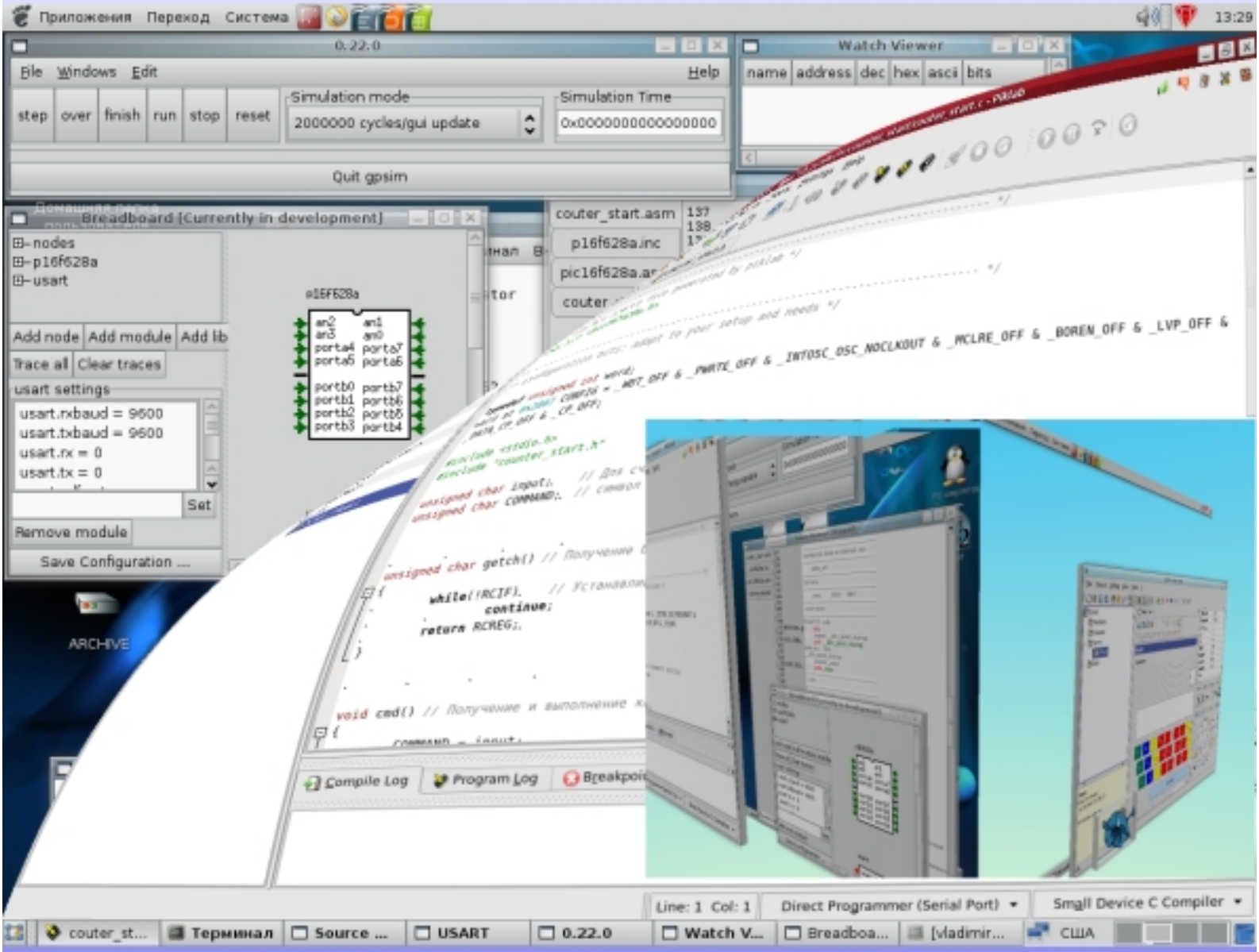
В.Н. Гололобов

Игра в программирование



Сказка для младшего и среднего радиолюбительского возраста

Москва 2007

Оглавление

Предисловие....3
Глава 1. Поиск печки, от которой танцевать....5
Два берега....5
Первое знакомство с Gambas....9
Компьютер может помочь с программированием....34
Почему Емеля ездил на печи?....40
А можно с этого места поподробнее?....51
Первый блин....56
Глава 2. Бряцая железом....63
Хорошее начало....63
Возвращение....72
Отладка в gpsim....87
Глава 3. У камелька....98
Начало проекта «Машинистка»....99
Развиваем успех....108
Заметки и пометки «на память»....115
Глава 4. Охота на кентавра....118
Засада в интерфейсе....118
Железное решение....123
Лирическое отступление....128
Грустное завершение рассказа о счетчике....137
Глава 5. Сказка о неудачливом радиолюбителе....147
Возвращение на круги своея....147
Расширение кругов (на воде?)....156
gpsim как зерцало грешника....162
Вялая попытка оправдаться....181
И оргвыводы....183
Глава 6. Сказка о ловком программисте....186
Предварительное рассмотрение проекта «Генератор»....186
Продолжение работы над проектом «Генератор»....191
Завершение....199
Конспекты....201
Gambas дружелюбен к пишущим на VB, но используя Linux....201
Разработка приложений в Gambas....209

Предисловие

Не получается у меня задумка. Не работает схема. Все, вроде бы, правильно, но не желает работать это «транзисторно-микросхемное месиво». Раньше было проще – измерил напряжения, посмотрел осциллографом, все понял и исправил. Так то было раньше.

Не то, чтобы я испереживался, не спал по ночам. Сплю, и вам советую. Спите крепко, чтобы проснувшись назавтра порадоваться новому дню и солнцу, если оно светит, или дождю, если на улице пасмурно. Как бы плотно ни укрыли тучи ваш город, солнце-то продолжает светить!

Любая деятельность человека всегда связана с периодами усталости или разочарования, с тем, что предмет его внимания начинает капризничать, становится непостижимым образом похож на дикобраза – кругом колючки, как ни подступись. Даже если то, чем вы занимаетесь, не ваша работа, а ваше увлечение, ведет оно себя временами столь же противно, что и каждодневная рутина. Бесполезно браниться, бесполезно сердится, полезно, но не всегда, к сожалению, помогает, обратиться за советом или помощью. Важнее всего помнить, что как бы плотно ни укрыли тучи...

Программирование, как обязательный курс, по которому приходилось сдавать экзамен, может больше напугать, чем заинтересовать. Между тем, все больше людей, никак с программированием прямо не связанных, или не связанных с ним прежде, подвергается, я бы сказал, «прямому шантажу» со стороны этого явления природы.

Я не программист. И не пытаюсь примазаться к этой славной профессии. Но и нет моей вины в том, что время от времени появляется необходимость возвращаться к давно забытым со студенческих времен понятиям процедуры, функции, цикла... Однако сегодня, когда мне не нужно сдавать экзамены, возвращаясь к этим пугающим нормального человека понятиям, я начинаю с грустью думать об их простой и ясной сущности. И происходит это потому, что сегодняшнее программирование изобилует новыми таинственными сущностями – класс, наследование, контейнер и т.д. Но, признаться, если не обращать на них внимания, то современные системы программирования позволяют сделать легко и быстро многое из того, что прежде отнимало много времени и сил. Сегодня я не могу сказать, что проще – написать небольшую программу, которая решит ваши проблемы, или с карандашом и бумагой проделать эту же работу? Обычно первая мысль о бумаге и карандаше кажется более разумной, но лишь до тех пор, пока не примешься претворять эту мысль в жизнь.

Я не умею программировать, и компьютер для меня сродни кофеварке, удобно и включается быстро, но я хочу предложить такую игру: я сделаю вид, что я умею программировать, что знаком с компьютером не по наслышке, а вы сделаете вид, что верите мне, – посмотрим, сумею ли я рассказать что-нибудь полезное о программировании. Это будет не более, чем игра, и я не буду, выскакивая из-за угла, пугать вас громкими криками: «Грядет итератор!».

Для игры в футбол есть футбольное поле, для игры в гольф – поле для гольфа. Для игры в программирование я предлагаю великолепнейшее поле – язык и среда программирования Gambas. Он имеет в своей основе классический язык Бэйсик. Работает в среде операционной системы Linux. Если по какой-то причине вы не хотите или не можете пользоваться этой операционной системой, то можете пользоваться средой программирования Visual Basic. Они достаточно похожи.

Для тех, кого смущают родственные отношения между Basic и Gambas, я готов привести множество высказываний опытных профессионалов о том, что программирование не соотносится напрямую с языком кодирования. В этом смысле нет ничего зазорного, что основой языка служит Basic, тем более, что Gambas – это современный, мощный и хорошо построенный объектно-ориентированный язык программирования. А насколько он хорош, можно судить по тому, что вся графическая пользовательская оболочка Gambas написана на самом языке Gambas.

По мне же, чем проще, тем лучше. И лично меня смущает только одно, достаточно ли это просто. С выяснения этого я, пожалуй, и начну.

Спасибо, что дочитали это предисловие до конца, а если нет, то и ничего страшного.

Глава 1. Поиск печки, от которой танцевать

Под печкой в любительской практике будем на протяжении этой главы понимать удобные программные средства, которые легко использовать в практике радиолюбительства с не меньшим успехом, чем в профессиональной.

Два берега

Если сравнивать программирование с рекой, то конкретная платформа или язык программирования – это берега могучей и широкой реки. К какому берегу вам причалить, к какому месту на берегу – ваш выбор! Плыть ли по реке на скутере, побивая рекорды скорости, или неспешно перемещаться, отдавшись на волю течения и уютно расположившись в палатке, установленной посреди плота, и наслаждаться видом окрестностей – ваш выбор!

Я отдаю предпочтение одному берегу – платформе Linux и среде программирования Gambas, основанной на языке Бейсик, вы можете сделать другой выбор. Программы, написанные в среде Visual Basic и Gambas, мало чем отличаются друг от друга. Более того, даже использование Delphi потребует скорее не переделки программы, а некоторой правки текста. Конечно, я не могу сказать этого о любой программе, но только о небольших проектах, с которыми собираюсь иметь дело.

То же будет относиться и к среде программирования микроконтроллеров, когда до этого дойдет дело. В Windows можно использовать MPLAB, в Linux – Piklab. В этом случае даже правки кода не потребуется, обе среды программирования используют одинаковый ассемблер, а программирование на языке «С» можно осуществить с использованием компилятора Hi-Tech, бесплатная версия которого (с ограничением по памяти) есть для обеих сред программирования.

Я уже говорил, что не буду пытаться научить кого-либо программированию, для этого есть опытные программисты, опытные преподаватели, я хочу только рассказать о пользе навыков в программировании с пониманием основных подходов к этому процессу даже для тех, кому нет нужды постоянно заниматься программированием. И, главное, что само программирование – увлекательнейшее занятие для любого, у кого есть компьютер и остается хотя бы немного времени, чтобы включить его и заняться чем-то менее бессмысленным, чем стрельба «из мышки по воробьям».

Много лет назад, когда мой сын был еще ребенком, я собрал для него, с большими приключениями, домашний компьютер, который назывался ZX-Спектрум. Когда сегодня я вспоминаю этот компьютер, то, возможно я ошибаюсь, мне кажется, что он был гораздо интереснее, чем мой сегодняшний рабочий Pentium. Он был гораздо проще, и было много легче разобраться с необходимыми переделками, если такая нужда возникала. Помню, когда появилась возможность подключить к нему дисковод, а первоначально этот домашний компьютер записывал все на магнитофон, и появились программы, которые загружались с дискет, то многие из них были не

полностью переделаны под работу с дисководом. Так компилятор языка «С» действительно загружался с дискеты, но при попытке сохранить текст программы обращался к магнитофону. Исправление этого недостатка доставило мне больше удовольствия, чем неприятностей. То же касается и игрушек, которые не работали, но зачем-то обязательно были нужны сыну. Для этого же компьютера мы с ним написали первую законченную программу, которая позволяла научиться печатать «вслепую». Не знаю, остались ли у него приятные воспоминания о программировании, но помню, что завершив работу над программой, мы поняли, что учиться печатать на той клавиатуре, собранной из «подручных средств», которая у нас была, не имеет смысла, но... позже, когда появился компьютер IBM PC, я восстановил эту программу, правда, убрав все излишества, для нового компьютера, и мы оба учились печатать, используя эту программу, поскольку других обучающих программ тогда не смогли найти. И мы оба по сей день пользуемся полученным навыком, что экономит много сил и времени и мне, и сыну, когда приходится что-то печатать. В тот раз я понял две важные вещи: во-первых, при желании можно написать любую программу, которую есть необходимость написать; во-вторых, что многое отнимает гораздо меньше времени, чем это может показаться на первый взгляд. Это касается овладения печатью «вслепую». Мне казалось, что пройдет не меньше года, прежде чем я смогу печатать глядя на экран, а не на клавиатуру, и использовать все пальцы для этого процесса, а не обходится двумя. Я ошибался. Через две-три недели я печатал, печатал «вслепую», а через месяц печатал и на русском, и на английском, переходя с языка на язык пусть не мгновенно, но без особых проблем. А возможности просиживать за компьютером часами тогда у меня не было. Только по 15-20 минут каждый вечер. К сожалению впоследствии я перестал пользоваться печатью латиницей хотя бы для сохранения умения, да и по-русски писать приходилось крайне редко, но последний навык все-таки сохранился. С программированием, кстати, такая же история. Я не программист, но когда приходилось этим заниматься по тем или иным причинам, я вынужденно овладевал какими-то базовыми подходами. Эти знания впоследствии не единожды меня очень выручали. Мне ни разу не пришлось пожалеть о потраченном на их приобретение времени, а если и приходилось о чем-то пожалеть, так это о том, что я мало времени потратил на изучение программирования.

Я не скажу, что сейчас, если возникает необходимость обратиться к программированию, я испытываю восторг от предвкушения предстоящей работы. Нет. Я отдаю себе отчет, что будет необходимость обращаться к документации, к книгам, к подсказкам среды программирования. Я прекрасно понимаю, что наделаю много ошибок, с которыми придется как-то справляться, что ни раз буду чертыхаться и злиться (на кого?), разочаровываться и начинать все заново, но за этими неприятными эпизодами, как за занавесом театра, будет скрываться увлекательнейшая история. Я точно знаю, что временами очень сердился на все и на вся, но когда позже пытался вспомнить, что же происходило на самом деле, то не мог вспомнить ничего кроме удовольствия от «возни» с непокорными программными

сущностями.

У меня нет пристрастия к какому-либо языку программирования в силу дилетантского подхода с одной стороны, и в силу разных обстоятельств с другой. В один из моментов работы потребовалось, а исходные коды имелись в наличии, дописать программу для временной демонстрации работы устройства. Позже это все предполагалось сделать профессионально, иначе, и на другом языке программирования. Программа, написанная на «С++», была документирована, что называется, «для себя», и я точно знал, что с этой задачей мне не справиться. Хотя бы потому, что работать с чужой, достаточно большой программой – занятие безнадежное и очень трудоемкое. Но, начав работать, я с удивлением отметил, что, отдавшись на волю течения, продвигаюсь вперед. Конечно, не дело, «лезть в воду, не зная броду», но компьютер удивительное создание. Если не делать ничего сверх необходимого, он всегда выручит, и при этом можно не опасаться фатальных последствий.

Продолжая аналогию с рекой, но считая рекой электронику, то два ее берега превратятся в программирование и «собственно железо». Достаточно давно электронные устройства имеют в своем составе нечто, что так или иначе программируется. И сегодня, занимаясь электроникой, трудно обойтись без нежданных встреч с программированием. К этому надо быть готовым, и я еще раз хочу подчеркнуть, пусть меня поправят, если я искренне заблуждаюсь, не следует путать программирование с конкретным языком программирования или той или иной средой программирования. Совсем не обязательно быть знатоком языка «Си-шарп», чтобы иметь возможность начать работу над программой. Совсем не обязательно ставить своей задачей освоить в полной мере ассемблер, если вы хотите использовать микроконтроллер в своей практике. В первую очередь по той причине, что каждый конкретный микроконтроллер будет иметь свой ассемблер. Гораздо проще поискать такой контроллер, такую среду программирования контроллера, для которой есть язык высокого уровня. Часто это язык «С». И из самого языка вы, скорее всего, будете использовать только часть возможностей, понять и освоить которую гораздо легче и быстрее, чем язык полностью. Я не имею ввиду профессионалов, у которых другой подход и иные критерии выбора.

Если продолжать аналогию с рекой, но в качестве реки принять ваше занятие электроникой, то берега превратятся в два основных вопроса, которые всегда будут сопровождать вас – что проще, выполнить работу не прибегая к помощи программирования и элементов, связанных с программированием, или программировать?

В качестве примера того, как подступиться к программированию, я намереваюсь написать (по ходу написания книги) программу, которая будет обслуживать микроконтроллер, который, в свою очередь, должен проверить двоичный счетчик с предустановкой. Это простенькая задачка – проверка счетчика. Давайте попробуем присмотреться к обоим берегам.

Как я стал бы действовать, причалив к «железному» берегу реки?

Для задания числа можно подпаять соответствующие входы к общему проводу и плюсу питания (через «подтягивающие» резисторы). Ко входу переписи припаять любой «микрик» – небольшой кнопочный переключатель, включив его соответствующим образом, чтобы сформировать фронт сигнала переписи. Аналогично можно было бы поступить и с входом счета, если бы не одно но – дребезг контактов. Дребезг вызовет серию импульсов вместо одного единственного. У некоторых импульсных генераторов есть режим генерации одиночного импульса, но такого генератора у меня нет. Придется спаять схему, предотвращающую дребезг контактов. Ну, а паять, так паять! Для проверки состояния выходов я могу на выходы припаять светодиоды, что избавит меня от необходимости проводить каждый раз по несколько измерений мультиметром. Припаяв светодиоды с токоограничительными резисторами, я нарисую в рабочей тетради таблицу-заготовку для полученных значений, и все. Я готов проверить работу счетчика. Не слишком меня это «напрягло». Правда, сделав пару циклов, я опять берусь за паяльник, чтобы изменить число на входе предварительной установки. Но и это не слишком сложно. Беспрограммный берег реки полог, порос зеленой сочной травой, красив и живописен.

Единственное, что вызывает у меня сомнения, а не болото ли это? Я проверил работу счетчика в пошаговом режиме на очень низкой частоте переключений. Не возникает ли проблем при ускорении работы? Проверить этот аспект я тоже могу. Достаточно импульсом обнуления счетчика запустить небольшое устройство, которое сформирует мне импульс переписи, затем засинхронизироваться этим импульсом и посмотреть на осциллографе результат работы счетчика при тактовой частоте, положим, в один мегагерц. Паять, так паять!

Лежа на плоту, плывя по течению реки, я посматриваю то на один берег, то на другой. Один я описал выше. А как выглядит другой?

Крутой и неприступный, отчего он кажется неприветливым, берег порос деревьями столь густо, что сквозь поросль не видно ни одной тропинки. От того, что солнце бьет прямо в глаза, заросли кажутся еще более мрачными, темными и непролазными. Ни птица не порхнет, ни зверь не шелохнется. Только неясные тени виртуальных программистов осмеливаются потревожить покой чащобы.

И что за напасть! Прибило меня течением именно к этому берегу.

Первое знакомство с Gambas

Не будучи представлен человеку, желая набиться ему в знакомые, ты никогда не начнешь с жалоб на свои личные проблемы. Самое лучшее, что можно сделать при первом знакомстве – это узнать у человека, как идут его дела, пригласить его выпить чашечку кофе, а за кофе поговорить о погоде, поискать общих знакомых.

Набиваясь в знакомые языку программирования я хочу следовать тому же правилу. Вначале поинтересуюсь о его делах (законспектирую руководство), потом поговорю об общих знакомых (например, языках программирования, или программах). И только после этого попробую обратиться к своим проблемам с его помощью.

Вы не поверите, но набиваться в знакомые не так-то просто, конечно, если не полагаться на случай, на везение. Для подготовки к предстоящему разговору я, вопреки обыкновению, отправился в книжный магазин. Обилие общих знакомых, дружными рядами оккупировавших полки, не сулило скорого продвижения в работе, предстояло перечитать все названия книг, чтобы попытаться найти ту, которую я хотел купить. А завершив процедуру, я, вдобавок, не нашел нужной. Но это только по причине моей привередливости, книг по программированию очень много. Еще можно поискать в Интернете.

И Gambas. В папке «help» кроме традиционных подсказок большое количество статей, относящихся к языку, к объектно-ориентированному программированию. Много примеров, над созданием которых потрудился не только автор, но и многочисленные добровольцы. Думаю, если бы я искал прямой путь к цели, этого хватило бы, чтобы вымостить широченный проспект, а не только подъездную дорогу.

Немного о Linux. Запуск программы, как во многих сегодняшних, если не всех, операционных системах, осуществляется щелчком левой клавиши мышки в соответствующем разделе меню. В Linux меню выбора программ хорошо систематизировано – офисные приложения в разделе «Офис», у меня там много всего, потому что я люблю, когда всего много, когда есть из чего выбирать.

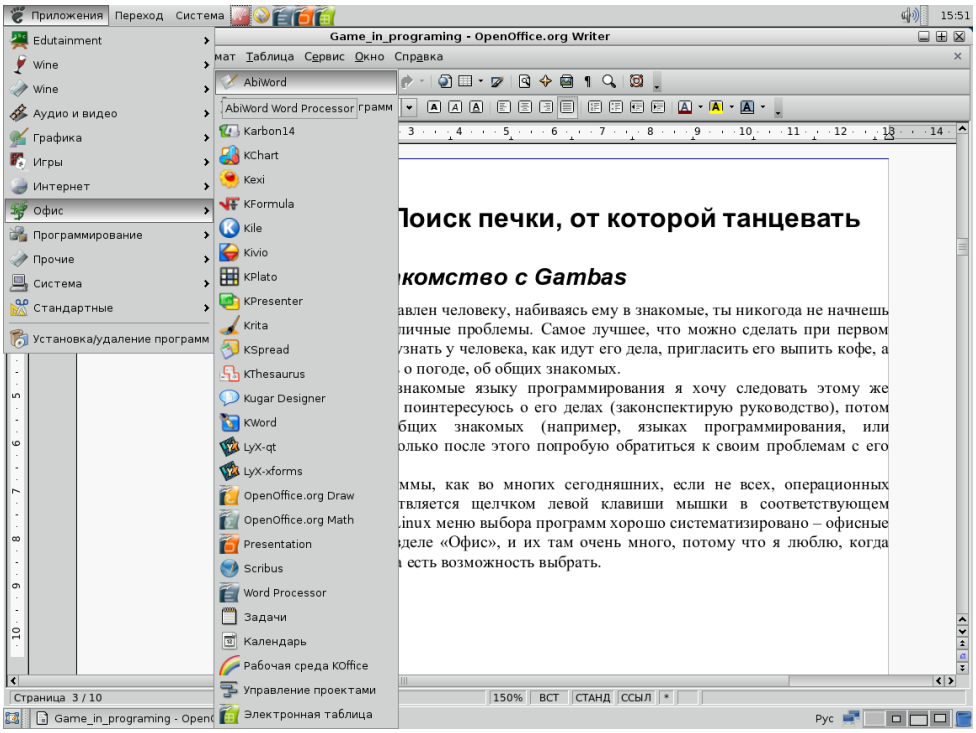


Рис. 1.1. Раздел офисных приложений в операционной системе Linux

Для работы только с текстом я часто пользуюсь текстовым процессором AbiWord. Он легкий и быстрый. Если приходится рисовать что-то в тексте, исключая необходимость создания чертежей профессионального качества, можно использовать KOffice, но чаще я пользуюсь OpenOffice.org.Writer. Из него я могу экспортировать текст с иллюстрациями в форматы «doc», «pdf», «html».

Не менее насыщенным выглядит у меня раздел приложений для программирования. Здесь несколько великолепных сред программирования, позволяющих работать с разными языками, на них я поглядываю с тоской и завистью. Есть программы создания web-сайтов. Свой сайт, в его последней ипостаси, я создавал с помощью Nvu. Времени заниматься сайтом было мало, и не умею я это делать, сайт вышел весь в меня – страшненький, но работающий. Смотрю в меню, и недоумеваю, откуда это накопилось? Недавно обновлял операционную систему, давал себе слово, что все по минимуму! Не понимаю.

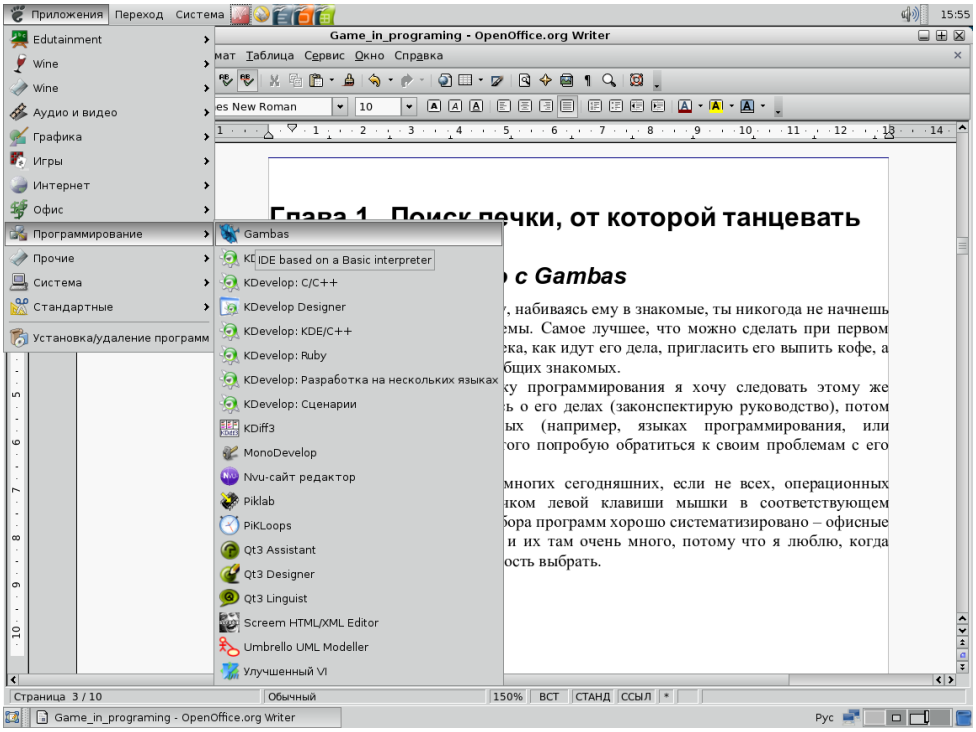


Рис. 1.2. Раздел «Программирование» в Fedora Core 6

И очень мне хочется похвастаться (не могу оказать себе в этом скромном удовольствии) теми возможностями анимации и трехмерного вида рабочих столов, которые есть в последней версии Fedora Core 6 (сегодня, завтра уже Fedora 7). Когда я устаю работать, но еще не начинаю дремать у телевизора, я включаю все эти красоты и начинаю просматривать почту, новости в мире Linux, форумы, где больше читаю, чем пишу, по той простой причине, что едва успеваю прочитать все то полезное, что накапливается за день.

Еще хочу добавить, что трехмерные и возможности анимации не относятся непосредственно к моему дистрибутиву. Нет. Второй дистрибутив на моем компьютере – это Ubuntu. Хотелось с ним познакомиться в работе, чтобы порекомендовать своему другу в качестве основной операционной системы. Так вот, в Ubuntu все эти красоты работают не менее успешно.

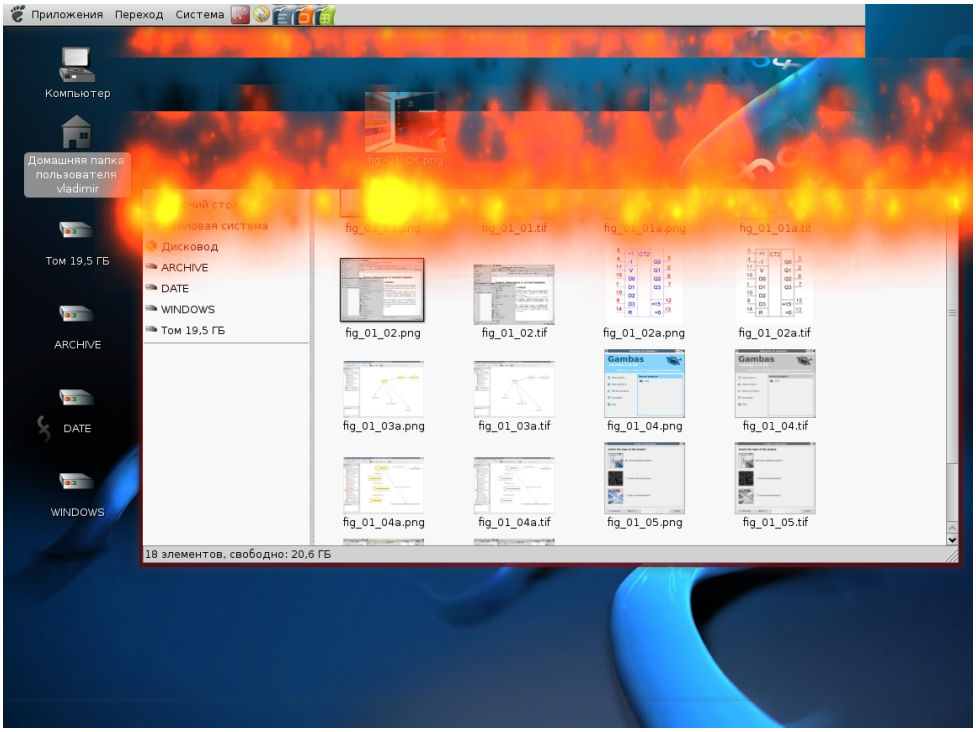


Рис. 1.3. Закрывание домашней папки с поддержкой Beryl

Я бы сказал, что операционная система просто «горит на работе»! И особенно мне этот фокус нравится, когда я закрываю логический раздел диска, где у меня установлен эмулятор Windows для Linux – Wine, закрывая его думаешь: «Вот и сожжены мосты!». В разделе установлено несколько «free» и «demo»-программ EDA для Windows, которые я использовал при работе над предыдущей книгой. По завершении хотел было снести раздел, но программы мне понравились, и я решил оставить их вместе с Wine на разделе Windows. Собственно, я не противник этой операционной системы, просто дороговаты для меня и система, и программы. И поскольку уж зашла об этом речь, все что будет сказано о конкретных программах, которые я, если получится, напишу в Gambas, а язык стоит того, чтобы постараться, так вот, все это можно повторить в Visual Basic, если для кого-то он кажется более подходящим.

Одна из особенностей операционной системы Linux – наличие нескольких рабочих столов, их может быть один или два, часто после установки их бывает четыре, но вы можете легко задать нужное именно вам количество рабочих столов. Я так привык использовать несколько рабочих столов, что не представляю себе, как

умещался раньше на одном единственном?

Сегодняшние дистрибутивы позволяют перемещаться между столами, используя трехмерный куб (и количество граней можно, похоже, увеличивать).

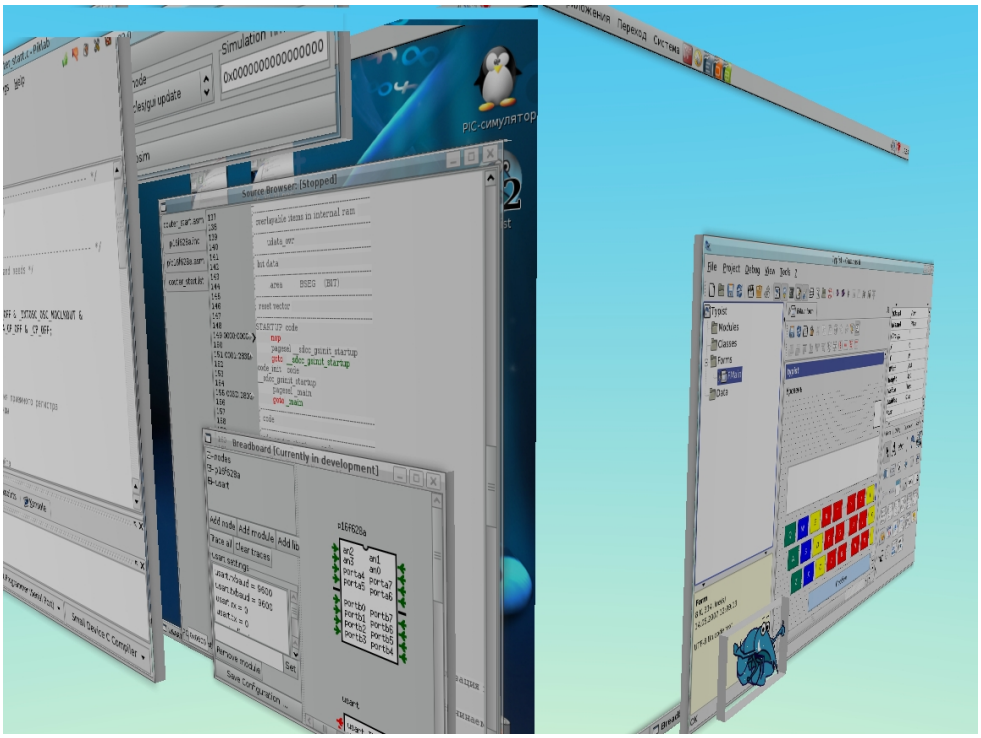


Рис. 1.4. Куб рабочих столов

В глазах многих, как мне кажется, Linux выглядит, как операционная система для профессионалов. Этому представлению может способствовать и посещение форумов, связанных с платформой Linux. Множество советов, как добиться желаемого результата, начинаются словами: «Открой скрипт...», или «Дай команду...». Это правильные советы. Это универсальные (или почти универсальные) советы, обязанные большой гибкости и открытости Linux. В ней можно переделать все, все изменить, настроить, что угодно и как угодно. Важно только знать как это сделать, и разбираться в последствиях своих действий. Многим нравится работать даже с последними версиями без графической оболочки, используя команды, вводимые вручную. Но я, например, только ленивый пользователь. Я не склонен изучать и совершенствовать операционную систему. Я хочу ею пользоваться, выполняя свою работу или потакая своим прихотям. Позволяет ли Linux оставаться пользователем и

только пользователем, и никем, кроме пользователя. Думаю, да.

Простой пример: сегодня в Москве многие владельцы компьютеров подключаются к домовым компьютерным сетям, подключение к Интернету проходит через vpn-сервер провайдера. Если в Windows все настройки кажутся простыми (что не совсем так), то создание подключения в Linux может показаться занятием «не для слабонервных». Как выглядят рекомендации, которые в изобилии встретишь на форумах:

*«Убедитесь, что в системе установлен пакет **pptp-linux-1.2.0-1asp.i386.rpm** и если нет, то установите его со второго диска дистрибутива. Также проверьте, что у вас уже установлено соединение с локальной сетью, используя команду **ifconfig eth0**. Если соединение есть, то вывод команды будет выглядеть примерно так:*

```
[user@localhost user]$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:11:22:A3:B4:C5
             inet addr:192.168.1.20  Bcast:192.168.1.255  Mask:255.255.255.0

          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 Mb)  TX bytes:0 (0.0 Mb)
                 Interrupt:3 Base address:0xa400
```

*Создайте файл **/etc/sysconfig/network-scripts/ifcfg-pptp0** приблизительно такого содержания:*

```
PEERDNS="no"
DEVICE="pptp0"
ONBOOT="no"
USERCTL="yes"
PERSIST="no"
DEBUG="yes"
DEFROUTE="yes"
PPPOPTIONS="require-mppe require-mppe-128"
MRU=""
MTU=""
IDLETIMEOUT=""
VPN_HOST="host_name"
VPN_USER="username"
ROUTES=""
```

где VPN_HOST – это имя или ip-адрес VPN-сервера, а VPN_USER – имя, под которым вы известны VPN-серверу...», и т.д.

Видимо, это правильные рекомендации. Когда лет пять назад я купил первый дистрибутив Linux, то обратился с проблемой подключения по VPN в отдел технической поддержки дистрибутива, где получил такого же рода советы. Однако,

повторяя все операции шаг за шагом, я так и не смог настроить подключение. Позже я нашел удобный графический вариант создания vpn-подключения с помощью программы (или сервиса?) Webmin, что было, возможно, не «по науке», но работало.

Подключение к Интернету в последнем дистрибутиве Fedora Core я делал с помощью Webmin. Создание подключения мало отличается от аналогичного в Windows и требует ввода почти стандартной информации о vpn-сервере (но требует IP адреса сервера), данных доступа и т.п. Но без ложки дегтя не обходится.

Не столь давно, установив в качестве второй операционной системы Ubuntu, еще один дистрибутив Linux, я настраивал подключение по VPN к Интернету в программе Kvpnc. Как это выглядит, чтобы легко было сравнить эти настройки с Windows, я сейчас покажу, предварительно уточнив, что Ubuntu я ставил с LiveCD (один загружающий систему CD-диск с возможностью установки системы в логический раздел жесткого диска) с графическим менеджером Gnome. Для работы Kvpn дополнительно потребовались некоторые библиотеки, относящиеся к другой графической оболочке. Если бы я использовал Kubuntu с графической оболочкой KDE, то установка дополнительных библиотек, возможно, не потребовалась бы.

Основная проблема возникает из-за необходимости скачивания файлов через Интернет, к которому ты еще не подключен. В частности, при установке Ubuntu сама установка застревает на настройке менеджера обновлений, который долго ищет подходящие зеркала для обновления дистрибутива, хотя подключение к Интернету еще предстоит создать. Обычно я отключаю сеть, чтобы закончить установку. Network Manager имеет раздел подключения по vpn, но тоже с дополнительными файлами. Я как-то попытался выполнить эту настройку, успеха, увы, не добился. После установки первой для меня версии Ubuntu я, используя основную свою ОС Fedora, скачал программу pptp-config со всеми необходимыми библиотеками и дополнительными файлами. Удобство этой программы для меня в том, что дополнительные файлы, которые обычно связаны зависимостями, имеют разрыв в этих зависимостях и позволяют найти конец цепочки, с которого последовательная установка проходит.

Для соединения с Интернетом по VPN можно было бы использовать после настройки этот графический интерфейс. Но программу следует запускать с правами root из терминала, либо сделав значок запуска, и вдобавок, я не знаю, как организовать ввод двух необходимых строк для маршрутизации без использования ручного ввода в терминале после того, как соединение установлено.

После установки, настройки и первого запуска этой программы (с вводом строк маршрутизации) появляется возможность установить Kvpnc. Для этого я вначале устанавливаю Kpdf.

Зачем? Затем, чтобы установились все необходимые библиотеки для работы Kvpnc. Затем Kvpnc устанавливается и можно запустить настройку, как это показано ниже.

После установки программы необходимо создать новый профиль (аналогично новому подключению), что лучше всего сделать с помощью мастера создания этого

профиля.

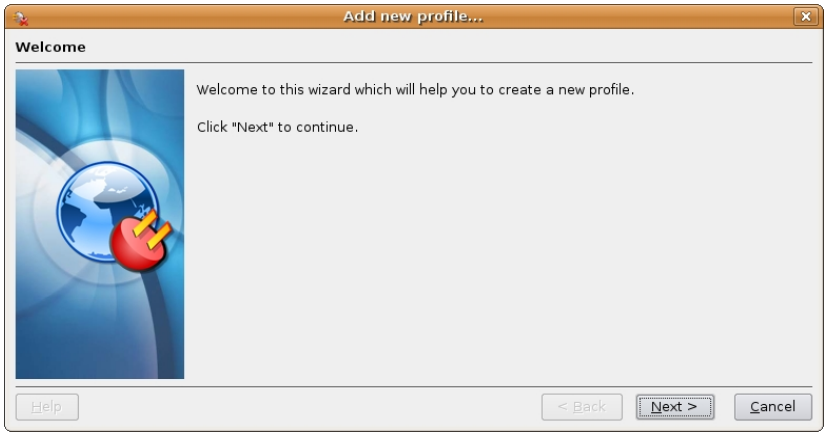


Рис. 1.5. Запуск мастера создания VPN подключения в kpvnc

Рисунки, которые вы видите, относятся к дистрибутиву Ubuntu. Можно бы все проделать в Fedora 7 (последний дистрибутив моей основной ОС), но я придерживаюсь правила, не создавать дополнительных трудностей без особой необходимости. Интернет настроен и работает, зачем переделывать?

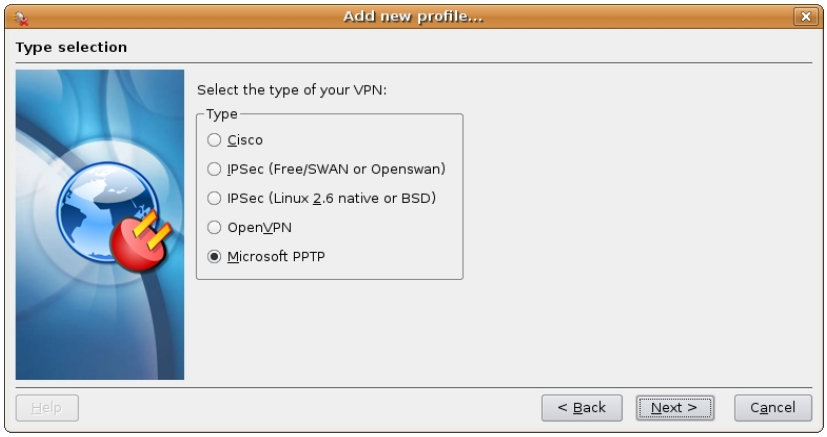


Рис.1.6. Продолжение настройки в диалоге создания VPN подключения

Едва ли мастер подключения сильно отличается от аналогичного в Windows. Как видно из рисунка, возможны разные варианты подключения по VPN. Но меня

интересует один вариант, определяемый моим провайдером, который я и выбираю. После нажатия на клавишу **Next** попадаешь в продолжение диалога, где и вводишь обычные данные об имени пользователя, пароле (как и в Windows) и т.д. Единственное отличие в добавлении двух строк, первая из которых касается IP адреса VPN-сервера (на рисунке ниже это 1.1.1.1) и IP адреса шлюза (на рисунке это 2.2.2.2), а вторая выглядит одинаково, я думаю, для всех пользователей.

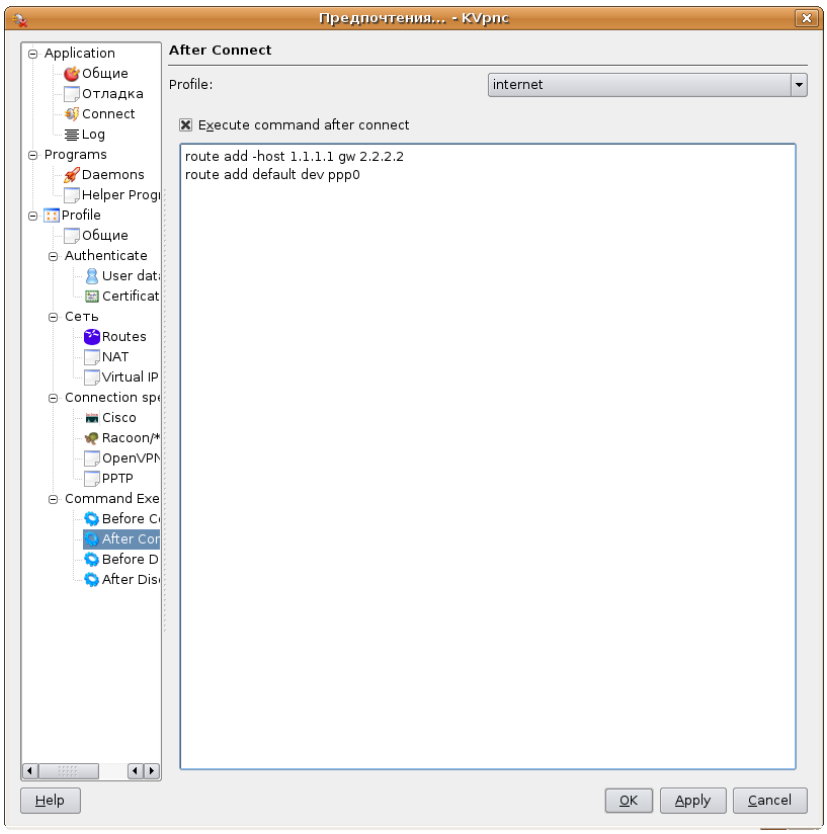


Рис. 1.7. Добавление двух строк в раздел After Connection

Это добавление необходимо сделать после завершения работы мастера настроек, запустив программу Kvpnc, и войдя в подменю After Connection меню настроек Settings. Нажатием клавиши **Apply**, как обычно, мы подтверждаем сделанные изменения, нажимаем клавишу **OK**, и теперь можем запустить соединение, найдя в разделе Приложения, в подразделе Интернет программу Kvpnc.

Запущенная программа запросит пароль администратора компьютера (или root), а

затем выведет окно подключения.

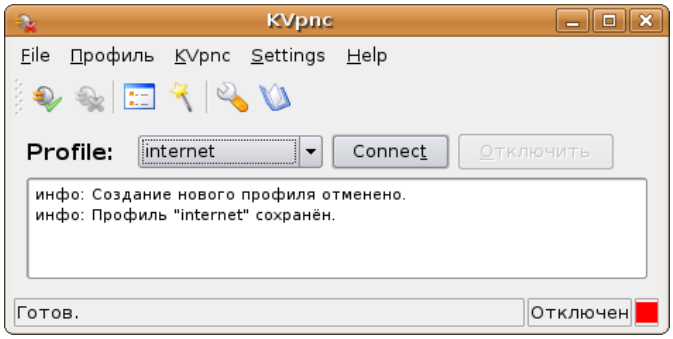


Рис. 1.8. Запуск программы kvpnc после настройки соединения

Осталось нажать клавишу **Connect**. После чего программа выводит сообщение о состоянии подключения (в нижней части экрана).

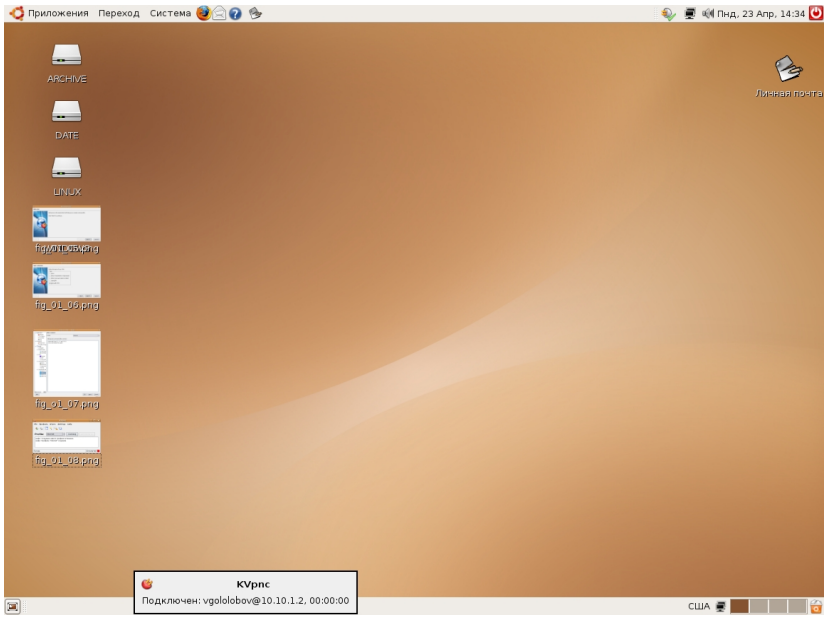


Рис. 1.9. Сообщение о подключении к Интернету в Ubuntu

Как и при настройке vpn-подключения, в Ubuntu я встретил несколько мелких

проблем. Например, выбрав русскую раскладку клавиатуры при установке Ubuntu, я получил раскладку, в которой знаки препинания: точка, запятая, двоеточие и т.д., – расположены не вполне удобно для меня. Да, достаточно в файле xorg.config найти строку раскладки и исправить us, ru на us, ru(winkeys), а затем выбрать настройки X в выводимом сообщении после перезапуска X-сервера, то есть, завершения сеанса и начала нового, чтобы проблема была устранена, но приходится вспоминать, как разобраться с этой особенностью каждый раз после установки ОС. Или другая нелепость, появившаяся недавно. Попытка запустить привычный мне почтовый клиент Evolution выводит сообщение о фатальной ошибке, да и то при запуске из терминала, обычный запуск «умирает молча». Проблема решается почему-то весьма своеобразным образом: необходимо изменить часовой пояс в настройках даты-времени, задав, как рекомендовали на форуме, часовой пояс Киева. Запустить Evolution, настроить почту, после чего можно вернуть нормальный часовой пояс. Такие особенности настроек, я думаю, доставляют большое удовольствие любителям «поковыряться» в ОС, но очень мешают остальным пользователям переходить на Linux. Я пользуюсь vpn-сервером, где для ускорения работы отключается MPPE. Это заставляет меня в файле options.pptp (/etc/ppp) удалять (или закомментировать) строку require-mppe-128. Это моя частная проблема, но есть и еще одна, которая может оказаться тоже частной, не знаю. В том виде, который настройки приобрели после создания vpn-подключения, я могу зайти на многие сайты, но не могу зайти на www.yandex.ru. У меня там почта, у меня там мой сайт, а зайти я не могу. Обидно.

В чем же проблема? В том, что по умолчанию используется MRU (не MTU, заметьте) со значением 1000. Графический интерфейс позволяет задать пользовательское значение MTU, но не MRU. После настроек в папке /etc/ppp/peers появляется файл сохранения настроек Kvpnc, где это отображено. Но попытки ввести исправление в этот файл не дают окончательного и бесповоротного решения. Во всяком случае на моем компьютере. Дело в том, что запуск Kvpnc проходит с вводом пароля root, при этом файл конфигурации переписывается. Чтобы он переписывался с нужным значением MRU необходимо в домашней папке root найти файл в скрытой папке .kde с именем kvpncrc, где и изменить значение MRU (для отображения в файловом менеджере скрытых файлов необходимо установить опцию «Показывать скрытые файлы» в разделе «Вид» основного меню).

Если не придавать значения двум строкам маршрутизации подключения после его установления, и тех правок в файлах, которые мне приходится делать, то формально остальные процессы не сложнее, чем в Windows. Да и вид экрана в операционной системе Linux мало отличен от Windows. А если кого-то это смущает, он может выбрать графический интерфейс KDE, или даже другой дистрибутив, который еще больше приближен по внешнему виду к Windows. Мне как пользователю, гораздо удобнее выполнять, скажем прямо, совсем мне не нужную процедуру создания подключения к Интернету (я предпочел бы иметь выход в Интернет сразу после физического подключения к сети) в графическом диалоге, а не писать коды в нескольких файлах. Тем обиднее, что все-таки приходится записывать что-то для

маршрутизации, обидно, что приходится прописывать IP адрес vpn-сервера, что приходится искать, где изменить MRU. Все это наводит на грустные размышления, как то: был бы я столь же доволен использованием Linux, если бы несколько лет назад не получал удовольствия от возни с настройками, поиска решения проблем на Linux-форумах, если бы не привык к некоторым особенностям любого дистрибутива Linux...

Но я совсем отвлекся от рассказа. Красоты последних дистрибутивов, досадные мелочи, всего много, так и не доберешься до дела.

Запускаем Gambas, который находится в разделе Программирование основного меню Приложения. К слову, я года два использую графическую оболочку Gnome, хотя изредка захожу в KDE, прежде основную рабочую графическую оболочку. Gnome установился по умолчанию, я решил посмотреть, насколько сильно он изменился, да так и остался в нем. Но я опять отвлекся:

Рис. 1.10. Первый запуск среды программирования Gambas

Даже первый запуск, кроме традиционных и необходимых действий, таких как: создать новый проект, открыть существующий, использовать недавно использовавшийся проект, - имеет раздел примеров (Examples). И примеров очень много. Я обязательно хочу ими воспользоваться, когда начну работу проектами. А вот первый проект, все-таки не могу я все делать «по правилам», первый проект я возьму из книги М. Мозгового «Занимательное программирование», да простит меня автор, не только потому, что пример сам по себе интересен, но и потому, что автор

приводит его в среде программирования Delphi. Я хочу повторить этот пример в качестве первого своего Gambas проекта, что называется «в лоб». В этом я доверюсь автору книги «Занимательное программирование», который не соотносит приведенные им программы с каким-то конкретным языком, он использует Pascal, язык системы программирования Delphi, как знакомый многим по студенческим временам. Мне же крайне любопытно, насколько возможно, не умея программировать, повторить программу, написанную на другом языке программирования. Если ничего не получится, брошу все. Или не брошу.

Я еще подумаю.

После выбора New project... (создание нового проекта) открывается диалог, в котором можно ввести название проекта, его заголовок, указать место, где проект будет располагаться. Gambas хранит все, что относится к проекту, в заданной директории, которая будет носить имя проекта. Для перемещения по диалогу достаточно нажимать клавишу **Next**, но в самом начале следует выбрать, какого рода проект вы намерены создать. На выбор три варианта – графический проект, терминальный проект, использование копии существующего (вашего или нет) проекта:

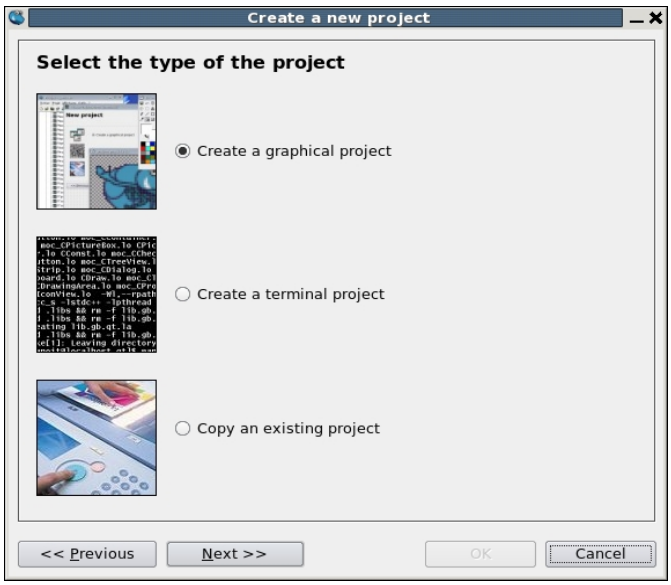


Рис. 1.11. Диалоговое окно выбора типа проекта

Меня интересует графический проект, и я оставляю флажок выбора там, где он есть: «Create a graphical project» – знай наших! Многие расчетные задачи, к чему и должен быть применен компьютер по его замыслу, прекрасно можно выполнить в

терминале. При этом вы получите программу, которая будет весьма неприхотлива к возможностям компьютера, будет работать, как я себе представляю, без загрузки графической оболочки, а для удобства использования, если вы, как и я, не очень любите набивать множество команд, можно использовать MC – программу (или графический менеджер), которая называется «Midnight Commander», чтобы его не путали с «Norton Commander», которого в Linux нет. Вообще я давно «подсел» на графические возможности современных операционных систем, и сегодня, каждодневно обновляя систему, не столько озабочен ее стабильностью, давно уже не замечал каких-либо проблем с этим, не столько радуюсь обновлениям, связанным с вопросами защиты системы, вирусы в Linux не очень свирепствуют, сколько жду перемен в ее внешнем облике. Хотя понимаю, что самое разумное, это использовать всю мощь операционной системы без «графических» тормозов. Но...

Итак. Выбрав графический проект, дав ему имя и место жительства, мы попадаем в среду программирования, которая позволит сделать первый шаг – создать форму. Для этого в правом окне менеджера проекта, где отображается «дерево» проекта, я щелкаю правой клавишей мышки по разделу Forms, а в выпадающем меню выбираю раздел New, что приводит меня к меню выбора, в котором есть пункт Form. Этот процесс я понимаю так – создать новую форму. Форма же, насколько мне известно, а я уже немного «законспектировал» пояснения к языку Gamabs, особенно для моего графического выбора, является основным окном программы, которое появится, когда я заполню форму некоторым содержанием, и построю сам проект. В окне диалога создания формы можно задать имя формы, что важно для средних и больших проектов, содержащих много окон, можно установить ряд опций, назначения которых я пока не понимаю, поэтому не хочу их устанавливать:

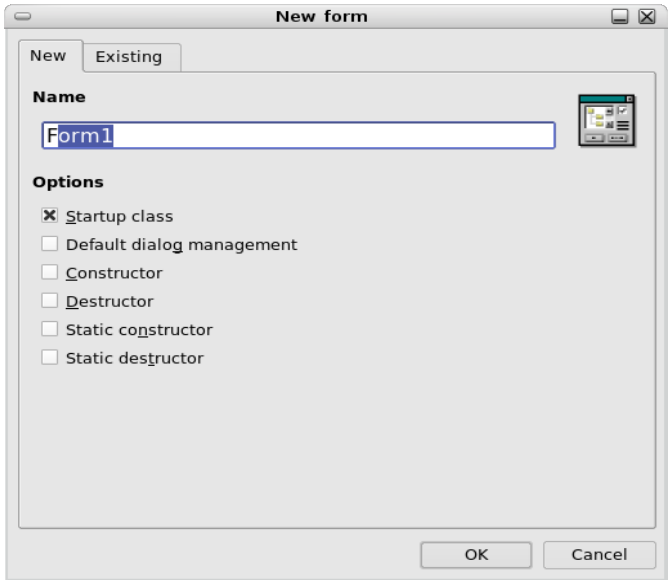


Рис. 1.12. Диалоговое окно создания формы в Gambas

Я не готов создавать средние и большие проекты, поэтому оставлю название формы без изменения и поскорее щелкну по клавише **OK**. В результате появляется форма и редактор программы. На форме, видимо, можно будет разместить разные кнопочки, окошки и другие средства управления программой. А в редакторе поместится все то, что и есть код программы. Следуя совету, прочитанному в книге «Занимательное программирование», я добавлю на форму область рисования (DrawingArea) и клавишу управления (Button). И то, и другое я выбираю на инструментальной панели справа, щелкнув по ним левой клавишей мышки, а затем с помощью мышки «нарисовав» контур окна и клавиши. На рисунке ниже кнопка управления на инструментальной панели изображена с надписью «OK», а область рисования имеет внутри пиктограмму, изображающую цветные геометрические фигуры: квадрат, круг, треугольник.

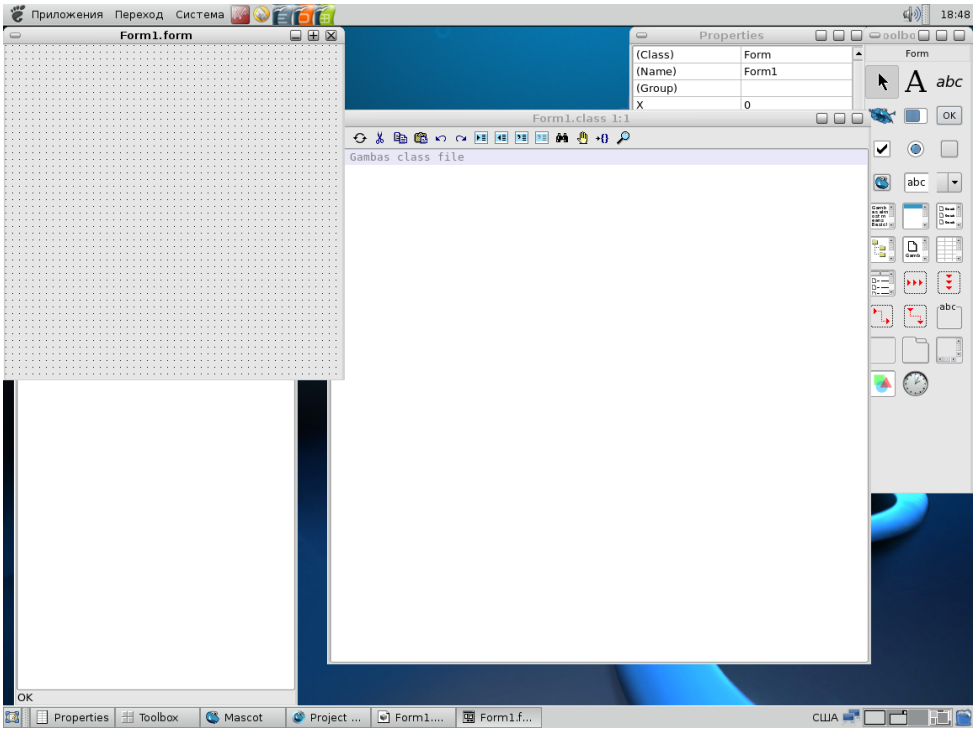


Рис. 1.13. Основная форма проекта и редактор кода

Форма, как и многие элементы управления, имеет свойство Text. Я полагаю, что это текст, который появится вместо надписи Form1.Form при выполнении программы. Этот текст я заменю своим, и, если программа заработает, то увижу его.

После нескольких входов в Gamabas и выходов из него, я сталкиваюсь с маленькой, но для меня очень неприятной особенностью – я не знаю, как вызвать редактор кода. Мне же понадобится куда-то вписывать код. Я знаю, уже знаю, что двойной щелчок по кнопке управления открывает редактор кода и создает первую и последнюю строку текста события, связанного с щелчком по клавише при работе программы. Двойной щелчок по форме тоже вызывает аналогичное действие. Но это действие мне не нужно, я удаляю ненужный код, а сам прием использую для вызова редактора кода. Но не делать же это всякий раз, когда тебе нужно что-то исправить в программе. Меня это немного раздражает. Но только до тех пор, пока я не догадываюсь щелкнуть по форме правой клавишей мышки. В выпадающем меню есть пункт Code, который вызывает редактор кода и умиротворение в моей душе.

Программа, мой первый «блин комом», выглядит (я скопирую ее из книги

«Занимательное программирование», чтобы не наделать ошибок, которые очень люблю делать при переписывании) следующим образом:

Листинг 1.1. Молекула газа в закрытом сосуде (первая версия)[1]

```
procedure TForm1.StartStopBtnClick(Sender: TObject),
    var X, Y   : Integer; { координаты молекулы }
        Vx, Vy : Integer; { составляющие скорости молекулы }
        angle  : Real;    { угол, задающий начальное направление полета }

    const R : Integer = 10; { радиус молекулы }
          V : Integer = 7;  { скорость молекулы }
begin
    Randomize;
    Screen.Refresh;                          { очистка экрана }
    X := RandomRange(R, Screen.Width - R);   { выбор начального (1)[2] }
    Y := RandomRange(R, Screen.Height - R);  { положения молекулы }
    angle := Random(360)* Pi / 180;          { и ее направления (2) }

    Vx := Round(V * Sin(angle));             { получение составляющих }
    Vy := Round(V * Cos(angle));             { скорости молекулы }

    while true do                            { основной цикл }
    begin
        Screen.Canvas.Pen.Color := clBtnFace;    { стираем молекулу (3) }
        Screen.Canvas.Ellipse(X - R, Y - R, X + R, Y + R);

        X := X + Vx;                       { сдвигаем на новую позицию }
        Y := Y + Vy;
                                           { определяем, не вышла ли }
        if X > Screen.Width - R then       { молекула за границы аквариума }
            begin  X := Screen.Width - R; Vx := -Vx; end;   { правая граница }
        if X < R then
            begin  X := R; Vx := -Vx; end;                  { левая граница }
        if Y > Screen.Height - R then
            begin  Y := Screen.Height - R; Vy := -Vy; end;  { нижняя граница }
        if Y < R then
            begin  Y := R; Vy := -Vy; end;                  { верхняя граница }

        Screen.Canvas.Pen.Color := clBlue;             { рисуем молекулу на }
        Screen.Canvas.Ellipse(X - R, Y - R, X + R, Y + R);   { новой позиции }

        Sleep(10);                                   { пауза 10 миллисекунд }
    end;
end;
```

Рис. 1.14. Листинг программы из книги М. Мозгового «Занимательное программирование»

Очень советую купить книгу, а я своими словами опишу программу. Представьте себе двумерный закрытый сосуд прямоугольной формы, в котором мечется, не находя себе места, шальная молекула газа. И молекула, и сосуд – все идеально. С точки зрения программы это означает, что молекула при своем движении сохраняет величину скорости, а ее направление меняется только при столкновении со стенками сосуда. Не думаю, что без посторонней помощи я смог бы написать такую программу. Я и не уверен, что без посторонней помощи смогу воспроизвести ее. Но попробую.

Первым делом я нарисую, как и собирался, область рисования и кнопку запуска программы на основной (в моем случае единственной) форме. Чтобы изменить название клавиши, после того как она появилась на форме под именем Button1, я выделяю эту клавишу щелчком по ней левой клавиши мышки. Клавиша приобретает вид контура с несколькими квадратиками, с помощью «таскания» за которые можно менять размеры клавиши. А окно свойств Properties открывается на свойствах клавиши, что можно сразу увидеть по первой строке (Class) – Button. В этом окне свойств есть пункт под названием Text в левой колонке. Если правее, в следующей колонке, ввести «Старт», то клавиша изменит свое название, видимое на форме. Аналогично я поступлю с областью рисования, которую назову Screen, как в оригинале, но это будет уже не надпись, как на клавише, а, собственно, имя. И меняю я его напротив (Name) в окне свойств после выделения области рисования щелчком левой клавиши мышки по ней.

Теперь можно начать писать код программы (я прочитал в отличие от вас все, что автор говорит о программе). Поскольку программа должна запускаться после нажатия на клавишу «Старт», а в руководстве к Gamabs я нашел, что двойной щелчок по элементам управления вписывает код, который создает заголовок для кода программы, отвечающего на событие – щелчок мышки по клавише управления, я дважды щелкаю по клавише и получаю в редакторе кода первую запись (увы, пока не мою) кода программы. В листинге на рисунке 1.9 аналогичная запись начинается словом **procedure,** а ниже начинается описание переменных и констант. Не буду фантазировать, открою с разделе помощи пункт означенный «Объявление локальных переменных» и объявлю их, а заодно и константы:

```
PUBLIC CONST R AS Integer = 10
PUBLIC CONST V AS Integer = 7

PUBLIC SUB Button1_Click()
       DIM X AS Integer
       DIM Y AS Integer
       DIM Vx AS Integer
       DIM Vy AS Integer
       DIM angle AS Float
END
```

А вот и не заодно. Пришлось константы вынести из блока программы, обслуживающего щелчок по клавише «Старт», иначе возникает ошибка при

компиляции программы. Осерчав на себя, из-за того, что не понимаю, почему так, я обозначил их PUBLIC, то есть, доступные всем другим классам (так написано в help). Возможно со временем я и пойму, почему и отчего, но пока мне приходит в голову только одно соображение – константа вещь постоянная и ее следует рассматривать глобально, но это мои фантазии. Лучше продолжим. В листинге первая операция в программе Randomize. Попробуем отыскать аналогичную в описании языка Gamabas:

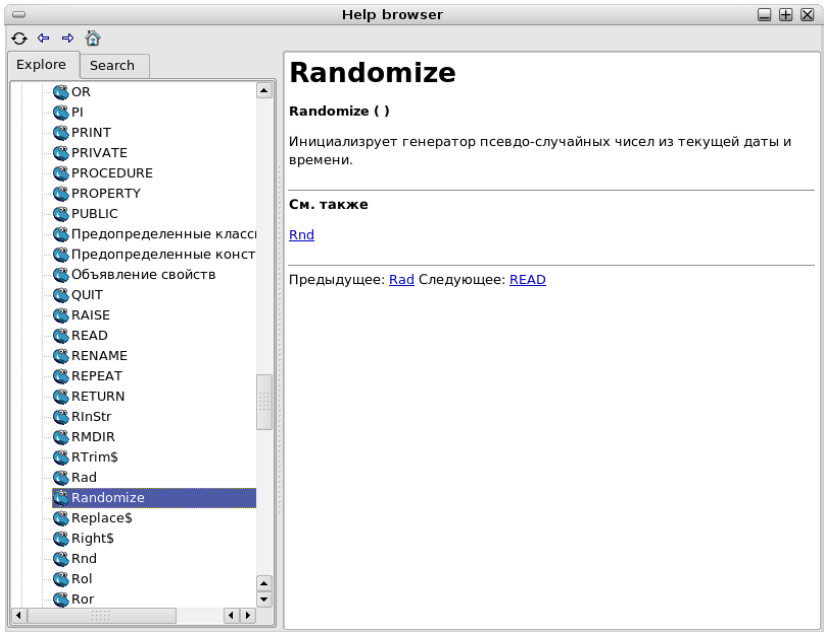


Рис. 1.15. Подсказка по языку Gambas

Следующий оператор в листинге, если я не ошибаюсь, обновляет область рисования. Для получения этого я воспользуюсь свойством редактора кода – после точки, которую можно поставить после имени объекта, открывается окно свойств и методов объекта. Так построены многие среды разработки программ, так построена и среда программирования Gambas. Конечно не слишком трудно вписать операторы, методы или свойства, это так, но мне не хочется начинать с опечаток и ошибок в написании текста кода. Поэтому я воспользуюсь возможностями, предоставляемыми хорошо сделанной системой:

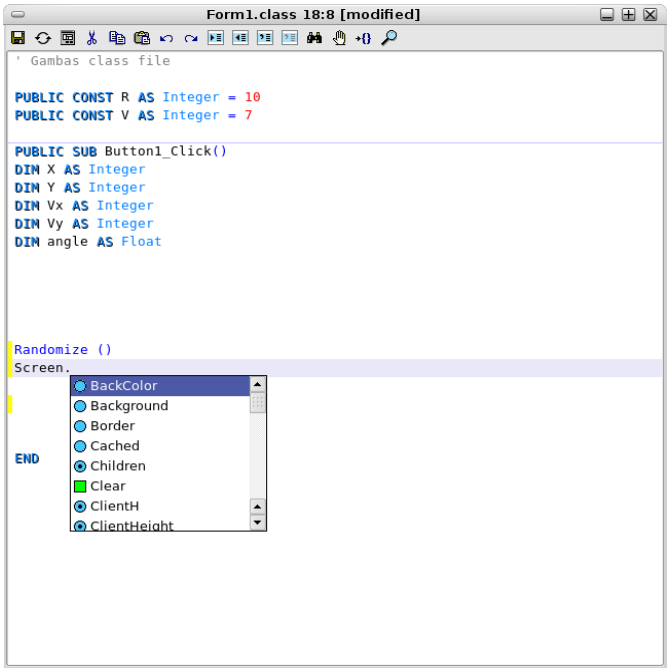


Рис. 1.16. Меню методов и свойств объекта

В открывающемся окне я постараюсь найти что-то похожее. И нахожу. Затем, используя оператор Rnd, я повторю записи, сделанные в листинге. Теперь программа выглядит так:

```
' Gambas class file

PUBLIC CONST R AS Integer = 10
PUBLIC CONST V AS Integer = 7

PUBLIC SUB Button1_Click()
        DIM X AS Integer
        DIM Y AS Integer
        DIM Vx AS Integer
        DIM Vy AS Integer
        DIM angle AS Float

        Randomize ()
        Screen.Refresh
        X = Rnd (R, Screen.Width - R)
        Y = Rnd (R, Screen.Height - R)
        angle = Rnd (0, 360)*Pi/180
        Vx = Round (V*Sin(angle))
```

```
        Vy = Round (V*Cos(angle))
END
```

Теперь осталось вставить цикл. В коде цикла есть несколько записей, которые я не знаю, как выполнить, и пока оставлю программу без них, попробовав ее запустить:

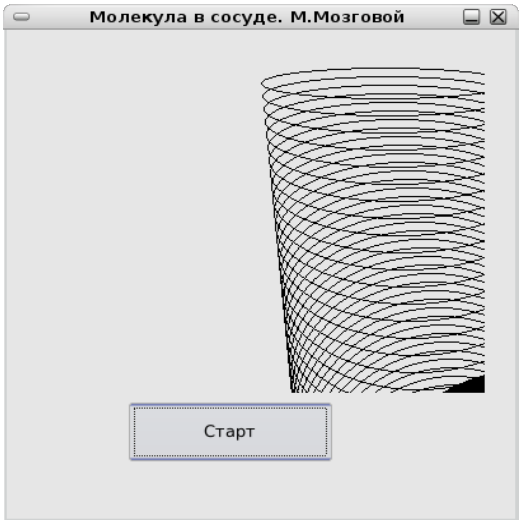


Рис. 1.17. Первый запуск незавершенной программы

Это не то, конечно, что должно получиться. Первое, что я пока не знаю как сделать, это – стереть молекулу. Или, если быть точным, не знаю как ее нарисовать цветом фона, как это сделано в оригинальном листинге, а позже перерисовать в новой позиции синим цветом. И я не знаю, как сделать паузу, поскольку не нахожу подходящего оператора. Кроме того, я ожидал, что молекула будет меньше, и у меня она, похоже, не круглая. Но с этим, я надеюсь, разобраться будет проще. Поищем ответы на основные вопросы.

Один ответ нашелся быстро: пауза в 10 миллисекунд получается с оператором WAIT 0.01, где число – необходимая длительность паузы. Теперь рисунок образуется не сразу, как прежде, а постепенно. Устранив ошибку, которой я обязан своей невнимательности, и из-за которой не происходило отражения от правой и нижней границ области рисования (в одном месте вместо Y я написал X), и заменив оба параметра размера эллипса радиусом, отчего «молекула» перестала менять размер при движении, я получаю результат близкий к ожидаемому. Но я все еще не умею стирать молекулу. Без этого результат работы программы выглядит примерно так:

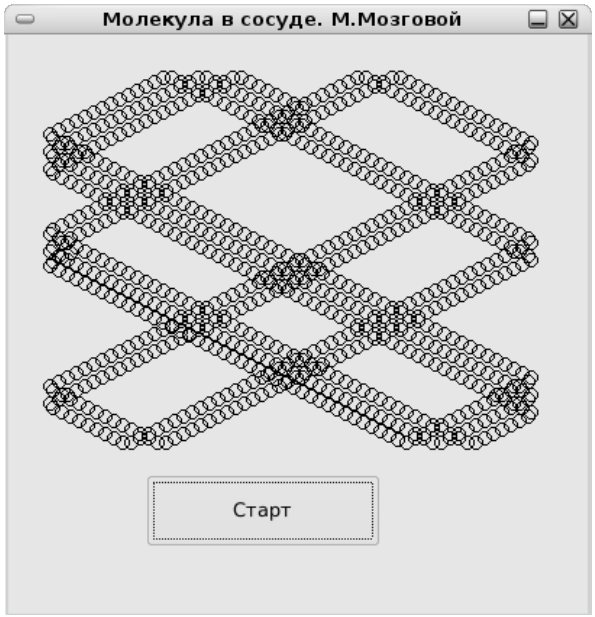


Рис. 1.18. Запуск программы после устранения ошибок

И, наконец, с помощью операции присваивания объекту рисования цвета формы

```
Draw.ForeColor = Form.Background
```

мне удается «стереть» молекулу в начале цикла, а с присвоением синего цвета ближе к концу цикла

```
Draw.ForeColor = 255
```

получить то, что я представляю, как программу из книги «Занимательное программирование». Выглядит она в Gambas так:

```
' Gambas class file

PUBLIC CONST R AS Integer = 10
PUBLIC CONST V AS Integer = 7

PUBLIC SUB Button1_Click()
        DIM X AS Integer
        DIM Y AS Integer
        DIM Vx AS Integer
        DIM Vy AS Integer
        DIM angle AS Float
```

```
        Randomize ()
        Screen.Refresh
        X = Rnd (R, Screen.Width - R)
        Y = Rnd (R, Screen.Height - R)
        angle = Rnd (0, 360)*Pi/180
        Vx = Round (V*Sin(angle))
        Vy = Round (V*Cos(angle))

        WHILE TRUE
                Draw.Begin (Screen)
                Draw.ForeColor = Form.Background
                Draw.Ellipse (X-R, Y-R, R, R)
                X = X + Vx
                Y = Y + Vy
                IF X > Screen.Width - R THEN
                        X = Screen.Width - R
                        Vx = -Vx
                ENDIF

                IF X < R THEN
                        X = R
                        Vx = -Vx
                ENDIF

                IF Y > Screen.Height - R THEN
                        Y = Screen.Height - R
                        Vy = -Vy
                ENDIF

                IF Y<R THEN
                        Y = R
                        Vy = -Vy
                ENDIF

                Draw.ForeColor = 255
                Draw.Ellipse (X-R, Y-R, R, R)
                Draw.End
                WAIT 0.01
        WEND
END
```

И работает это:

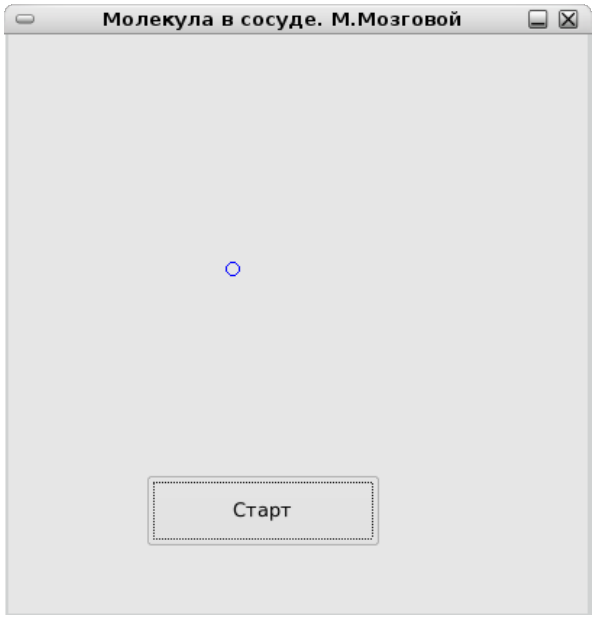


Рис. 1.19. Окончательный результат работы программы

Если отбросить некоторые языковые различия, такие, например, как операторные скобки рисования:

```
Draw.Begin (Screen)
Draw.End
```

и различия в написании, что и должно было иметь место, то в остальном код программы действительно переписан «в лоб». Следовательно, предположение, что можно на протяжении книги использовать другую среду программирования, достаточно хорошо подтверждается. А сама среда программирования, по крайней мере для небольших проектов, работает не хуже, чем Delphi. Так что, надеюсь, к концу книги, хотя программистом я и не стану, но немного программировать научусь.

А сейчас я вспомнил, что хотел посмотреть раздел примеров в Gambas. Вот и посмотрю. Но прежде хочу добавить в программу, то что предлагает ее автор в самом начале разговора о кодировании программы – добавить возможность останавливать программу не принудительно с помощью средств разработки, нажимая на красную клавишу «Стоп» основного меню, а использовать ту же кнопку управления, что уже есть на форме. В этом случае начало программы в Gambas выглядит так:

```
 ' Gambas class file

PUBLIC CONST R AS Integer = 10
PUBLIC CONST V AS Integer = 7
PUBLIC Running AS Boolean

PUBLIC SUB Button1_Click()
        DIM X AS Integer
        DIM Y AS Integer
        DIM Vx AS Integer
        DIM Vy AS Integer
        DIM angle AS Float

        IF Running THEN
                Running = FALSE
                QUIT
        ENDIF

        Button1.Text = "СТОП"
        Running = TRUE
        ..........................
```

Далее программа остается без изменения. Так, действительно, удобнее. А примеры, которые есть в Gambas заставляют меня тихо завидовать тем, кто умеет так программировать.

Рис. 1.20. Пример работы программы из раздела Examples системы Gambas

Часы не только ходят после запуска программы, но и показывают правильное время. Что ж, будем учиться.

Компьютер может помочь с программированием

Во всяком случае, так говорят знающие люди. И они же говорят, что программирование – это не написание самой программы, то есть, кода программы на каком-либо языке программирования, программа, это в первую очередь описание того, что должно происходить с того момента, как программа запущена. Для этого предварительного описания можно использовать, как мне кажется, обычный язык. В одной из книг, которые я сейчас просматриваю, пытаясь научиться программировать, говорится о псевдо-языке программирования, с помощью которого можно описать программу. А на моем компьютере есть программа Umbrello, назначение которой мне давно хотелось понять. У меня эта программа в разделе «Программирование» основного меню «Приложения». Запустить программу не составляет труда – это я уже освоил. И руководство в разделе «Справка» основного меню, то, что сейчас надо.

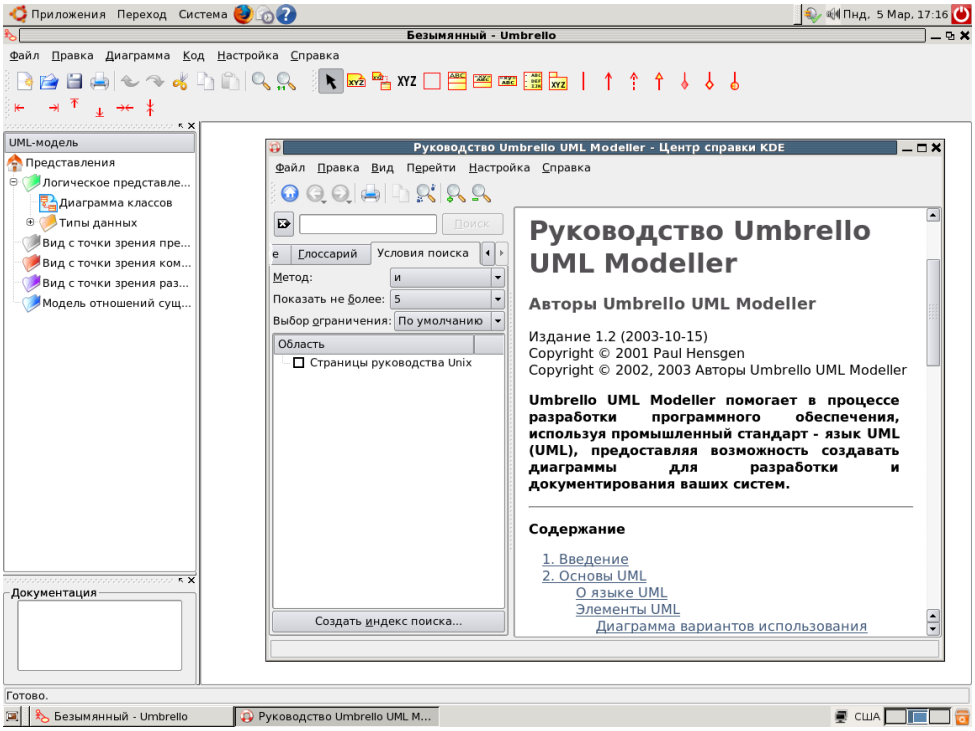


Рис. 1.21. Программа Umbrello

На рисунке можно разглядеть, что Umbrello помогает в процессе разработки программного обеспечения. Это-то мне и нужно. Ничто так ни важно для

начинающего, как помощь на первом этапе освоения нового.

Руководство к программе русифицировано, как и сама программа, и против своего обыкновения, надеюсь вы лишены этого недостатка, прежде, чем начать пользоваться программой, я попробую прочитать руководство. Отдельные моменты, чтобы вам не искать, процитирую. Например, вот это:

> *Umbrello UML Modeller — это средство для работы с диаграммами UML, которое может быть полезным в процессе разработки программного обеспечения, особенно на стадиях анализа и проектирования...*
> *Хорошая модель очень важна для средних и больших проектов, но также будет полезна и для небольших проектов. Даже если вы один работаете над маленьким проектом, хорошая модель предоставит вам общую картину разработки, что поможет вам сразу написать качественный код.*
> *UML является языком диаграмм, используемым для описания таких моделей. Вы можете выражать свои идеи в UML с помощью различных типов диаграмм...*
> *Унифицированный язык моделирования (UML) является языком диаграмм или обозначений для спецификации, визуализации и документации модели объектно-ориентированных программных систем. UML не является методом разработки, то есть он не определяет последовательность действий при разработке программного обеспечения. Он помогает описать свою идею и взаимодействовать с другими разработчиками системы.*

У меня проект маленький. И других разработчиков не предвидится. Но это и не обязательно. Первое, что мне хочется сделать, иначе трудно будет разбираться с программой позже, это привести руководство в соответствие с имеющейся у меня версией. Руководство, видимо, написано для предыдущей версии, и, хотя принципиальных различий может не быть, детали, пока не освоился, тоже могут помешать пониманию. Для себя я запишу соответствие названий диаграмм в руководстве и моей версии:

> *Диаграмма класса - Диаграмма класса*
> *Диаграмма последовательности - Диаграмма последовательности*
> *Диаграмма взаимодействий - Диаграмма отношений сущностей*
> *Диаграмма вариантов использования - Диаграмма прецедентов*
> *Диаграмма состояний - Диаграмма состояний*
> *Диаграмма действий - Диаграмма деятельности*
> *Диаграмма компонентов - Диаграмма компонентов*
> *Диаграмма выпуска - Диаграмма развертки*

Есть еще в моей версии «Диаграмма кооперации», но куда ее приткнуть? Почитаем.

В руководстве часто упоминается слово «класс». Запишу-ка я, что это значит:

> *Класс определяет атрибуты и методы набора объектов. Все объекты класса (называемые экземплярами) имеют одинаковое поведение и одинаковый набор атрибутов (у каждого объекта - собственный набор атрибутов). Иногда вместо класса используется термин «Тип», но важно понимать, что эти термины неодинаковы. Термин «Тип» имеет более общий смысл.*

Примеры использования диаграмм в руководстве есть. Пора и нарисовать что-нибудь. Но я на свет, конечно, не вчера родился. Знаю, что первый шаг в любом деле, определение цели. Это потом можно подумать, как достичь намеченного. А моя цель сейчас – разобраться со схемой, которая не желает работать. Сразу разобраться со всей схемой я не осилю, поэтому постараюсь сделать из проблемы маленькую, но чем-то похожую, задачку.

Пусть будет такая маленькая задачка: есть цифровой счетчик с предустановкой и сменой направления счета. Как его проверить? Я пытался посмотреть сигналы осциллографом, но ничего не получилось. Не пойму, работает счетчик, нет; может микросхема плохая попалась, или я ее успел «сжечь». Изобразим счетчик.

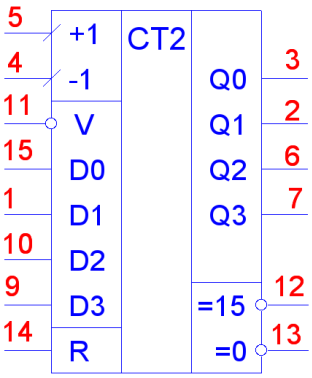


Рис. 1.22. Двоичный реверсивный счетчик с предустановкой

Специалисту ясно, что данные для записи следует подать на входы D0-D3, а перепись данных осуществится при низком уровне на входе V. Для прямого отсчета служит вход +1, фронт от низкого состояния к высокому, и т.д. Вход, который не используется при подключении к тактовому генератору, это я прочитал в книге В.Л. Шило «Популярные цифровые микросхемы», много раз она меня выручала,

неиспользуемый вход направления счета следует установить в высокое состояние.

Собственно, я пытался проверить это на макетной плате, но с одним генератором и осциллографом... Вот и хочу я, чтобы компьютер заменил мне и все нужные генераторы, и числа для предустановки задавал, и... и потом еще придумаю, чтобы он такое делал.

Безусловно, я понимаю, что сам компьютер не догадается помочь мне, пока не попросишь его с помощью программы. Придется попросить. Для начала нарисую, как я буду работать с программой. Это только первые соображения.

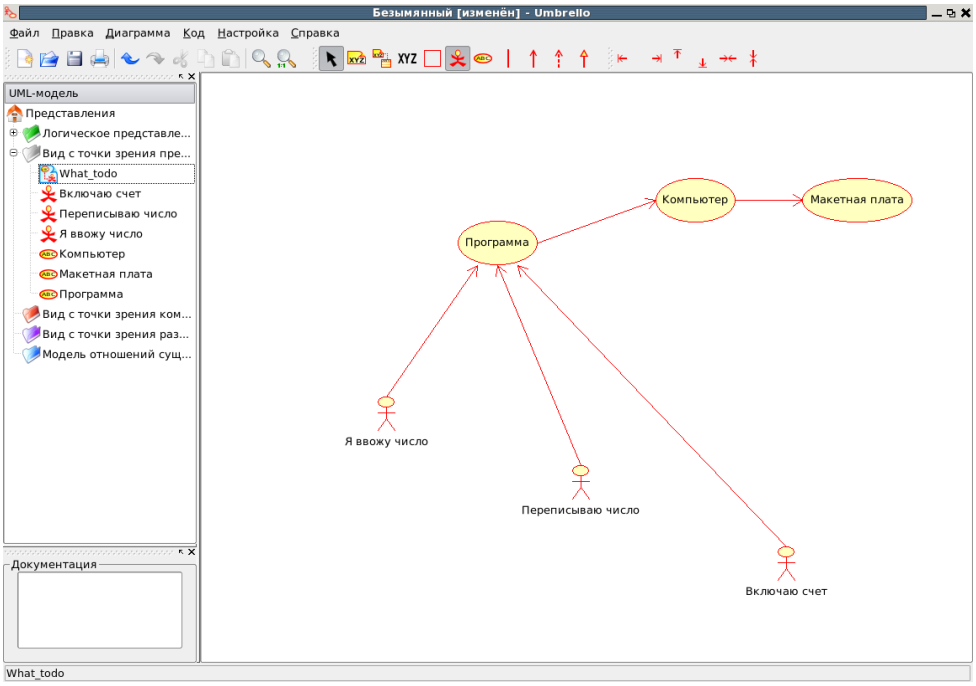


Рис. 1.23. Первые соображения о том, что следует делать

Меня можно спросить, почему «меня» так много на рисунке? Отвечу, так сложилось исторически. Меня можно спросить, а как компьютер будет «общаться» с макетной платой? Отвечу – на каверзные вопросы не отвечаю!

Очень часто устройства, добавляемые к компьютеру, подключают к его шине внешних устройств, но я не намерен разбирать компьютер. Очень часто внешние устройства подключают к порту принтера. Такое подключение обеспечивает достаточно быстрый обмен информацией между компьютером и подключенным устройством, дает возможность, переделав программу, привести характер работы

порта в соответствие с текущей необходимостью, однако я предпочитаю поступить иначе. Мне нет необходимости в быстром обмене информацией, скорее, наоборот.

В предыдущей серии экспериментов я использовал последовательный порт компьютера, и у меня остался интерфейс RS485 и макетная плата с панелькой для микроконтроллера. Пусть микроконтроллер обслуживает проверяемую схему, а с компьютера будут отправляться команды микроконтроллеру.

Есть несколько соображений в пользу такого решения. Первое из них – мне хочется рассказать о программе Piklab больше, чем я это сделал в прошлый раз. Программа работает с PIC-контроллерами в Linux, как MPLAB в среде Windows. И если о работе MPLAB я уже рассказывал, то о Piklab, скорее, упоминал.

Кроме того, задумав сделать из компьютера нечто похожее на испытательный стенд, я подумал о том, что не всегда у меня будет желание переносить его к рабочему столу, на котором у меня разложены макетная плата, лабораторный блок питания и прочие аксессуары «железячного» бытия. Бросить же на время экспериментов от компьютера к макету тонкий и легкий четырех-жильный телефонный провод можно просто по полу через всю квартиру.

Но самым существенным аргументом для меня в подобном решении будет один – если мне понадобится посмотреть осциллографом сигналы, то мне понадобятся короткие и «жесткие» по времени последовательности импульсов. Микроконтроллер справится с этой задачей лучше компьютера. И в этом смысле, если я не ошибаюсь, а если ошибаюсь пусть опытные программисты меня поправят, я буду использовать программу, работающую на компьютере, в основном, в качестве удобного графического интерфейса пользователя. А вся аппаратная часть задачи будет реализована на макетной плате, возглавляемой микроконтроллером. Но пока это мечты.

Если запускаешь программу, то обычно открывается окно, где сверху есть меню, а в окне можно писать, или рисовать, или что-то строить. Мне строить еще рано, а на первых порах нужно только одно маленькое окошко и одну-две кнопки. В окошко я собираюсь вписывать число для предустановки, одна кнопка мне нужна для установки этого числа на выводы микросхемы, вторая кнопка для переписи числа и начала счета.

После этого программа должна заставить компьютер осчастливить меня в качестве генератора всех необходимых мне сигналов. Сильно подозреваю, что если я положу макетную плату на компьютер, то она не найдет общего языка с ним. Придется что-то еще добавить. Скорее всего выглядеть моя задумка будет как-нибудь так:

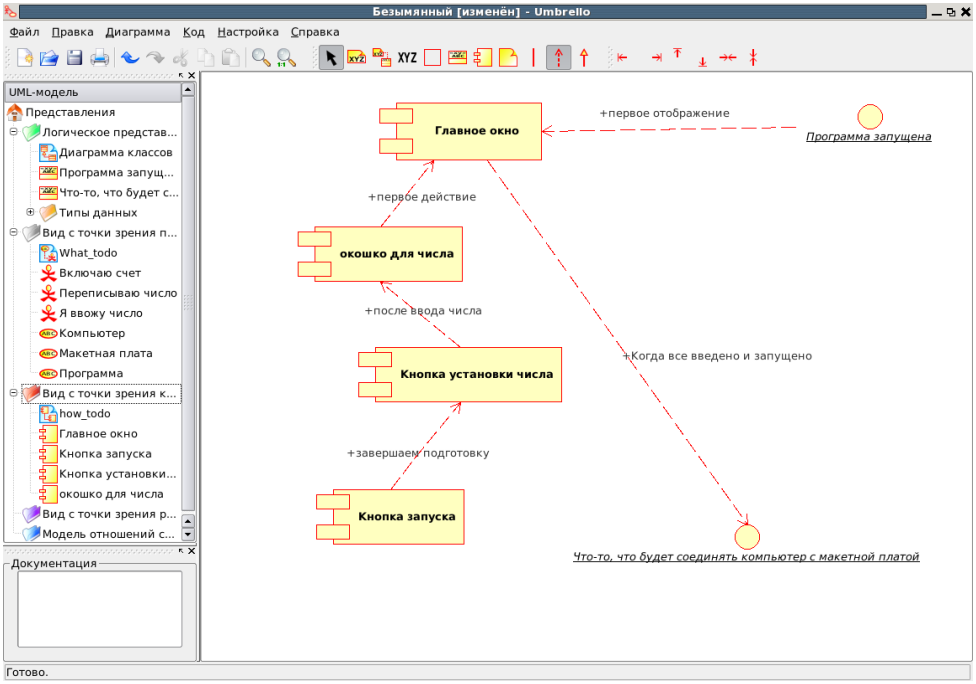


Рис. 1.24. Окончательные соображения перед началом программирования

Я люблю рисовать квадратики, кружочки, рожицы, когда пытаюсь сообразить, что же делать дальше. Я понимаю, что программа Umbrello не для того предназначена, но я еще вернусь к ней. А сейчас у меня план такой – познакомиться с таинственными сущностями, что окружают программиста.

Почему Емеля ездил на печи?

Думаю, и это только мое личное мнение, что тогда он еще не заработал на приличный «Мерседес».

Примерно так же я представляю себе соотношение процедурного и объектно-ориентированного языков программирования, и это тоже только мои предположения. В моих ветхозаветных воспоминаниях о программировании, если отбросить вопрос о типах данных, таких как целое, символ, число с плавающей точкой и т.д., не обращать внимание на переменные, без которых я и сегодня не представляю, может ли существовать программа, и операторах языка, если забыть о подпрограммах, функциях и процедурах, то программа сводится к набору последовательных инструкций, предназначенных для компьютера (или процессора). Эта последовательность может прерываться, когда нужно что-то проверить: выходя из дома, проверяешь, а взял ли ты ключи, надел ли башмаки? Если надел, то можно идти дальше, иначе нужно вернуться и сменить обувь. Иногда последовательность прерывается неким образом, похожим на тот случай, когда уходя из дома, задумываешься, погасил ли ты свет? Ты возвращаешься, проверяешь, уходишь, но по дороге тебе приходит в голову, а вспомнил ли ты о том, что надо проверить, погасил ли ты свет? Этот круговорот рассеянности заканчивается только тогда, когда все у тебя сходится: и проверил, и не забыл, и выключено все.

В таком программировании мне понятно назначение подпрограмм – если часто делаешь одно и то же, как, например, поездка на работу, нет смысла каждый раз расписывать это в последовательных инструкциях. А компьютеру нужно все объяснить до мелочей. Можно один раз эти мелочи расписать, назвать этот набор инструкций каким-то образом, скажем, «На работу», и впоследствии, как каждое утро, позавтракав, радостно восклицаешь: «На работу!», – вызывать одну и ту же рутинную последовательность действий.

Не могу сказать, что все было ясно и просто прежде, если приходилось сталкиваться с программированием. Одни только указатели в языке С или передача параметров функции по значению и по ссылке вызывали у меня тихую внутреннюю дрожь. А написание программы заставляло открывать справочники и за каждым оператором отправляться в долгое путешествие по этим полезным книгам. Но если код программы был мне не ясен, то сущность процесса, в какой-то мере, была ясна.

Теперь же, объектно-ориентированное программирование лишает меня даже этих скромных познаний: классы, их дети и родители, наследство и права на него...

Вместе с тем, без понимания того, что такое объект и класс, может статься, что и не получится у меня ничего с Gambas.

Одно из определений класса я приводил, взяв его из описания программы Umbrello, приведу второе, взятое из help'a Gambas:

Класс - разновидность шаблона (template), он не используется непосредственно. Для его использования создается копия. Эта копия называется объектом. Вы

можете сделать разные объекты того же самого класса.

Класс - это тип данных, так что объявление объекта похоже на обычное объявление переменной:

object_name AS Class_name

но объект должен быть подтвержден с помощью NEW:

object_name = NEW Class_name

Запишу еще несколько определений, прежде чем начну попытки разобраться с этим. Следующее взято мною из книги С.Липпмана:

Двумя первичными характеристиками объектно-ориентированного программирования являются наследование и полиморфизм. ***Наследование*** *позволяет нам группировать классы в семейства связанных типов, давая возможность разделять общие операции и данные.* ***Полиморфизм*** *позволяет нам программировать эти семейства, 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.*

Взаимосвязь между родительским или базовым классом и его потомками называется ***иерархия наследования.***

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.

Второй уникальный аспект объектно-ориентированного программирования - это ***полиморфизм****: возможность указателю на базовый класс или ссылке явно ссылаться на любой из объектов его производного класса.*

Если вам что-то стало понятно, то я вам завидую. У меня полнейшая каша в голове. А еще есть контейнеры и много еще более загадочных сущностей! Попробую найти еще что-нибудь полезное в Интернете.

Объектно-ориентированное программирование - технология программирования, при которой программа рассматривается как набор дискретных объектов, содержащих, в свою очередь, наборы структур данных и процедур, взаимодействующих с другими объектами.

Отыскалось много интересного и на сайтах университетов. Вот например:

Программы - это совокупность взаимодействующих объектов. Каждый объект отвечает за конкретную задачу. Вычисление осуществляется посредством взаимодействия объектов... Поведение объекта диктуется классом. Данные и поведение представлены в виде классов, экземпляры которых - объекты. Все экземпляры одного класса будут вести себя одинаковым образом в ответ на одинаковые запросы.

Объект проявляет свое поведение путем вызова метода в ответ на сообщение. Интерпретация сообщения зависит от объекта и может быть различной для различных классов объектов.

Для удобства создания нового типа из уже существующих типов, определенных пользователем используется механизм наследования. Классы могут быть организованы в виде иерархического дерева наследования.

Таким образом, Объектно-ориентированный язык должен обладать свойствами абстракции, инкапсуляции, наследования и полиморфизма.

***Инкапсуляция** с сокрытием данных - способность отличать внутреннее состояние объекта и поведение от его внешнего состояния и поведения.*

***Абстракция** - расширяемость типов - способность добавлять типы, определяемые пользователем для того, чтобы дополнить ими встроенные типы. Один из принципов ООП заключается в том, что типы, определяемые пользователем, должны обладать теми же привилегиями, что и встроенные типы.*

***Наследование** - способность создавать новые типы, повторно используя, описание существующих типов.*

***Полиморфизм** с динамическим (поздним) связыванием - способность объектов быть ответственными за интерпретацию вызова функции.*

Принципы Объектно-ориентированного подхода:

- *Действие в объектно-ориентированном программировании инициируется посредством передачи сообщений объекту. Сообщение содержит запрос на осуществление действия. В качестве реакции на сообщение получатель запустит некоторый метод, чтобы удовлетворить принятый запрос.*
- *Все объекты являются экземплярами классов. Все объекты одного класса используют одни и те же методы в ответ на одинаковые сообщения.*
- *Принцип наследования. Классы могут быть организованы в иерархическую структуру с наследованием свойств. Дочерний класс наследует атрибуты родительского класса.*
- *Принцип полиморфизма. Объекты реагируют на одно и тоже сообщение строго специфичным для них образом.*

С удовольствием, и без зависти, я прочитал все найденные мною статьи. И слова, и даже почти все термины в них понятны, но лишь пока читаешь. Стоит на мгновение отвлечься, как, возвращаясь к прерванному чтению, ты уже не так уверен

в том, что тебе все понятно. В этом смысле, боюсь, мне более всего подойдет следующее:

Для тех, кто не имеет об этом ни малейшего понятия, мы не будем подробно объяснять, что такое объектно-ориентированное программирование. В этот вопрос и так уже внесено достаточно путаницы. Поэтому забудьте о том, что люди говорили вам об объектно-ориентированном программировании. Наилучший способ (и, фактически, единственный) изучить что-либо полезное об объектно-ориентированном программировании - это сделать то, что вы уже почти сделали: сесть и попытаться узнать все самостоятельно.

Последую этому совету. А чтобы не сожалеть о времени, потраченном на чтение книг, статей и методических указаний, на первом этапе я определю для себя, что объектно-ориентированное программирование отлично от традиционного в первую очередь тем, что его основным элементом является объект с набором свойств и возможных действий для общения с другими сущностями программы. Это не определение – это маленький островок в той каше-размазне, что у меня сейчас в голове.

По мере работы с языком Gambas я постараюсь возвращаться к найденным мной статьям, проверяя не рассеялся ли туман в голове? А для начала попробую перевести все то немногое, что удалось понять (или кажется, что удалось) на область, которая мне привычнее – электронику.

Как и программирование, электроника развивалась, и если когда-то, как при процедурном программировании, схема строилась на отдельных элементах: это были резисторы, конденсаторы, транзисторы, – то позже появились микросхемы, а с их появлением и другой подход к созданию схем. Схема, как и раньше, рассматривается в функциональном плане. Но если раньше эти функции реализовывались с помощью построения схемы из отдельных элементов, то сегодня для выполнения необходимых действий (операций с сигналами) выбирается подходящая микросхема. Она имеет определенные свойства, как питающее напряжение, цепи коррекции или задания режимов, уровни входных и выходных сигналов, и имеет возможность действовать, преобразуя входные сигналы в выходные, генерируя выходные сигналы при определенных условиях, за которыми микросхема следит и т.д.

Интересный, как мне кажется, пример видения программы в традиционном подходе и с «правильной» точки зрения я встретил в книге С.Н. Лукина «Понятно о Visual Basic.NET». Хотя программа относится к Visual Basic, ее без труда можно реализовать в среде Gambas, что я и хочу сделать. Речь в книге идет о создании программы собственного калькулятора. В моей операционной системе Linux есть штатный калькулятор, приводимый к виду научных расчетов, калькулятор, который позволяет записать достаточно сложное выражение и произвести расчет нажатием на клавишу «=», и я часто им пользуюсь, но... Но в некоторых ситуациях даже он не спасает меня от ворчания пополам с четырханием. Однако вначале о программе.

Запускаем Gambas. Кстати о Gambas. Почти готова вторая стабильная версия программы, и, после некоторого размышления о быстротечности всего сущего, я решил, что далее буду использовать ее. Это версия 1.9.48 (разрабатываемая). Итак, запускаем все-таки Gambas:

Рис. 1.25. Gambas 2, окно запуска

По всей видимости новая версия среды программирования претерпела существенные изменения, и к моменту выхода стабильной версии эти изменения могут еще в большей мере коснуться программы, что, видимо, важно для профессионалов, а не для таких дилетантов, как я. По этой причине, имея первый опыт работы с Gambas, я выбираю раздел **New project**, получая следующие возможности для разработки проекта:

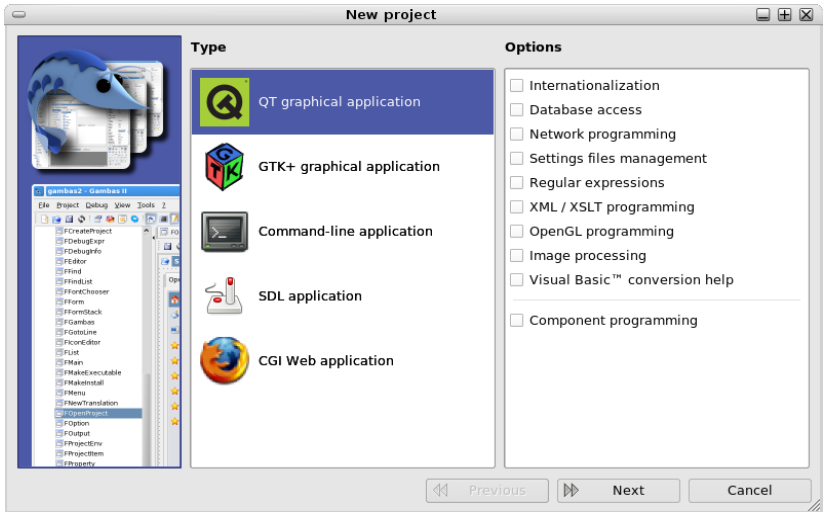


Рис. 1.26. Возможные виды проектов в Gambas 2

Как и ранее, я выбираю для своих целей графический проект с применением средств Qt. Пока я не знаю назначение опций, находящихся в правом окне, отчего и не использую их. Проект я назову «Calc» или как-нибудь похоже. Все элементы управления должны располагаться на форме, для вызова которой я дважды щелкаю в левом окне «дерева» проекта. Что мне сейчас нужно, так это три окошка TextBox и одна клавиша Button. Окошки позволят ввести в них текст, а клавиша пригодится для выполнения расчетов – нажал клавишу, и посчитал. В первом окне будем записывать первое число, во втором окне – второе. Результат отобразится в третьем. Для получения доступа к объектам я открываю Toolbox в разделе основного меню View. Можно нажать клавишу F6, как во многих программах. Затем, используя окно диалога свойств, оно открылось при запуске, но можно его открыть с помощью того же пункта основного меню View (Properties), я нахожу в свойствах окошек TextBox1 – TextBox3 пункт Text, где и записаны эти имена, чтобы удалить их, как ненужный мне текст. Это не обязательно, но тогда в работающей программе придется удалять их вручную каждый раз, когда программа запускается. Но это все не столь важно, хотя на рисунке ниже можно увидеть, где удаляется лишняя надпись. Одновременно можно заметить, что вид рабочего окна, сравнительно со стабильной версией, изменился.

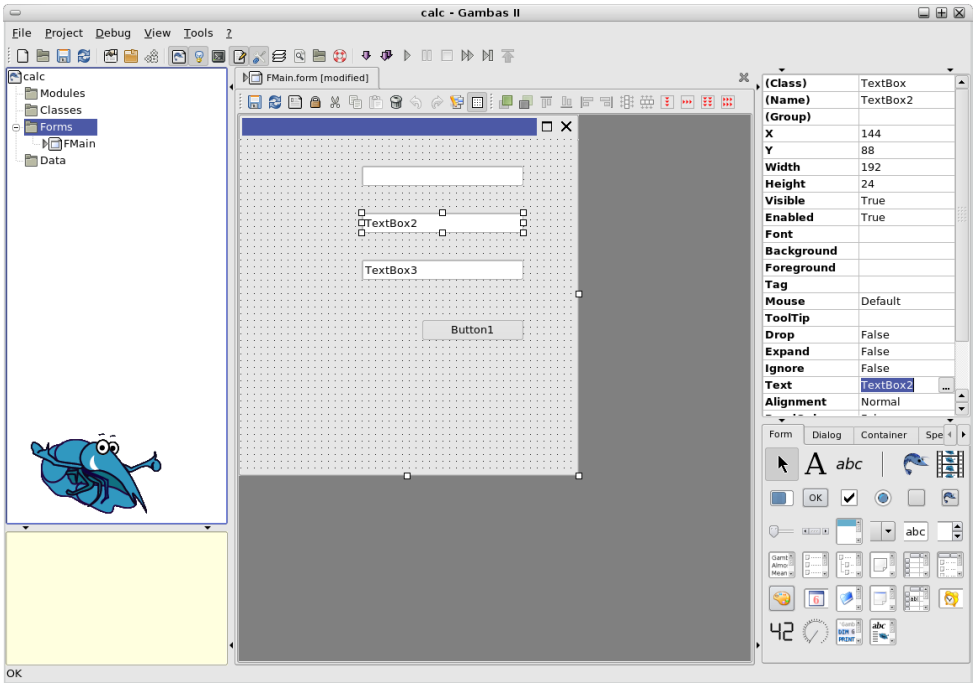


Рис. 1.27. Подготовка формы программы калькулятора

Проделав подготовительные операции по расчистке формы, можно, дважды щелкнув по клавише **Button1**, открыть редактор кода с готовым заголовком процедуры. Альтернатива двойного щелчка, и для меня единственная находка в поисках того, как добиться других возможностей ввести события, это щелкнуть по клавише **Button1** правой клавишей мышки. Выпадающее меню содержит много полезных пунктов, и, в частности, первый из них для меня полезный пункт **Code**, всегда ломал голову, как перейти в редактор кода без добавления ненужных процедур, и пункт **Event**, который и позволяет выбрать множество прописанных в нем событий помимо щелчка клавишей мышки по кнопке.

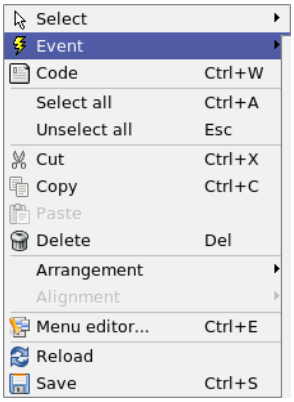


Рис. 1.28. Выпадающее меню после щелчка по объекту Button

И пункты выбора и остальное следовало бы, и хотелось бы, попробовать, но не сейчас. Я думаю, что еще очень много интересного и полезного встретится по пути. А пока, вот как выглядит набор возможных событий, связанных с любой кнопкой, помещенной на форму:

Рис. 1.29. Перечень возможных событий, связанных с объектом Button

Если выбрать событие Click, то получится такой же результат, как и после двойного щелчка по кнопке Button1 на форме. Но и это сейчас не важно, а важна строка, которую я впишу в редакторе кода:

```
TextBox3.Text = Val(TextBox1.Text) + Val(TextBox2.Text)
```

Как можно убедиться ниже, это единственный текст кода, который я ввожу в программу, да и то, используя подсказки Gambas, что, впрочем, удобно, как и в других интегрированных средах разработки.

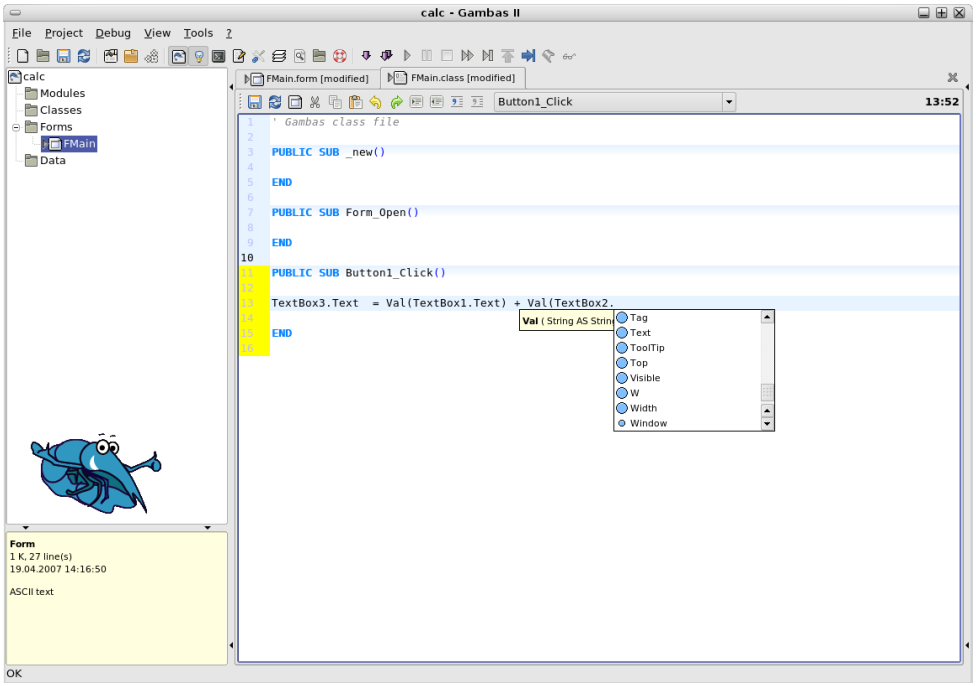


Рис. 1.30. Ввод кода программы в редакторе кода Gambas

Этой строки достаточно для того, чтобы после запуска программы иметь возможность ввести любое число в верхнее окошко, а второе ниже (если это нецелые десятичные числа, то следует использовать запятую при вводе числа), а затем, нажав на кнопку Button, получить результат операции, в качестве которой я выбрал сложение, но мог выбрать и другую операцию.

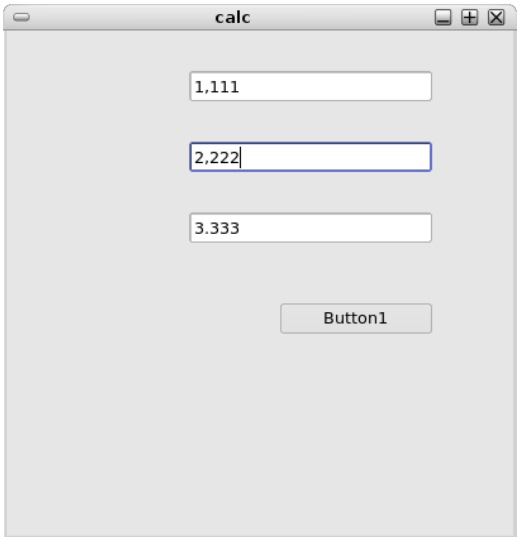


Рис. 1.31. Работа «калькулятора» в Gambas

Вот тот удивительнейший для меня результат, на который открыл мне глаза С.Н. Лукин. Всего одна строка кода, немного перетаскивания картинок «с места на место», и работающая программа. И никогда бы я не догадался, как написать эту строку кода. Я уверен, что стал бы создавать переменные, в которые передавал бы введенные числа, складывал бы их, а потом пытался как-то отобразить их. И уверен, что подобная возможность, не что иное, как свойство объектно-ориентированного программирования. Мне не надо заботится о множестве процедур, мне достаточно добавить к программе нужные объекты – окна ввода чисел и вывода результатов, и записать что я хочу от программы: «Получить результат сложения двух чисел, которые введены в первые два окна». Здесь **Val ()** – это преобразование строки в число, а ввожу я в калькулятор текстовую строку. При этом мне даже не надо заботиться о типе получаемого числа. Если я ввожу целое число, то получаю тип Integer, ввожу дробное число, получаю тип Float, с плавающей точкой. По свойствам введенной мной строки Gambas определяет тип данных.

Все, кому приходилось держать в руках паяльник, верно сталкивались с необходимостью получить резистор из двух, включенных параллельно. Обычно выбирается опорный резистор подходящего номинала, к которому подбирается добавочный. Я «навскидку» пытаюсь угадать его величину, затем измеряю результат мультиметром, делая несколько шагов в направлении наилучшего приближения. Если повезет, то цель достигается в несколько шагов. Если нет, приходится выбирать другой опорный резистор и повторять весь процесс. Можно попытаться использовать калькулятор для этой цели, но большое количество операций с сопутствующими

«опечатками» просто удручает. Так не проще ли потратить несколько минут на создание программы, аналогичной выше описанной, чтобы впоследствии, экономя время и силы, проделать все за компьютером!

А можно с этого места поподробнее?

Можно. Если подробнее о среде программирования Gambas. Или, если имеется ввиду подробнее о том, зачем вообще программировать, когда перечень готовых программ, как платных, так и бесплатных, с краткими аннотациями, займет множество пухлых томов. Или, наконец, подробнее о выборе языка и среды программирования.

Всем этим вопросам я намерен посвятить оставшуюся часть книги.

Чуть подробнее о Gambas. Язык и среда программирования написаны для собственного удовольствия молодым французом Бенуа Минисини (Benoit Minisini) из предместий Парижа. Ему нравится создавать компиляторы, ему нравится играть на флейте. Он получает удовольствие, обучаясь актерскому мастерству, и работает программистом. К работе над проектом в течение нескольких лет его подвигла не только любовь к программированию, но и те идеи, а особенно то воплощение языка Бейсик, что были реализованы в Microsoft Visual Basic. Настолько это все ему НЕ ПОНРАВИЛОСЬ, что он решил создать свой собственный вариант современного языка Basic. Что и осуществил. Логичным завершением компилятора стало создание графической среды программирования, написанной уже на языке Gamabas.

Рост популярности операционной системы Linux, как рабочей среды персональных компьютеров, привлекает все большее внимание профессионалов. Среди них не только программисты, но много педагогов. И если первые используют любой язык, подходящий к их задаче, то вторым чаще приходится обращаться к языкам, задуманным именно для обучения.

Как любая программа, предназначенная к работе в составе операционной системы, Gambas поддерживает привычный для пользователя интерфейс – ту графическую пользовательскую среду, а сегодняшние операционные системы не обходятся без нее, которая привычна для пользователя, которая помогает ему быстро найти нужный раздел в меню, быстро начать работу над собственной задачей.

Рис. 1.32. Основное меню и основная инструментальная панель Gambas

Пункт меню File, который обычно служит для работы с файлами, в данном случае служит для работы с файлами проекта: New project... (создание нового проекта), Open project... (открыть проект), Open recent (открыть недавний проект), Open example (открыть пример), Save project (сохранить проект), Quit (и выйти).

Для управления проектом, помимо множества возможностей явно не выведенных в области основного меню, служит раздел Project:

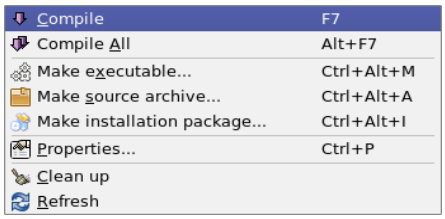


Рис. 1.33. Подменю раздела Project

Это, соответственно, компиляция файла, компиляция проекта, создание исполняемого файла, создание архива исходного текста, создание установочного пакета, свойства проекта, очистка и обновление проекта.

Два пункта меню непривычны для меня – создание архива исходных текстов и установочного пакета. Но не следует забывать, что в Linux программы, как и сама операционная система, всегда существуют в виде открытых исходных кодов. А установить программу можно двумя способами, либо используя исходный код и средства операционной системы для создания и установки программы, либо используя готовый установочный пакет. Правда в разных дистрибутивах Linux такой установочный пакет может быть разным.

Следующий пункт меню относится к отладчику вновь создаваемой программы на Gambas, и имеет более или менее стандартный набор средств отладки, таких как запуск, пауза, остановка работающей программы, шаг программы и передвижение вперед, и очистку всех точек остановки.

Далее следуют разделы вида – View (открывающий и закрывающие все окна диалога рабочей области), инструментов (Tools) и подсказок (?). В последнем разделе достаточно подробный Help browser, в последней версии программы основанный на Wiki. Мне, конечно, не хватило терпения «прощелкать» все пункты подсказки, но предыдущая версия имела Help, состоящий более чем из 3000 файлов, среди которых ряд статей и множество примеров.

Основная инструментальная панель, как это принято, повторяет наиболее часто используемые разделы основного меню, имеет всплывающие подсказки назначения клавиш и дает возможность таким «торопыгам», как я, быстрее нестись к завершающей стадии своей работы – мучительным раздумьям: «Почему ничего не работает?». Очень быстро нестись.

В новой редакции программы графическая работа над формой проекта и кодом программы может вестись быстрым переключением с помощью ярлычков (на рисунке ниже, Fmain.form и Fmain.class), щелкая мышкой по которым попадаешь в графический редактор или редактор кода. Каждый из редакторов имеет свое основное инструментальное меню. На рисунке ниже – инструментальное меню графического редактора.

Рис. 1.34. Инструментальное меню графического редактора и ярлычки перехода

Кроме обычных средств, отображающих основное меню, таких как сохранение и обновление, откат и возврат, удаление и вставки из буфера, панель имеет и некоторые специфические компоненты, такие как вход в редактор меню. Ведь почти все формы, которые при работе программы станут окнами программы, должны иметь основное меню. Именно редактор меню позволяет создавать иерархию основного меню с его разделами и подменю.

Остальные клавиши позволяют управлять размещением объектов на форме, выравниванием их размеров. Кроме переноса объектов на форме с помощью мышки – выделение объекта и его захват, удержание левой клавиши мышки и перемещение с помощью мышки – так вот, кроме этого объекты можно перемещать с помощью клавиш курсорной клавиатуры, что очень помогает, когда нужны точные движения. У меня, например, мышка обычно либо убегает в одну строну, либо в другую, я долго прицеливаюсь, а когда, наконец, попадаю в нужное место, то от движения при отпускании клавиши мышки, та сдвигается с места и все нужно начинать сначала. Если бы ни клавиатура, то и не знаю, что делать.

Я думаю, что по мере создания сложного проекта со множеством форм и собственных классов, это дополнительное меню будет расширяться, позволяя быстро перемещаться по проекту. Но для этой же цели можно использовать и окно «дерева» проекта.

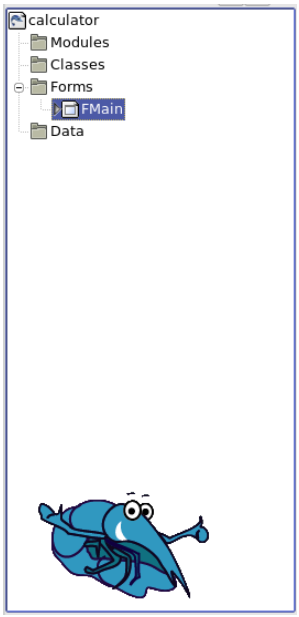


Рис. 1.35. Окно дерева проекта

У меня оно пустое, что хорошо, впрочем, отражает мои познания в части программирования. Но это так, к слову. И пока не забыл, хочу привести еще одно правило, почерпнутое мною из книги С. Н. Лукина, которому постараюсь следовать сам: не следует пытаться сразу создать весь проект, лучше делать это частями, открывая новые проекты (из старых) по мере продвижения, отлаживать и сохранять каждую часть, продвигаясь пусть скромными, но последовательными шажками к своей цели. Отлаживать процедуры, если это возможно, «в гордом одиночестве». Я назвал бы это «штучным производством». Не буду пересказывать дальше, лучше почитайте сами. Книжка не дорогая, и, думаю, будет весьма полезна и в начале изучения языка, и в дальнейшем.

Отдельную область занимает инструментарий проекта – Toolbox. Построенный аналогично рабочей области, он тоже имеет различные ярлычки, открывающие страницы, на которых размещаются средства создания проекта. Моему разумению пока доступны такие графические элементы как клавиши, текстовые окошки, переключатели и иже с ними. Но если вас не пугают такие ужасы, как контейнеры, можете пощелкать по закладкам и рассмотреть все страницы.

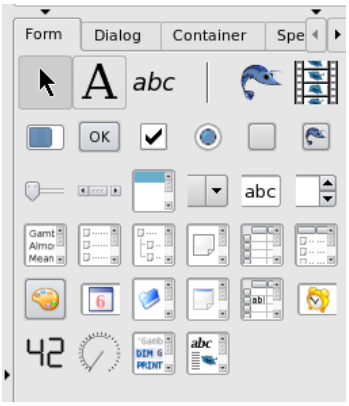


Рис. 1.36. Вид инструментария проекта

А сейчас я зажмурюсь, у меня высотобоязнь, и отрою свойства проекта. Это сделать несложно – вошел в раздел основного меню Project, выбрал пункт выпадающего меню Properties, нажал левую клавишу мышки и ты уже в свойствах проекта, где есть закладка Components. Вот такой «высоченный» список:

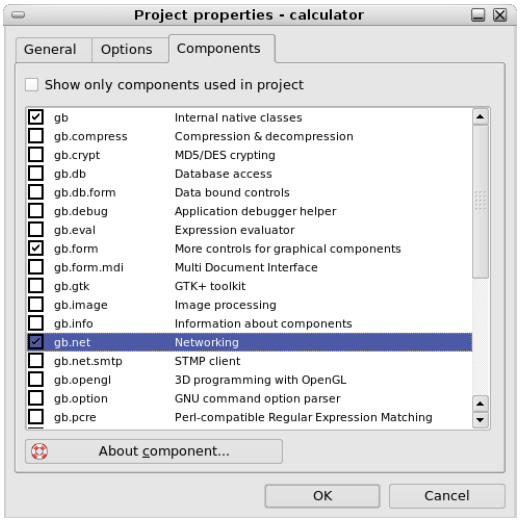


Рис. 1.37. Окно списка доступных компонент Gambas

Если выбрать еще один компонент из списка, то инструментарий пополняется еще одной страницей с закладкой.

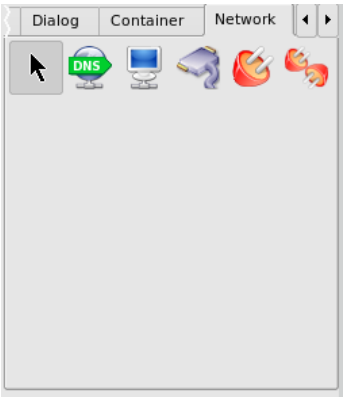


Рис. 1.38. Пополнение инструментария проекта после добавления компонента

Но если вы подобно мне не готовы стать монтажником-высотником, всегда можно выключить ненужные компоненты, оставив только то, что ясно и нужно в данный момент.

Первый блин

Профессионалы, как правило, специализируются на чем-то. Одни предпочитают, если говорить о программировании, работать с базами данных, другие системное программирование. Специализация позволяет ясно очертить круг задач, приобрести необходимый набор инструментов и создать собственный «склад» заготовок. Всегда полезно использовать отлаженную и проверенную заготовку, если она подходит для выполнения предстоящей задачи. В этом смысле нам дилетантам приходится труднее. И задача не слишком ясна, и цель не видна, и из заготовок – пустой ящик с одним погнутым гвоздем (как мои зарисовки в программе Umbrello).

Хотя, пожалуй, не совсем все так плохо. Есть заготовка от проекта «Калькулятор». Я нашел компонент в Gb.net, который, надеюсь, даст возможность использовать COM-порт компьютера. Попробуем эти находки положить в основу проекта «Счетчик» (Counter). И еще раз, а то я стал забывать о чем идет речь, перечислю, что я хочу сделать:

- Создать графический интерфейс с помощью Gambas для проверки двоичного счетчика с предустановкой.
- С помощью интерфейса будет вводиться число, которое при нажатии клавиши, положим, «Запись» будет переписываться в счетчик.
- Для выполнения этой и последующих операций компьютер по COM-порту будет соединяться с микроконтроллером, отдавая последнему

соответствующую команду, которую контроллер должен обработать.

- После переписи числа в счетчик микроконтроллер будет прочитывать число на выходе счетчика и отправлять результат на компьютер, а графический интерфейс отображать это число.
- Интерфейс будет иметь клавишу запуска счета, после нажатия на которую счетчик должен начинать отсчет, микроконтроллер должен прочитывать результат и предавать компьютеру.
- Вероятно, нужно будет иметь еще клавишу интерфейса для того, чтобы открыть COM-порт и закрыть.

Пока этого хватит. Для выполнения задуманного я, как описано выше, добавлю еще один компонент, сетевой, с помощью диалогового окна свойств проекта, и, как и в калькуляторе, создам два окна TextBox, две клавиши Button и еще одну, о свойствах которой пока ничего не знаю. Называется она ToggleButton. Если она поможет мне переключать COM-порт, хорошо, нет, заменим. В результате форма выглядит следующим образом:

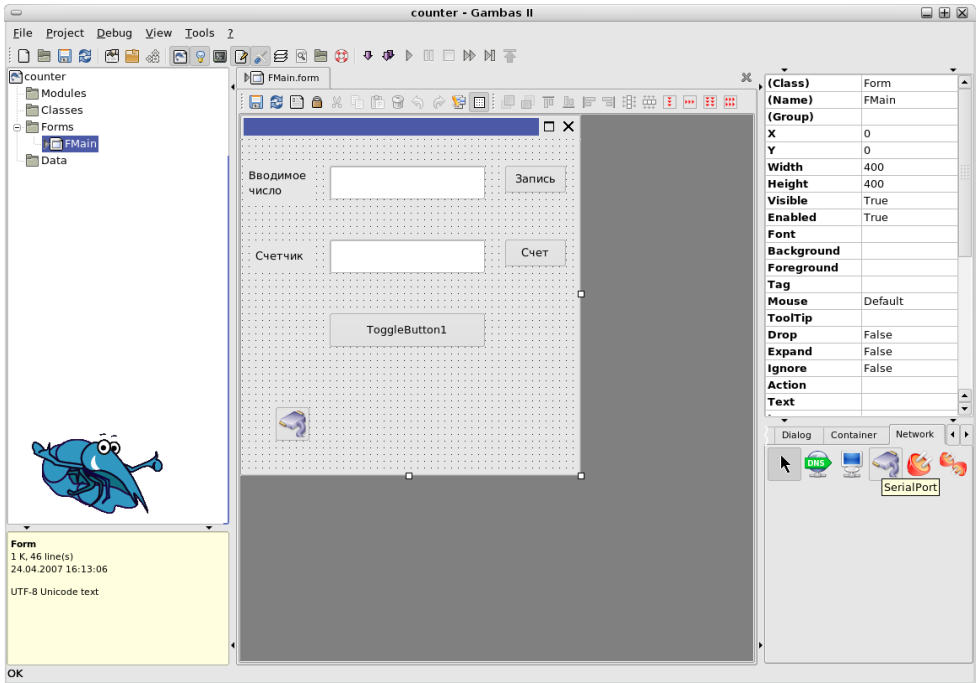


Рис. 1.39. Вид формы для проекта «Counter»

На форме, слева внизу, виден добавленный мной компонент для работы с COM-портом, на инструментальной панели надписью SerialPort отмечено, откуда я его извлек. Теперь самое время обратиться к разделу помощи программы, чтобы прочитать, как пользоваться компонентом порта последовательной передачи данных, и что такое клавиша ToggleButton.

Выяснение последнего вопроса о назначении клавиши при обращении к подсказке не прояснило для меня ситуацию. Пришлось проверить, как это работает, с помощью самой программы, добавив в процедуру отработки щелчка по клавише строку вида:

```
PUBLIC SUB ToggleButton1_Click()
        TextBox1.Text = ToggleButton1.Value
END
```

При нажатой клавише она остается «включена», при повторном нажатии «выключается», а ее значение в виде текста при нажатии получается как «T», а при выключении ничего не видно. Пока решим, что при нажатии клавиша получает значение True. Впрочем, это можно проверить, заменив код на:

```
IF ToggleButton1.Value THEN
        TextBox1.BackColor = Color.Blue
ELSE
        TextBox1.BackColor = Color.Red
ENDIF
```

Теперь при нажатии на клавишу верхнее окошко становится синим, а после повторного нажатия становится красным.

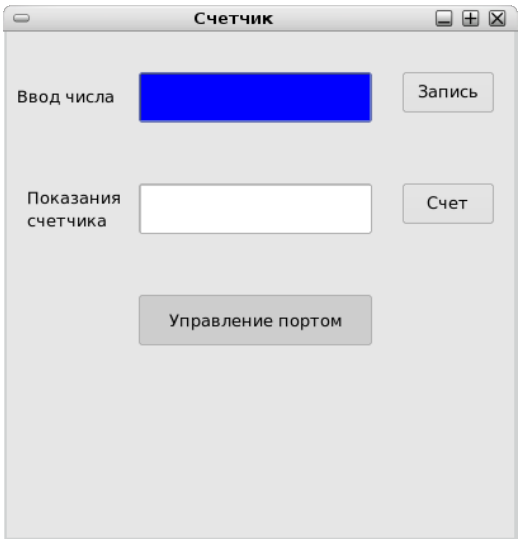


Рис. 1.40. Проверка работы клавиши ToggleButton

Посмотрим, как включать и выключать COM-порт, как задать, какой именно порт использовать, и как установить его свойства.

Поначалу я хотел задавать все свойства порта перечислением их в программе. Потом я подумал, что можно использовать окно свойств компонента, где эти свойства, скорость работы и количество бит и т.д., можно установить. Что и сделал. Теперь по нажатию кнопки на форме порт будет включаться, а при повторном нажатии выключаться.

Увы! Я перебрал несколько вариантов, как мне казалось разумных строк в программе, но, если отбросить явные ошибки, наилучший результат выглядел сообщением об ошибке при открывании порта. Код программы выглядел с моей точки зрения правильно, но кого волнует моя точка зрения?

```
PUBLIC SUB ToggleButton1_Click()
        IF ToggleButton1.Value THEN
                SerialPort1.Open
                TextBox1.Text = SerialPort1.Status
        ELSE
                SerialPort1.Close
                TextBox1.Text = SerialPort1.Status
        ENDIF
        IF SerialPort1.Status THEN
                TextBox1.BackColor = Color.Green
        ELSE
                TextBox1.BackColor = Color.Red
        ENDIF
END
```

В первом окне для проверки я хотел бы вывести состояние порта после выполнения операции открывания порта. Затем, в подтверждение этого, если порт открыт, изменить цвет этого окошка на зеленый. При закрывании порта цвет должен смениться на красный. Этого хотелось бы, но...

Не получилось.

Просмотрев все советы, которые я нашел, посмотрев то, что я делал в Visual Basic, когда работал над книгой об «умном доме», я не нашел ничего, что указывало бы на ошибку. Осталось перейти к прямой проверке. Для начала я решаю сменить операционную систему, точнее дистрибутив, благо на компьютере у меня есть Ubuntu с установленной программой Gambas. Перезагрузив компьютер, я открываю проект, запускаю программу, нажимаю на клавишу открывания порта и получаю результат:

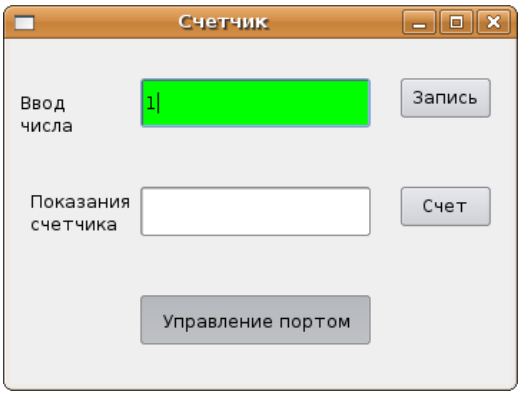


Рис. 1.41. Первая удачная попытка открыть COM-порт

В этом дистрибутиве Linux все работает соответственно моим ожиданиям. Сообщения об ошибке при открывании порта не появляется. Вперед я продвинулся, но понимания, за счет чего это произошло, пока нет.

Я вновь возвращаюсь к своему основному дистрибутиву, и вновь получаю сообщение об ошибке, но теперь уже с полной уверенностью, что программа ДОЛЖНА работать.

Я не знаток операционных систем, чтобы утверждать, что это особенность именно Linux – строгое распределение прав доступа ко всем ресурсам системы. Но, если щелкнуть правой клавишей мышки по любой папке или файлу и выбрать из выпадающего меню пункт свойств, то можно на странице прав доступа увидеть, кто и на что имеет право.

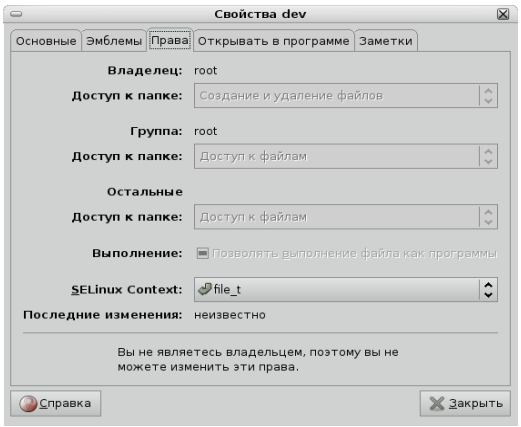


Рис. 1.42. Свойства папки dev на странице прав доступа

Не знаю, как устроен Linux, но для меня вся его структура – это набор папок и файлов. В частности и COM-порт, который имеет название ttyS0, и который расположен в папке dev файловой структуры, для меня просто файл с правами доступа. И, похоже, к этому файлу у меня доступа нет. Результат ли это особенностей дистрибутива, или результат моих экспериментов, не знаю. Но эту ситуацию можно попытаться изменить. Чтобы убедиться в правильности выбранного пути, я завершаю свой сеанс работы и захожу в симтему «под root'ом», то есть, вхожу как администратор компьютера. Теперь, запустив программу в Gambas, я не получаю сообщения об ошибке, программа ведет себя ровно так же, как в Ubuntu – работает, выдает все, что я «заказывал».

В основном меню графической среды Gnome, которую я использую, в пункте Система есть раздел Администрирование, в котором есть пункт Пользователи и группы. В открывающемся диалоговом окне можно добавлять и удалять пользователей, можно изменить состав групп пользователей, что меняет их права доступа к различным ресурсам компьютера.

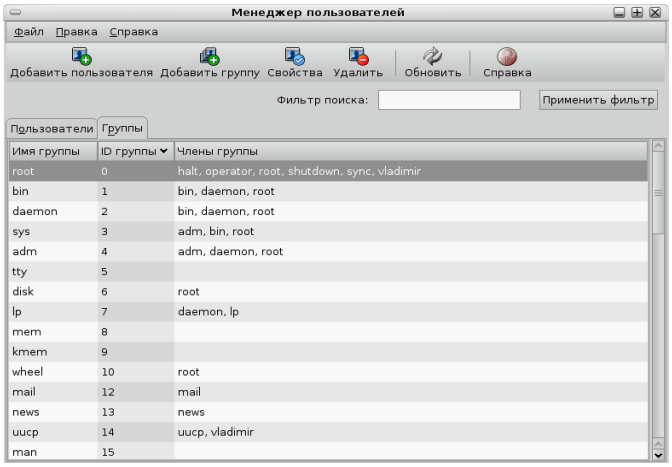


Рис. 1.43. Диалоговое окно управления пользователями и группами

Выбрав пользователя root, администратора компьютера, и нажав на клавишу **Свойства** основного меню диалога, можно войти в диалог свойств пользователя, где есть страница, определяющая, кто входит в административную группу.

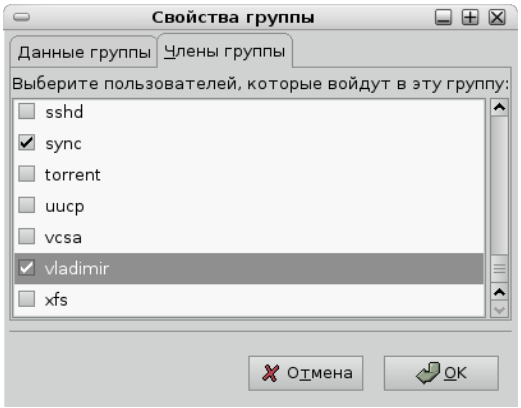


Рис. 1.44. Диалоговое окно определения состава административной группы

В этом окне диалога мне достаточно поставить галочку (щелкнуть левой клавишей мышки по квадратику напротив имени пользователя) рядом с собой, чтобы получить права доступа к тем ресурсам, которые может использовать администратор. Конечно, в рамках того, как обозначены эти права для административной группы, поскольку они могут отличаться от прав главного администратора.

После присвоения себе «повышенных» прав я снимаю вопрос об ошибке при открывании последовательного порта. Порт открывается.

Но работает ли он? Правильно ли он работает? Все это я намерен проверить после того, как завершу часть проекта, относящуюся к микроконтроллеру. Именно с ним, а не сам с собой, я намереваюсь беседовать при помощи созданного интерфейса. Интерфейс предстоит пополнить чем-то для переключения скорости счета, а код программы всем необходимым для диалога интерфейса с микроконтроллером. Хотя мне не терпится проверить работу COM-порта, воспользовавшись прежней наработкой и быстро перепрограммировав контроллер – если бы я занимался разработкой, я так и сделал бы – для рассказа мне эта идея кажется слишком большим жульничеством, можно иногда, но надо знать меру. Пожалуй, придется потерпеть до конца следующей главы.

Глава 2. Бряцая железом

Можно сколь угодно долго спорить о том, что нужнее начинающему любителю электроники, научиться паять или освоить компьютер; лучше ли начинать с конструктора, или с подходящей программы; ни спор, плавно переходящий в ссору, ни самые веские аргументы не способны отменить того, что стало реальностью: компьютер вторгся в мир радиолюбительства, как некогда транзисторы, а позже микросхемы, чтобы остаться в нем надолго, если ни навсегда.

Хорошее начало

Итак. Первое приближение интерфейсной части проекта готово. Теперь время поговорить о «железной» части проекта. Я предполагаю использовать микроконтроллер PIC16F628A. Два основных аргумента в пользу этого выбора – наличие самой микросхемы в продаже, и наличие всего необходимого для работы с микросхемой: среда разработки с поддержкой языка Си и программатор, который успешно работает в этой среде. Лично для меня важно и то, что от прошлых экспериментов осталась и микросхема, и макетная плата, и, если и придется что-то покупать, то не все. Далее я расскажу о среде программирования и приведу схему программатора, но начать хочу со схемы эксперимента, как если бы я задался целью реализовать ее без применения компьютера. Для этого, правда, я использую программу EDA, что много проще, чем рисовать схему на бумаге, а затем сканировать ее.

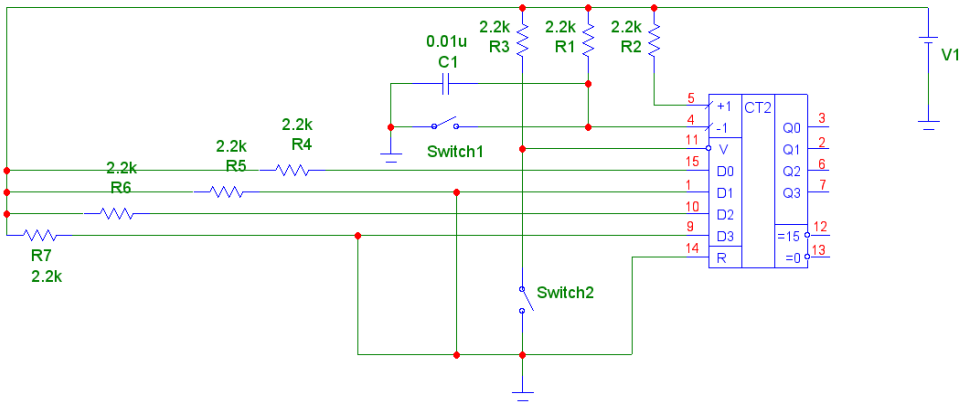


Рис. 2.1. Схема проведения прямой проверки работы счетчика

Теперь о программе Piklab. К сожалению, когда я в книге об «умном доме» описывал работу с микроконтроллером, я не нашел этой программы, хотя, с другой стороны, программа MPLAB великолепна, и ничего плохого о ней сказать не могу. Для работы с программой Piklab, особенно при использовании компилятора языка Си, потребуются дополнительные пакеты. Если программа устанавливается стандартным способом, то пакеты gputils и gpsim, скорее всего установятся по зависимостям. Так установилась программа в моем дистрибутиве Fedora Core 6. Второй дистрибутив Ubuntu не предложил ничего, но есть пакет для Debian, и в этом случае приходится дополнительно устанавливать утилиты, хотя этот процесс проходит быстро и безболезненно. В частности требуются пакеты для графической оболочки KDE. Пакет gputils необходим для программирования на ассемблере, gpsim для отладки программы. Для работы с языком Си можно воспользоваться бесплатным компилятором picc-lite, который можно найти на сайте www.htsoft.com. Я использую версию picc-lite_9.50PL2. Компилятор знаком мне по работе с программой MPLAB под Windows.

Схему программатора я выбрал под названием JDM picprogrammer. С программой работают и другие программаторы, хотя я проверял помимо самодельного только один, купленный при работе над предыдущей книгой. Чтобы не отсылать вас на поиски схемы в Интернет, или к своим прежним книгам, я приведу схему программатора в оригинальном виде. При сборке этого программатора я обнаружил, что забыл купить стабилитрон на 8.2 В, тогда как стабилитронов на 5.1 В купил в избытке. По этой причине я к имевшемуся стабилитрону последовательно подключил два светодиода (из расчета, что при прямом включении они будут вести себя, как стабилитрон с напряжением стабилизации около 3 В). Такой вариант сработал, и оказалось, что иметь светодиоды в этой цепи даже полезно – при обращении к программатору они светятся, и видно, что программатор работает.

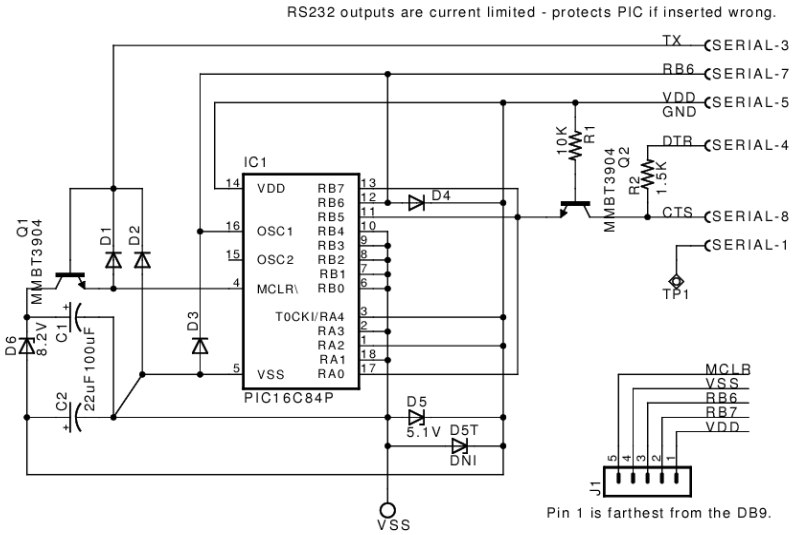


Рис. 2.2. Схема программатора для работы с программой piklab

На схеме переделанный стабилитрон – это D6.

Сейчас я извлеку программатор из своей большой «хламежки», и подключу его к первому COM-порту...

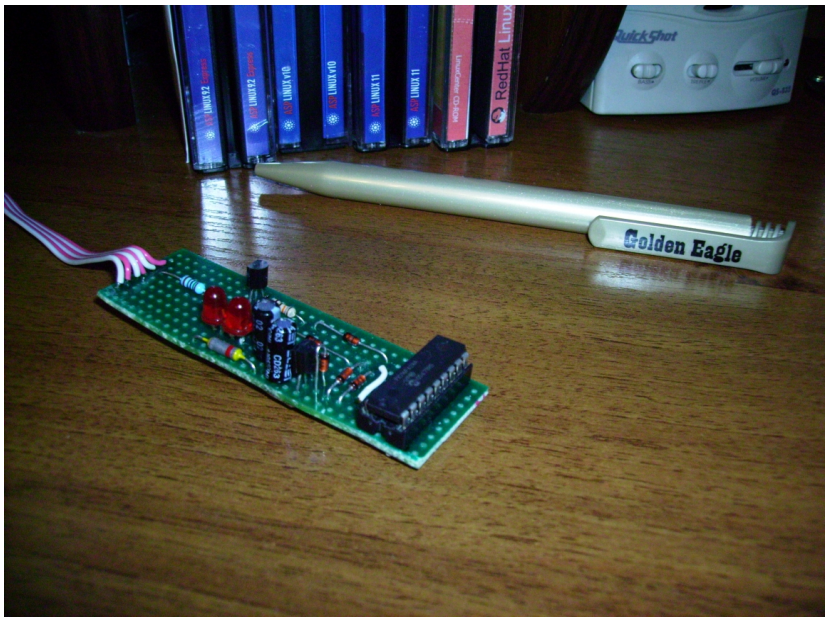


Рис. 2.3 Внешний вид программатора

Это второй самодельный программатор, первый работал с программой PonyProg, в обоих случаях были рекомендации сделать кабель, соединяющий программатор с компьютером максимально коротким, но у меня он получался не короче метра, иначе слишком неудобно работать. Особых претензий к программаторам у меня не появилось, но, возможно, длинный кабель и вреден для «здоровья» программатора.

Итак, первый запуск программы Piklab.

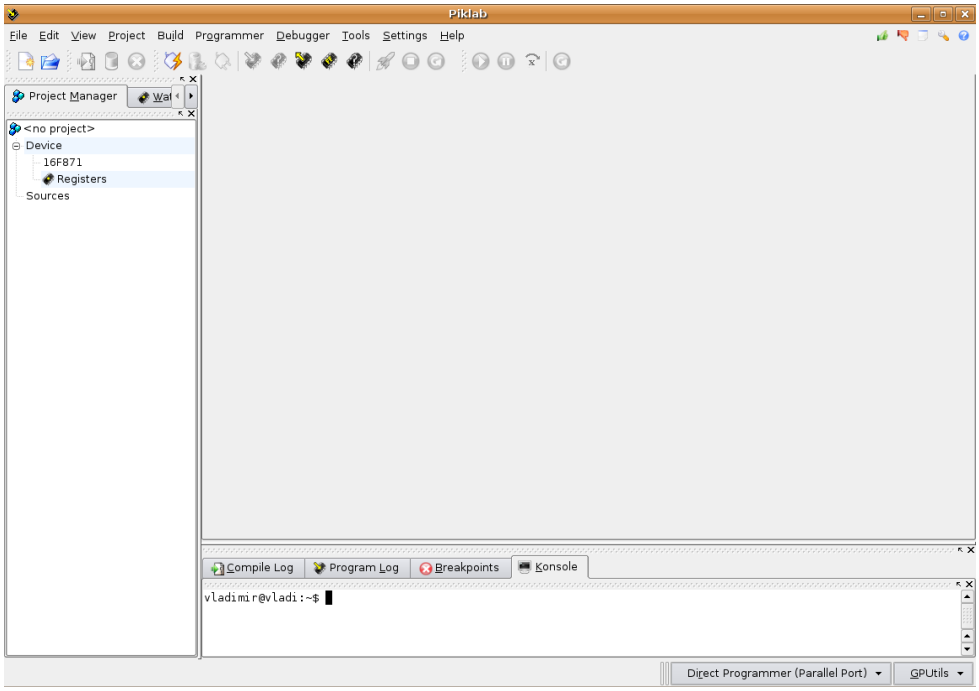


Рис. 2.4. Первый запуск программы Piklab

Внешне среда программирования микроконтроллера, как вы можете судить, очень похожа и на среду программирования Gambas, и на среду программирования KDevelop, если вы знаете такую. Что не удивительно, поскольку эти программы созданы для работы с одной операционной системой. Слева окно менеджера проекта, центральное место занимает встроенный редактор программирования, а в нижней части расположены окна сообщений, включая окно для работы с точками останова и консолью. На рисунке окно консоли открыто, чтобы в этом окне можно было ввести любую команду.

Начну с основного меню программы. Кроме обычных для любой программы таких разделов меню, как раздел работы с файлами, кроме обычных разделов, характерных для любой среды программирования, связанных с отладкой программы, есть и специфический раздел, относящийся к работе с программатором.

Рис. 2.5. Основное меню программы Piklab

Хотя первым в списке разделов меню стоит File, относящийся к работе с файлами, я зайду в раздел Settings, раздел установок программы.

Рис. 2.6. Подменю раздела Settings

И более всего меня пока интересует установка программатора. По причине чего я захожу в раздел Configure Piklab..., чтобы открыть установки, относящиеся именно к программатору.

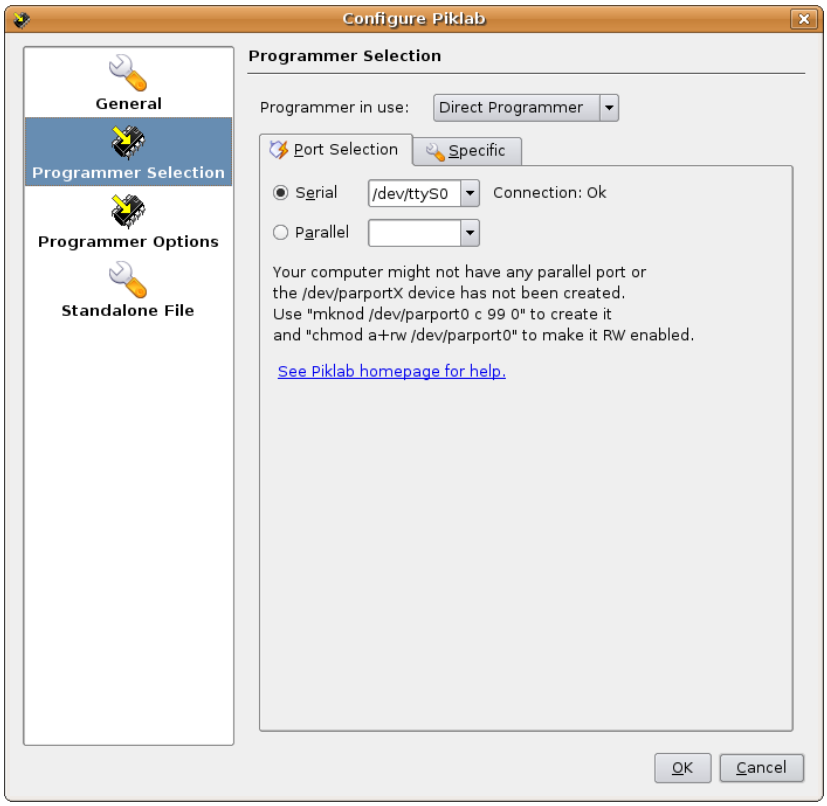


Рис. 2.7. Диалоговое окно установок программатора

После выбора последовательного порта COM1 (ttyS0 для Linux) и типа программатора Direct Programmer я сразу вижу, что рядом с надписью Connection: появляется OK, то есть порт обнаружен. А следом за выбором типа контроллера в окне менеджера проекта (Progect Manager) я могу прочитать содержимое микроконтроллера, который остался в панельки со времен предыдущих экспериментов. Для этого можно зайти в раздел основного меню Programmer, на всякий случай выбрать пункт Connect, проверить, чтобы на панели состояния отображался правильный выбор Direct Programmer (Serial Port), и можно прочитать содержимое микроконтроллера.

Успокоенный этими удачными пробами, я собираюсь плавно перейти к рассказу об основном меню программы и соответствующих клавиш основного инструментального меню. Но мне приходит в голову проверить процесс отладки программы. Без достаточно удобной отладки работать с программой будет слишком трудно. Я знаю, что для отладки используется gpsim, но как это осуществить в

программе piklab? Файл помощи, который я пытаюсь открыть, показывает, что документация отсутствует. В Интернете на странице проекта есть краткое введение в работу с программой (я ищу работу в графическом приложении), описание, касающееся этой части выглядит весьма кратким:

*After your successful compilation (as you can see in the Compile Log) you only have to press the button **connect** to "virtually connect" the programmer **Gpsim**, to be able to start the simulation of your program. Just press the blue button with the "play" symbol, to start. After this, the **Program Log** shows some information output from gpsim.*

Что означает, примерно, следующее:

*После успешной компиляции программы (как можно видеть в Compile Log) вы должны только нажать клавишу **connect** для «виртуального подключения» к программатору **Gpsim**, чтобы иметь возможность начать симуляцию вашей программы. Достаточно нажать синюю клавишу с символом «проиграть», чтобы начать симуляцию. После этого **Program Log** покажет некоторую выходную информацию от gpsim.*

Я повторяю эти действия вначале после трансляции программы на Си (я всегда для проверки использую простую и удобную программу, которую некогда нашел в Интернете, и уже, к сожалению, не помню автора). В окне Program Log появляется сообщение:

Connected.
Setting up debugging session.
Ready to start debugging.
Running...

И это все, что я вижу. Этот же результат и при трансляции ассемблерной программы. Этот результат дают оба дистрибутива, установленные на моем компьютере. Я прерываю работу на несколько дней, понимая, что если каким-то образом не решить эту проблему, то следует остановиться на этом месте и честно сказать себе: «Строили мы строили, и, наконец, построили!». Книга неожиданным образом завершена, добавить нечего.

Можно попытаться связаться с авторами проекта, но они могут быть либо заняты, либо могут охладеть к работе над проектом. В любом случае ответа от них можно ждать долго и не дождаться. Ассемблерный вариант выглядит несколько лучше в этом плане – при трансляции он дает файл с расширением .cod, который можно загрузить в gpsim, который, в свою очередь, запускается у меня из терминала. Но едва ли вам захочется сразу начать работу с микроконтроллером, используя ассемблер. Это возможно только в том случае, если вас интересует именно работа на

ассемблере. Для остальных удобнее использовать более легкое создание программ на языке высокого уровня. Беда же в том, что компилятор не дает такого файла. Стоп работа!

Возвращение

За это время я еще раз перечитал документацию, перебрал варианты, твердо решил, что бросаю эту книгу, потом решил еще раз попробовать как-то обойти проблему. Один из возможных вариантов заключается в том, чтобы использовать полученный после трансляции hex-файл для дизассемблирования, а последний файл с программой на ассемблере повторно транслировать, чтобы получить .cod файл, необходимый для загрузки в симулятор работы микроконтроллера gpsim. Сознаюсь честно, что я даже пытался установить программу MPLAB под Wine (эмулятор работы Windows). Но работала только очень ранняя версия для Widows 95, с которой проблем могло оказаться не меньше, если не больше. Последняя версия MPLAB работать не захотела. Чтобы честно сказать, что я перепробовал все, я даже поставил программу VirtualBox – великолепно работающую виртуальную машину для Linux, в которой можно установить из iso-образа на своем компьютере другой дистрибутив Linux или Windows. В ней я установил последнюю версию MPLAB, где попробовал установить последнюю версию транслятора фирмы HI-TECH, обнаружив, что этот транслятор не входит в перечень, устанавливаемых по умолчанию, хотя несколько лет назад это было не так. Я вспомнил, что несколько лет назад, пытаясь работать с микроконтроллером в Linux я, похоже, оказался в сходной ситуации, пробовал установить другой компилятор, но безуспешно, и стал работать в MPLAB. Но с тех пор могло измениться многое.

По умолчанию на моем компьютере загружается дистрибутив Fedora Core 6. Там я и начал поиск другого компилятора для работы с программой Piklab. На домашней странице проекта в разделе документации я быстро обнаружил краткое описание настроек sdcc – компилятора, используемого с Piklab. Возможность установить этот компилятор в Fedora есть, я его скачиваю, устанавливаю, но не могу настроить. Его не обнаруживает программа Piklab.

Порою я бываю упрям без меры. Видимо, это упрямство заставляет меня перезагрузить компьютер, чтобы войти в дистрибутив Ubuntu, построенный на основе Debian. В этом дистрибутиве я повторяю установку компилятора sdcc, и он великолепно обнаруживается программой Piklab. Мало того, он работает, есть возможность работать с микроконтроллером PIC16F628A, то есть с полной памятью контроллера. И самое главное, он при трансляции дает cod-файл, который можно загрузить в симулятор gpsim.

Порою я бываю упрям без меры. Видимо, это упрямство заставляет меня перезагрузить компьютер, возвращаясь к основной моей операционной системе Linux в дистрибутиве Fedora Core 6 (если этим летом не сменю его либо на привычный мне ASPLinux, либо на Fedora 7). Я загружаю исходный текст компилятора, чтобы использовать штатный режим (в моем понимании) создания программы из исходных кодов. Это не слишком сложно, хотя может занимать достаточно много времени, если программа большая и сложная. Процедура для меня сводится к тому, чтобы запустить терминал (это больше, но аналогично командной строке в Windows). В

терминале командой **cd /home/vladimir/Desktop/sdcc/** перейти в папку, где находятся исходные тексты. Эта папка лежит у меня на рабочем столе. Дальше обычная для Linux команда **./configure** и следом **make**. Следом, не значит, сразу. В этом месте, после команды **make** трансляция исходного текста прекращается. Насколько я понимаю нужен дополнительно пакет **yacc**. Такого пакета программа установки не обнаруживает. Но есть пакет **byacc**. Устанавливаю его, вновь запускаю трансляцию командой **make**, трансляция продолжается и успешно завершается. Теперь, поскольку у меня не работает команда **sudo**, опять приходится искать обходные пути. Обойти проблему, впрочем, не сложно. Вместо команды **sudo** использую команду **su**, и после ввода пароля даю команду **make install**, как и положено. Как и положено, «оттарахтев» минут пятнадцать, трансляция успешно завершается. И теперь программа Piklab обнаруживает компилятор sdcc. И он работает так же удачно, как и в Ubuntu.

Конечно, меня огорчает, что симулятор не запускается, как я ожидал, из программы работы с текстом, что-то я не допонимаю в этом, видимо, но для реальной работы такой вариант даже удобнее – работать на одном рабочем столе все равно не получится, а так можно разложиться со всеми удобствами на двух рабочих столах. Есть еще одна небольшая деталь, касающаяся дистрибутивов. В Fedora не работает механизм добавления модулей, который работает в Ubuntu, но это решается добавлением пакета gpsim-devel с помощью программы установки и удаления приложений.

Моя ошибка, то что я не подготовился в должной мере к работе, несколько выбила меня из колеи, но теперь есть все необходимое для продолжения работы: программатор, компилятор, симулятор. Ничто не мешает продолжить повествование о назначении разделов меню программы Piklab.

Первый пункт основного меню привычен. Он относится к работе с файлами проекта. Но в данном случае в силу того, что проект имеет две составляющие, программную и физическую в виде загрузки откомпилированного файла в программатор, если открыть раздел, то можно обнаружить некоторое отличие предлагаемых действий от обычных для меня – пользователя текстового и графических редакторов, редкого гостя сред программирования и создания web-сайтов.

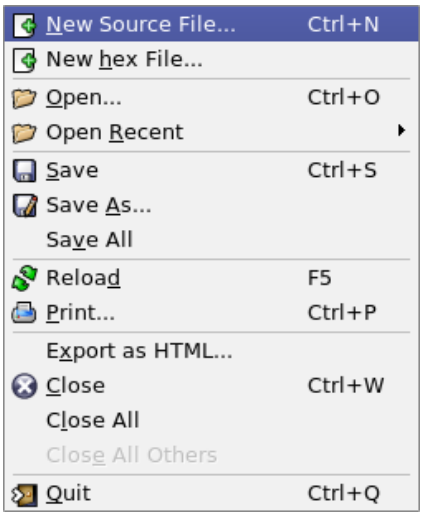


Рис. 2.8. Подменю раздела File основного меню

Первый пункт подменю New Source File... подразумевает создание нового исходного файла программы. Однако лучше создать новый проект. Следующий пункт New hex File... подразумевает создание нового готового для загрузки в микроконтроллер файла кодов. Это еще рано. Далее следуют два пункта меню, позволяющих открыть файл и открыть недавно использованный файл. Затем три пункта сохранения файлов, практически не отличающихся от аналогичных в любом приложении. Что означает Reload я пока не знаю, а остальные пункты меню – вывод на печать, экспорт в виде HTML, закрывание файлов и выход из программы, – говорят сами за себя.

Следующий раздел касается возможностей работы в редакторе текста программы.

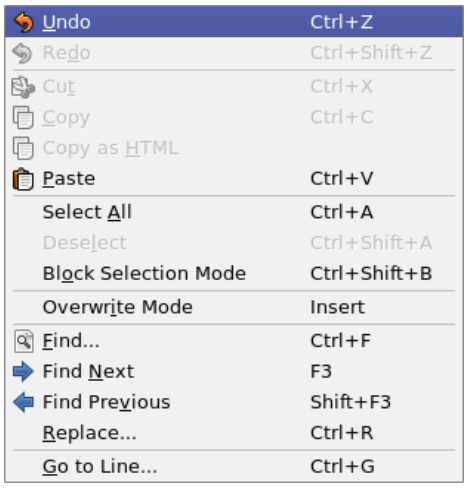


Рис. 2.9. Подменю раздела Edit основного меню программы piklab

Возможности эти соответствуют возможностям любого достаточно мощного текстового редактора, включая отмену последней операции Undo, повтор отмененного действия Redo, вырезание текста Cut, копирования Copy и вставки Paste. Небольшое отличие – копирование в формате HTML (Copy as HTML). Меню чувствительно к операциям с текстом. Например, пока вы не выделите слово или фрагмент текста, операции копирования и вырезания текста не будут активированы. Это удобно, если учесть, что в процессе разработки приходится многократно проделывать одни и те же операции, быстро щелкая по разным разделам меню, и мышка не всегда поспевает за вашими мыслями, отчего, щелкнув по одному из разделов (как вы полагали), вы в действительности щелкаете по другому, который расположен чуть выше или ниже. В подобной ситуации, чем меньше активных разделов, тем лучше.

Конфигурацию редактора можно изменять. При появлении файла с текстом программы подменю раздела Settings пополняется пунктом Configure Editor...

Раздел выделения кроме обычного выделения всего текста Select All, и снятия выделения Deselect, имеет режим выделения блока текста Block Selection Mode. Чем это удобно? Я говорил, что из-за отсутствия файла для загрузки в симулятор gpsim компилятор HI-TECH заставил меня задуматься о повторной трансляции ассемблерного кода. Но для получения ассемблерного кода из файла листинга, например, удобно применить такой режим, как выделение блока текста, иначе придется вручную удалять много полезного в общем случае, но ненужного в конкретной ситуации.

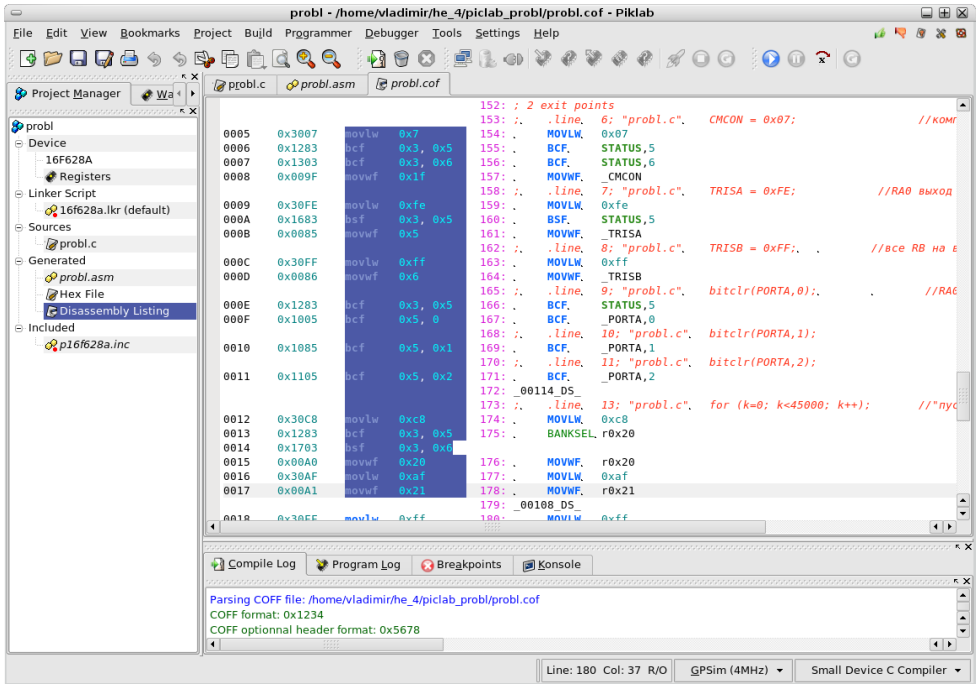


Рис. 2.10. Режим выделения блока текста в редакторе piklab

Выделив в блоке текста только программу, написанную на ассемблере, вы можете вставить ее в новый файл, перенося нужный фрагмент за одно действие. Совсем не лишняя возможность.

Далее в подменю следует изменение режима ввода текста: вставка-замещение. Затем функции поиска Find и замещения текста Replace. Вы, как мне кажется, не раз пользовались этими функциями при работе в текстовом процессоре, так что ничего нового я вам не скажу. И завершает подменю редактирования функция перехода к номеру строки, Go to Line... При отладке программы это может оказаться очень полезным. Забегая вперед хочу сказать, что редактор позволяет включить отображение номеров строк. Сообщения при компиляции, относящиеся к ошибкам, обычно соотносятся с номером строки, и переход по номеру строки не будет лишним.

Подменю следующего раздела основного меню Вид (View) относится к виду открываемых окон редактирования.

Рис. 2.11. Вид подменю View в режиме редактора текста

Так выглядит это подменю, когда вы редактируете текст программы, но его содержание изменится, если у вас открыт hex-файл.

Команды Back (Назад) и Forward (Вперед) относятся к навигации, как в любом обозревателе, например, web-браузере. Когда работаешь с маленькими проектами дополнительные возможности редко используются. Обычно предпочитаешь обходиться небольшим набором хорошо знакомых операций. Но даже маленькие проекты имеют свойство «разбухать» как-то сами по себе. Ты и не хотел этого, но оно происходит. И тогда есть смысл присмотреться к различным возможностям, которые предоставляет тебе программа. Это же замечание я готов отнести к разделу Switch to... подменю View. Каждому открытому файлу соответствует закладка, что позволяет быстро переключиться с файла на файл. Эту же операцию легко сделать в окне менеджера проекта. Но если вы заинтересуетесь работой с программой, работой с микроконтроллерами, вы не остановитесь на создании только одиночных проектов, вам наверняка захочется создавать удобные, проверенные и отлаженные программные модули, которые вы будете включать в новые проекты как объектные модули. Их может набираться много. Переходы с помощью закладок уже не будут выглядеть столь удобными. Попробуйте переключение с помощью этой команды.

Следующий раздел подменю имеет свое собственное подменю, позволяющее отображать или выключать разные окна проекта. Это особенно полезно, когда щелкая по клавишам управления открытыми окнами закрываешь окно по ошибке, и не знаешь, как открыть его вновь. А это и окна менеджера проекта, и окно наблюдения, вывода компиляции и программирования контроллера, точек останова и окно консоли.

Следующий раздел основного меню позволяет вам сделать закладки (Bookmarks) или удалить их. Текст программы быстро становится достаточно длинным, чтобы

«кручение» колесика мышки перестало доставлять вам радость. Проще поставить закладки и переходить к ним при необходимости. Меню тоже чувствительно к вашим действиям. Если вы не делали закладок, то увидите только один активный раздел подменю – Set Bookmark. Но вид подменю изменится, как только вы сделали закладку.

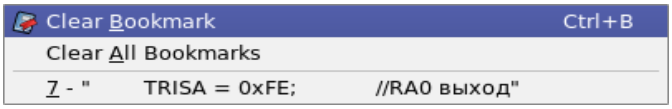


Рис. 2.12. Вид подменю закладок в тексте программы после создания закладки

Следующий раздел меню Project напоминает мне о необходимости немного поворчать.

Самая разумная работа над проектом, это поэтапная его реализация. Начав работу, не стоит пытаться написать код всей программы. Лучше написать часть кода, ограниченную рамками разумной достаточности, проверить и отладить эту часть. После чего сохранить проект, а затем создать новый проект, куда можно включить уже отлаженную часть или отложить это до более подходящего случая. Иначе, как это часто получается у меня, проект, содержащий ошибки, при попытке их устранить обогащается множеством новых, не менее увлекательных ошибок, и, в конце концов, превращается в полное собрание моих ошибок и заблуждений. Вернуть его к первоначальному виду, который хотя бы как-то работал, задача, которая по плечу такому атлету, как Сизиф, но не мне.

Выход простой – почаще использовать первый пункт подменю раздела Project – New Project... Далее в подменю есть пункты открывания существующего проекта (Open Project...), и недавно использовавшегося проекта (Open Recent Project).

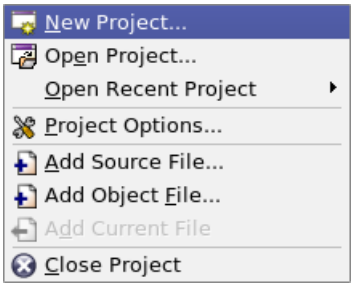


Рис. 2.13. Подменю раздела Project

Перемахнув через один из пунктов, я хочу закончить обзор этого подменю, сказав, что Add Source File..., Add Object File... и Add Current File относятся к

заполнению проекта файлами. Первым добавляется файл исходного текста, например, ассемблерного или Си, вторым добавляется объектный файл, а последний добавляет в проект текущий (активный) файл, который вы могли создать с помощью меню File. Последний пункт закрывает проект, и если вы не сохранили его перед этим, открывается диалог сохранения файлов. Тот пункт меню, о котором я не упомянул, а именно, Project Options..., открывает диалоговое окно установок проекта.

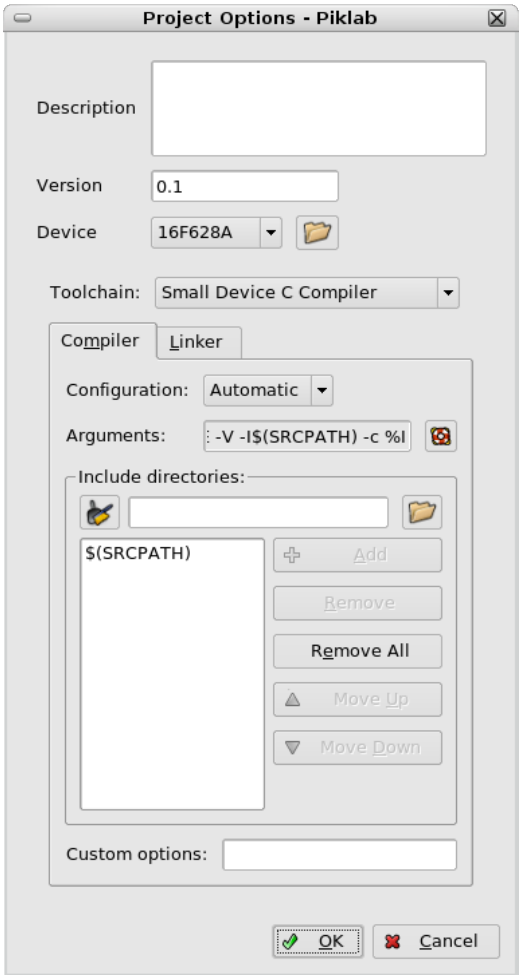


Рис. 2.15. Диалоговое окно установок проекта

Здесь можно выбрать ваш микроконтроллер, транслятор языка, добавить

описание проекта. Опции, относящиеся к таким возможностям, как аргументы или пользовательские опции, скорее для опытных разработчиков, я лично, предпочитаю их не трогать до тех пор, пока не начну в полной мере осознавать последствия этого эксперимента.

Следующий раздел основного меню Build позволяет вам «построить» свой проект: скомпилировать только один файл, обработать весь проект или очистить проект, если вы многократно (и безуспешно) пытались транслировать написанный вами код. Последний пункт Stop относится, насколько я понимаю, к остановке процесса трансляции кода. Если текст кода большой, его трансляция может занимать достаточно много времени, а чтобы не ждать завершения, когда вы вспомнили, что собирались заменить имена переменных, можно остановить трансляцию.

Следующий раздел основного меню относится к работе с программатором. Его установку можно выполнить в разделе Settings, а работу осуществить либо с помощью меню, либо использовать клавиши основной инструментальной панели. После выбора программатора подменю выглядит следующим образом:

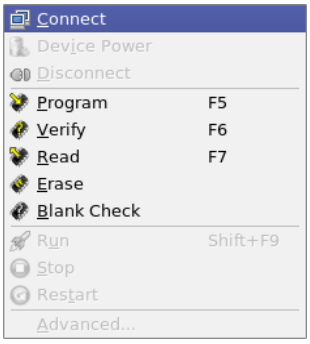


Рис. 2.16. Вид подменю работы с программатором

Первый пункт – подключение программатора, после активизации подключения возможно отключение (Disconnect). Процесс программирования (Program) отображается в нижней части экрана проекта. Далее следует проверка (Verify), чтение (Read), очистка (Erase) и проверка очистки. К чему относятся остальные, не активизированные пункты меню, я пока не разобрался, но, возможно, они предназначены для дальнейшего расширения проекта.

С программированием микроконтроллера у меня есть небольшая «заморочка». Перед программированием я не могу задать слово конфигурации (по адресу 2007). Чтобы задать его, мне приходится прочитывать запрограммированную микросхему, после чего я могу вписать необходимое мне значение (3F18), которое выглядит теперь как 2118, и записать его. Не самая сложная операция. Но почему-то это не получается сделать сразу.

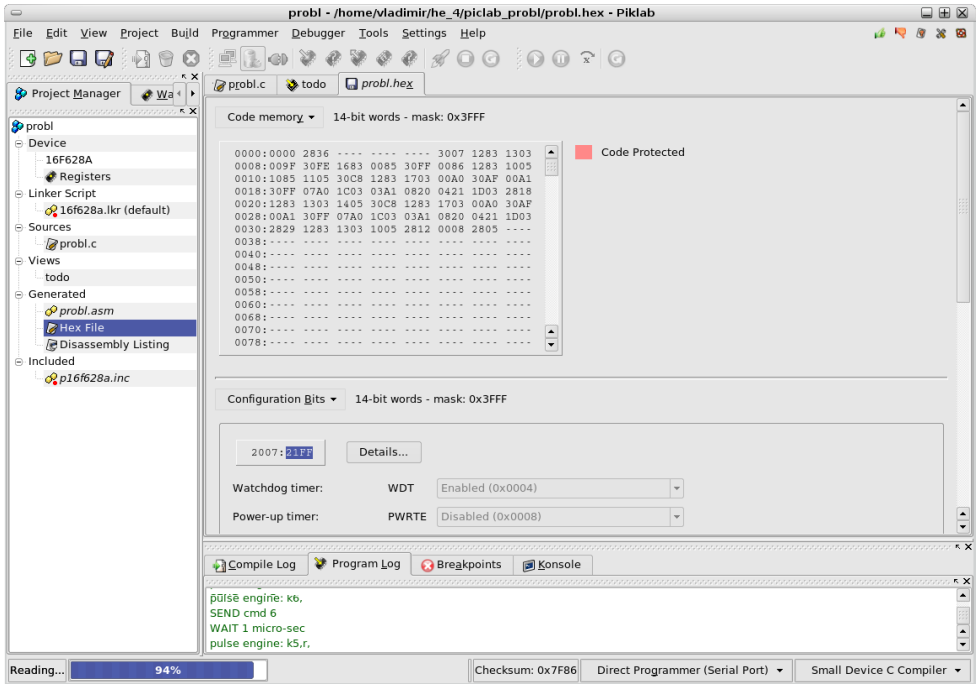


Рис. 2.17. Отображение процесса чтения из микросхемы

Я пропущу раздел Debugger, поскольку решил отладку программы осуществлять непосредственно в программе симуляции gpsim, иначе отладка у меня не работает, да и удобнее, мне кажется, работать с двумя программами на разных рабочих столах.

За разделом Debugger следует раздел Tools – инструменты программы Piklab. Количество доступных инструментов и активность разделов зависят от того, с каким файлом вы в данный момент работаете. Наибольшее количество пунктов в подменю появляется при работе с исходным текстом на языке Си.

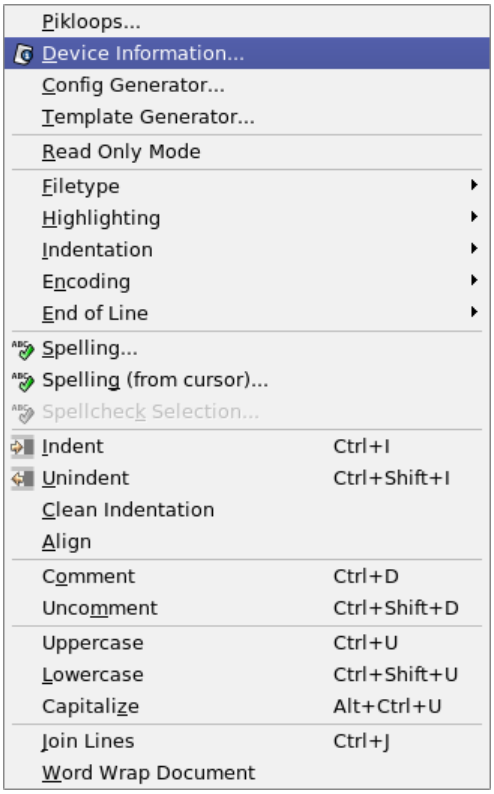


Рис. 2.18. Подменю раздела инструментов программы piklab

Pikloops – это дополнение к программе, позволяющее получать коды на ассемблере для организации временных циклов. Если это приложение установлено, то выбор этого пункта подменю запускает приложение.

Многие языки высокого уровня позволяют сделать ассемблерные вставки. Не помню, говорил или нет о том, как я однажды наступил (будучи оптимистом, надеюсь не в последний раз) на «грабли». Задав время с помощью пустого цикла for в программе на языке Си я долго удивлялся, почему у меня время получается неправильным, пока не сообразил посмотреть, как выглядит этот цикл на ассемблере. Оказалось, что совсем не так, как я это представлял себе. Словом, если нужно сделать время точным, надо использовать ассемблерную вставку, в чем поможет программа pictools, или использовать другие варианты, но обязательно проверить точность их работы.

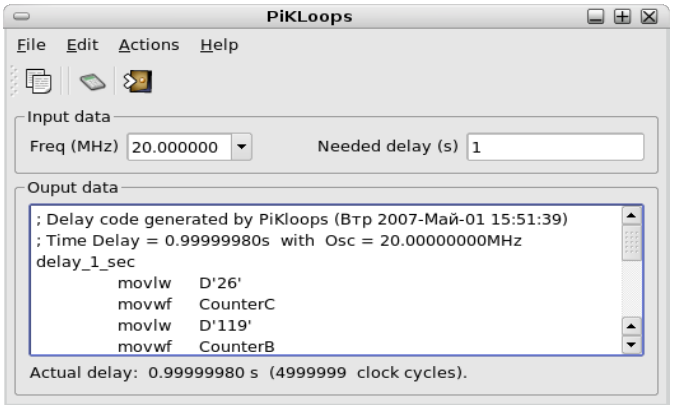


Рис. 2.19. Приложение piktools

Можно задать тактовую частоту микроконтроллера, необходимое время цикла и получить готовый код.

Следующий пункт подменю этого раздела тоже, с моей точки зрения, очень полезен. Я постоянно забываю расположение выводов контроллера, приходится обращаться к документации производителя, или рисовать цоколевку микросхемы постоянно поглядывая на рисунок. Работая с программой Piklab можно обращаться к пункту меню – Device Information...

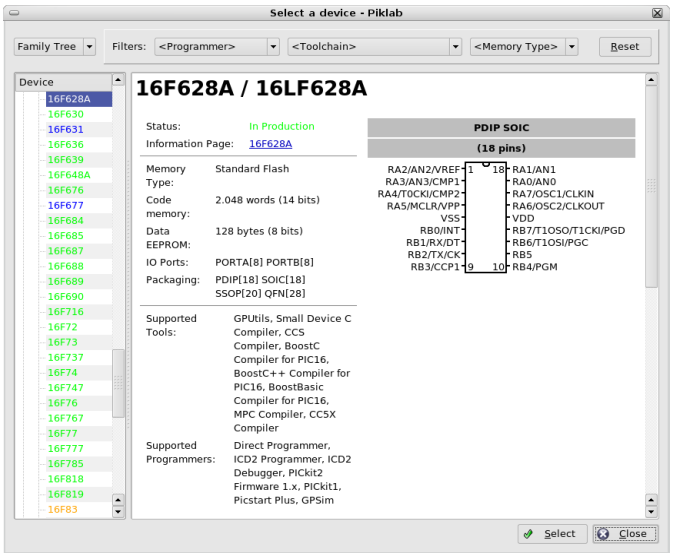


Рис. 2.20. Информация о текущем контроллере

Следующий инструмент раздела Tools должен полностью снять вопрос о том, как записать слово конфигурации по адресу 2007. Запуск этого инструмента дает диалог установки всех необходимых бит конфигурации для выбранного микроконтроллера и языка программирования. В окне диалога можно выбрать необходимую конфигурацию, используя клавиши управления установками, а затем скопировать необходимый текст с помощью клавиши Copy to clipboard, чтобы, закрыв окно диалога, в текстовом редакторе просто вставить текст в нужное место из буфера обмена.

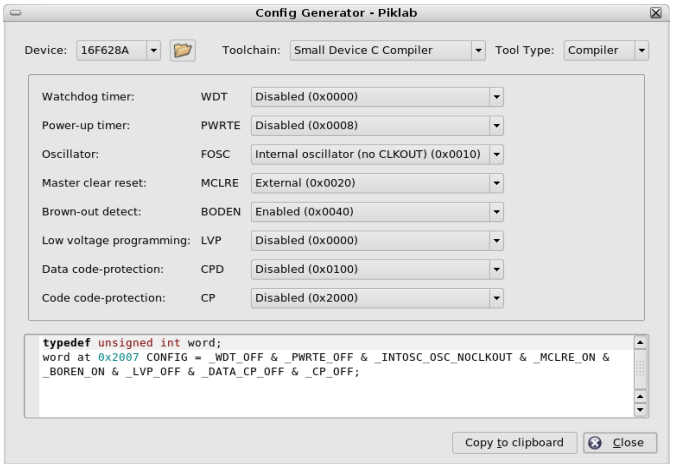


Рис. 2.21. Диалоговое окно задания слова конфигурации контроллера

После того, как текст скопирован и вставлен в код программы, а сама программа откомпилирована, слово конфигурации будет записано при программировании микросхемы. До этого «открытия» мне приходилось поступать иначе. Теперь буду знать.

Хотя я чаще пользуюсь тем, что было сделано ранее, чем начинаю проект «с нуля», видимо есть смысл использовать и те возможности по оформлению кода программы, которые предоставляет Piklab. В частности, возможность генерировать шаблон, правильно понимаемый компилятором. Для этих целей служит инструмент под названием Template Generator...

Аналогично тому, как создается слово конфигурации в диалоговом окне его генерации, в окне диалога шаблона для выбранного языка программирования создается шаблон основного текста программы. Его можно скопировать в буфер обмена с тем, чтобы вставить в текст программы.

Можно поступить и иначе, используя мастер создания нового проекта, где на одной из страниц диалога есть возможность сразу установить опцию шаблона для

выбранного языка программирования и используемого транслятора кода программы. Оба варианта доступны. Использовать инструментальное меню можно при создании проектов, содержащих более одного файла текста кода.

Использование пункта подменю Read Only Mode (режим только чтения) в данный момент не актуально, но при работе над программой может и понадобиться.

Далее следует достаточно длинный перечень средств, которые опытному разработчику сильно облегчат жизнь, но не думаю, что они понадобятся в любительской практике. А если и понадобятся, то все эти возможности можно освоить по ходу дела.

О следующем пункте меню Settings я рассказывал выше. В этом пункте меню можно установить видимость (или невидимость) панелей, задать все свойства программы и программных средств. Этим пунктом меню я пользовался, когда подключал программатор, и тогда, когда подключал компилятор языка Си.

Разделы установок открывают соответствующее диалоговое окно, устроенное достаточно удобно, чтобы не только настроить среду программирования, но и убедиться в том, что настройка произведена. Так, напомню, что когда я немного «заблудился» с выбором компилятора языка Си, когда нашел и установил пакет SDCC (rpm пакет для легкой и быстрой установки), то обнаружил именно в окне диалога, что программа не обнаруживает этого компилятора. И только после повторной установки компилятора Small Device C Compiler из исходных текстов в разделе Configure Toolchains... я увидел, что и сам компилятор, и все необходимые составляющие программой Piklab обнаружены.

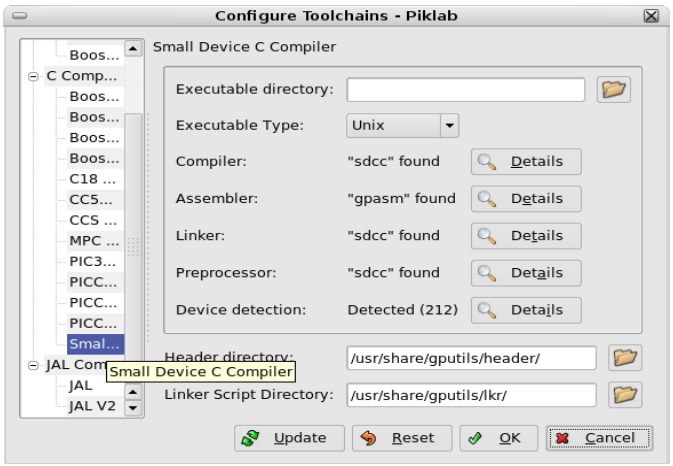


Рис. 2.22. Диалоговое окно настройки языка программирования

Как видно из рисунка, программа обнаружила и сам компилятор («sdcc» found), и

ассемблер, и компоновщик, и препроцессор.

В целом в программе на текущий момент только раздел меню Help оказывается бесполезен. Думаю, это временно, и прежде, чем я допишу книгу, он существенно пополнится.

Основная инструментальная панель программы повторяет наиболее употребительные разделы и пункты меню. Клавиши инструментальной панели активизируются соответственно тому, с каким из файлов в настоящий момент ведется работа, а при отладке активизируются клавиши отладчика, если на нижней панели рабочей области программы выбрать в качестве программатора Gpsim.

Рис. 2.23. Основная инструментальная панель программы piklab

Многие диалоговые окна и подменю основного меню программы чувствительны к типу файла, с которым в текущее время происходит работа. По этой причине я очень рассчитываю на то, что при реальной работе над программой объявятся новые возможности, о которых я буду сообщать по мере их появления. А сейчас пришло время рассказать об отладке программ.

Для этих целей я использую программу gpsim. Как рекомендовано в руководстве, загружать в программу лучше файл с расширением cod, создаваемый при трансляции исходного кода на ассемблере или языке Си.