix = 0; ix < 4;
    ++ix)
    for (int jx = 0; jx < 4; ++jx)_matrix[ix][jx] =
        array[array_index++];}

```

```

Matrix::Matrix(elemType a11, elemType a12, elemType a13, elemType a14,elemType a21,
               elemType a22, elemType a23, elemType a24,elemType a31, elemType a32,
               elemType a33, elemType a34,elemType a41, elemType a42, elemType a43,
               elemType a44)
{
    _matrix[0][0] = a11; _matrix[0][1] = a12;_matrix[0][2] = a13; _matrix[0][3] =
    a14;_matrix[1][0] = a21; _matrix[1][1] = a22;_matrix[1][2] = a23; _matrix[1][3]
    = a24;_matrix[2][0] = a31; _matrix[2][1] = a32;_matrix[2][2] = a33;
    _matrix[2][3] = a34;_matrix[3][0] = a41; _matrix[3][1] = a42;_matrix[3][2] =
    a43; _matrix[3][3] = a44;}

```

Here is a small program that Упражнения a portion of the Matrix class interface:

```

int main()
{
    Matrix m;cout << m << endl;

    elemType ar[16]={
        1., 0., 0., 0., 0., 1., 0., 0.,
        0., 0., 1., 0., 0., 0., 0., 1. };

    Matrix identity(ar);
    cout << identity << endl;

    Matrix m2(identity);m = identity;
    cout << m2 << endl; cout << m << endl;

    elemType ar2[16] = {
        1.3, 0.4, 2.6, 8.2, 6.2, 1.7, 1.3, 8.3,
        4.2, 7.4, 2.7, 1.9, 6.3, 8.1, 5.6, 6.6 };
    Matrix m3(ar2); cout << m3 << endl;

    Matrix m4 = m3 * identity; cout << m4 << endl;
    Matrix m5 = m3 + m4; cout << m5 << endl;
    m3 += m4; cout << m3 << endl;}

```

When compiled and executed, this program generates the following output:

```

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1
1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1
1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1
1.3 0.4 2.6 8.2 6.2 1.7 1.3 8.3
4.2 7.4 2.7 1.9 6.3 8.1 5.6 6.6
1.3 0.4 2.6 8.2 6.2 1.7 1.3 8.3
4.2 7.4 2.7 1.9 6.3 8.1 5.6 6.6
2.6 0.8 5.2 16.4 12.4 3.4 2.6 16.6
8.4 14.8 5.4 3.8 12.6 16.2 11.2 13.2
2.6 0.8 5.2 16.4 12.4 3.4 2.6 16.6
8.4 14.8 5.4 3.8 12.6 16.2 11.2 13.2

```

## Упражнение 5.1

**Implement a two-level stack hierarchy. The base class is a pure abstract Stack class that minimally supports the following interface: pop(), push(), size(), empty(), full(), peek(), and print(). The derived classes are LIFO\_Stack and Peekback\_Stack. The Peekback\_Stack allows the user to retrieve the value of any element in the stack without modifying the stack itself.**

The two derived classes implement the actual element container using a vector. To display the vector, I use a `const_reverse_iterator`. This supports traversing the vector from the back to the front.

```
#include <string>#include <iostream>
#include <vector> using namespace std;

typedef string elemType; class Stack {
public:virtual ~Stack(){}virtual bool pop(elemType&) = 0;virtual bool push(const
    elemType&) = 0;virtual bool peek(int index, elemType&) = 0;

    virtual int top() const = 0;
    virtual int size() const = 0;

    virtual bool empty() const = 0;
    virtual bool full() const = 0;
    virtual void print(ostream& =cout) const = 0;
};

ostream& operator<<(ostream &os, const Stack &rhs){ rhs.print(); return
    os; }

class LIFO_Stack : public Stack {public:LIFO_Stack(int
    capacity = 0) : _top(0)
    { if (capacity) _stack.reserve(capacity); }int size() const { return
    _stack.size(); }bool empty() const { return ! _top; }bool full() const { return
    size() >= _stack.max_size(); }int top() const { return _top; }void print(ostream
    &os=cout) const;

    bool pop(elemType &elem);
    bool push(const elemType &elem);
    bool peek(int, elemType&){ return false; }

private:vector< elemType > _stack;int _top;};

bool LIFO_Stack::pop(elemType &elem){if (empty()) return
    false;elem = _stack[--_top];_stack.pop_back();return
    true;}

bool LIFO_Stack::push(const elemType &elem){if (full()) return
    false;_stack.push_back(elem);++_top;return true;}

void LIFO_Stack::print(ostream &os=cout) const {
    vector<elemType>::const_reverse_iterator
        rit = _stack.rbegin(),
        rend = _stack.rend();

    os << "\n\t";
```

```

while (rit != rend)
os << *rit++ << "\n\t";

```

```

os << endl;
}

```

The implementation of the `Peekback_Stack` duplicates that of `LIFO_Stack` except for the implementation of `peek()`:

```

bool Peekback_Stack::
peek(int index, elemType &elem)
{
    if (empty())
        return false;

    if (index < 0 || index >= size())
        return false;

    elem = _stack[index];return true;}

```

Here is a small program that Упражнения the inheritance hierarchy. The nonmember `peek()` instance accepts an abstract `Stack` reference and invokes the virtual `peek()` member function, which is unique to each derived class.

```

void peek(Stack &st, int index)
{ cout << endl;string t;if (st.peek(index, t))
    cout << "peek: " << t;
  else cout << "peek failed!";
  cout << endl;
}

int main()
{ LIFO_Stack st;string str;while (cin >> str && ! st.full())
    st.push(str);

  cout << '\n' << "About to call peek() with LIFO_Stack" << endl;
  peek(st, st.top()-1);
  cout << st;

  Peekback_Stack pst;

  while (! st.empty()){
    string t;
    if (st.pop(t))

      pst.push(t);
  }

  cout << "About to call peek() with Peekback_Stack" << endl;
  peek(pst, pst.top()-1);
  cout << pst;
}

```

```
}
```

When compiled and executed, this program generates the following output:

```
once upon a timeAbout to call peek() with LIFO_Stackpeek failed!
    time
    a
    upon
    once

About to call peek() with Peekback_Stack
peek: once

    once
    upon
    a
    time
```

## Упражнение 5.2

**Reimplement the class hierarchy of Упражнение 5.1 so that the base Stack class implements the shared, type-independent members.**

The reimplementing of the class hierarchy illustrates a concrete class hierarchy; that is, we replace our pure abstract Stack class with our implementation of LIFO\_Stack, renaming it Stack. Although Stack serves as a base class, it also represents actual objects within our applications. Thus it is termed a concrete base class. Peekback\_Stack is derived from Stack. In this implementation, it inherits all the members of Stack except peek(), which it overrides. Only the peek() member function and the destructor of Stack are virtual. The definitions of the member functions are the same and are not shown.

```
class Stack {
public:Stack(int capacity = 0): _top(0){ if (capacity)
    _stack.reserve(capacity);
}
virtual ~Stack(){}

    bool pop(elemType&);
    bool push(const elemType&);
    virtual bool peek(int, elemType&)

        { return false; }

    int size() const { return _stack.size(); }
    int top() const { return _top; }

    bool empty() const { return ! _top; }
    bool full() const { return size() >= _stack.max_size(); }
    void print(ostream&=cout);
```

```
protected:vector<elemType> _stack;int
    _top;};

class Peekback_Stack : public Stack {public:Peekback_Stack(int
capacity = 0): Stack(capacity) {}
    bool peek(int index, elemType &elem);};
```

## Упражнение 5.3

**A type/subtype inheritance relationship in general reflects an *is-a* relationship: A range-checking `ArrayRC` is a kind of `Array`, a `Book` is a kind of `LibraryRentalMaterial`, an `AudioBook` is a kind of `Book`, and so on. Which of the following pairs reflects an *is-a* relationship?**

(a) member function `isA_kindOf` function

This reflects an *is-a* relationship. A member function is a specialized instance of a function. Both have a return type, a name, a parameter list, and a definition. In addition, a member function belongs to a particular class, may or may not be a virtual, const, or static member function, and so on. Inheritance correctly models the relationship.

(b) member function `isA_kindOf` class

This does not reflect an *is-a* relationship. A member function has a class that it is a member of, but it is not a specialized instance of a class. Inheritance incorrectly models the relationship.

(c) constructor `isA_kindOf` member function

This reflects an *is-a* relationship. A constructor is a specialized instance of a member function. A constructor must be a member function; however, it has specialized characteristics. Inheritance correctly models the relationship.

(d) `airplane isA_kindOf vehicle`

This reflects an *is-a* relationship. An airplane is a kind of vehicle. A vehicle is an abstract class. Airplane is also an abstract class and subsequently is likely to be inherited from. Inheritance correctly models the relationship.

(e) `motor isA_kindOf truck`

This does not reflect an *is-a* relationship. A motor is part of a truck. A truck has a motor. Inheritance incorrectly models the relationship.

(f) `circle isA_kindOf geometry`

This reflects an *is-a* relationship. A circle is a specialized instance of geometry — of 2D geometry. Geometry is an abstract base class. Circle is a concrete specialization of geometry. Inheritance correctly models the relationship.

(g) `square isA_kindOf rectangle`

This reflects an *is-a* relationship. Like a circle, a rectangle is a specialized instance of geometry. A square is a further specialization — a rectangle in which each side is equal. Inheritance correctly models the relationship.

(h) `automobile isA_kindOf airplane`

This is neither a has-a nor an *is-a* relationship. Both an automobile and an airplane are kinds of vehicles.

Inheritance incorrectly models the relationship.

```
(i) borrower isA_kindOf library
```

This does not reflect an is-a relationship. A borrower is an object (or component) of a library. A library has one or more borrowers. A borrower is a member of a library but is not a kind of library! Inheritance incorrectly models the relationship

## Упражнение 5.4

**A library supports the following categories of lending materials, each with its own check-out and check-in policy. Organize these into an inheritance hierarchy:**

```
book audio book record children's puppetvideo Sega  
video gamental book Sony Playstation video gameCD-  
ROM book Nintendo video game
```

An inheritance hierarchy moves from the most abstract to the most specific. In this example, we are given concrete instances of materials loaned by a library. Our task is twofold. First, we must group common abstractions: the four book abstractions and the three video game abstractions. Second, we must provide additional classes that can serve as an abstract interface for the concrete instances. This must be done at two levels: at the level of each family of concrete classes, such as our books, and at the level of the entire library lending hierarchy.

For example, Sega, Sony Playstation, and Nintendo video games are specific instances of video games. To tie them together, I've introduced an abstract video game class. I've designated the book class as a concrete base class with the three other kinds of books as specialized instances. Finally, we need a root base class of our library lending materials.

In the following hierarchy, each tab represents an inheritance relationship. For example, audio, rental, and CD-ROM are inherited from book, which in turn is inherited from the abstract library\_lending\_material.

```
library_lending_material  
  book  
    audio book  
    rental book  
    CD-ROM book  
  children's puppet  
  record  
  video  
  video game  
    Sega  
    Sony Playstation  
    Nintendo
```

## Упражнение 6.1

**Rewrite the following class definition to make it a class template:**

```
class example {
```

```

public:example(double min, double max);example(const double
    *array, int size);

    double& operator[](int index);
    bool operator==(const example& const);

    bool insert(const double*, int);
    bool insert(double);

    double min() const { return _min; }
    double max() const { return _max; }

    void min(double);
    void max(double);

    int count(double value) const;
private:int _size;double *_parray;double
    _min;double _max;};

```

To transform the example class into a template, we must identify and factor out each dependent data type. `_size`, for example, is of type `int`. Might that vary with different user-specified instances of `example`? No. `_size` is an invariant data member that holds a count of the elements addressed by `_parray`. `_parray`, however, may address elements of varying types: `int`, `double`, `float`, `string`, and so on. We want to parameterize the data type of the members `_parray`, `_min`, and `_max`, as well as the return type and signature of some of the member functions.

```

template <typename elemType>
class example {
public:

    example(const elemType &min, const elemType &max);example(const
    elemType *array, int size);

    elemType& operator[](int index);
    bool operator==(const example& const);

    bool insert(const elemType*, int);
    bool insert(const elemType&);

    elemType min() const { return _min; };
    elemType max() const { return _max; };

    void min(const elemType&);
    void max(const elemType&);

    int count(const elemType &value) const;
private:int _size;elemType
    *_parray;elemType _min;elemType
    _max;};

```

Because `elemType` now potentially represents any built-in or user-defined class, I declare the formal parameters

as `const` references rather than pass them by value.

## Упражнение 6.2

**Reimplement the Matrix class of Упражнение 5.3 as a template. In addition, extend it to support arbitrary row and column size using heap memory. Allocate the memory in the constructor and deallocate it in the destructor.**

The primary work of this Упражнение is to add general row and column support. We introduce a constructor that takes a row and a column size as arguments. The body of the constructor allocates the necessary memory from the program's free store:

```
Matrix(int rows, int columns)
    : _rows(rows), _cols(columns){ int size =
    _rows * _cols;
    _matrix = new elemType[size];
    for (int ix = 0; ix < size; ++ix)
        _matrix[ix] = elemType();}
```

`elemType` is the template parameter. `_matrix` is now an `elemType` pointer that addresses the heap memory allocated through the `new` expression.

```
template <typename elemType>
class Matrix {
public:

    // ...private:int _rows;
    int _cols;
    elemType *_matrix;};
```

It is not possible to specify an explicit initial value for which to set each element of the Matrix. The potential actual types that might be specified for `elemType` are simply too diverse. The language allows us, rather, to specify a default constructor:

```
_matrix[ix] = elemType();
```

For an `int`, this becomes `int()` and resolves to 0. For a `float`, this becomes `float()` and resolves to 0.0f. For a string, this becomes `string()` and invokes the default string constructor, and so on.

We must add a destructor to delete the heap memory acquired during the class object's construction:

```
~Matrix(){ delete [] _matrix; }
```

We also must provide a copy constructor and a copy assignment operator now. Default memberwise initialization and copy are no longer sufficient when we allocate memory in a constructor and deallocate it in a destructor. Under the default behavior, the `_matrix` pointer member of both class objects now addresses the same heap memory. This becomes particularly nasty if one object is destroyed while the other object continues to be active within the program: Its underlying matrix has been deleted! One solution is to *deep copy* the underlying matrix so that each class object addresses a separate instance:

```

template <typename elemType>Matrix<elemType>::Matrix(const
Matrix & rhs){
    _rows = rhs._rows; _cols = rhs._cols;
    int mat_size = _rows * _cols;
    _matrix = new elemType[mat_size];
    for (int ix = 0; ix < mat_size; ++ix)

        _matrix[ix] = rhs._matrix[ix];}

template <typename elemType>
Matrix<elemType>& Matrix<elemType>::operator=(const Matrix &rhs)
{

    if (this != &rhs){
        _rows = rhs._rows; _cols = rhs._cols;
        int mat_size = _rows * _cols;
        delete [] _matrix;
        _matrix = new elemType[mat_size];
        for (int ix = 0; ix < mat_size; ++ix)

            _matrix[ix] = rhs._matrix[ix];

    }

    return *this;}

```

**Here is the full class declaration and the remaining member functions:**

```

#include <iostream>
template <typename elemType>
class Matrix
{

    friend Matrix<elemType>operator+(const Matrix<elemType>&, const
        Matrix<elemType>&);

    friend Matrix< elemType >operator*(const Matrix<elemType>&, const
        Matrix<elemType>&);

public:Matrix(int rows, int columns);Matrix(const Matrix&);~Matrix();Matrix&
    operator=(const Matrix&);

    void operator+=(const Matrix&);
    elemType& operator()(int row, int column)
    { return _matrix[row * cols() + column]; }

    const elemType& operator()(int row, int column) const
    { return _matrix[row * cols() + column]; }

    int rows() const { return _rows; }
    int cols() const { return _cols; }

    bool same_size(const Matrix &m) const{ return rows() == m.rows() && cols() ==
        m.cols(); }

```

```

    bool comfortable(const Matrix &m) const
        { return (cols() == m.rows()); }

    ostream& print(ostream&) const;

protected:int _rows;int _cols;elemType
    *_matrix;
};

template <typename elemType>
inline ostream&
operator<<(ostream& os, const Matrix<elemType> &m)

    { return m.print(os); }

// end of Matrix.h

template <typename elemType>
Matrix< elemType >
operator+(const Matrix<elemType> &m1, const Matrix<elemType> &m2)
{

    // make sure m1 & m2 are same size
    Matrix<elemType> result(m1);
    result += m2;
    return result;

}

template <typename elemType>
Matrix<elemType>
operator*(const Matrix<elemType> &m1, const Matrix<elemType> &m2)
{

    // m1's columns must equal m2's rows ...
    Matrix<elemType> result(m1.rows(), m2.cols());
    for (int ix = 0; ix < m1.rows(); ix++) {

        for (int jx = 0; jx < m1.cols(); jx++) {result(ix, jx) = 0;for (int kx = 0;
            kx < m1.cols(); kx++)result(ix, jx) += m1(ix, kx) * m2(kx, jx);
        }

    }
    return result;

}

template <typename elemType>
void Matrix<elemType>::operator+=(const Matrix &m){// make sure m1 & m2 are same
    sizeint matrix_size = cols() * rows();for (int ix = 0; ix < matrix_size; ++ix)
        (*(_matrix + ix)) += (*(m._matrix + ix));}

template <typename elemType>
ostream& Matrix<elemType>::print(ostream &os) const {int col = cols();int

```

```

matrix_size = col * rows();for (int ix = 0; ix < matrix_size; ++ix){if (ix %
col == 0) os << endl;
    os << (*(_matrix + ix)) << ' ';
}
os << endl;

return os;}

```

Here is a small program to Упражнение our Matrix class template:

```

int main()
{ ofstream log("C:\\My Documents\\log.txt");if (! log)
    { cerr << "can't open log file!\n"; return; }

Matrix<float> identity(4, 4);
log << "identity: " << identity << endl;
float ar[16]={ 1., 0., 0., 0., 0., 1., 0., 0.,
              0., 0., 1., 0., 0., 0., 0., 1. };

    for (int i = 0, k = 0; i < 4; ++i)for (int j = 0; j < 4;
        ++j)identity(i, j) = ar[k++];log << "identity after
        set: " << identity << endl;

Matrix<float> m(identity);
log << "m: memberwise initialized: " << m << endl;

Matrix<float> m2(8, 12);
log << "m2: 8x12: " << m2 << endl;
m2 = m;
log << "m2 after memberwise assigned to m: "
    << m2 << endl;

float ar2[16]={ 1.3, 0.4, 2.6, 8.2, 6.2, 1.7, 1.3, 8.3,4.2, 7.4, 2.7, 1.9, 6.3,
              8.1, 5.6, 6.6 };

    Matrix<float> m3(4, 4);
    for (int ix = 0, kx = 0; ix < 4; ++ix)
    for (int j = 0; j < 4; ++j)
        m3(ix, j) = ar2[kx++];

log << "m3: assigned random values: " << m3 << endl;

Matrix<float> m4 = m3 * identity; log << m4 << endl;Matrix<float> m5 = m3 +
m4; log << m5 << endl;

m3 += m4; log << m3 << endl;}

```

When compiled and executed, the program generates the following output:

```

identity:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

identity after set:1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 m: memberwise initialized: 1 0 0
0 0 1 0 0 0 0 1 0 0 0 0 1

m2: 8x12: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```



the C-style character string representation, we invoke the `c_str()` string member function:

```
ifstream infile(file_name.c_str());
```

Following the definition of `infile`, we should check that it opened successfully:

```
if (!infile) // open failed ...
```

If `infile` did open successfully, the third statement executes but may not succeed. For example, if the file contained text, the attempt to place an element in `elem_cnt` fails. Alternatively, it is possible for the file to be empty.

```
if (!infile) // gosh, the read failed
```

Whenever we deal with pointers, we must be concerned as to whether they actually address an object. If `allocate_array()` was unable actually to allocate the array, `pi` is initialized to 0. We must test that:

```
if (!pi) // geesh, allocate_array() didn't really
```

The assumption of the program is that `elem_cnt` represents a count of the elements contained within the file. `index` is unlikely to overflow the array. However, we cannot guarantee that `index` is never greater than `elem_cnt` unless we check.

## Упражнение 7.2

**The following functions invoked in `alloc_and_init()` raise the following exception types if they should fail:**

```
allocate_array() noMemsort_array()
int register_data() string
```

**Insert one or more try blocks and associated catch clauses where appropriate to handle these exceptions. Simply print the occurrence of the error within the catch clause.**

Rather than surround each individual function invocation with a separate try block, I have chosen to surround the entire set of calls with a single try block that contains three associated catch clauses:

```
int *alloc_and_init(string file_name)
{
    ifstream infile(file_name.c_str());
    if (!infile) return 0;
    int elem_cnt;
    infile >> elem_cnt;
    if (!infile) return 0;

    try {
        int *pi = allocate_array(elem_cnt);
        int elem;
        int index = 0;
        while (infile >> elem && index < elem_cnt)
```

```

        pi[index++] = elem;

    sort_array(pi, elem_cnt);
    register_data(pi);
}
catch(const noMem &memFail) {

    cerr << "alloc_and_init(): allocate_array failure!\n" <<
        memFail.what() << endl;
    return 0;
}
catch(int &sortFail) {

    cerr << "alloc_and_init(): sort_array failure!\n" << "thrown
        integer value: " << sortFail << endl;
    return 0;
}
catch(string &registerFail) {

    cerr << "alloc_and_init(): register_data failure!\n" << "thrown
        string value: " << registerFail << endl;
    return 0;
}
return pi; // reach here only if no throw occurred ...
}

```

## Упражнение 7.3

**Add a pair of exceptions to the Stack class hierarchy of [Упражнение 6.2](#) to handle the cases of attempting to pop a stack that is empty and attempting to push a stack that is full. Show the modified `pop()` and `push()` member functions.**

We'll define a `PopOnEmpty` and a `PushOnFull` pair of exception classes to be thrown, respectively, in the `pop()` and `push()` `Stack` member functions. These classes no longer need to return a success or failure value:

```

void pop(elemType &elem){ if (empty())
    throw PopOnEmpty();
    elem = _stack[--_top];
    _stack.pop_back();
}
void push(const elemType &elem){
    if (! full()){
        _stack.push_back(elem);
        ++_top;
        return;
    }
    throw PushOnFull();
}

```

To allow these two `Stack` class exceptions to be caught in components that do not explicitly know about `PopOnEmpty` and `PushOnFull` exception classes, the class exceptions are made part of a `StackException` hierarchy that is derived

from the standard library `logic_error` class.

The `logic_error` class is derived from `exception`, which is the root abstract base class of the standard library exception class hierarchy. This hierarchy declares a virtual function `what()` that returns a `const char*` identifying the exception that has been caught.

```
class StackException : public logic_error {
public:StackException(const char *what) : _what(what){}const char *what() const {
    return _what.c_str(); }
protected:string _what;};

class PopOnEmpty : public StackException {public:PopOnEmpty()
: StackException("Pop on Empty Stack"){};};

class PushOnFull : public StackException
{public:PushOnFull() : StackException("Push on Full
Stack"){};};
```

Each of the following catch clauses handles an exception of type `PushOnFull`:

```
catch(const PushOnFull &pof){ log(pof.what());
    return; }

catch(const StackException &stke){
    log(stke.what()); return; }

catch(const logic_error &lge){ log(lge.what());
    return; }

catch(const exception &ex){ log(ex.what());
    return; }
```

## Appendix B. Generic Algorithms Handbook

Each generic algorithm, with the fistful of exceptions that make the rule, begins with a pair of iterators that marks the range of elements within the container over which to traverse. The range begins with the first iterator and ends with but does *not* include the second:

```
const int array_size = 7;
int iarray[array_size] = { 1, 10, 8, 4, 3, 14, 8 };
vector<int> vec(iarray, iarray+ array_size);

vector<int>::iterator it = find(vec.begin(), vec.end(), value);
int *pi = find(iarray, iarray+array_size, value);
```

The algorithms are generally overloaded to support two versions: one that uses either the built-in equality or the less-than operator of the underlying element type, and a second one that accepts either a function object or a pointer to function providing an alternative implementation of that operator. For example, by default `sort()` orders the container elements using the less-than operator. To override that, we can pass in the predefined greater-than function object:

```
sort(vec.begin(), vec.end()); sort(vec.begin(), vec.end(),
greater<int>());
```

Other algorithms, however, are separated into two uniquely named instances; the predicate instance in each case ends with the suffix `_if`, as in `find_if()`. For example, to find an element less than 10, we might write

```
find_if(vec.begin(), vec.end(), bind2nd(less<int>, 10))
```

Many of the algorithms that modify the container they are applied to come in two flavors: an in-place version that changes the container, and a version that returns a copy of the container with the changes applied. For example, there is both a `replace()` and a `replace_copy()` algorithm. The copy version always contains `_copy` in its name. It accepts a third iterator that points to the first element of the container in which to copy the modified elements. By default, the copy is achieved by assignment. We can use one of three inserter adapters to override the assignment semantics and have the elements instead inserted. (See [Section 3.9](#) for a discussion and examples.)

As programmers, we must quickly be able to look up which algorithms are available and how they are generally used. That is the purpose of this handbook. <sup>[1]</sup> The following built-in arrays, vectors, and lists are used as arguments to the algorithms:

[1] This handbook is an abridged and recast version of the C++ Primer Appendix, which provides a program example and discussion of each generic algorithm. Here I've listed a subset of the algorithms that are, in my opinion, the most frequently used.

```
int ia[8]={ 1, 3, 6, 10, 15, 21, 28, 36 };
vector<int> ivec(ia, ia+8);
list<int> ilist(ia, ia+8);
```

```
string sa[10] = { "The", "light", "untonsured", "hair",
    "grained", "and", "hued", "like", "pale", "oak" };
vector<string> svec(sa, sa+10);
list<string> slist(sa, sa+10);
```

Each listing provides a brief description of the algorithm, indicates the header file that must be included (either algorithm or numeric), and provides one or two usage examples.

### [accumulate\(\)](#)

By default, adds the container elements to an initial value specified by the third argument. A binary operation can be passed in to override the default addition.

```
#include <numeric>

iretult = accumulate(ia, ia+8, 0);
iretult = accumulate(ilist.begin(), ilist.end(), 0, plus<int>());
```

### [adjacent\\_difference\(\)](#)

By default, creates a new sequence in which the value of each new element, other than the first one, represents the difference of the current and the preceding element. Given the sequence `{0, 1, 1, 2, 3, 5, 8}`, the new sequence is

{0,1,0,1,1,2,3}. A binary operation can be passed in to override subtraction. For example, `times<int>` yields the sequence {0,0,1,2,6,15,40}. The third argument is an iterator that addresses the container into which to copy the results.

```
#include <numeric>

adjacent_difference(ilist.begin(), ilist.end(),
iresult.begin()); adjacent_difference(ilist.begin(), ilist.end(),
iresult.begin(),
times<int>());
```

### adjacent\_find()

By default, looks for the first adjacent pair of duplicate elements. A binary operator can override the built-in equality operator. Returns an iterator that addresses the first element of the pair.

```
#include <algorithm> class TwiceOver
{public:
    bool operator() (int val1, int val2){ return val1 == val2/2 ? true :
        false; }};

piter = adjacent_find(ia, ia+8); iter = adjacent_find(vec.begin(),
vec.end(), TwiceOver());
```

### binary\_search()

Assumes that the container is sorted by the less-than operator. If the container is sorted by some other ordering relationship, the binary operator must be passed in. The algorithm returns true or false.

```
#include <algorithm>
found_it = binary_search(ilist.begin(), ilist.end(), value);
found_it = binary_search(vec.begin(), vec.end(), value,
greater<int>());
```

### copy()

Copies the elements of one container into a second container.

```
#include <algorithm>
ostream_iterator<int> ofile(cout, " ");
copy(vec.begin(), vec.end(), ofile);
vector<string> target(svec.size());
copy(svec.begin(), svec.end(), target.begin());
```

### copy\_backward()

Behaves the same as `copy()` except that the elements are copied in the reverse order.

```
#include <algorithm>
copy_backward(svec.begin(), svec.end(), target.begin());
```

## count()

Returns a count of the number of elements within the container equal to `value`.

```
#include <algorithm>

cout << value << " occurs "
     << count(svec.begin(), svec.end(), value)
     << " times in string vector.\n";
```

## count\_if()

Returns a count of the number of times the operator evaluated as true.

```
#include <algorithm>
class Even {
public:
bool operator()(int val){ return !(val%2); }
};
ires = count_if(ia, ia+8, bind2nd(less<int>(),10));
lres = count_if(ilist.begin(), ilist_end(), Even());
```

## equal()

Returns true if the two sequences are equal for the number of elements contained within the first container. By default, the equality operator is used. Alternatively, a binary function object or pointer to function can be supplied.

```
#include <algorithm>class
EqualAndOdd{public:
    bool operator()(int v1, int v2){ return ((v1==v2) && (v1%2)); };};

int ia1[] = { 1,1,2,3,5,8,13 };
int ia2[] = { 1,1,2,3,5,8,13,21,34 };
res = equal(ia1, ia1+7, ia2); // true
res = equal(ia1, ia1+7, ia2, equalAndOdd()); // false
```

## fill()

Assigns a copy of `value` to each element within the container.

```
#include <algorithm>fill(ivec.begin(), ivec.end(),
value);
```

## fill\_n()

Assigns a copy of `value` to `count` elements within the container.

```
#include <algorithm>
fill_n(ia, count, value);
fill_n(svec.begin(), count, string_value);
```

## find()

The elements within the `container` are compared for equality with `value`. If a match is found, the search ends. `find()` returns an iterator to the element. If no match is found, `container.end()` is returned.

```
#include <algorithm>
piter = find(ia, ia+8, value);
iter = find(svec.begin(), svec.end(), "rosebud");
```

## find\_end()

This algorithm takes two iterator pairs. The first pair marks the container to be searched. The second pair marks a sequence to match against. The elements within the first container are compared for the last occurrence of the sequence using either the equality operator or the specified binary operation. If a match is found, an iterator addressing the first element of the matched sequence is returned; otherwise, the iterator marking the end of the first container is returned. For example, given the character sequence `Mississippi` and a second sequence `ss`, `find_end()` returns an iterator to the first `s` of the second `ss` sequence.

```
#include <algorithm>
int ia[17] = { 7, 3, 3, 7, 6, 5, 8, 7, 2, 1, 3, 7, 6, 3, 8, 4, 3 };
int seq[3] = { 3, 7, 6 };
```

```
// found_it addresses ia[10]
found_it = find_end(ia, ia+17, seq, seq+3);
```

## find\_first\_of()

`find_first_of()` accepts two iterator pairs. The first pair marks the elements to search. The second pair marks a collection of elements to search for. For example, to find the first vowel in the character sequence `synesthesia`, we define our second sequence as `aeiou`. `find_first_of()` returns an iterator to the first instance of an element of the sequence of vowels, in this case pointing to the first `e`. If the first sequence does not contain any of the elements, an iterator that addresses the end of the first sequence is returned. An optional fifth parameter allows us to override the default equality operator with any binary predicate operation.

```
#include <algorithm>
string s_array[] = { "Ee", "eE", "ee", "Oo", "oo", "ee" };
string to_find[] = { "oo", "gg", "ee" };

// returns first occurrence of "ee" -- &s_array[2]
found_it = find_first_of(s_array, s_array+6, to_find, to_find+3);
```

## find\_if()

The elements within the `container` are compared for equality with the specified binary operation. If a match is found, the search ends. `find_if()` returns an iterator to the element. If no match is found, `container.end()` is returned.

```
#include <algorithm>
find_if(vec.begin(), vec.end(), LessThanVal(ival));
```

## for\_each()

`for_each()` takes a third parameter that represents an operation that is applied to each element in turn. The operation cannot modify the elements (we can use `transform()` for that). Although the operation may return a value, that value is ignored.

```
#include <algorithm>template <typename
Type>void print_elements(Type elem) { cout
<< elem << " "; }

for_each(ivec.begin(), ivec.end(), print_elements);
```

## generate()

`generate()` fills a sequence by applying the specified generator.

```
#include <algorithm>class GenByTwo {public:
    void operator() () {
        static int seed = -1; return seed += 2; };list<int> ilist(10);

// fills ilist: 1 3 5 7 9 11 13 15 17
19generate(ilist.begin(), ilist.end(), GenByTwo());
```

## generate\_n()

`generate_n()` fills a sequence by applying `n` successive invocations of the generator.

```
#include <algorithm>class gen_by_two
{public:
    gen_by_two(int seed = 0) : _seed(seed){}int operator() () { return _seed += 2;
}private:
    int _seed;};vector<int> ivec(10);

// fills ivec: 102 104 106 108 110 112 114 116 118
120generate_n(ivec.begin(), ivec.size(), gen_by_two(100));
```

## includes()

`includes()` returns true if every element of the second sequence is contained within the first sequence; otherwise, it

returns false. Both sequences must be sorted, either by the default less-than operator or by the same operation passed as an optional fifth parameter.

```
#include <algorithm>
int ia1[] = { 13, 1, 21, 2, 0, 34, 5, 1, 8, 3, 21, 34 };
int ia2[] = { 21, 2, 8, 3, 5, 1 };

// includes must be passed sorted containers
sort(ia1, ia1+12); sort(ia2, ia2+6);
res = includes(ia1, ia1+12, ia2, ia2+6); // true
```

### inner\_product()

`inner_product()` accumulates the product of two sequences of values, adding them in turn to a user-specified initial value. For example, given the two sequences  $\{2, 3, 5, 8\}$  and  $\{1, 2, 3, 4\}$ , the result is the sum of the product pairs  $(2*1)+(3*2)+(5*3)+(8*4)$ . If we provide an initial value of 0, the result is 55.

A second version allows us to override the default addition operation and the default multiply operation. For example, if we use the same sequence but specify subtraction and addition, the result is the difference of the following addition pairs:  $(2+1)-(3+2)-(5+3)-(8+4)$ . If we provide an initial value of 0, the result is -28.

```
#include <numeric>
int ia[] = { 2, 3, 5, 8 };
int ia2[] = { 1, 2, 3, 4 };
int res = inner_product(ia, ia+4, ia2, 0);

vector<int> vec( ia, ia+4);
vector<int> vec2(ia2, ia2+4);

res = inner_product(vec.begin(), vec.end(), vec2.begin(), 0, minus<int>(),
                    plus<int>());
```

### inplace\_merge()

`inplace_merge()` takes three iterator parameters: `first`, `middle`, and `last`. Two input sequences are marked by  $[first, middle]$  and  $[middle, last]$  (`middle` marks 1 past the last element of the first sequence). These sequences must be consecutive. The resulting sequence overwrites the two ranges beginning at `first`. An optional fourth parameter allows us to specify an ordering operation other than the default less-than operator.

```
#include <algorithm>int ia[20] = {
    29, 23, 20, 17, 15, 26, 51, 12, 35, 40, 74, 16, 54, 21, 44, 62, 10, 41, 65, 71
};

int *middle = ia+10, *last = ia+20;

// 12 15 17 20 23 26 29 35 40 51 10 16 21 41 44 54 62 65 71 74
sort(ia, middle); sort(middle, last);
```

```
// 10 12 15 16 17 20 21 23 26 29 35 40 41 44 51 54 62 65 71
74inplace_merge(ia, middle, last);
```

## iter\_swap()

Swaps the values contained within the elements addressed by two iterators.

```
#include <algorithm>
typedef list<int>::iterator iterator;
iterator it1 = ilist.begin(), it2 = ilist.begin()+4;
iter_swap(it1, it2);
```

## lexicographical\_compare()

By default, the less-than operator is applied, although an optional fifth option allows us to provide an alternative ordering operation. Returns true if the first sequence is less than or equal to the second sequence.

```
#include <algorithm>class size_compare
{public:
    bool operator()(const string &a, const string &b) {
        return a.length() <= b.length();

    }};string sa1[] = { "Piglet", "Pooh", "Tigger" };string sa2[] = { "Piglet",
"Pooch", "Eeyore" };

// false: 'c' less than 'h'res = lexicographical_compare(sa1, sa1+3,
sa2, sa2+3);

list<string> ilist1(sa1, sa1+3);list<string>
ilist2(sa2, sa2+3);

// true: Pooh < Pooch
res = lexicographical_compare(ilist1.begin(), ilist1.end(), ilist2.begin(),
    ilist2.end(), size_compare());
```

## max(), min()

Returns the larger (or smaller) of the two elements. An optional third argument allows us to provide an alternative comparison operation.

## max\_element(), min\_element()

Returns an iterator pointing to the largest (or smallest) value within the sequence. An optional third argument allows us to provide an alternative comparison operation.

```
#include <algorithm>int mval =
max(max(max(max(ivec[4],
```

```

        ivec[3]), ivec[2]), ivec[1]), ivec[0]);

mval = min(min(min(min(ivec[4], ivec[3]),
                      ivec[2]), ivec[1]), ivec[0]); vector<int>::c
onst_iterator iter; iter = max_element(ivec.begin(), ivec.end()); iter
= min_element(ivec.begin(), ivec.end());

```

## merge()

Combines two sorted sequences into a single sorted sequence addressed by the fifth iterator. An optional sixth argument allows us to indicate an ordering other than the default less-than operator.

```

#include <algorithm>
int ia[12] = {29,23,20,22,17,15,26,51,19,12,35,40};
int ia2[12] = {74,16,39,54,21,44,62,10,27,41,65,71};

vector<int> vec1(ia, ia+12), vec2(ia2, ia2+12);

vector<int> vec_result(vec1.size()+vec2.size());

sort(vec1.begin(), vec1.end(),
greater<int>()); sort(vec2.begin(), vec2.end(),
greater<int>());

merge(vec1.begin(), vec1.end(),
      vec2.begin(), vec2.end(),
      vec_result.begin(), greater<int>());

```

## nth\_element()

`nth_element()` reorders the sequence so that all elements less than the `nth` element occur before it and all elements that are greater occur after it. For example, given

```
int ia[] = { 29,23,20,22,17,15,26,51,19,12,35,40 };
```

an invocation of `nth_element()` marking `ia+6` as `nth` (it has a value of 26)

```
nth_element(ia, ia+6, &ia[12]);
```

yields a sequence in which the seven elements less than 26 are to its left, and the four elements greater than 26 are to its right:

```
{ 23,20,22,17,15,19,12,26,51,35,40,29 }
```

The elements on either side of the `nth` element are not guaranteed to be in any particular order. An optional fourth parameter allows us to indicate a comparison other than the default less-than operator.

## `partial_sort()`, `partial_sort_copy()`

`partial_sort()` accepts three parameters — `first`, `middle`, and `last` — and an optional fourth parameter that provides an alternative ordering operation. The iterators `first` and `middle` mark the range of slots available to place the sorted elements of the container (`middle` is 1 past the last valid slot). The elements stored beginning at `middle` through `last` are unsorted. For example, given the array

```
int ia[] = {29,23,20,22,17,15,26,51,19,12,35,40 };
```

an invocation of `partial_sort()` marking the sixth element as `middle`

```
partial_sort(ia, ia+5, ia+12);
```

yields the sequence in which the five smallest elements are sorted:

```
{ 12,15,17,19,20,29,23,22,26,51,35,40 }
```

The elements from `middle` through `last-1` are not placed in any particular order, although all their values fall outside the sequence actually sorted.

## `partial_sum()`

Creates a new sequence in which, by default, the value of each new element represents the sum of all the previous elements up to its position. For example, given the sequence `{0,1,1,2,3,5,8}`, the new sequence is `{0,1,2,4,7,12,20}`. The fourth element, for example, is the partial sum of the three previous values (`0,1,1`) plus its own (`2`), yielding a value of 4. An optional fourth parameter allows the user to specify an alternative operation to apply.

```
#include <numeric>
int ires[7], ia[7] = { 1, 3, 4, 5, 7, 8, 9 };
vector<int> vres(7), vec(ia, ia+7);
```

```
// partial_sum(): 1 4 8 13 20 28 37
partial_sum(ia, ia+7, ires);
```

```
//partial sum using times<int>(): 1 3 12 60 420 3360 30240
partial_sum(vec.begin(),vec.end(),vres.begin(),times<int>());
```

## `partition()`, `stable_partition()`

`partition()` reorders the elements based on the true/false evaluation of a unary operation. All the elements that evaluate as true are placed before the elements that evaluate as false. For example, given the sequence `{0,1,2,3,4,5,6}` and a predicate that tests for elements that are even, the true and false element ranges are `{0,2,4,6}` and `{1,3,5}`. Although all the even elements are guaranteed to be placed before any of the odd elements, the relative position of the elements within the reordering is not guaranteed to be preserved. `stable_partition()` guarantees to preserve the relative order of the elements within the container.

```
#include <algorithm>
```

```
class even_elem {public:bool operator()(int elem){ return
```

```

    elem%2 ? false : true; });

int ia[] = { 29,23,20,22,17,15,26,51,19,12,35,40 };
vector<int> vec(ia, ia+12);
// partition based on whether element is even:
// 40 12 20 22 26 15 17 51 19 23 35 29
stable_partition(vec.begin(), vec.end(), even_elem());

```

## random\_shuffle()

By default, `random_shuffle()` reorders the elements randomly based on its own algorithm. An optional third parameter allows us to pass in a random-number-generating operation that must return a value of type `double` within the interval `[0,1]`.

```

#include <algorithm>random_shuffle(ivec.begin(),
ivec.end());

```

## remove(), remove\_copy()

`remove()` separates out all instances of a `value` within the sequence. It does not actually erase the matched elements (the container's size is preserved). Rather, each nonmatching element is assigned in turn to the next free slot. The returned iterator marks 1 past the new range of elements.

For example, consider the sequence `{0,1,0,2,0,3,0,4}`. Let's say that we wish to remove all 0 values. The resulting sequence is `{1,2,3,4,0,3,0,4}`. The 1 is copied into the first slot, the 2 into the second slot, the 3 into the third slot, and the 4 into the fourth slot. The 0 at the fifth slot represents the *leftover* of the algorithm. The returned iterator addresses that slot. Typically, this iterator is then passed to `erase()`. (The built-in array is not suited to the `remove()` algorithm because it cannot be resized easily. For this reason, the `remove_copy()` is the preferred algorithm for use with an array.)

```

#include <algorithm>

int ia[] = { 0, 1, 0, 2, 0, 3, 0, 4, 0, 5 };
vector<int> vec(ia, ia+10);

// vector after remove, without applying erase():
// 1 2 3 4 5 3 0 4 0 5
vec_iter = remove(vec.begin(), vec.end(), 0);

// vector after erase(): 1 2 3 4 5
vec.erase(vec_iter, vec.end());

int ia2[5];
// ia2: 1 2 3 4 5
remove_copy(ia, ia+10, ia2, 0);

```

## remove\_if(), remove\_copy\_if()

`remove_if()` removes all elements within the sequence for which the predicate operation evaluates as true. Otherwise, `remove_if()` and `remove_copy_if()` behave the same as `remove()` and `remove_copy()` — see the earlier discussion.

```
#include <algorithm>

    class EvenValue {public:bool operator()(int value) {return
    value % 2 ? false : true; }};

int ia[] = { 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 };vector<int>
vec(ia, ia+10);

iter = remove_if(vec.begin(), vec.end(),
bind2nd(less<int>(),10));vec.erase(iter, vec.end()); // sequence now: 13 21 34

int ia2[10]; // ia2: 1 1 3 5 13 21remove_copy_if(ia,
ia+10,ia2, EvenValue());
```

## replace(), replace\_copy()

`replace()` replaces all instances of `old_value` with `new_value` within the sequence.

```
#include <algorithm>
string oldval("Mr. Winnie the Pooh");
string newval("Pooh");
string sa[] = { "Christopher Robin", "Mr. Winnie the Pooh",
               "Piglet", "Tigger", "Eeyore" };

vector<string> vec(sa, sa+5);

// Christopher Robin Pooh Piglet Tigger Eeyorereplace(vec.begin(),
vec.end(), oldval, newval);

vector<string> vec2;

    // Christopher Robin Mr. Winnie the Pooh Piglet Tigger
    Eeyorereplace_copy(vec.begin(),
vec.end(),inserter(vec2,vec2.begin()), newval, oldval);
```

## replace\_if(), replace\_copy\_if()

`replace_if()` replaces all elements within the sequence with `new_value` for which the predicate comparison operation evaluates as true.

```
#include <algorithm>
int new_value = 0;
int ia[] = { 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 };
vector<int> vec(ia, ia+10);

    // new sequence: 0 0 0 0 0 0 0 13 21 34 replace_if(vec.begin(),
vec.end(),bind2nd(less<int>(),10), new_value);
```

## reverse(), reverse\_copy()

Reverses the order of elements in a container.

```
#include <algorithm>
list<string> slist_copy(slist.size());

reverse(slist.begin(), slist.end());
reverse_copy(slist.begin(), slist.end(), slist_copy.begin());
```

## rotate(), rotate\_copy()

`rotate()` is passed three iterators: `first`, `middle`, and `last`. It exchanges the two ranges marked by the iterators `first`, `middle-1` and `middle`, `last-1`. For example, given the following C-style character string "boohiss!!", `char ch[] = "boohiss!!";`

To change it to "hissboo!!", the call to `rotate()` looks like this:

```
rotate(ch, ch+3, ch+7);
```

Here is another example:

```
#include <algorithm>
int ia[] = { 1, 3, 5, 7, 9, 0, 2, 4, 6, 8, 10 };
vector<int> vec(ia, ia+11), vec2(11);
```

In this first invocation, we exchange the last six elements, beginning with 0, with the first five elements, beginning with 1:

```
// rotate on middle element(0) : 0 2 4 6 8 10 1 3 5 7 9
rotate(ia, ia+5, ia+11);
```

In this second invocation, we exchange the last two elements, beginning with 8, with the first nine elements, beginning with 1:

```
// rotate on next to last element(8): 8 10 1 3 5 7 9 0 2 4
rotate_copy(vec.begin(), vec.end()-2, vec.end(), vec2.begin());
```

## search()

Given two sequences, `search()` returns an iterator that addresses the first position in the first sequence in which the second sequence occurs. If the subsequence does not occur, an iterator that addresses the end of the first sequence is returned. For example, within `Mississippi`, the subsequence `iss` occurs twice, and `search()` returns an iterator to the start of the first instance. An optional fifth parameter allows the user to override the default equality operator.

```
#include <algorithm>
```

```
char str[25] = "a fine and private place";
char substr[4] = "ate";
int *piter = search(str, str+25, substr, substr+4);
```

### [search\\_n\(\)](#)

`search_n()` looks for the first occurrence of `n` instances of a `value` within a sequence. In the following example, we search `str` for two occurrences of the character `o` in succession, and an iterator to the first `o` of `moose` is returned. If the subsequence is not present, an iterator that addresses the end of the first sequence is returned. An optional fifth parameter allows the user to override the default equality operator.

```
#include <algorithm>const char oh =
'o';

char str[26] = "oh my a mouse ate a moose";char *found_str =
search_n(str, str+26, 2, oh);
```

### [set\\_difference\(\)](#)

`set_difference()` constructs a sorted sequence of the elements present in a first sequence but not present in a second. For example, given the two sequences `{0, 1, 2, 3}` and `{0, 2, 4, 6}`, the set difference is `{1, 3}`. All the set algorithms (three additional algorithms follow) take five iterators: The first two mark the first sequence, and the second two mark the second sequence. The fifth iterator marks the position of the container into which to copy the elements. The algorithm presumes that the sequences are sorted using the less-than operator; an optional sixth argument allows us to pass in an alternative ordering operation.

### [set\\_intersection\(\)](#)

`set_intersection()` constructs a sorted sequence of the elements present in both sequences. For example, given the two sequences `{0, 1, 2, 3}` and `{0, 2, 4, 6}`, the set intersection is `{0, 2}`.

### [set\\_symmetric\\_difference\(\)](#)

`set_symmetric_difference()` constructs a sorted sequence of the elements that are present in the first sequence but not present in the second, and those elements present in the second sequence are not present in the first. For example, given the two sequences `{0, 1, 2, 3}` and `{0, 2, 4, 6}`, the set symmetric difference is `{1, 3, 4, 6}`.

### [set\\_union\(\)](#)

`set_union()` constructs a sorted sequence of the element values contained within the two sequences. For example, given the two sequences `{0, 1, 2, 3}` and `{0, 2, 4, 6}`, the set union is `{0, 1, 2, 3, 4, 6}`. If the element is present in both containers, such as 0 and 2 in the example, the element of the first container is copied.

```
#include <algorithm>
string str1[] = { "Pooh", "Piglet", "Tigger", "Eeyore" };
```

```

string str2[] = { "Pooh", "Heffalump", "Woozles" };
set<string> set1(str1, str1+4),

        set2(str2, str2+3);

// holds result of each set operation
set<string>
res;

//set_union(): Eeyore Heffalump Piglet Pooh Tigger
Woozles
set_union(set1.begin(), set1.end(),
        set2.begin(), set2.end(), inserter(res, res.begin()));

res.clear(); // empties the container of elements

// set_intersection(): Pooh
set_intersection(set1.begin(), set1.end(), set2.begin(), set2.end(),
        inserter(res, res.begin()));

res.clear();

        // set_difference(): Eeyore Piglet
        Tigger
set_difference(set1.begin(), set1.end(),
        set2.begin(), set2.end(), inserter(res, res.begin()));

res.clear();

// set_symmetric_difference():
// Eeyore Heffalump Piglet Tigger Woozles
set_symmetric_difference(set1.begin(), set1.end(), set2.begin(),
        set2.end(), inserter(res, res.begin()));

```

## sort(), stable\_sort()

By default, sorts the elements in ascending order using the less-than operator. An optional third parameter allows us to pass in an alternative ordering operation. `stable_sort()` preserves the relative order of elements within the container. For example, imagine that we have sorted our words alphabetically and now wish to order them by word length. To do this, we pass in a function object `LessThan` that compares two strings by length. Were we to use `sort()`, we would not be guaranteed to preserve the alphabetical ordering.

```

#include <algorithm>
stable_sort(ia, ia+8);
stable_sort(svec.begin(), svec.end(), greater<string>());

```

## transform()

The first version of `transform()` invokes the unary operator passed to it on each element in the sequence. For example, given a sequence `{0, 1, 1, 2, 3, 5}` and a function object `Double`, which doubles each element, the resulting sequence is `{0, 2, 2, 4, 6, 10}`. The second version invokes the binary operator passed to it on the associated elements of a pair of sequences. For example, given the sequences `{1, 3, 5, 9}` and `{2, 4, 6, 8}`, and a function object `AddAndDouble` that adds the two elements and then doubles their sum, the resulting sequence is `{6, 14, 22, 34}`. The resulting sequence is copied into the container pointed to by either the third iterator of the first

version or the fourth iterator of the second.

```
#include <algorithm>
int double_val(int val) { return val + val; }
int difference(int val1, int val2) { return abs(val1 - val2); }

int ia[] = { 3, 5, 8, 13, 21 };
vector<int> vec(5), vec2(5);

// first version: 6 10 16 26 42
transform(ia, ia+5, vec.begin(), double_val);

// second version: 3 5 8 13 21
transform(ia, ia+5, vec.begin(), vec2.begin(), difference);
```

### [unique\(\), unique\\_copy\(\)](#)

All consecutive groups of elements containing either the same value (using the equality operator) or evaluating as true when passed an optional alternative comparison operation are collapsed into a single element. In the word *Mississippi*, the *semantic* result is "Misisipi." Because the three *i*'s are not consecutive, they are not collapsed, nor are the two pairs of *s*'s. To guarantee that all duplicated elements are collapsed, we would first sort the container.

As with `remove()`, the container's actual size is not changed. Each unique element is assigned in turn to the next free slot, beginning with the first element of the container. In our example, the *physical* result is "Misisippi," where the character sequence *ppi* represents the leftover piece of the algorithm. The returned iterator marks the beginning of the refuse. Typically this iterator is then passed to `erase()`. (Because the built-in array does not support the `erase()` operation, `unique()` is less suitable for arrays; `unique_copy()` is more appropriate.)

```
#include <algorithm>
int ia[] = { 0, 1, 0, 2, 0, 3, 0, 4, 0, 5 };
vector<int> vec(ia, ia+10);

sort(vec.begin(), vec.end());
iter = unique(vec.begin(), vec.end());
vec.erase(vec_iter, vec.end()); // vec: 0 1 2 3 4 5

int ia2[10];
sort(ia, ia+10);
unique_copy(ia, ia+10, ia2);
```