В.Н. Гололобов

О ПЕРЕВОДЕ ОДНОГО РУКОВОДСТВА

Часть 3. Читая уроки по wxWidgets

Москва -2015

Оглавление

Читая рассказ о C++ при переводе («О переводе одного руководства»), и тогда, когда я уничтожал многочисленные опечатки, я твёрдо решил, что сам воспользуюсь этим руководством, чтобы освежить свои скудные знания о языке и пополнить их с помощью специалиста, написавшего этот хорошее повествование. Но понимать прочитанное, понимать то, о чём это написано, а, главное, уметь применить это на практике – разные этапы освоения предмета. После нескольких попыток что-то сделать, я решил обратиться к опыту профессионалов по использованию wxWidgets, найдя в Интернете оригинальные уроки:

http://zetcode.com/gui/wxwidgets/

Перевод этих уроков есть на сайте:

http://www.sl-alex.com.ua/ru/page/wxwidgets_tutorial_00_introduction

Привет, Мир!

Повторение консольных программ путём простого копирования примеров в текстовый редактор Code::Blocks с последующей сборкой и запуском – это не то, чему мне хотелось бы научиться. Поэтому я решил повторить примеры в графическом виде с помощью wxSmith для начала. Благо с первыми шагами в создании wxWidgets project справиться удалось. Остались некоторые сомнения в правильности выбранных настроек, которые можно проверить. В частности, мне не очень было понятно с дополнительными файлами – в прошлый раз я просто скопировал эти настройки. Сейчас можно неспешно, создавая новый проект, проверить достаточный выбор для первых опытов. Вот какие настройки в диалоге создания нового проекта:

Рис. 1.1. Первые настройки проекта

Но при попытке собрать проект командой *Build* (я использую иконку на инструментальной панели) появляется сообщение, что нет *Tiff* (чего-то). Не будучи специалистом, я удаляю неудачную версию проекта и создаю её заново, но теперь отмечаю и *wxTiff* тоже. После этого проходит сборка и программа запускается.

Надпись на панели заголовка окна я хотел бы вывести из программы. Я знаю, что окно (по имени проекта) *probaFrame* наследник *wxWindow*. Последнее имеет функцию *SetTitle(),* которую я и хотел бы использовать. В моём представлении это должно выглядеть так:

```
probaFrame.SetTitle("Привет, Мир!");
```

В файл «probaMain.cpp» я вписываю то, что надумал. Результат при сборке выведен в панель сообщений о сборке:

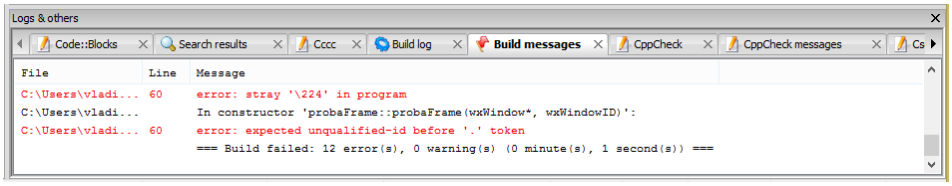

Рис. 1.2. Сообщения о результатах сборки

В поисках ответа на вопрос, в чём проблема, я встретил упрёк участнику форума на аналогичный вопрос, что он плохо читал учебник в части статических членов (так я понял написанное на английском, не вдаваясь в детали). Предложено было исправить фразу:

```
probaFrame::SetTitle(“Привет, Мир!”);
```

Совет помогает, сообщений об ошибке нет. Впрочем, если оставить только:

```
SetTitle(“Привет, Мир!”);
```

То результат тот же, ошибок нет, но и окно выглядит так, как если бы команд и не было – заголовок окна отсутствует. И что я заметил ещё, вот фрагмент текста файла «probaMain.cpp»:

```
55
56  probaFrame::probaFrame(wxWindow* parent,wxWindowID id)
57  {
58      //(*Initialize(probaFrame)
59
60          SetTitle("Привет, Мир!");
61
62      wxMenuItem* MenuItem2;
63      wxMenuItem* MenuItem1;
64      wxMenu* Menu1;
65      wxMenuBar* MenuBar1;
66      wxMenu* Menu2;
67
```

Рис. 1.3. Фрагмент текста

Если перейти на страницу формы, дважды щёлкнув по файлу «probaframe.wxs» в менеджере проекта, на закладке предыдущего файла появляется метка изменения текста, а этот файл изменится следующим образом:

```
56  probaFrame::probaFrame(wxWindow* parent,wxWindowID id)
57  {
58      //(*Initialize(probaFrame)
59      wxMenuItem* MenuItem2;
60      wxMenuItem* MenuItem1;
61      wxMenu* Menu1;
62      wxMenuBar* MenuBar1;
63      wxMenu* Menu2;
```

Рис. 1.4. Тот же фрагмент текста после перехода к окну формы

То выражение, что я добавлял, исчезло. Конечно, я могу справиться с этой задачей, если просто впишу в свойство Title формы то, что хотел:

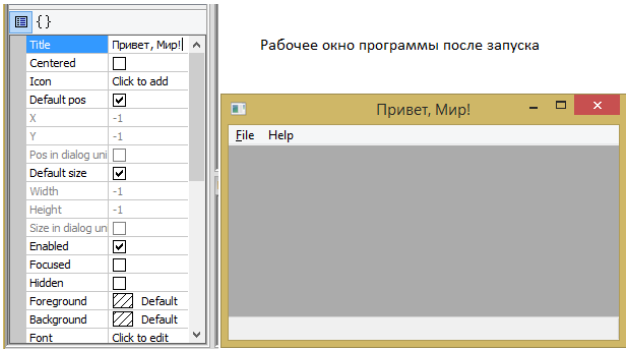

Рис. 1.5. Текст в свойстве Title формы и форма после запуска

Все эти манипуляции я проделал в Fedora 21, где пришлось добавить:

```
SetTitle(wxT("Привет, Мир!"));
```

Но результат в точности повторил всё показанное выше. Конечно, когда я узнаю больше, когда научусь разбираться лучше, я, возможно, пойму, в чём здесь дело, а сейчас я хочу попробовать другие возможности вывода фразы «Привет, Мир!».

В рабочее окно я могу поместить кнопку и некое текстовое поле. Если щёлкнуть по кнопке, то в текстовое поле пусть будет выведено «Привет, Мир!».

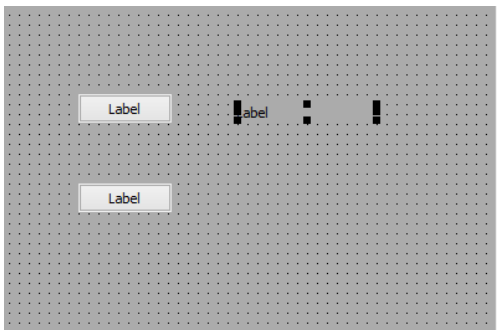

Рис. 1.6. Заготовка формы для эксперимента

После двойного щелчка по первой кнопке в файле «proba1Main.cpp» появляется заготовка под ответ на щелчок по кнопке. Туда я вписываю то, что мне кажется понятным и правильным:

```
void proba1Frame::OnButton1Click(wxCommandEvent& event)
{
    StaticText1.SetLabel("Привет,Мир!");
}
```

Сборка даёт ошибку, но выводит и подсказку:

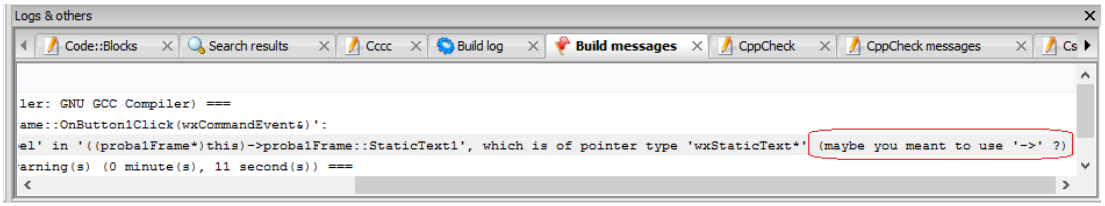

Рис. 1.7. Сообщение об ошибке и подсказка

Если заменить точку на оператор стрелки, то…

```
void proba1Frame::OnButton1Click(wxCommandEvent& event)
{
    StaticText1 -> SetLabel("Привет,Мир!");
}
```

После сборки проекта, которая теперь проходит без ошибок, можно запустить программу, а щелчок по первой кнопке даёт такой результат:

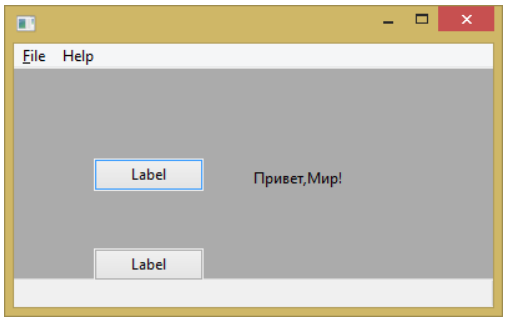

Рис. 1.8. Выполнение программы

Что такое оператор стрелки? Вот, что написано в руководстве по языку C++:

> *Оператор стрелки (->) – это оператор разыменовывания, который используется исключительно с указателями на объекты, которые имеют своих членов. Этот оператор используется для доступа к членам объекта непосредственно по их адресам*.

Подсказку я, признаться, увидел не сразу, а перед этим пробовал разные варианты. В частности, а что будет, если оставить только:

```
void proba1Frame::OnButton1Click(wxCommandEvent& event)
{
    SetTitle("Привет,Мир!");
}
```

Сборка проекта не дала ошибки. А запуск программы дал такой результат после щелчка по первой кнопке:

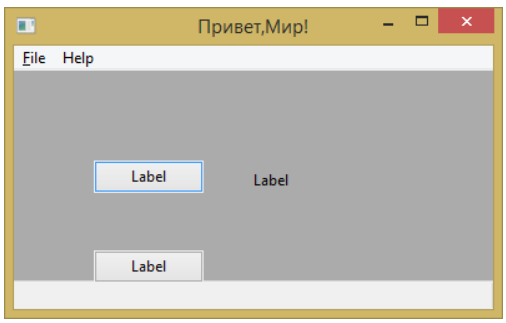

Рис. 1.9. Неожиданный результат эксперимента

То есть, выражение установки этикетки для окна было правильным, но появилось только после некоторого события. Возможно, отсутствие событий в том месте программы, куда я пытался записать выражение для задания заголовка, и стало причиной первой неудачи. Возвращаясь к первой программе, я отыскиваю место, где с моей точки зрения есть событие.

```
55
56  probaFrame::probaFrame(wxWindow* parent,wxWindowID id)
57  {
58      //(*Initialize(probaFrame)
59      wxMenuItem* MenuItem2;
60      wxMenuItem* MenuItem1;
61      wxMenu* Menu1;
62      wxMenuBar* MenuBar1;
63      wxMenu* Menu2;
64
65      Create(parent, id, wxEmptyString, wxDefaultPosition, wxDefaultSize, wxDEFAULT_FRAME_STYLE, _T("id"
66
67      SetLabel("Привет, Мир!");
68
69      MenuBar1 = new wxMenuBar();
70      Menu1 = new wxMenu();
71      MenuItem1 = new wxMenuItem(Menu1, idMenuQuit, _("Quit\tAlt-F4"), _("Quit the application"), wxITEM
72      Menu1->Append(MenuItem1);
73      MenuBar1->Append(Menu1, _("&File"));
```

Рис. 1.10. Новое место для задания выражения

Если бы я был внимательнее, то заметил бы, что в этом месте создаётся окно с пустым заголовком. Если бы…

После трансляции и запуска получается следующее:

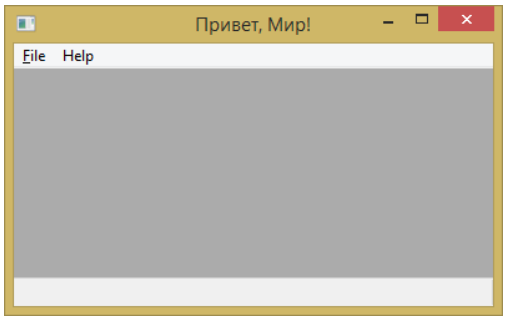

Рис. 1.11. Появление заголовка окна

То есть, выражение для заголовка было вставлено до события, в котором создаётся окно с пустым заголовком. Поэтому заголовка и не было.

И теперь, пожалуй, станет понятным, что происходит, когда мы вписываем заголовок в поле свойств формы (я написал «Привет»). Code::Blocs удаляет написанное мной выражение, а в команду создания окна (событие) вписывает заголовок:

```
55
56  probaFrame::probaFrame(wxWindow* parent,wxWindowID id)
57  {
58      //(*Initialize(probaFrame)
59      wxMenuItem* MenuItem2;
60      wxMenuItem* MenuItem1;
61      wxMenu* Menu1;
62      wxMenuBar* MenuBar1;
63      wxMenu* Menu2;
64
65      Create(parent, id, _("Привет"), wxDefaultPosition, wxDefaultSize, wxDEFAULT_FRAME_STYLE, _T("id"))
66      MenuBar1 = new wxMenuBar();
67      Menu1 = new wxMenu();
68      MenuItem1 = new wxMenuItem(Menu1, idMenuQuit, _("Quit\tAlt-F4"), _("Quit the application"), wxITEM
```

Рис. 1.12. Изменения, вносимые средой разработки

Вот такая простейшая программа «Hello World!» получается.

wxWidgets и Fedora 21

Программа Code::Blocks кросс-платформенная. Поэтому я решил, что следует повторить последнее упражнение из предыдущей главы в моём дистрибутиве Linux Fedora 21, используя графические возможности Code::Blocks.

Первое, что следует сделать, исходя из моего предыдущего опыта, это добавить ряд файлов, чтобы использовать wxSmith. Какие точно файлы следует добавлять, я не вполне понял, поэтому файлов, которые я добавил, может оказаться в переизбытке. И, кстати, это может зависеть от дистрибутива Linux. Несомненно, пожалуй, что следует установить нечто, что относится к wxWidgets. Если заглянуть в пакеты для Fedora 21 на mirror.yandex.ru, то можно обнаружить большое количество пакетов, начинающихся с «wx»:

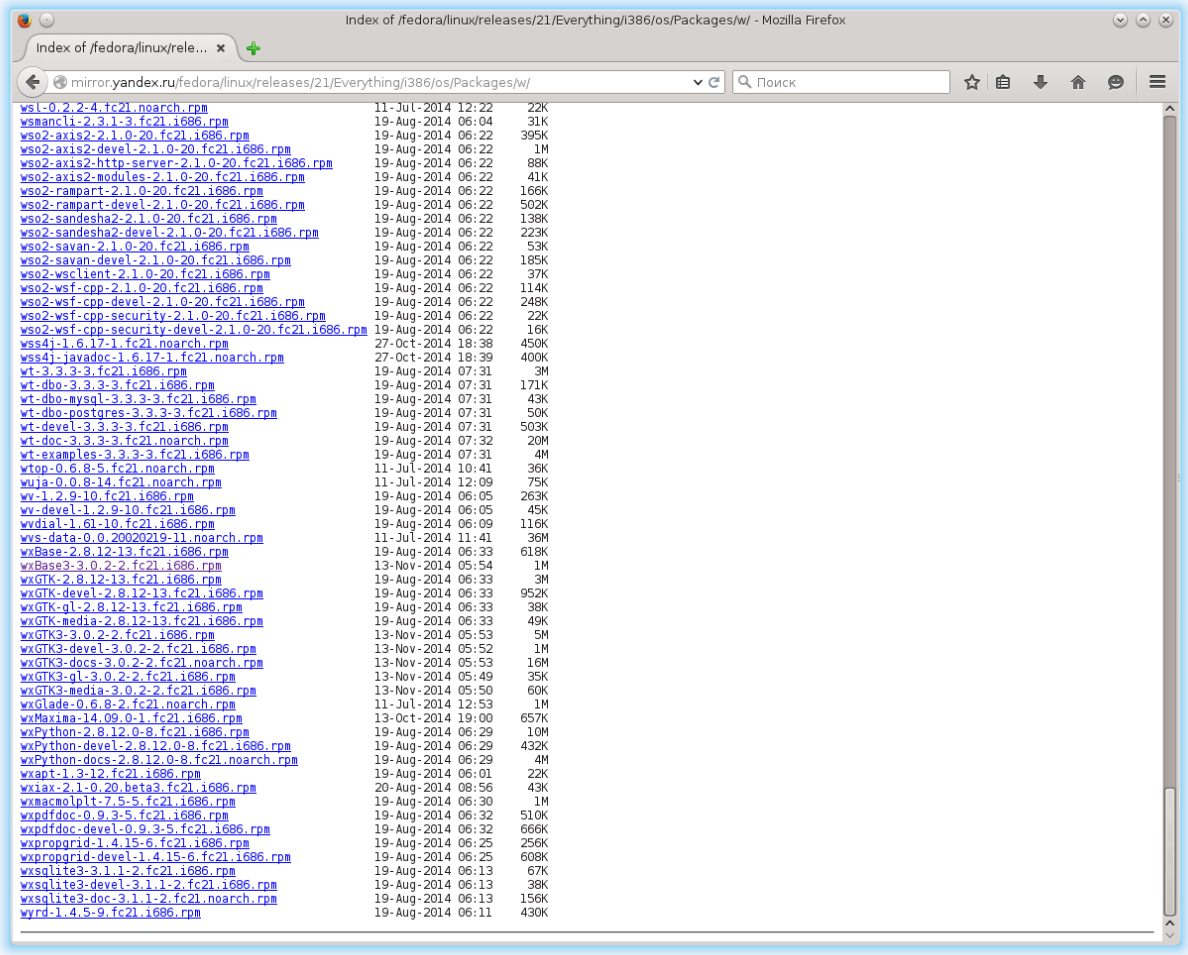

Рис. 2.1. Список пакетов для wxWidgets

Первое, что с моей точки зрения следует установить — это wxBase. Я установил обе версии, и wxBase-2.8.12, и wxBase-3.3.0, но не уверен, что не хватило бы и первой версии.

Создание проекта в Code::Blocks для Linux не отличается от аналогичной процедуры в Windows: *Create a new projects*, затем выбрать тип проекта.

Чтобы в проекте появилась возможность работы с wxSmith, я установил codeblocks-contrib. И, что называется «на всякий случай», codeblocks-contrib-libs и codeblocks-libs, установил wxGTK-2.8.12. Повторюсь, что-то, возможно, лишнее, а что-то я не упомянул.

После завершения процедуры создания проекта (я выбрал Widgets 2.8.x в диалоге, wxSmith Frame Based, Use default wxWidgets configuration, путь для папки проекта я постарался выбрать без русскоязычных имён), я пробую собрать пустой, с одним только окном по умолчанию проект. Эти попытки я делаю до тех пор, пока не проходит без ошибок трансляция и сборка проекта.

Теперь можно добавить, например, кнопку и повторить сборку проекта, запустить программу. Правда, если я добавляю одну кнопку, она занимает всё рабочее пространство окна, отчего я добавляю сразу две кнопки. Когда все необходимые для работы файлы установлены, программа собирается и запускается.

Чуть не забыл. Все команды включения графики для препроцессора имеют вид: #include <wx/msgdlg.h>. Чтобы программа нашла нужные файлы заголовков, следует в терминале выполнить две команды:

```
sudo cd /usr/include
sudo ln -sv wx-2.8/wx wx
```

Этим будет создана ссылка на нужную папку.

Убедившись, что с графическими возможностями Code::Blocks всё в порядке, можно добавить создание заголовка, как это сделано раньше. Отличие, напомню, в следующем:

```
SetTitle(wxT("Привет, Мир!"));
```

Запуск показывает, что всё проходит аналогично тому, как это имело место в Windows, но трансляция и сборка проходит гораздо быстрее.

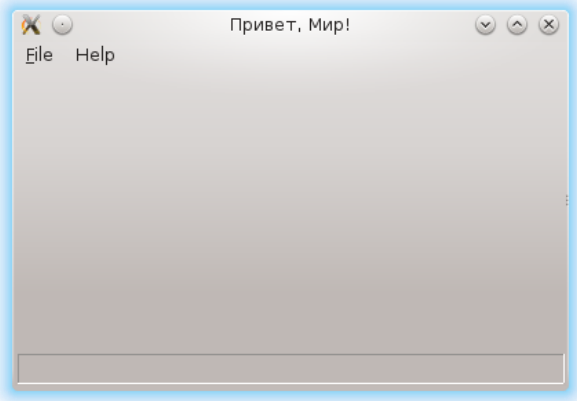

Рис. 2.2. Окно с заголовком

Сложение двух чисел

Но вернёмся к руководству. Одна из первых программ – это сложение двух чисел. В графическом изложении потребуется место, куда можно ввести слагаемые и куда можно вывести результат сложения. Для инициализации операции сложения я хочу использовать кнопку.

Среди графических элементов есть wxTextCtrl (не буду настаивать, что это единственный выбор).

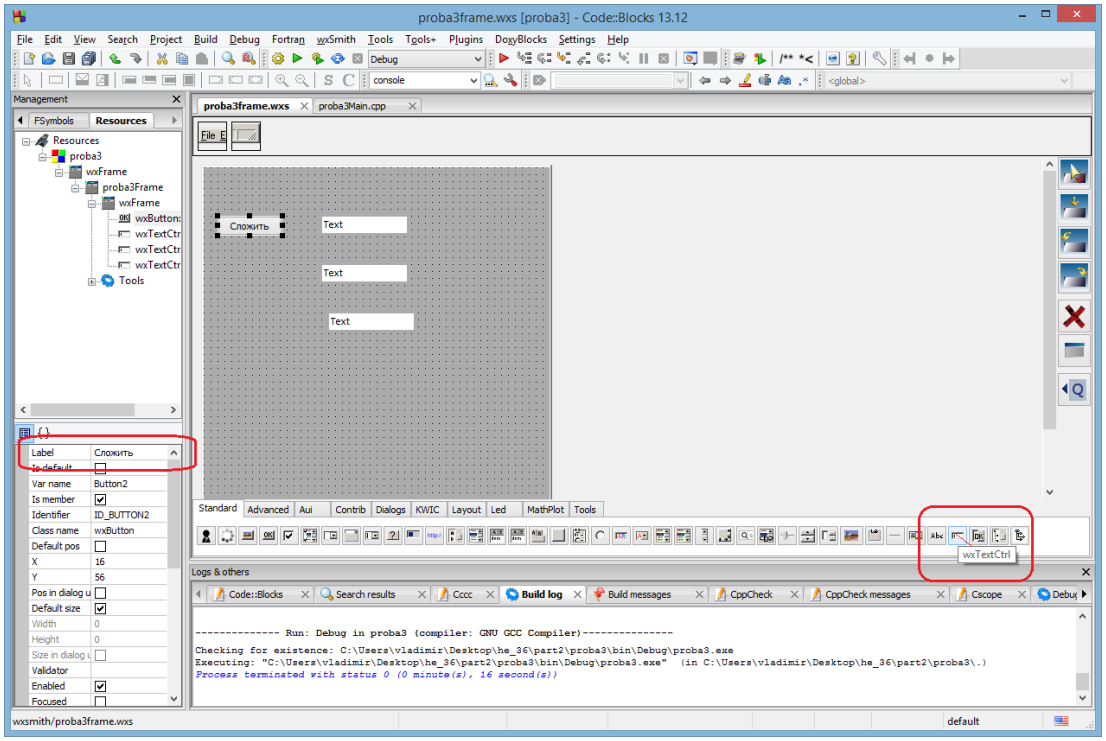

Рис. 3.1. Текстовое окно для ввода

В первое окно я введу первое слагаемое, во второе окно второе, а в третьем после щелчка по кнопке должен появиться результат. Вот такой у меня план. Для добавления элементов графики на рабочее поле окна достаточно щёлкнуть по этому элементу, перенести курсор в нужное место, где щёлкнуть повторно. В свойствах кнопки атрибут *Label* можно заменить надписью *Сложить*. Разместить все элементы в рабочем поле окна несложно – подцепить мышкой элемент и перенести в нужное место.

Работа с графическими элементами не отменяет обычного программирования: чтобы работать со слагаемыми я предпочитаю добавить в программу три переменные в заготовку щелчка по кнопке *Сложить*.

```
113
114  void proba3Frame::OnButton2Click(wxCommandEvent& event)
115  {
116      int a;
117      int b;
118      int result;
119  }
120
```

Рис. 3.2. Добавление переменных

Перед продолжением работы я хочу убедиться, что могу ввести нужные числа в текстовые поля. Оттранслировав заготовку программы, я мышкой выделяю в текстовом поле надпись *Text*, а затем ввожу какие-то числа.

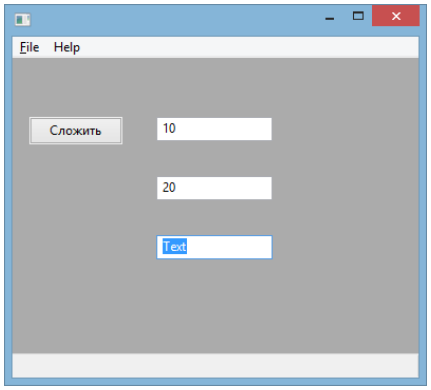

Рис. 3.3. Ввод чисел в текстовое поле

Теперь пора задуматься, как этим переменным присвоить значения, введённые в текстовые поля. Особенно, если учесть, что вводятся строки, а не числа.

Многие ответы на свои вопросы я нахожу здесь:

Рис. 3.4. Сайт с документацией по wxWidgets

Описание классов помогает найти подходящие функции. Для считывания введённых значений в текстовые поля можно использовать функцию:

```
TextCtrl1 -> GetValue();
```

Поскольку переменные целого типа, а функция принимает строку, требуется преобразование текста в целое значение. И функция преобразования тоже есть:

```
a = wxAtoi(TextCtrl1 -> GetValue());
```

Для обратного преобразования переменной *result* в текстовый вид для последующего ввода в окно результата функция выглядит несколько громоздко, но работает:

```
wxString some = wxString::Format(wxT("%i"),rezult);
```

А вывести её в текстовое окно можно с помощью:

```
TextCtrl3 -> SetValue(some);
```

Таким образом, реакция на щелчок по кнопке «Сложить» выглядит так:

```
void proba3Frame::OnButton2Click(wxCommandEvent& event)
{
    int a;
    int b;
    int rezult;

    a = wxAtoi(TextCtrl1 -> GetValue());
    b = wxAtoi(TextCtrl2 -> GetValue());
    rezult = a + b;
    wxString some = wxString::Format(wxT("%i"),rezult);

    TextCtrl3 -> SetValue(some);
}
```

Если на второй кнопке изменить надпись Label на «Вычесть», дважды щёлкнуть по этой кнопке, а в качестве реакции повторить всё, что показано выше, то… останется изменить результат:

```
rezult = a - b;
```

Теперь можно сложить два числа и вычесть:

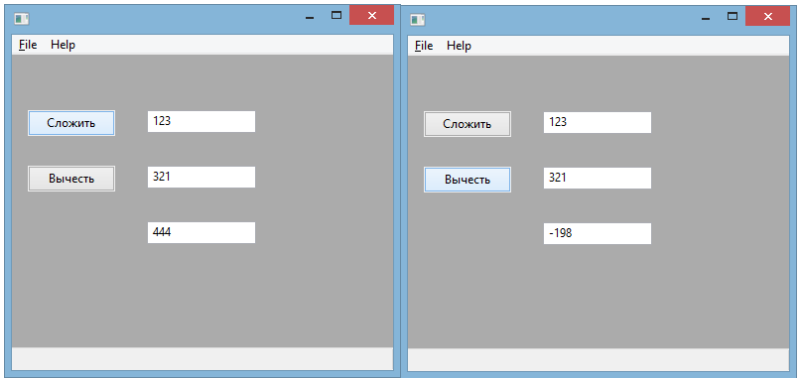

Рис. 3.5. Работа программы с двумя операциями

Прямое копирование блока выражений, где мы не меняем имена переменных, позволительно по той причине, что это локальные переменные. Мы можем добавить ещё две кнопки и повторить вышеописанное для реализации умножения и деления.

В первых главах рассказа о языке C++ много внимания уделено стандартным типам данных, и достаточно подробно объясняется важность указания типа.

Попробуем при вычислениях использовать не целые числа, а десятичные числа, то есть, числа с плавающей точкой. Для переменных этого типа есть два идентификатора: *float* и *double*. При преобразовании используется тип *double*. Поэтому сложение по щелчку «Сложить» будет выглядеть несколько иначе.

```
void proba4Frame::OnButton1Click(wxCommandEvent& event)
{
    double a;
    double b;
    double rezult;

    wxString f_rez1 = TextCtrl1 -> GetValue();
    wxString f_rez2 = TextCtrl2 -> GetValue();

    f_rez1.ToDouble(&a);
    f_rez2.ToDouble(&b);

    rezult = a + b;
    wxString some = wxString::Format(wxT("%f"), rezult);
    TextCtrl3 -> SetValue(some);
}
```

А программа выполняет все операции, как и в первом случае, но с десятичными числами:

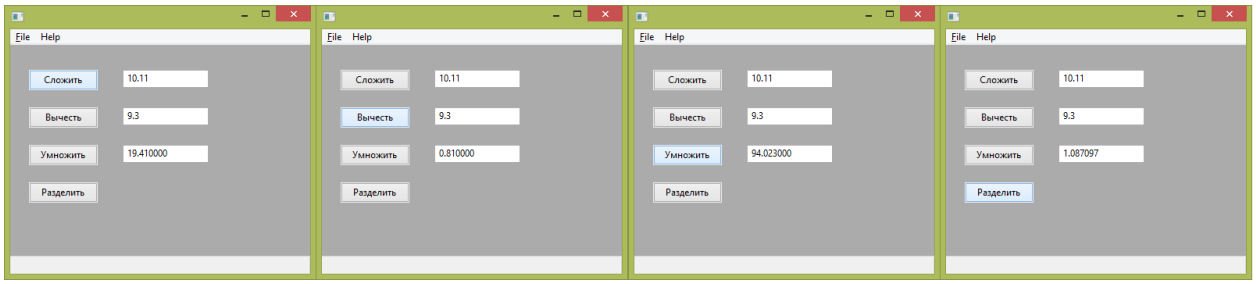

Рис. 3.6. Выполнение операций с десятичными числами

И ещё одно замечание: использование оператора стрелки связано с тем, что во многие функции передаются не значения, а ссылки на эти значения. Об этом достаточно подробно рассказано в главе, посвящённой передачи параметров в функции. При первом знакомстве с языком Си использование указателей и определение значения переменной вызывают трудности. В рассказе о языке C++, как мне кажется, очень доходчиво изложена эта тема.

А та простота, с которой можно написать программу калькулятора, обусловлена тем трудом, который был вложен разработчиками wxWidgwts в многочисленные классы графических объектов и функций-членов этих классов. Возможность использовать готовые библиотеки, написанные профессионалами, одна из самых сильных сторон языка Си.

Некоторые возможности Code::Blocks

Если вам больше по душе русифицированный вариант программы, то поищите в Интернете, воспользовавшись строкой поиска, например: code-blocks ru_ru. Скачав пакет, распакуйте его. Зайдите в папку C:\Program Files\CodeBlocks\share\CodeBlocks\ и создайте в ней папку "locale" после этого откройте её и скопируйте папку ru_RU. После этого запустите программу и зайдите на вкладку Settings->Environment, где и поставьте галочку на Internationalization, выберите Russian из выпадающего списка справа, нажмите **ОК**.

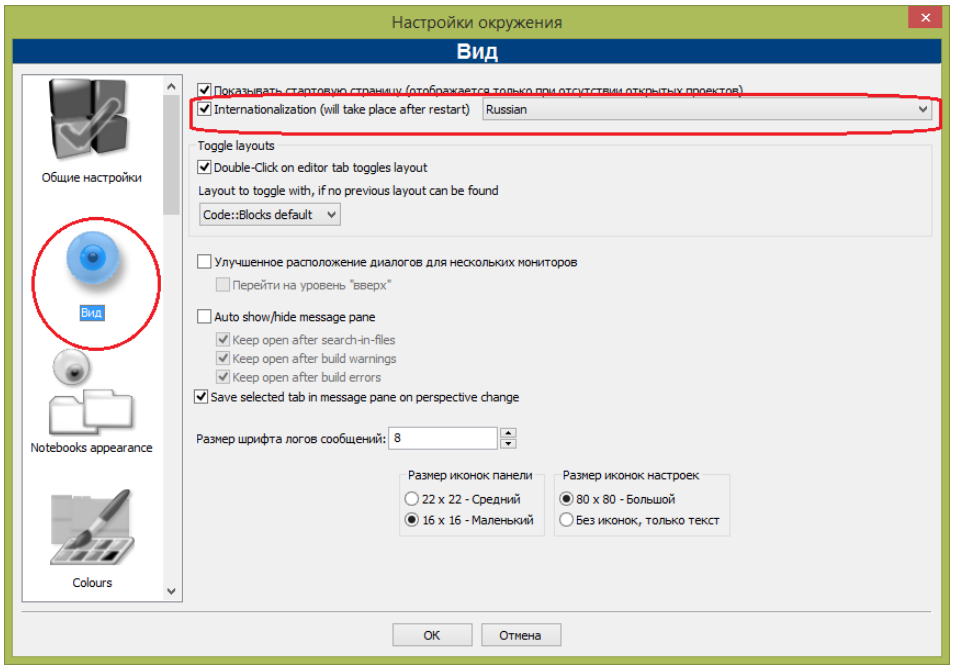

Рис. 4.1. Место изменения языка

После перезапуска программа будет иметь такой вид:

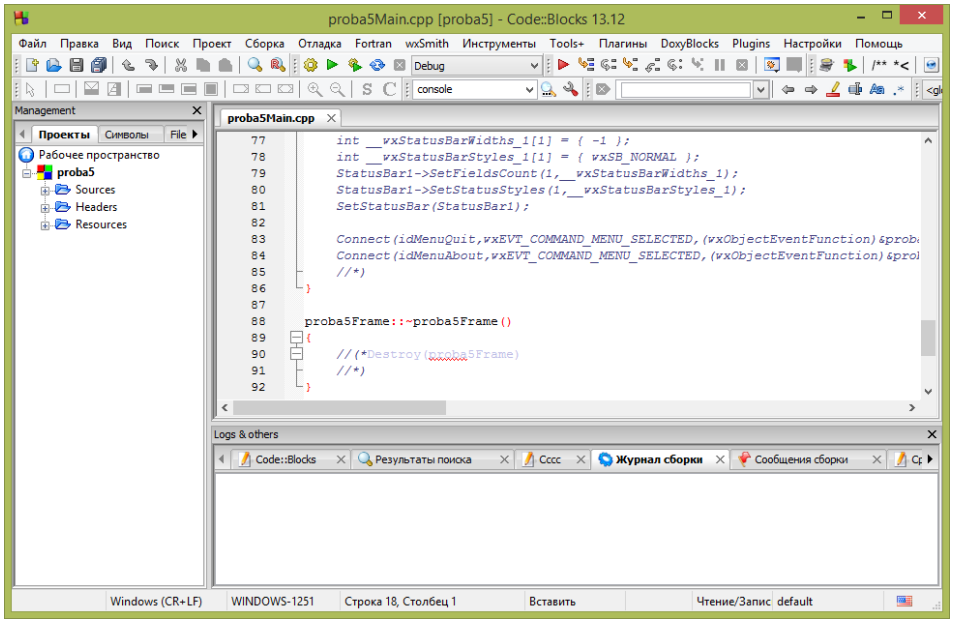

Рис. 4.2. Русифицированная версия Code::Blocks

Для радиолюбителей, изучающих язык Си, программа Code::Blocks может оказаться интересной ещё и тем, что позволяет писать программы для AVR-контроллеров, в частности, для модуля Arduino. Достаточно при выборе типа проекта выбрать AVR project. В качестве компилятора, возможно, используется WinAVR, который предварительно следует тоже установить.

У меня в модуле Arduino микроконтроллер ATMega168, который я и выбираю при создании проекта Code::Blocks. Вместо заготовки основного файла я копирую готовую, написанную кем-то, программу…

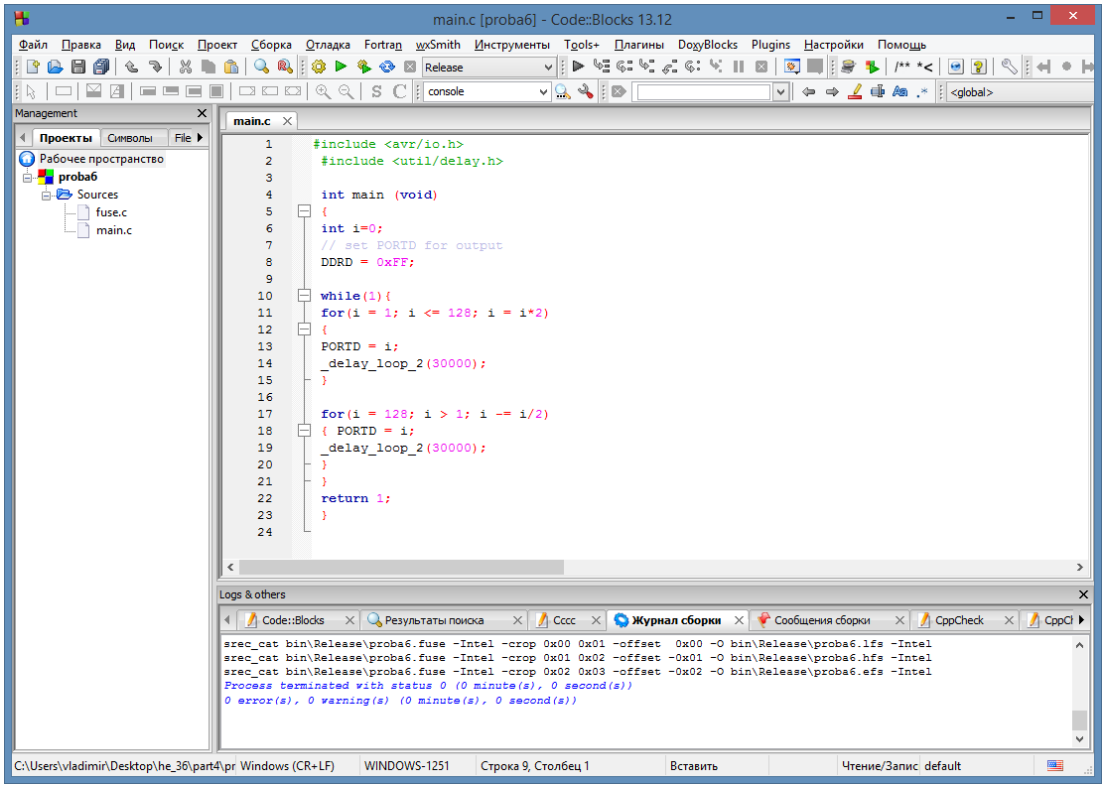

Рис. 4.3. Программа для микроконтроллера ATMega168

…а после трансляции в папке bin\Release нахожу hex-файл с именем проекта. Я предпочитаю проверять в программе ISIS (Proteus), если она есть.

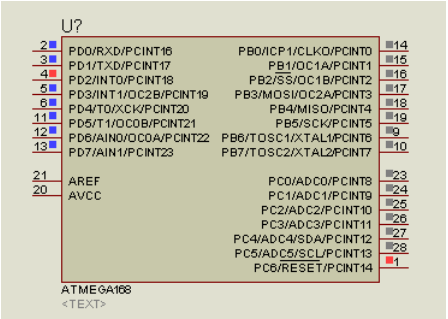

Рис. 4.4. Работа программы при проверке

Для загрузки программы в модуль Arduino можно использовать программу avrdude. В качестве интерфейса для работы с программой можно использовать программу SinaProg.

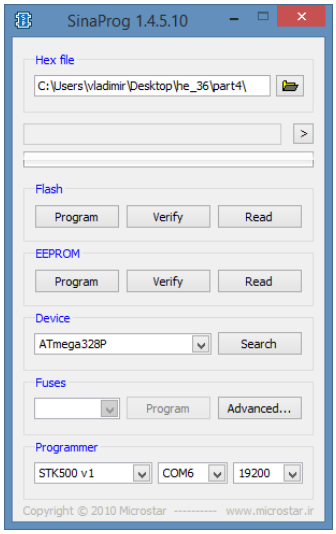

Рис. 4.5. Программа SinaProg

О настройках конфигурации микроконтроллера (fuses) лучше прочитать внимательно, чтобы не заблокировать программную память, установив бит защиты от программирования. Но поучиться использованию языка Си для программирования микроконтроллеров вполне можно, и даже, как мне кажется, можно удачно, с помощью Arduino.

Мне об этом рано говорить, не настолько хорошо я знаю язык, но программа Code::Blocks позволяет создавать ваши собственные библиотеки, как статические, так и динамические. Пример создания статической библиотеки, чтобы не быть голословным, я «срисую» из руководства одного из ВУЗов. Итак.

После запуска программы выбираем создание нового проекта, а тип будущего проекта определяем как:

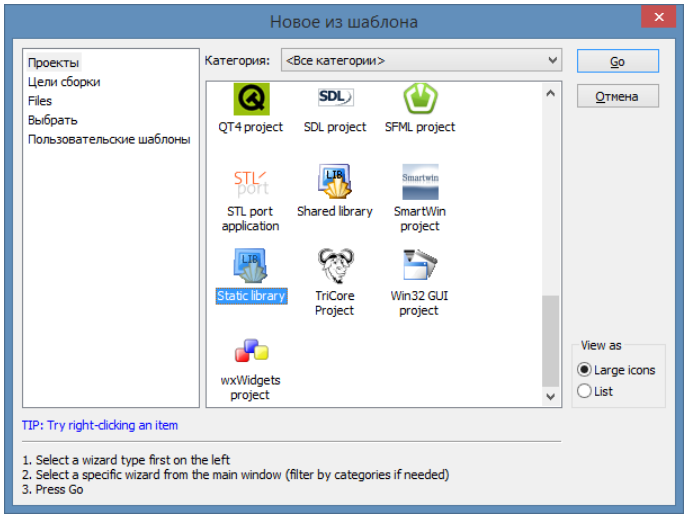

Рис. 4.6. Выбор создания проекта статической библиотеки

Имя исходного файла, созданного программой по завершении настроек проекта, следуя заветам руководства, я переименую: исходный файл языка Си хотелось бы видеть для C++, изменив его расширение. Если файл main.c открыт, его нужно закрыть. Выделив его в окне **Проект**, достаточно щёлкнуть по выделению правой клавишей мышки, чтобы выбрать команду *Переименовать*.

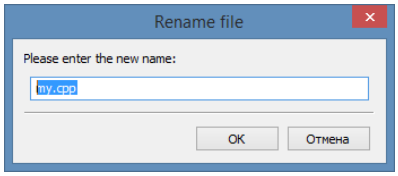

Рис. 4.7. Изменение расширения файла и имени

В пункте *Проект* основного меню выбираем *Properties (свойства)*, где на закладке *Цели сборки* отмечаем наш файл.

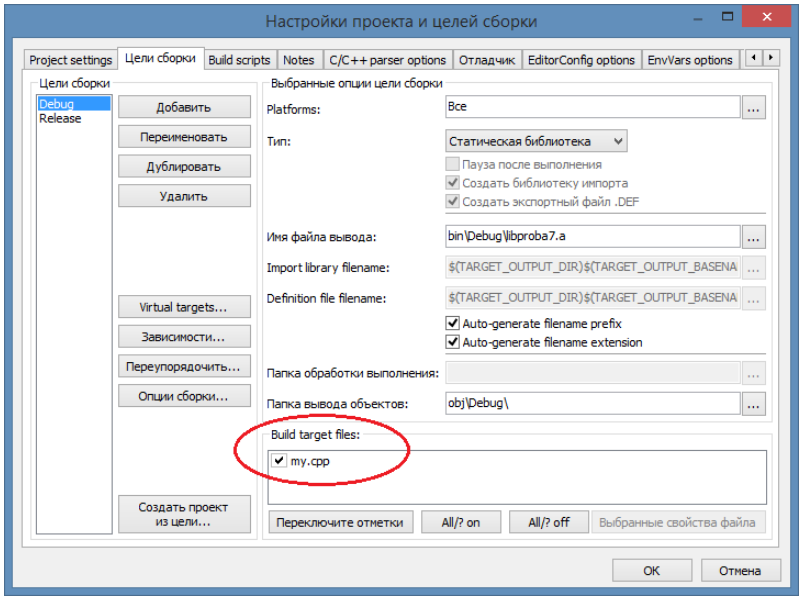

Рис. 4.8. Отметка цели сборки

В проект предстоит добавить заголовочный файл. Для этого выбираем в основном меню пункт Файл->New->File… И делаем надлежащий выбор:

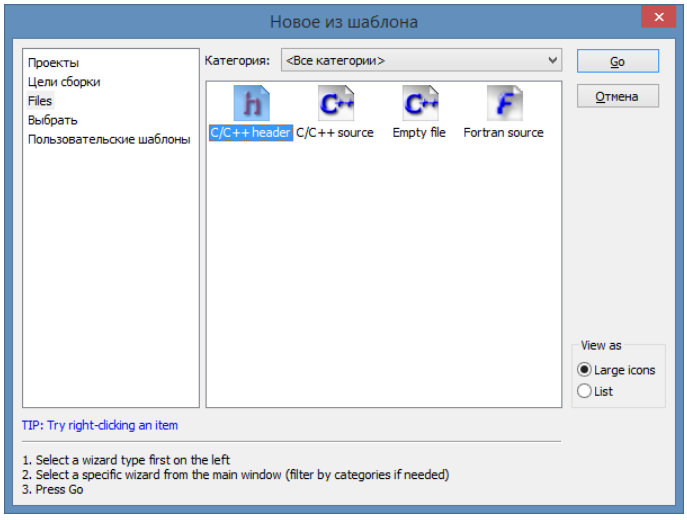

Рис. 4.9. Выбор создания заголовочного файла

Далее в диалоге указываем полный путь к проекту и задаём оба типа режима, как отладочный, так и конечный, исполняемый.

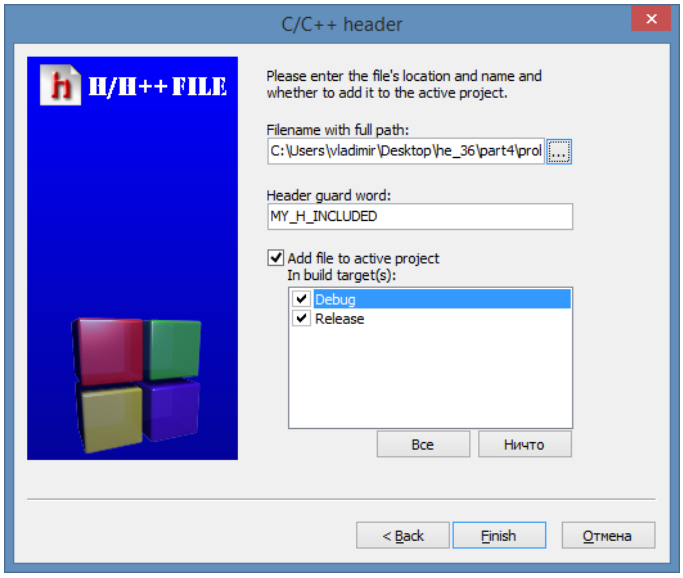

Рис. 4.10. Диалог настройки файла проекта

В заготовку файла заголовка следует вставить имя функции, которая была создана в main.c:

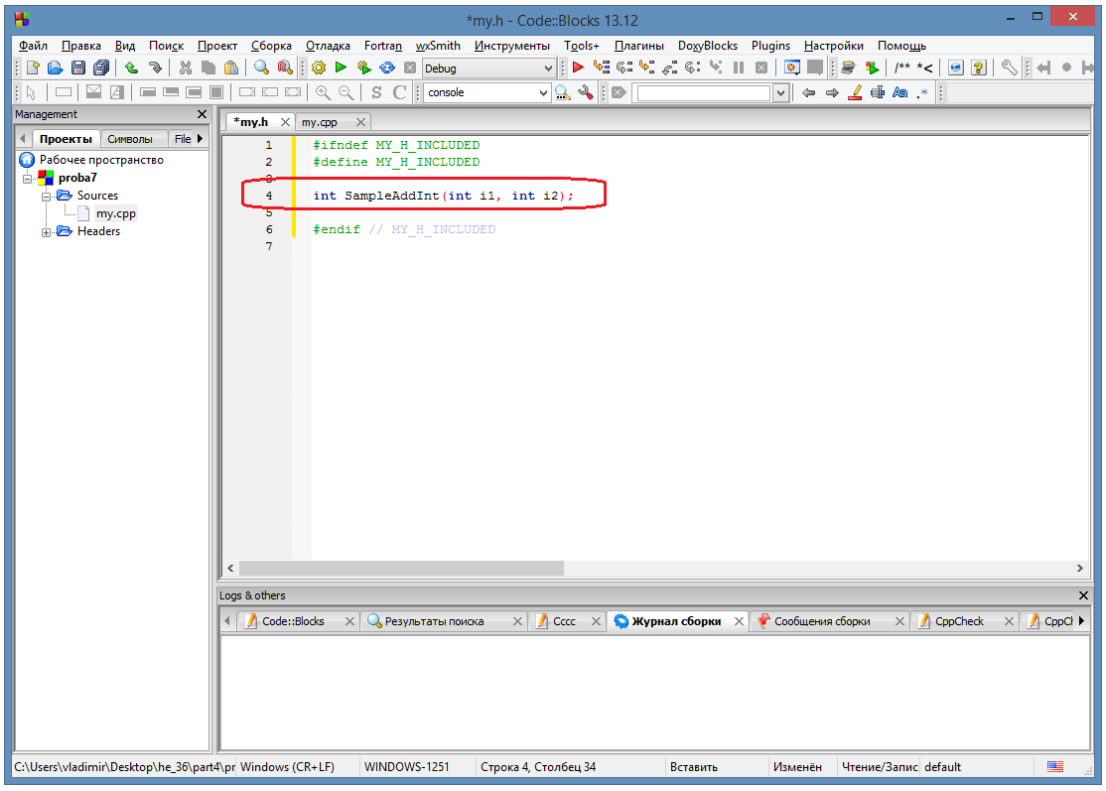

Рис. 4.11. Дополнение заголовочного файла

После трансляции и сборки проекта нужная библиотека появится в папке Debug. Теперь, чтобы не усложнять работу, используем консольный проект. Текст заготовки заменим собственным (мной скопированный из оригинала).

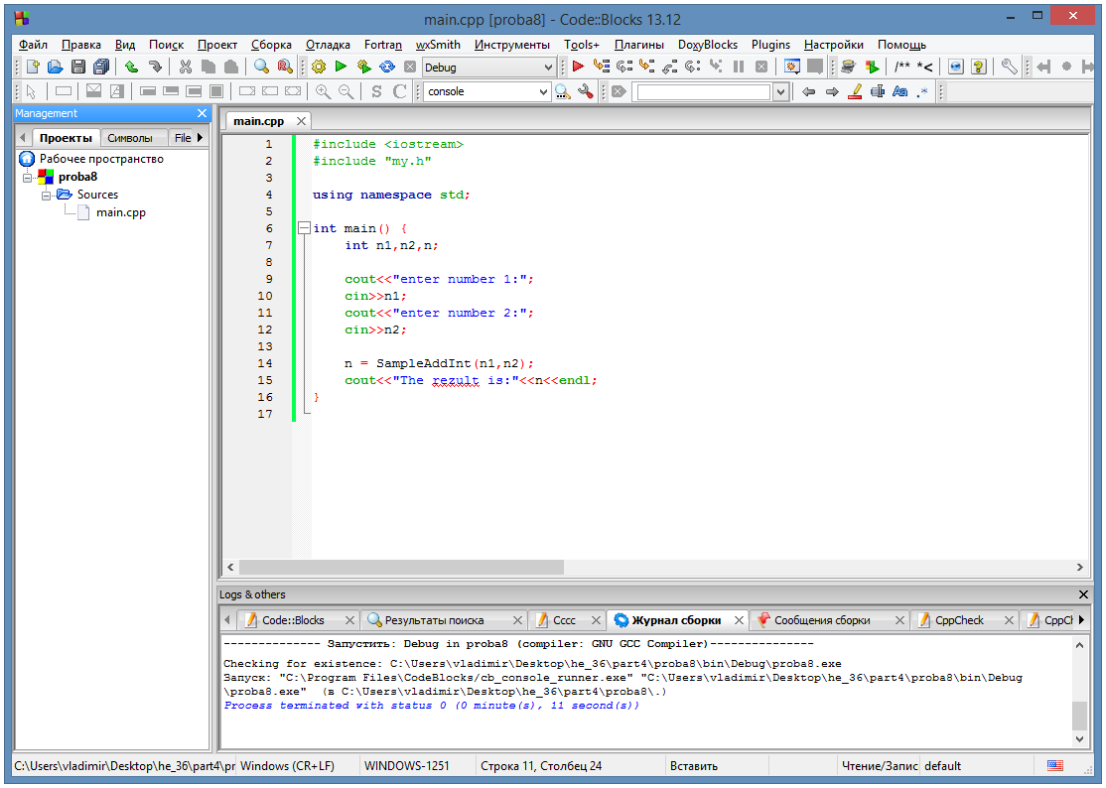

Рис. 4.12. Изменение текста программы

Для успешной компиляции и сборки необходимо указать ряд деталей сборки. В основном меню: Проект->Опции сборки…

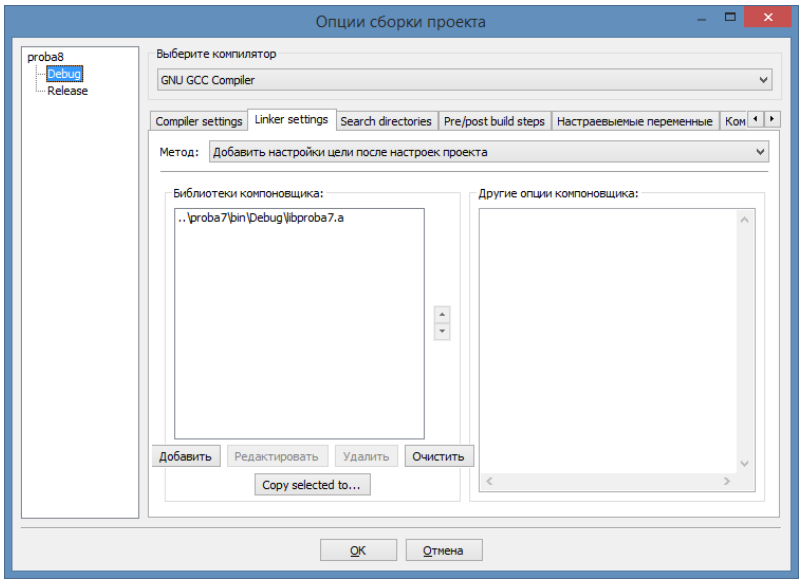

Рис. 4.13. Закладка опций сборки компоновщика для режима Debug

Путь к файлам моего предыдущего проекта proba7 должен быть относительным. Раздел Search directories (директории поиска) для компилятора и компоновщика:

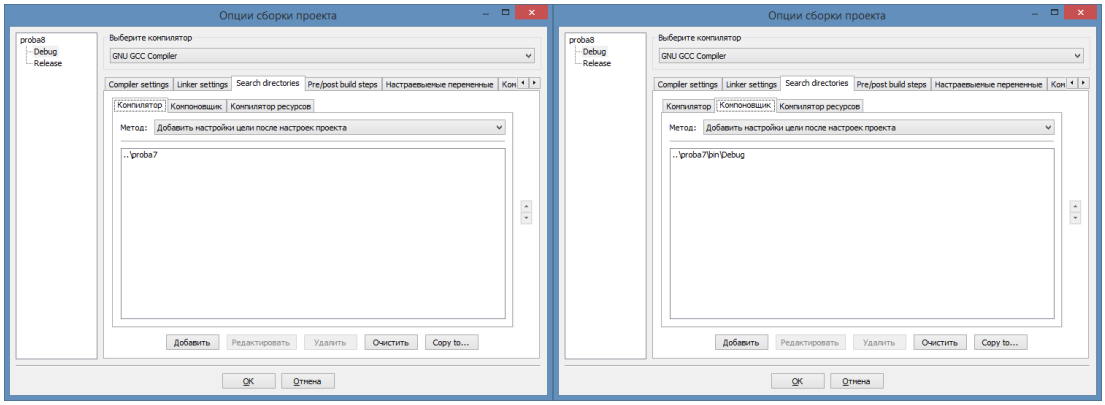

Рис. 4.14. Задание путей для компилятора и компоновщика

После компиляции и сборки можно запустить программу в режиме отладки, чтобы увидеть:

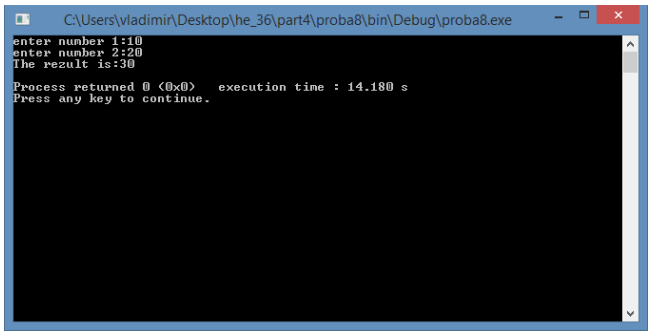

Рис. 4.15. Работа программы в консольном режиме

Программа использует библиотеку, которая была создана в предыдущем проекте. В этом вы можете убедиться, если неверно настроите консольный проект или не настроите его.

При создании исполняемого файла проекта требуется повторить оба проекта в режиме Release.

Прежде, чем перейти к следующей главе, хотелось бы упомянуть о ещё одной особенности.

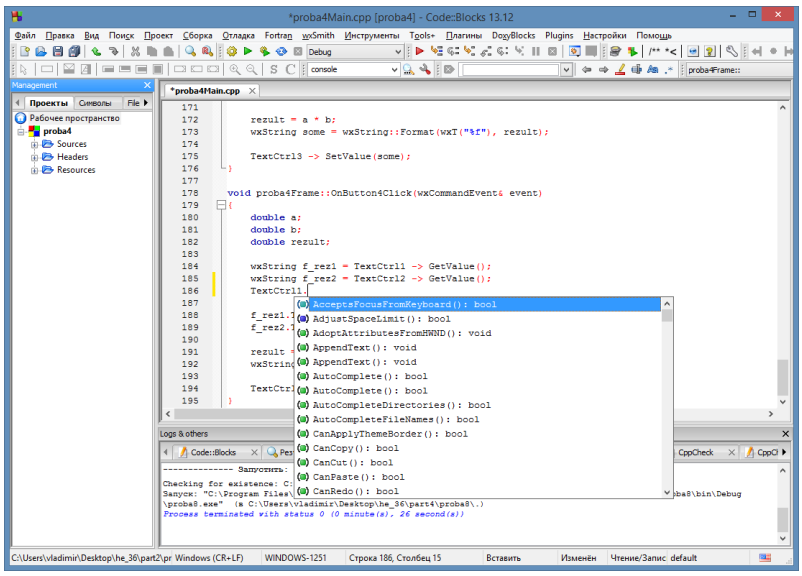

Рис. 4.16. Подсказки в текстовом редакторе проекта

Вверх и вниз по лестнице знаний

Используя wxWidgets и wxSmith, спасибо их создателям, можно создавать свои программы. Но в какой мере это приближает к пониманию языка C++?

Давайте возьмём несколько уроков по применению wxWidgets без wxSmith. Я нашёл эти уроки, с чего и начал рассказ, на сайте: http://zetcode.com/gui/wxwidgets/firstprograms/

Посмотрим, как создать самое простое окно собственными силами. Создавая wxWidgets проект, в диалоге изменим предыдущие установки:

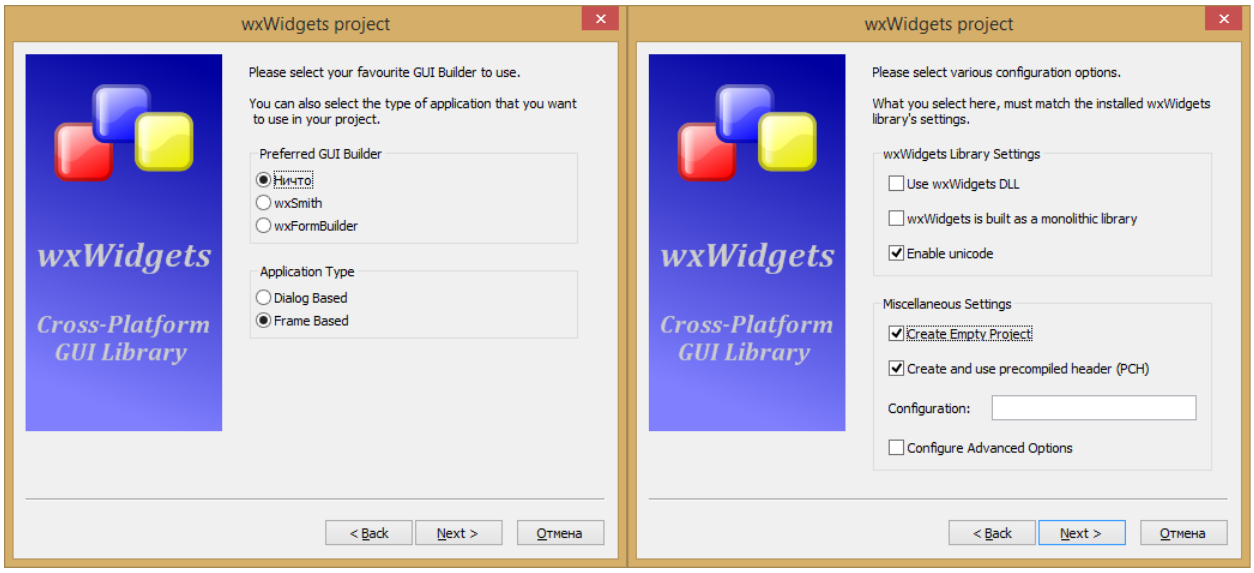

Рис. 5.1. Создание пустого проекта без wxSmith и его компонентов

Не как в прошлый раз, сейчас в окне *Проекты* нет ни одного файла, и рабочее поле Code::Blocks пусто.

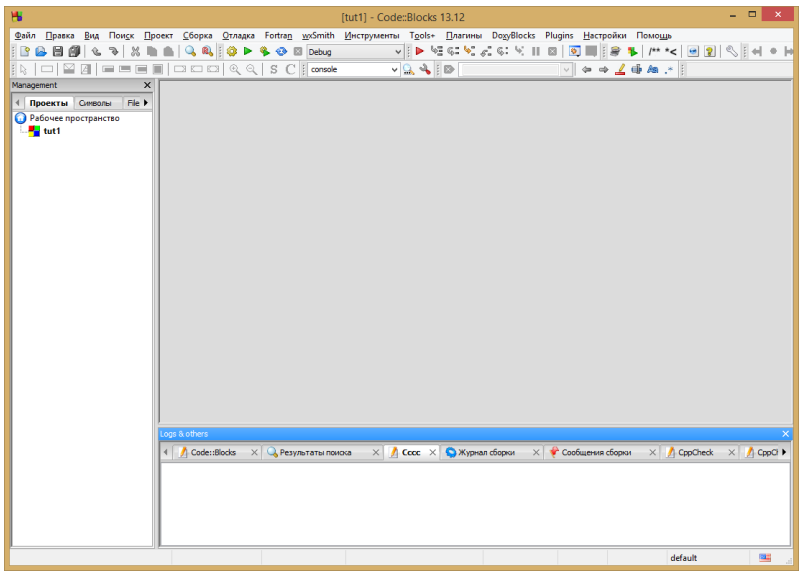

Рис. 5.2. Рабочее поле программы проекта tut1

Необходимо, следуя уроку, создать четыре файла: tut1.h, tut1.cpp, main.h и main.cpp. Для создания файлов используем Файл->New->Empty file. Первое, что предлагается сделать, это дать имя файлу. Текст первого файла tut1.h:

```
#include <wx/wx.h>
class Tut1 : public wxFrame
{
public:
    Tut1 (const wxString& title);
};
```

После записи текста файла содержимое окна Проекты изменилось:

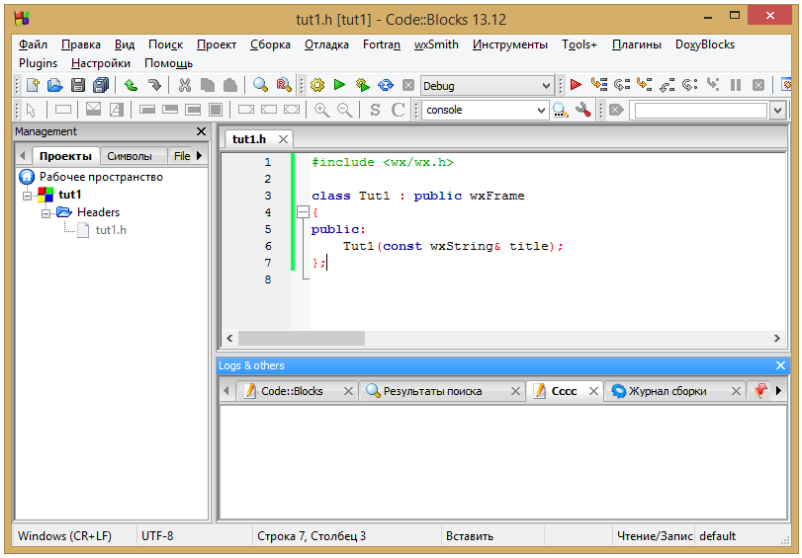

Рис. 5.3. Изменения в окне менеджера проекта

Следующий файл, который следует создать – это tut1.cpp:

```
#include "tut1.h"

Tut1::Tut1 (const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(250, 150))
{
  Centre();
}
```

Нет сомнений, что любая программа должна иметь функцию main(). Поэтому следующим будет файл main.h:

```
#include <wx/wx.h>
class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

Наконец, создадим файл main.cpp:

```
#include "main.h"
```

```
#include "tut1.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    Tut1 *tut1 = new Tut1 (wxT("Проба"));
    tut1->Show(true);

    return true;
}
```

Все вновь созданные файлы появились в окне *Проекты*. Прежде, чем включить трансляцию и сборку проекта, вспомним, что следует включить поддержку последнего стандарта (для целей и Debug, и Release):

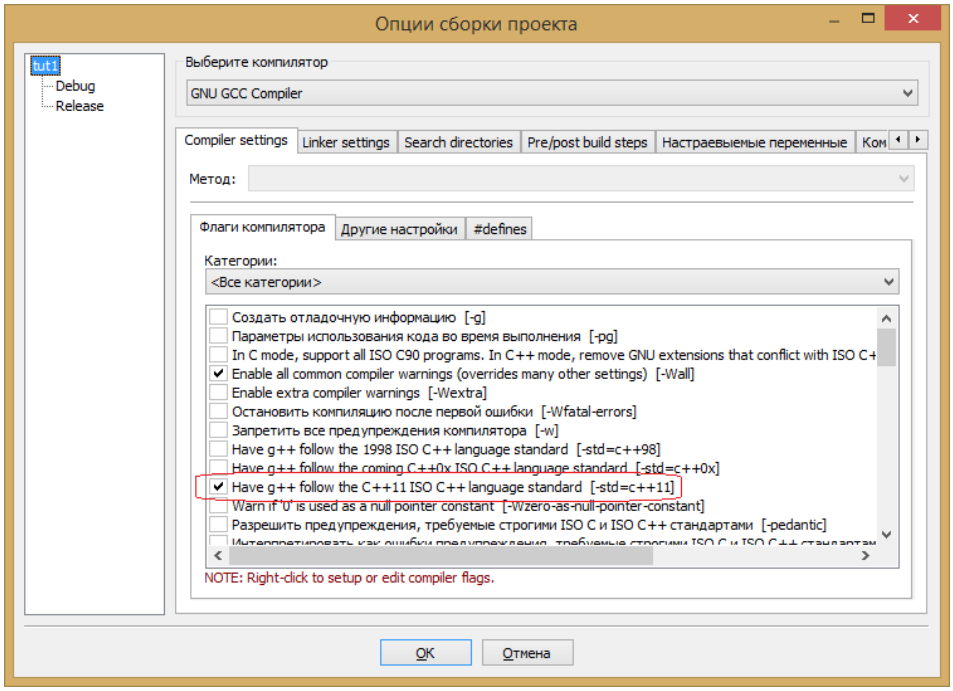

Рис. 5.4. Включение дополнительных возможностей языка C++

Выполнив команду *Сборка*, можно запустить программу.

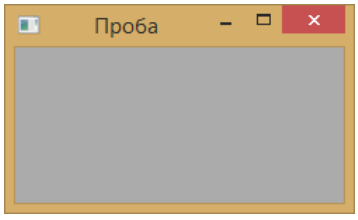

Рис. 5.5. Работа простейшей программы

Мы поднялись на одну ступеньку по лестнице освоения языка C++. Предлагаю спуститься чуть ниже, вспомнив то, что написано в рассказе о языке C++. Рассмотрим заголовочный файл tut1.h. Он объявляет новый класс Tut1 производный от класса wxFrame. Синтаксис в этом случае:

```
class имя_производного_класса: public имя_базового_класса { /*...*/ };
```

В фигурных скобках у нас объявлена общедоступная функция `Tut1(const wxString& title);`

В файле tut1.cpp реализуется то, что объявлено в заголовочном файле в виде конструктора копии с инициализацией:

```
Tut1::Tut1 (const wxString& title) : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(250, 150)) {  Centre(); }
```

Заголовочный файл основной функции main объявляет производный класс MyApp от базового класса wxAPP с виртуальной общедоступной функцией-членом OnInit();

```
class MyApp : public wxApp { public: virtual bool OnInit(); };
```

Основная функция main.cpp использует эту функцию инициализации окна. Синтаксис показан выше. В файле есть одно выражение, которое мне не совсем понятно:

```
IMPLEMENT_APP(MyApp)
```

Наведя справки, можно выяснить, что это используется в реализации класса приложения, чтобы класс приложения сделать известным для динамического конструктора wxWidgets.

Но меня интересует ещё один урок – мне хочется добавить в рабочее окно кнопку. Если научиться это делать, то можно добавлять и другие элементы без использования готовых решений wxSmith.

Итак. Начать я предполагаю с повторения урока, что называется «в лоб». Создаём новый проект с «пометкой» *Empty project*. Создаём, как написано в уроке, заголовочный файл button.h:

```
#include <wx/wx.h>

class Button : public wxFrame
{
public:
    Button(const wxString& title);

    void OnQuit(wxCommandEvent & event);
};
```

И второй файл button.cpp:

```
#include "button.h"

Button::Button(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(270, 150))
{
  wxPanel *panel = new wxPanel(this, wxID_ANY);

  wxButton *button = new wxButton(panel, wxID_EXIT, wxT("Quit"),
      wxPoint(20, 20));
  Connect(wxID_EXIT, wxEVT_COMMAND_BUTTON_CLICKED,
      wxCommandEventHandler(Button::OnQuit));
  button->SetFocus();
  Centre();
```

```
}

void Button::OnQuit(wxCommandEvent & WXUNUSED(event))
{
    Close(true);
}
```

Следующий файл main.h, похоже, не изменился с прошлого урока:

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

И последний файл main.cpp описывает всё, что будет происходить с кнопкой:

```
#include "main.h"
#include "button.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    Button *btnapp = new Button(wxT("Button"));
    btnapp->Show(true);

    return true;
}
```

Этот файл очень похож на аналогичный в первом примере. Но… вот, что появляется после трансляции программы и сборки.

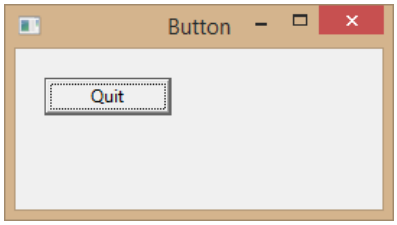

Рис. 5.6. Работа программы второго урока

Если щёлкнуть мышкой по кнопке, то окошко программы исчезает, программа закончила работу.

Автор уроков поясняет те изменения, что произошли:

```
wxPanel *panel = new wxPanel(this, wxID_ANY);
```

Вначале мы создаём wxPanel. Этот элемент будет располагаться внутри wxFrame.

Строка:

```
wxButton *button = new wxButton(panel, wxID_EXIT, wxT("Quit"), wxPoint(20, 20));
```

Мы создаём wxButton, – пишет автор. Кнопка расположится на панели. Мы используем предопределённый идентификатор wxID_EXIT для кнопки. Этим отобразится небольшая иконка выхода на кнопке. Название кнопки – это Quit. Кнопка будет позиционироваться в координатах x=20, y=20. Начало системы координат в верхнем левом углу.

```
Connect(wxID_EXIT, wxEVT_COMMAND_BUTTON_CLICKED,
    wxCommandEventHandler(Button::OnQuit));
```

Если мы щёлкнем по кнопке, генерируется событие `wxEVT_COMMAND_BUTTON_CLICKED`. Мы соединяем (connect) событие с методом `OnQuit()` класса Button. Таким образом, когда мы щёлкаем по кнопке, вызывается метод `OnQuit()`.

```
button->SetFocus();
```

*Мы задаём фокус для клавиатуры на кнопке. При этом, если мы нажмём на клавиатуре клавишу **Enter**, кнопка будет нажата.*

```
Close(true);
```

Внутри метода `OnQuit()` мы вызываем метод `Close()`. Этим мы закроем наше приложение.

Исчерпывающие пояснения. Используя их, я хочу в первой программе добавить кнопку. Мне интересно, получится ли что-то у меня. Я создаю новый пустой проект и повторяю всё, что было описано для первого урока. Поскольку я копирую тексты из этой главы (непосредственно с листа), я вношу изменения в имена, сопоставляя их с новым именем проекта tut3. Убедившись, что программа транслируется и запускается, я приступаю к изменениям программы.

Я уверен, что заголовочный файл для кнопки я могу повторить без изменения. Или нет…

Попытка совместить оба проекта полностью, повторяя оба урока, не слишком умное решение. Хотя программа транслируется и работает, но даёт два рабочих окна «имени первого и второго проектов». Копирование без понимания всегда лишено смысла. Достаточно сравнить файлы обоих уроков, чтобы понять – оба примера создают свои окна, которые производятся от класса wxFrame. Удалив из второго урока всё, что относится к реакции на нажатие кнопки, можно яснее увидеть то общее, что есть в обоих случаях. Постепенно становится понятно, что нет необходимости использовать что-то кроме одного выражения, которое я добавляю в файл (первого урока) tut3.cpp:

```
#include "tut3.h"

Tut3::Tut3(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(250, 150))
{
    wxButton *button = new wxButton(this, wxID_ANY, wxT("Button"));
  Centre();
}
```

Обращаясь к руководству по языку C++, можно выяснить, что:

Динамическое выделение памяти осуществляется с помощью оператора new. Оператор возвращает указатель на начало нового блока памяти. Синтаксис такой:

```
pointer = new type
```

После сборки проекта появляется следующее окно:

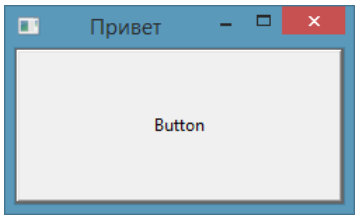

Рис. 5.7. Окно первого урока с кнопкой

Вспоминая свой проект с использованием wxSmith, я добавляю ещё одну кнопку. И получаю:

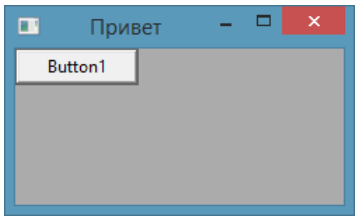

Рис. 5.8. Окно программы с двумя кнопками

И где вторая кнопка? Эта проблема решается с помощью параметра, который я не использовал:

```
wxPoint(20, 20)
```

Теперь для каждой кнопки можно указать координаты расположения с помощью этого параметра:

```
wxButton *button1 = new wxButton(this, wxID_ANY, wxT("Button1"), wxPoint(20, 20));
wxButton *button2 = new wxButton(this, wxID_ANY, wxT("Button2"), wxPoint(20, 50));
```

Итог этой части работы:

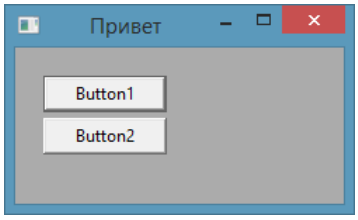

Рис. 5.9. Окно программы после использования параметров координат

Я добился решения поставленной задачи, но при этом меня заинтересовала проблема двух кнопок, которая возникла ещё при использовании сборки с помощью wxSmith графического помощника. Если я добавлял одну кнопку, то она занимала весь экран. Тот же эффект возник и без wxSmith. Но во втором уроке, напомню, автор уроков предупреждал:

Мы создаём wxButton, – пишет автор. Кнопка расположится на панели.

Мне интересно, если я добавлю панель при работе с wxSmith, смогу ли я получить одну кнопку?

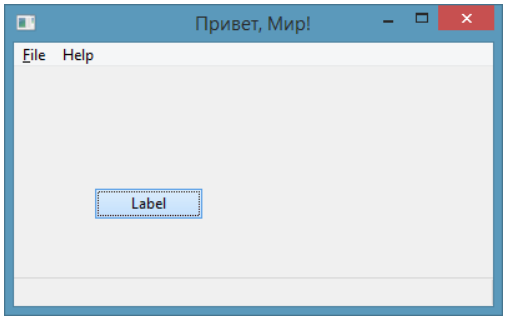

Рис. 5.10. Одна кнопка на панели с использованием wxSmith

Хотя автор уроков только мельком отметил тот факт, что все элементы добавляются на панели, понять это помог придуманный мною третий урок.

Кстати, выполняя его, я встретил ещё одну особенность – попытка изменить название кнопки, написав его как «Кнопка», оказалась неудачна. Программа не смогла транслироваться. Причины этого мне пока не понятны, но я надеюсь, что разберусь и с этим. Позже.

Уроки и современные технологии

Любой инженер, начиная создание чего-то собственного, обращается к уже существующим решениям. Это правильный подход. Но не следует забывать о годах учёбы инженера.

Повторяя уроки в предыдущей главе, я использую технологию копирования из одного окна (Интернет-браузера) в другое (окно текстового редактора Code::Blocks). Это позволяет быстро получить результат. Но будет ли эффект долговременным? Я хочу проверить это, повторив первый урок без копирования готовых текстов.

Создавая пустой проект, я вспоминаю, что необходимо добавить поддержку последнего стандарта языка C++. И первый файл – это заголовочный файл, который я назвал tutor5.h:

```
#include <wx/wx.h>

class Tutor5::Tutor5(const wxString &name): wxFrame(wxWindow, wxID_ANY,
label) { Centre(); };
```

В заголовочном файле, как мне запомнилось из руководства по языку и из уроков, достаточно объявить новый класс и инициализировать его с помощью конструктора (я пока не уверен в правильности этой фразы, но за подсказкой не хочу обращаться).

Следующий файл – это tutor5.cpp, где следует реализовать новый класс:

```
#include <wx/wx.h>
#include “tutor5.h”
Tutor5 = new Tutor5 * tutor(wxT("Привет"));
```

Это всё, что я вспомнил. Далее следует создать основной файл main, начиная с заголовочного файла. В этот момент я понимаю, что ничего вспомнить не могу. Придётся обратиться к подсказке – взглянуть на правильные файлы, чтобы по памяти воспроизвести их.

Для первого заголовочного файла, конечно, следует объявить приложение, создав новый класс:

```
#include <wx/wx.h>

class myAPP : public wxApp
{
public:
    myApp onInit();
};
```

И вот, что мне запомнилось для файла main.cpp:

```
#include <wx/wx.h>
#include "main.h"

myApp::OnInit
{
    Tutor5 = new Tutor * tutor(wxT("Привет"));
    tutor -> Show();

    return tutor;
}
```

После запуска сборки появляются сообщения об ошибках:

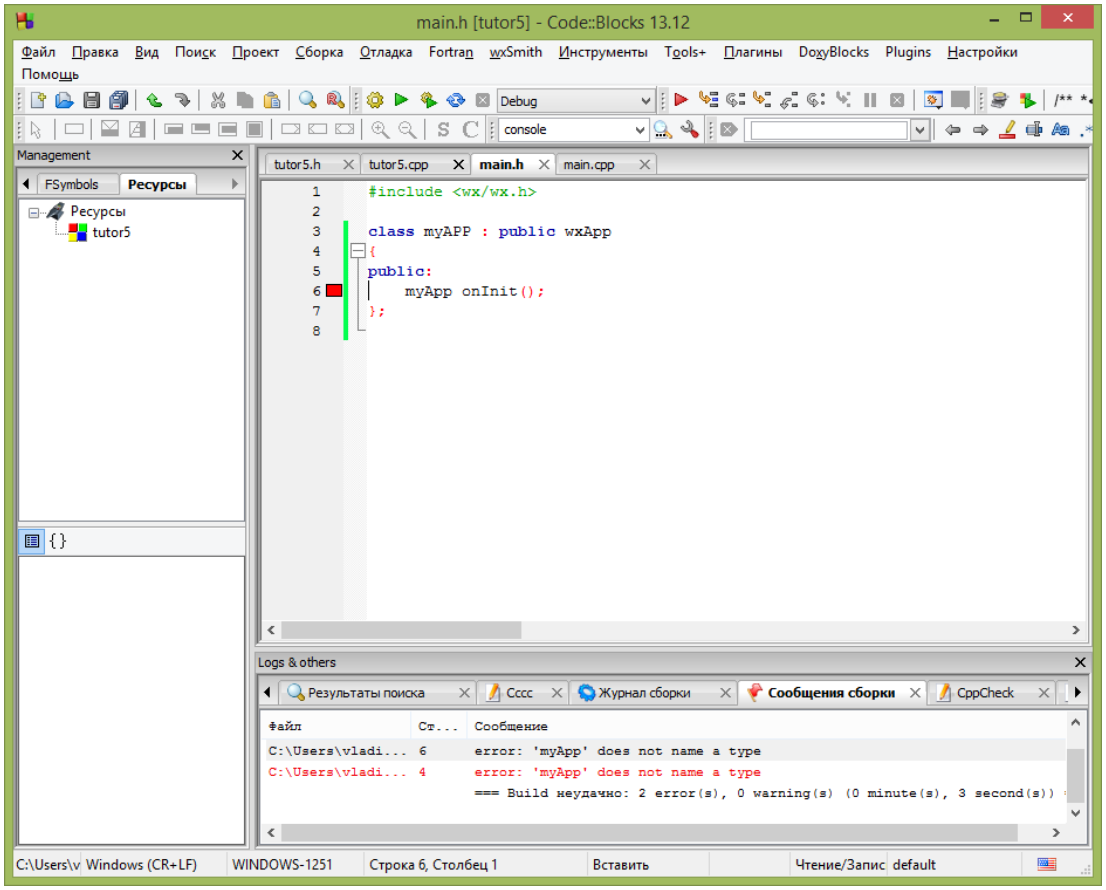

Рис. 6.1. Первый запуск программы на сборку

Заглянем в справочник (это нормальная практика) по классу wxApp:

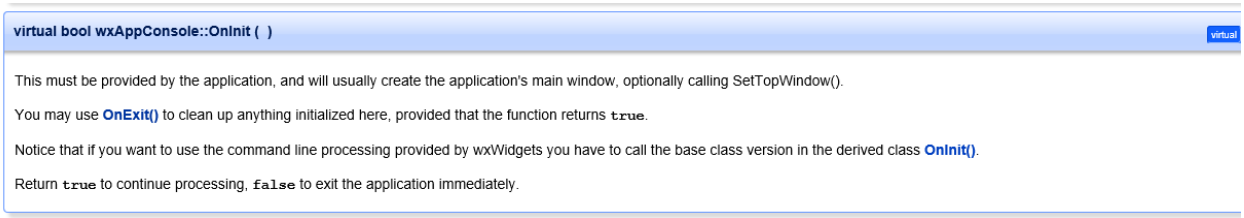

Рис. 6.2. Справка по классу wxApp и функции-члену OnInit()

Попробуем привести «ошибку» к должному виду в файле main.h.

```
#include <wx/wx.h>

class myAPP : public wxApp
{
public:
        virtual bool OnInit();
};
```

Следующая ошибка появляется в файле main.cpp:

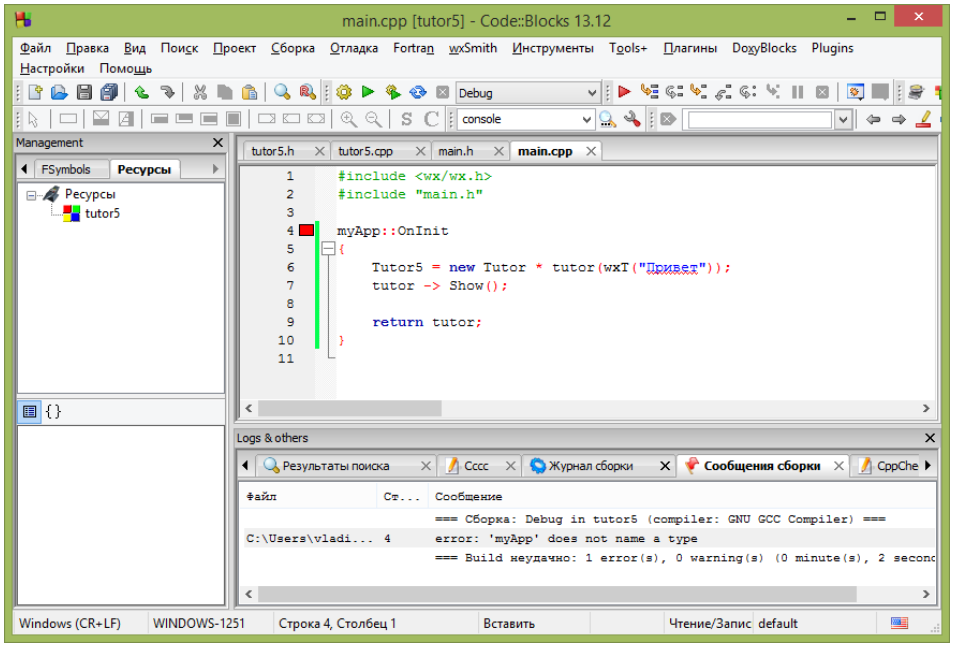

Рис. 6.3. Следующая ошибка в файлах проекта

Первый опыт показывает, что копировать текст программы гораздо легче, чем писать самостоятельно. Повторим этот опыт, но теперь начнём с правильной стороны, с «главной функции»: мы собираемся создать новое приложение, стало быть, начинать надо с создание нового класса приложения. От нового класса требуется, чтобы при запуске приложения можно было создать окно.

main.h

```
#include <wx/wx.h>
class MyApp : public wxApp
{
public: virtual bool OnInit();
};
```

Теперь обратимся к основному файлу.

main.cpp

```
#include “main.h”
IMPLEMENT_APP(MyApp)
bool MyApp:: OnInit()
{
return true;
};
```

Подглядеть пришлось, чтобы добавить IMPLEMENT_APP(MyApp).

Теперь необходимо создать новый класс для окна в файле заголовка. Но прежде оттранслируем то, что есть – достаточно пропустить одну букву в ключевом слове или забыть поставить скобку, чтобы появились проблемы.

tutor6.h

```
#include <wx/wx.h>
class Tutor6 : public wxFrame
{
public: Tutor6 (const wxString &name);
} ;
```

Если не пытаться вспоминать, что было когда-то сделано, то можно повторить уже написанный заголовочный файл, подправив идентификаторы и ту функцию-член, которая нужна. Осталось написать файл cpp к этому проекту… И вновь, как и ранее, я пытаюсь вспомнить, а не понять, что нужно делать… Видимо, я жертва современных технологий, мне проще скопировать:

```
#include "tutor6.h"

Tutor6::Tutor6(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(270,
150))
{
  Centre();
}
```

Добавив в файл main две строки:

```
Tutor6 *tutor = new Tutor6 (wxT("Tutor"));
tutor -> Show();
```

Последняя ошибка – не добавлено в основной файл #include “tutor6.h”. Итог:

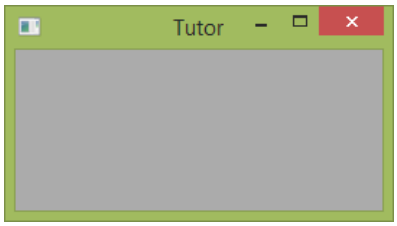

Рис. 6.4. Окно программы, полученное ручным написанием текста программы

Но главный итог, главный вывод, который я могу сделать – нужно повторять собственное написание простых программ много раз, стараясь не подглядывать, не копировать, повторять до тех пор, пока не станет ясно, что нужно писать во всех файлах.

Чтобы следовать этому совету, я добавляю в файл tutor.cpp (забыл добавить 6, когда создавал файл) две строки, создающие кнопку:

```
wxPanel *panel = new wxPanel(this, wxID_ANY);
wxButton *button = new wxButton (panel, wxID_ANY, wxT("button"), wxPoint(20,20));
```

Частью это происходит благодаря воспоминаниям, частью от понимания, а частью с помощью банального подглядывания. Меня интересует названия по-русски. Я меняю надпись в строке, создающей окно, где приходится внести правку:

```
Tutor6 *tutor = new Tutor6 (“Проба”);
```

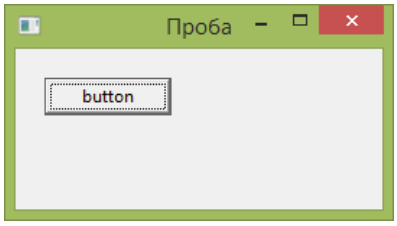

Рис. 6.5. Окно с кнопкой

В прошлый раз при замене этикетки на кнопке кириллической надписью я получал ошибки. Но не в этот раз. В чём разница… может быть, когда-нибудь и это будет понятно.

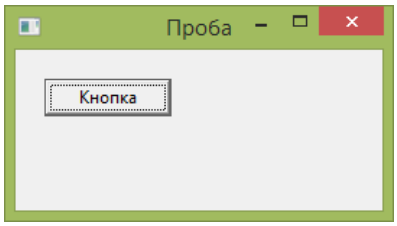

Рис. 6.6. Изменение названия кнопки

Повторение удавшегося приёма настраивает на оптимистический лад. Можно повторить строку, подправив её и добавив к уже работающим выражениям. Этим мы добавим текстовое окно. То есть:

```
        wxPanel *panel = new wxPanel(this, wxID_ANY);
        wxTextCtrl *control = new wxTextCtrl(panel, wxID_ANY);
        wxButton *button = new wxButton (panel, wxID_ANY, wxT("Кнопка"),
wxPoint(20,50));
```

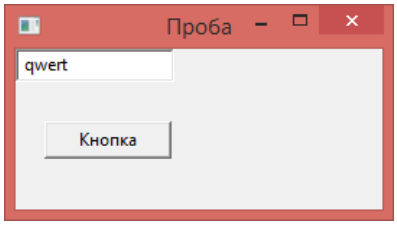

Рис. 6.7. Новая работа программы

Казалось бы, что ещё нужно? Однако попытка переместить текстовое поле, используя wxPoint(20,20), оказывается неудачна.

Что означает, что уроки выучены плохо, а отсутствие знаний не покрывается современными технологиями: готовые библиотеки wxWidgets, прекрасный текстовый редактор Code::Blocks с подсказками, хороший компилятор, сообщающий об ошибках, и возможность найти в Интернете решение своих проблем. Уроки следует выполнять хорошо.

Что задали на дом?

На дом задали разобраться, почему не получилось сместить текстовое поле?

Я не скажу, с какого раза я смог, не подглядывая и не копируя, создать новое приложение без остальных элементов, но попробую вспомнить, что мне мешало это сделать сразу. Создаём новый проект:

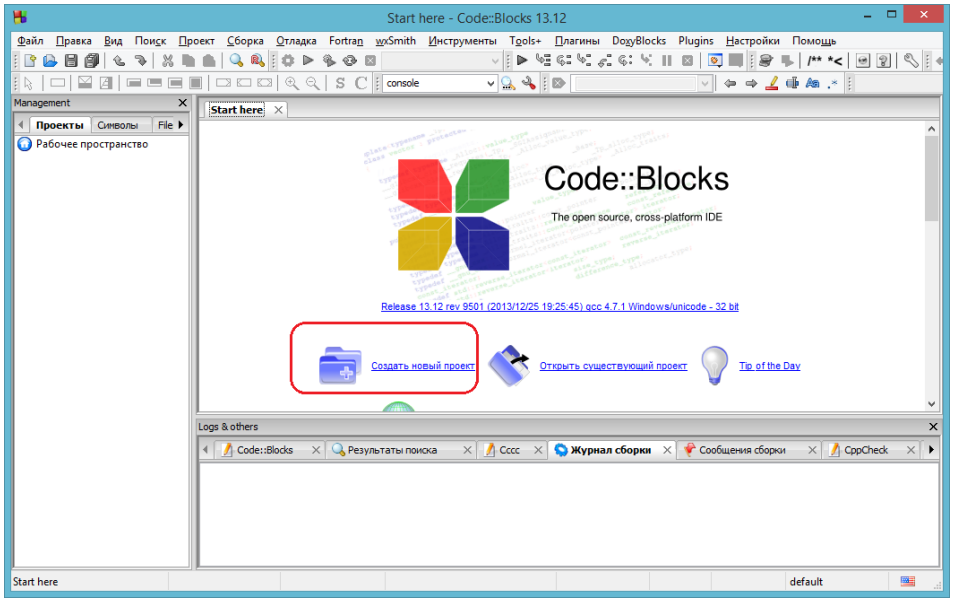

Рис. 7.1. Создание нового проекта

Выбираем *wxWidgets project* и нажимаем кнопку **Go**:

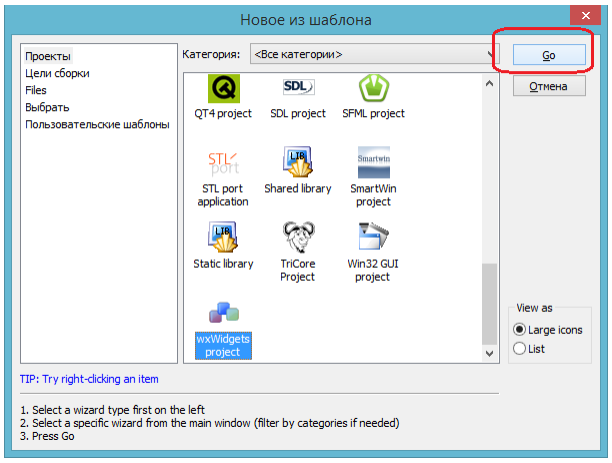

Рис. 7.2. Выбор типа нового проекта

В следующем диалоговом окне я ничего не меняю, нажимаю кнопку **Next>**. Затем выбираю ту версию wxWidgets, что установлена у меня, то есть, wxWidgets 3.0.x и нажимаю кнопку **Next>**.

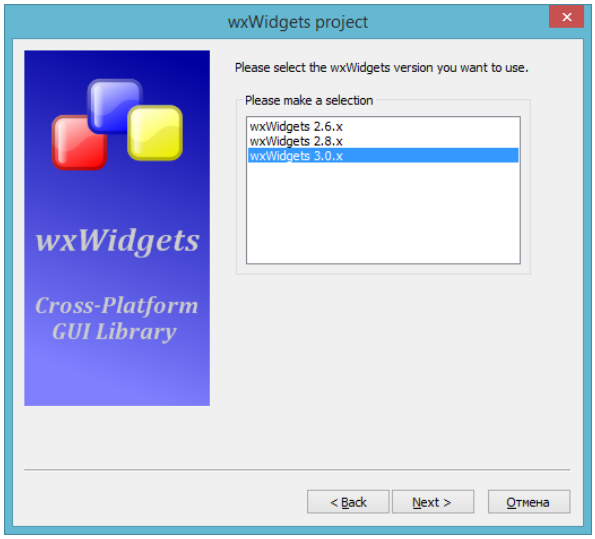

Рис. 7.3. Выбор версии wxWidgets

Далее следует указать имя проекта, используя отмеченную на рисунке кнопку, можно (и нужно, наверное) указать папку, в которой будет храниться проект.

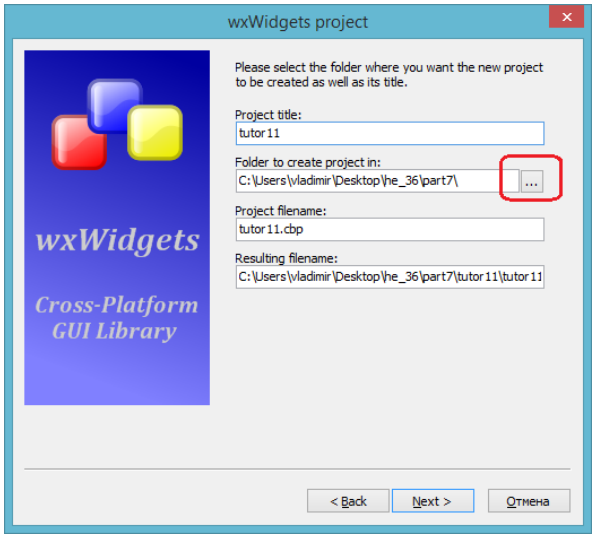

Рис. 7.4. Указание имени и места расположения проекта

Следующее диалоговое окно предлагает внести все авторские данные, это, если нужно, а можно только имя, не исключаю, что можно и ничего не добавлять. Затем следует выбор метода построения проекта. В данном случае я хочу самостоятельно разобраться со всеми проблемами:

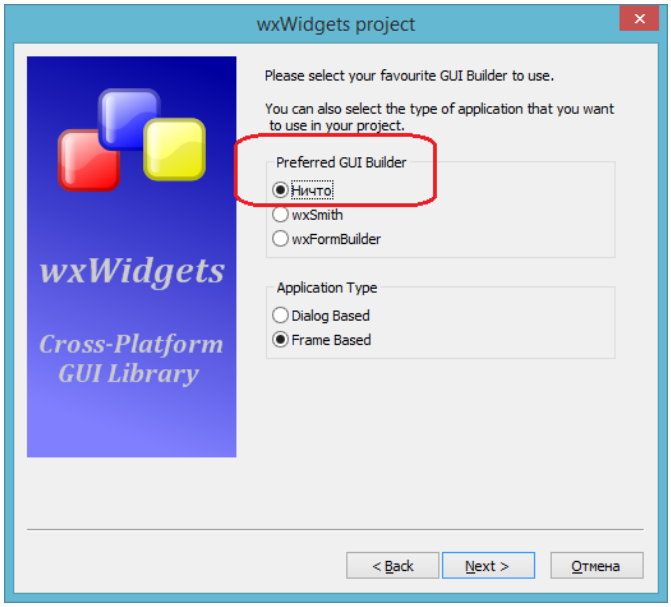

Рис. 7.5. Выбор метода сборки проекта

Далее следует указать место расположения wxWidgets – при первом запуске Code::Blocks, в следующий раз можно пользоваться кнопкой **Next>**, равно как и в других диалогах.

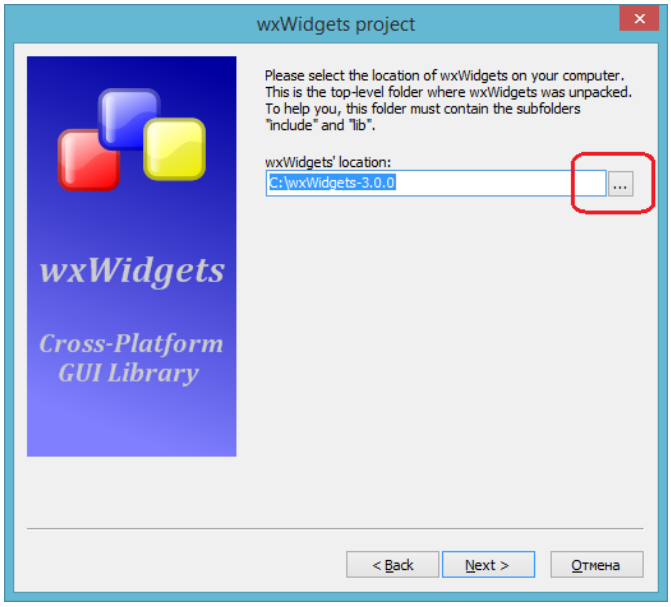

Рис. 7.6. Выбор места, где находится wxWIdgets

Следующий диалог я оставил без изменений, а в предпоследнем указал, что создаётся пустой проект.

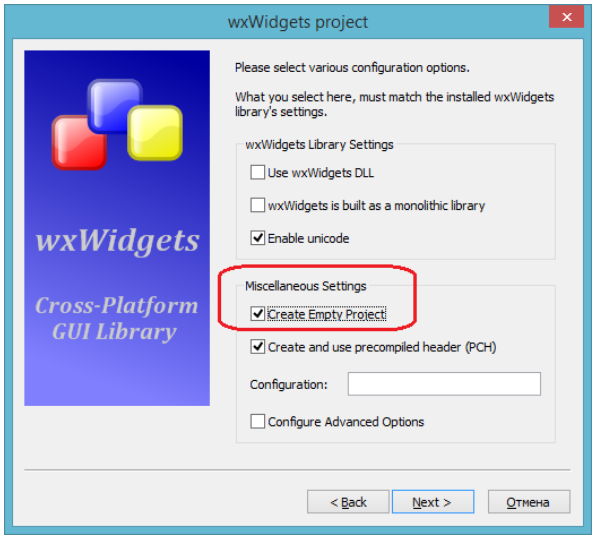

Рис. 7.7. Указание на создание пустого проекта

В появившемся сообщении есть возможность согласится, нажав кнопку **ОК**, что я делаю, а по причине пустого проекта в следующем диалоговом окне можно ничего не выделять, нажав кнопку **Finish**.

Прежде, чем создавать новые файлы, я хочу отметить, что проект может использовать современную версию компилятора: Проект->Опции сборки:

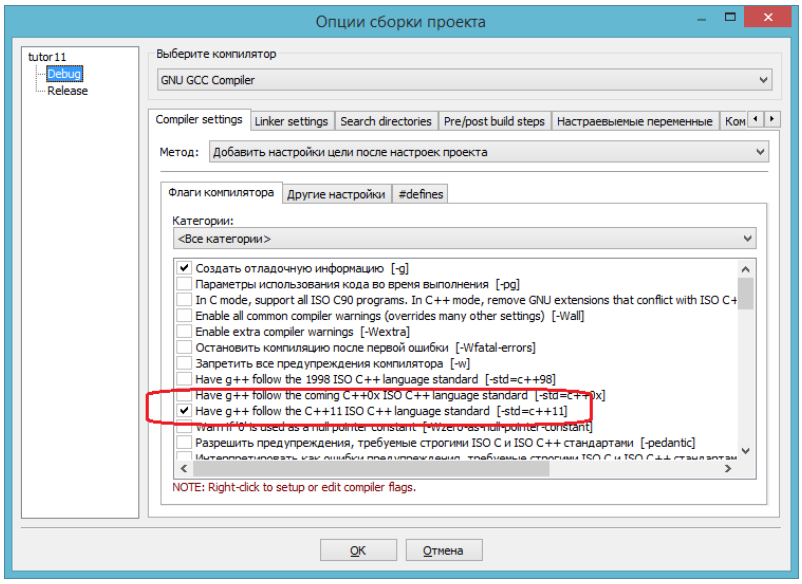

Рис. 7.8. Отметка о применении последнего стандарта в Debug и Release режимах

Теперь следует создать все необходимые файлы. Первым создадим заголовочный файл основной функции программы main. В нём объявляем создание собственного класса, производного от класса приложений: Файл->New->Empty file. Добавляем имя заголовочного файла и соглашаемся, что используем его в обоих режимах и отладки, и окончательной сборки программы.

В появившемся в редакторе поле начинаем с включения нужного заголовка (это я запомнил), стараясь не переврать слово #include <wx/wx.h>. Впрочем, достаточно начать со значка решётки, чтобы из выпадающего списка выбрать нужное:

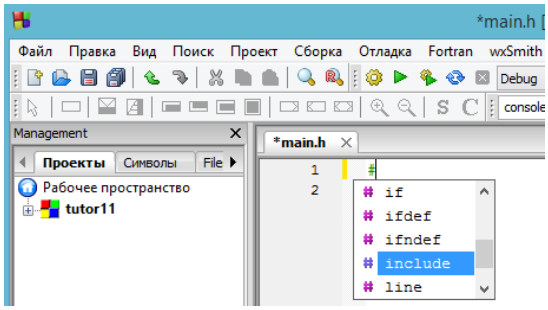

Рис. 7.9. Подсказка по выбору того, с чего следует начать

Итак, я создаю свой класс на базе wxApp, который я называю MyApp. Из основного класса мне нужна одна функция OnInit(), позволяющая при инициализации приложения увидеть окно, которое ещё предстоит создать.

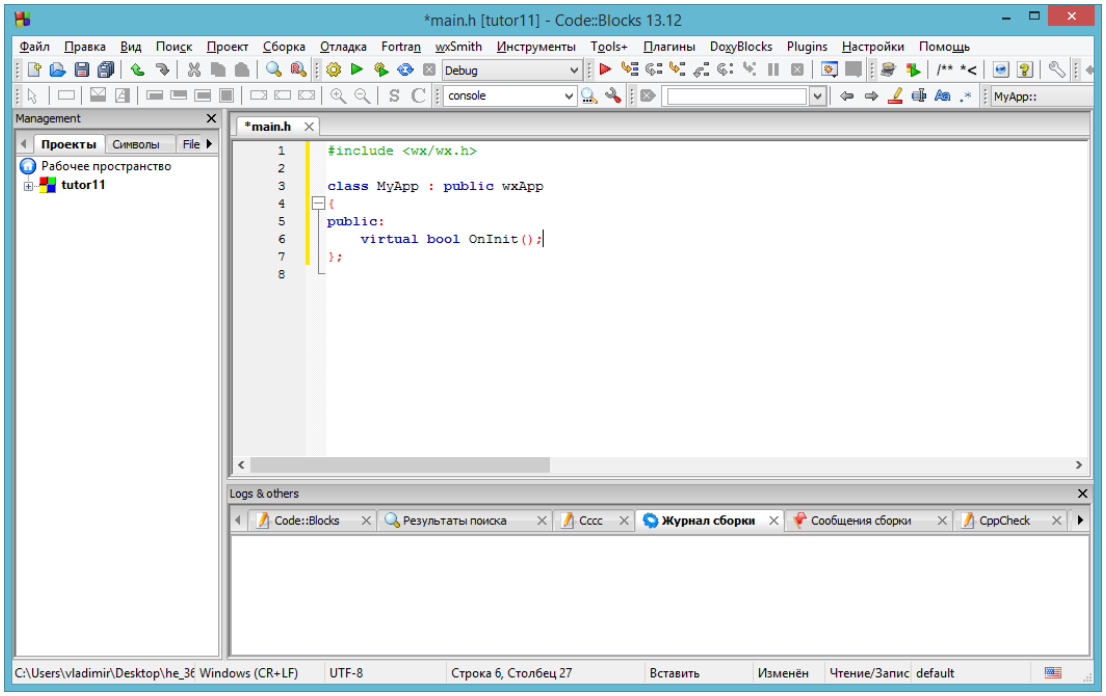

Рис. 7.10. Создание заголовочного файла основной функции проекта

Что здесь у меня вызывало проблемы – в первую очередь то, что я забывал, что это создание нового приложения, то есть, производного класса от wxApp. Затем, что это общедоступный класс. И тот факт, что мне нужна общедоступная (public) виртуальная функция, возвращающая булево значение.

Аналогично созданию заголовочного файла создаём основной рабочий файл main.cpp. В который не забываем включить ранее созданный заголовочный файл, поскольку нам нужно теперь уже не объявлять наше приложение, а реализовать его. Поскольку файл в папке проекта, его имя заключаем в кавычки.

Следующее, что я всегда забываю, таинственная фраза о реализации, но здесь тоже прекрасно работает подсказка:

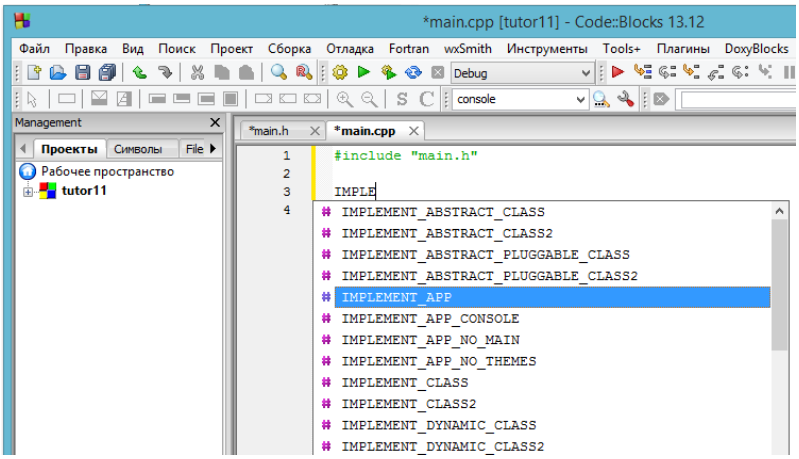

Рис. 7.11. Подсказка после ввода нескольких букв

В скобках указываем свой класс. Теперь можно вызвать функцию инициализации, указав границы применения. А проверить, что всё сделано правильно, можно запустив сборку проекта:

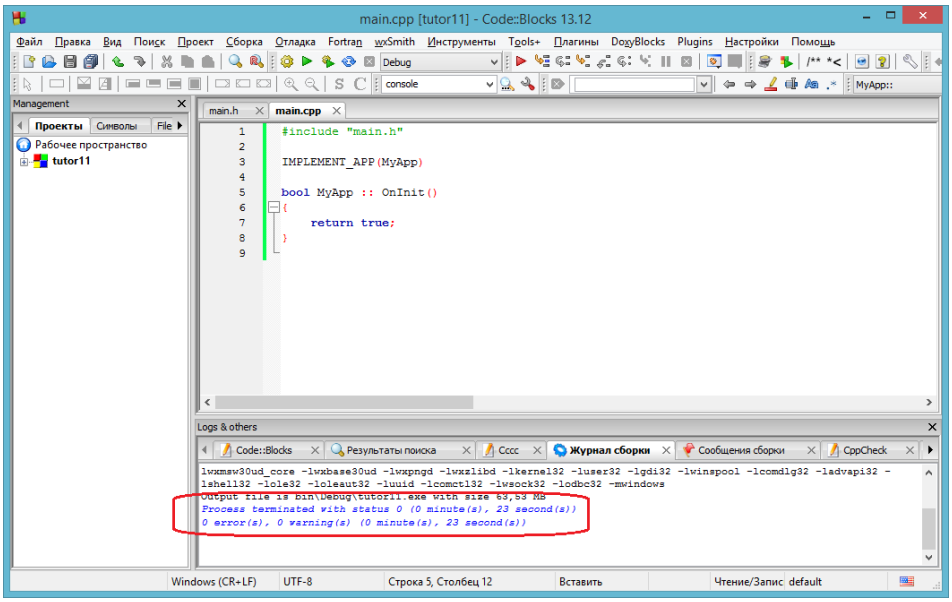

Рис. 7.12. Предварительная проверка правильности создания приложения

Теперь приступаем к созданию окна. Начинаем с создания заголовочного файла, где объявим новый класс (я использую имя проекта).

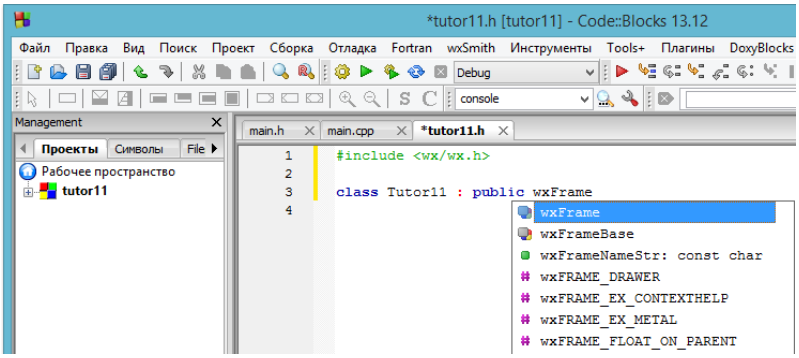

Рис. 7.13. Подсказка при выборе базового класса

В файле заголовка я тоже в прошлый раз забыл указать public, забыл, что мне нужна функция вставки заголовка.

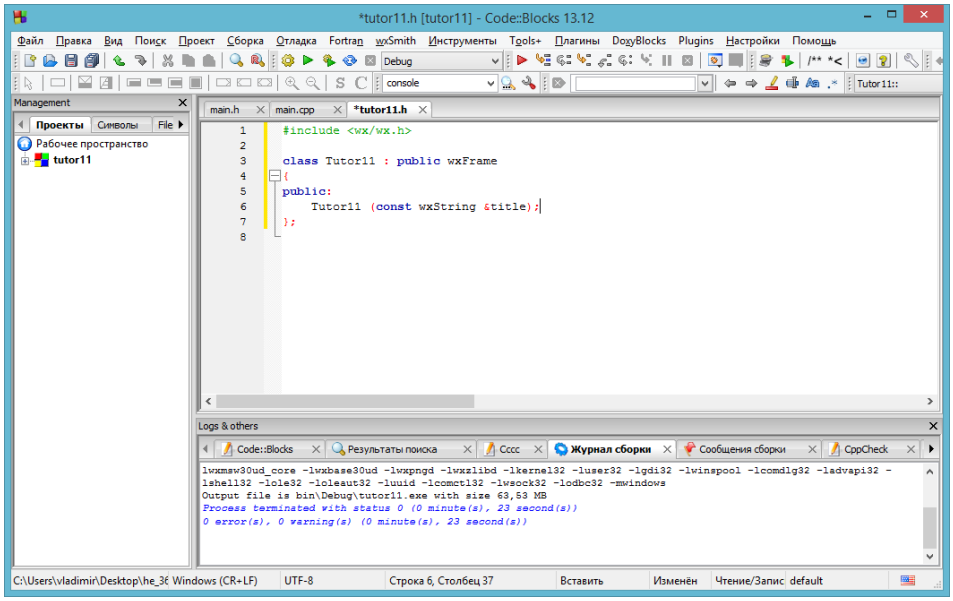

Рис. 7.14. Создания файла tutor11.h

И переходим к созданию файла tutor.cpp, начиная с включения файла заголовка. При создании окна его устанавливаем в центр.

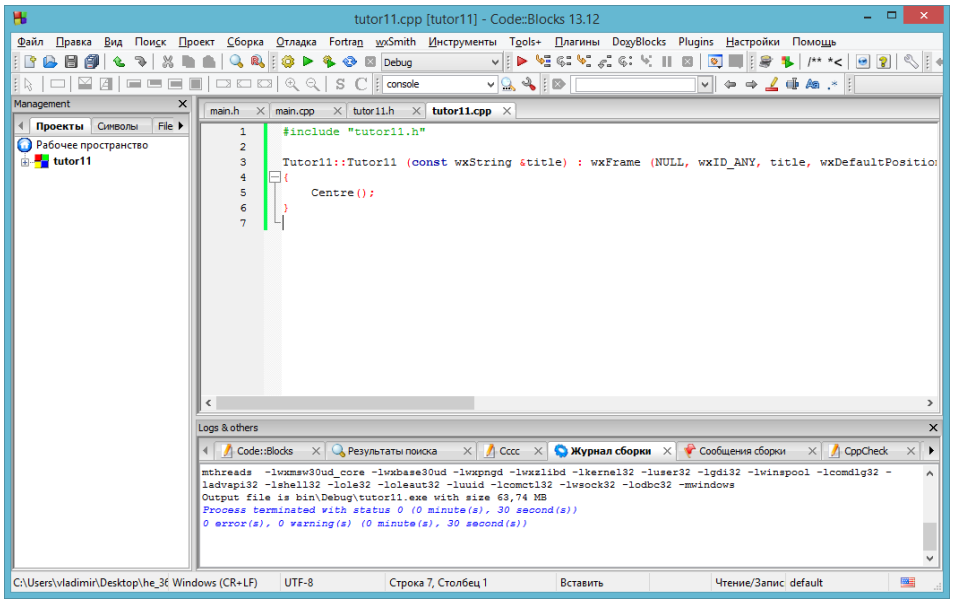

Рис. 7.15. Создание файла tutor11.cpp

В этом месте я, признаться, опять забуксовал, пришлось подсматривать и реализацию, и параметры. А при попытке транслировать текст возникла ещё одна проблема – исполняемый файл режима *Debug* оказался недоступен, отказано мне было в доступе (замучил я, наверное, программу). Попытка удалить файл .exe из папки *Debug* тоже не удалась, отказано. Пришлось перезагрузить компьютер, удалить этот файл, собрать проект заново. Чтобы увидеть результат своих трудов, мне осталось добавить два выражения в основной файл:

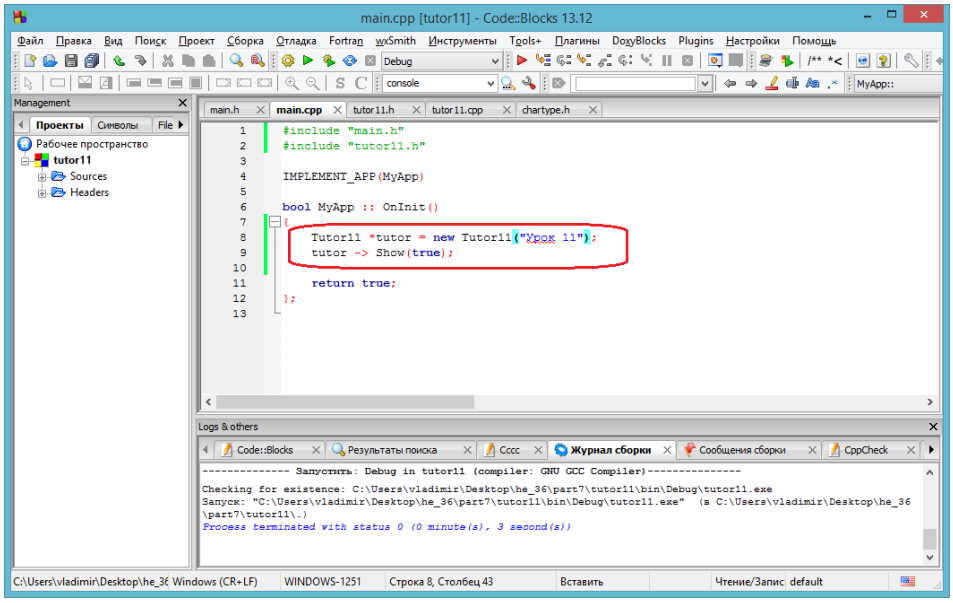

Рис. 7.16. Добавление в файл main.cpp

Хотя я сделал эти вставки по памяти, не обошлось тоже без ошибок: забыл вставить заголовочный файл своего класса Tutor11, при названии латиницей Tutor11(wxT("Lesson 11")), а для кириллицы потребовалась та запись, которая есть на рисунке. Осталось добавить в файл tutor11.cpp одно выражения для создания панели:

```
wxPanel *panel = new wxPanel (this, wxID_ANY);
```

Чтобы, наконец, перейти к тому, чего ради я всё это затеял. И не думайте, что всё выше написанное и показанное в картинках для вас, нет, это я сделал для себя, чтобы запомнить. И вам советую так делать уроки, записывая, что вы делаете. Лучше запоминается, проще разобраться, что плохо усвоено. Впрочем, у каждого свой стиль работы. Итак.

Попытка добавить в файл tutor11.cpp фразу:

```
wxTextCtrl *text = new wxTextCtrl (panel, wxID_ANY);
```

Даёт появление текстового поля на панели:

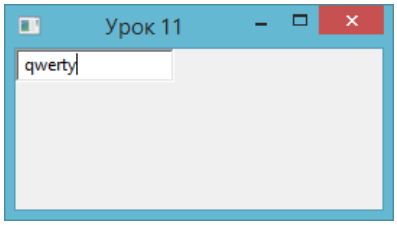

Рис. 7.17. Рабочее окно с текстовым полем

А добавление параметра wxPoint(20,20) приводит к ошибке.

Эта загадка решилась просто (хотя я и не понял, почему так): добавление в строку выше кроме указания места расположения ещё и пустого имени даёт искомый результат. А можно ещё, как оказалось, добавить и размер:

```
wxTextCtrl *text = new wxTextCtrl (panel, wxID_ANYб wxT(“”), wxPoint(20,20),
wxSize(80,20));
```

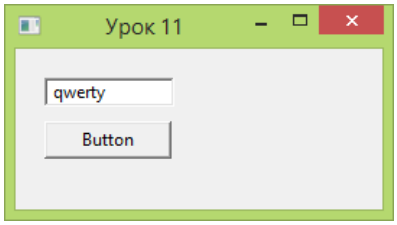

Рис. 7.18. Совместное размещение кнопки и текстового поля

Разыгравшийся аппетит (сознаюсь, что увидел это, пока разбирался с текстовым полем) заставляет меня…

Добавить в мой файл tutor11.cpp несколько строк:

```
wxMenuBar *menubar = new wxMenuBar;
wxMenu *file = new wxMenu;
wxMenu *help = new wxMenu;
menubar -> Append(file, "Файл");
menubar -> Append(help, "Помощь");
SetMenuBar(menubar);
```

И это даёт:

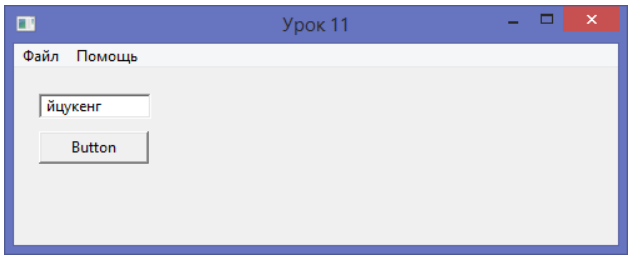

Рис. 7.19. Добавление основного меню

События в мире кнопки

Создание рабочего окна, меню, элементов рабочего поля окна важно и не так просто, как это могло показаться при использовании внешнего компоновщика интерфейса. Но оно бесполезно, если нельзя использовать элементы рабочего окна для обработки каких-то событий. Если есть кнопка, то её хочется нажать, чтобы…

Попробуем сделать так, чтобы при нажатии кнопки очищалось текстовое поле. Руководствуясь уроками, я для начала хочу внести в программу изменение, чтобы название кнопки при её нажатии изменилось. Не пытаясь написать что-то самостоятельно, я просто копирую в свой файл tut12.h дополнительно несколько строк:

```
#include <wx/wx.h>
class Tut12 : public wxFrame
{
public:
    Tut12(const wxString &title);
    void OnButtonClick(wxCommandEvent & event);
private:
    DECLARE_EVENT_TABLE()    };
```

И в файл tut12.cpp копирую:

```
void Tut12::OnButtonClick(wxCommandEvent& WXUNUSED(event))
{
      SetLabel(wxT("OK"));
}
BEGIN_EVENT_TABLE(Tut12, wxFrame)
      EVT_BUTTON(wxID_ANY,  Tut12::OnButtonClick)
END_EVENT_TABLE()
```

Конечно, я изменил, приведя в соответствие со своей «маркировкой» ряд названий. Кроме того, в классе wxButton подсмотрел выражение, касающееся задания этикетки. Почему я скопировал текст из уроков? Потому что достаточно одной опечатки, чтобы ничто не работало, чтобы появились ошибки, которые, возможно, понятны опытному программисту, но мне. Результат, который я получил… Впрочем, смотрите сами:

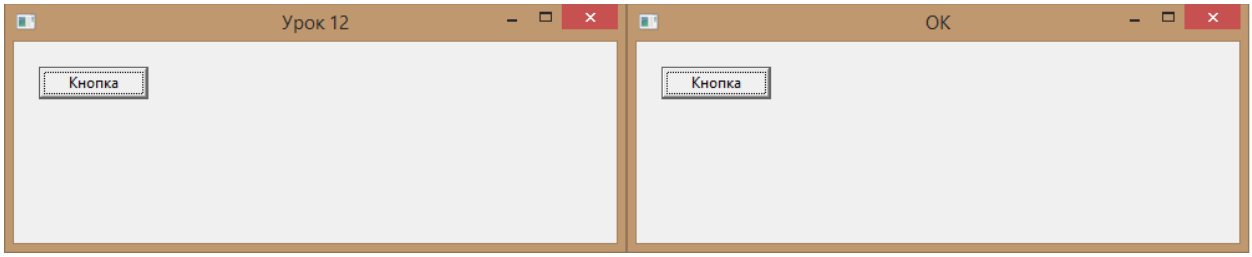

Рис. 8.1. Первый результат по использованию событий

Получается, что SetLabel(wxT("OK")) относится не к кнопке, а к окну. Попытка изменить эту строку на:

```
button -> SetLabel(wxT("OK"));
```

…приводит к появлению ошибки, связанной с тем, что моя кнопка либо не находится в границах моего класса, либо не является членом семейства wxButton. Проблема решилась тем, что в файле tut12.h я (следуя урокам) добавил: wxButton *button; В файле tut12.cpp изменил создание кнопки:

```
button = new wxButton (panel, wxID_ANY, wxT("Кнопка"), wxPoint(20,20));
```

И использовал изменение этикетки, как показано выше. Теперь всё получилось:

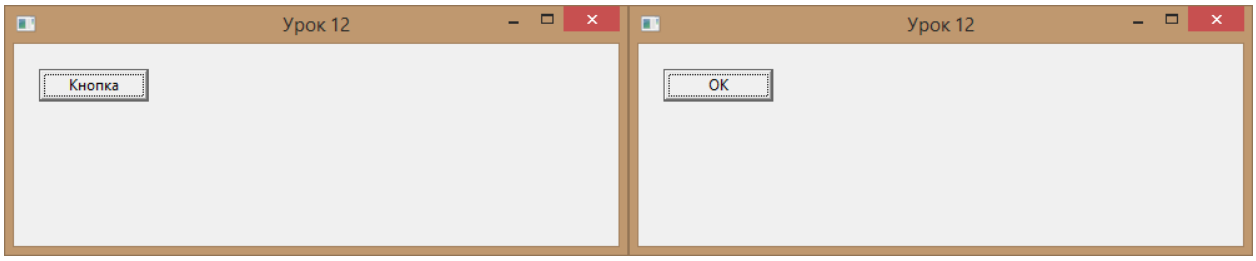

Рис. 8.2. Окончательный результат первого использования событий

Для достижения окончательного результата осталось добавить текстовое поле. Я учитываю предыдущие искания и повторяю создание тестового поля, как сделал это с кнопкой. Добавив команду по смене этикетки у текстового поля, я получаю результат, который намеревался получить изначально.

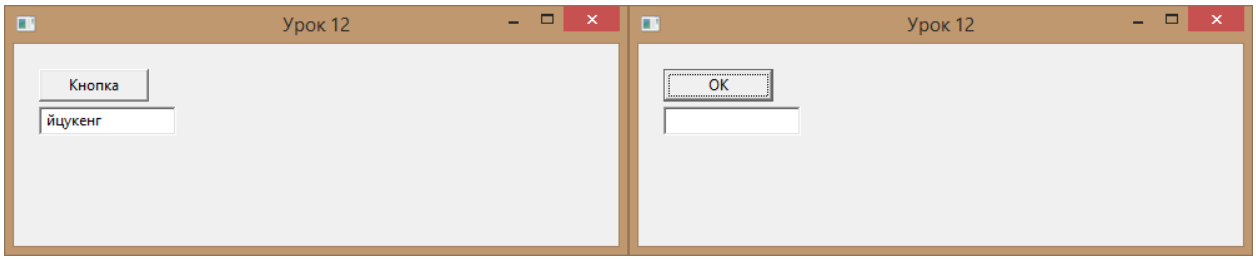

Рис. 8.3. Результат, который был запланирован

Теперь мой заголовочный файл tut12.h имеет вид:

```
#include <wx/wx.h>

class Tut12 : public wxFrame
{
public:
    Tut12(const wxString &title);
    void OnButtonClick(wxCommandEvent & event);
    wxButton *button;
    wxTextCtrl *text;
private:
    DECLARE_EVENT_TABLE()
};
```

И файл tut12.cpp:

```
#include "tut12.h"

Tut12::Tut12 (const wxString &title) : wxFrame (NULL, wxID_ANY, title,
wxDefaultPosition, wxSize(500,200))
```

```
{
    wxPanel *panel = new wxPanel(this, wxID_ANY);
    button = new wxButton (panel, wxID_ANY, wxT("Кнопка"), wxPoint(20,20));
    text = new wxTextCtrl (panel, wxID_ANY, wxT(""), wxPoint(20,50));
    Centre();
}
void Tut12::OnButtonClick(wxCommandEvent& WXUNUSED(event))
{
      text -> SetLabel(wxT(""));
      button -> SetLabel(wxT("OK"));
}
BEGIN_EVENT_TABLE(Tut12, wxFrame)
      EVT_BUTTON(wxID_ANY,  Tut12::OnButtonClick)
END_EVENT_TABLE()
```

Игра с названием текстового поля, которую я затеял, никак не связана с чем-то иным, как только с незнанием команды, которая могла бы очистить это текстовое поле. Возможно, такая команда (или функция) есть, но я её не нашёл.

Теперь, когда первый положительный результат достигнут, я хочу замахнуться на современный стиль выполнения событий. В уроке, которому я следую, написано, что применение таблиц событий унаследовано от языка Си, а современный метод подразумевает использование *Connect().*

Я не забываю, что повторение мать учения, поэтому не пытаюсь исправить предыдущий проект, а создаю новый, где постараюсь всё (или большую часть) написать «от руки».

Однако попытка перекопировать урок под мои нужды даёт ошибку, природу которой я не понимаю:

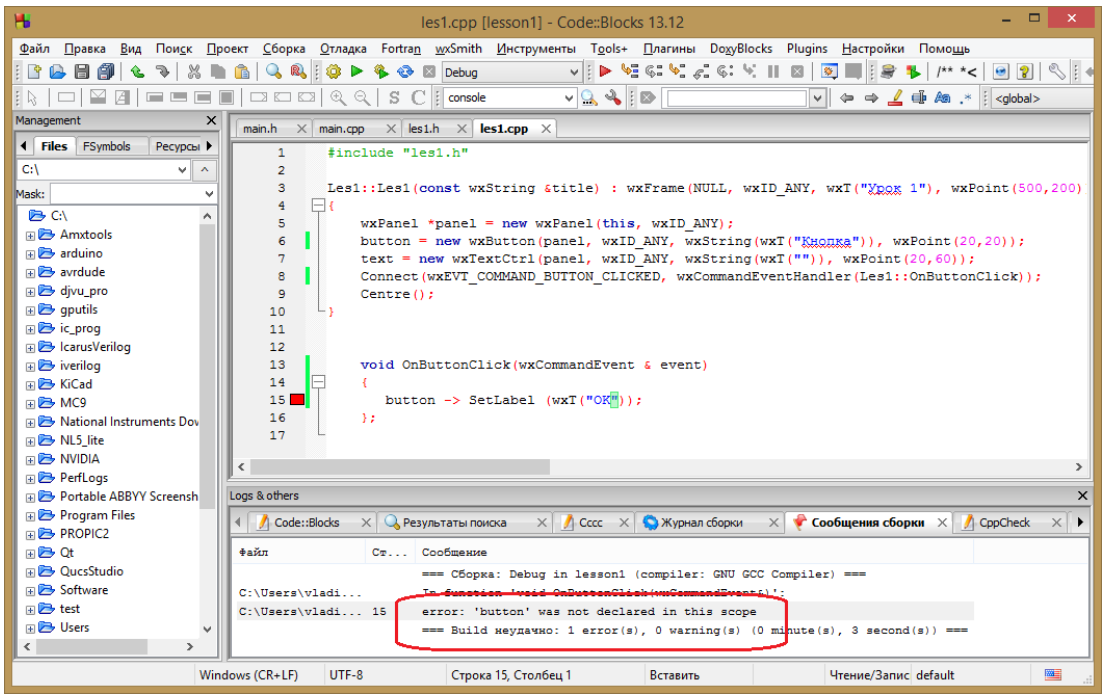

Рис. 8.4. Ошибка, связанная с нарушением границы?

Не помогло даже то, что я назвал этот проект Lesson1, чтобы избежать названия Tutor13. И я не понимаю, в чём причина, поскольку кнопка и объявлена и реализована…

После скитаний по форумам в поисках ответа, после разных попыток вернуть кнопку из-за границы я, наконец, замечаю, что, копируя, допустил ошибку. Я много раз сравнивал исходный текст урока со своей копией, я же смотрел… но не видел. Очевидно, что сообщение об ошибке настолько привлекло моё внимание к кнопке, что остальное осталось вне поля моего зрения. Подсказка для опытного человека – это помощь, а для неопытного не всегда.

Я не исключаю, что свою лепту внесли многочисленные страницы классов wxWidgets, когда я отыскивал необходимые для замены элементы, но это и не заменяет то, что следует лучше изучать предмет.

После исправления ошибки программа транслируется и выполняется так же, как и в предыдущем случае. А исправление было простым:

```
void Les1::OnButtonClick(wxCommandEvent & event)
{
   button -> SetLabel (wxT("OK"));
};
```

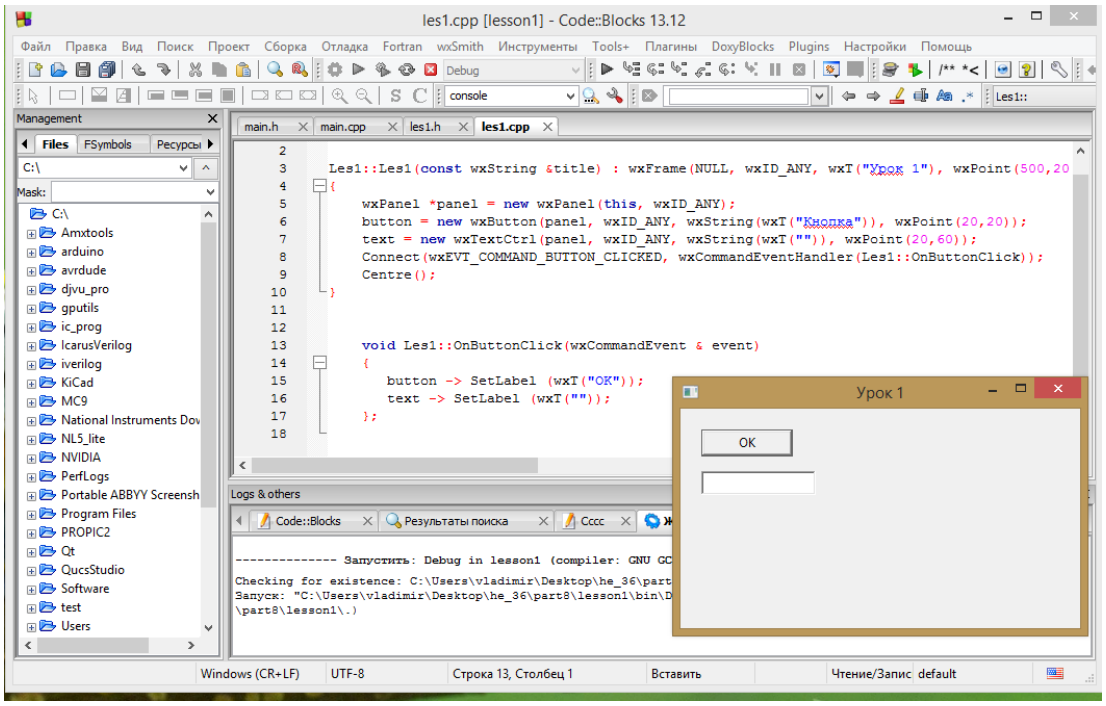

Рис. 8.5. Работа исправленной программы

Дойдя до этого места в рассказе, я вспомнил, что хотел отыскать руководство к программе Code::Blocks и перевести его. Чем и намерен заняться.

Полезная программа

Занимаясь переводом руководства пользователя Code::Blocks, я, а перевод занял не менее двух недель, я почти уверен, что забыл всё, о чем написал выше. Тем не менее, есть одна задача, которую я хотел бы решить: каждый месяц я снимаю показания счётчика электроэнергии, рассчитываю стоимость по дневному и ночному тарифу, чтобы заполнить счёт, который получаю от Мосэнерго. Не слишком сложно, но отчего бы не создать программу, которая упростит это процесс.

Итак, в окна программы вносятся предыдущие показатели счётчика, текущие данные, тарифы, а по нажатию кнопки выводятся: количество потреблённой электроэнергии и её стоимость.

Мне необходимо несколько текстовых полей, а именно: для ввода дневных и ночных данных, для ввода тарифов; для вывода результатов. Потребуется одна кнопка для выполнения расчётов, а также надписи.

То, что я за это время всё забыл (или почти всё), выяснилось сразу. Мне пришлось подглядывать в собственный рассказ, написанный ранее, подглядывать в тексты предыдущих проектов и пробовать.

Помимо тех графических компонентов, что уже использовались, пришлось отыскивать компонент, который я назвал бы Label, надпись. Получилась такая строка:

```
wxStaticText *label = new wxStaticText(panel, wxID_ANY,wxT("День"), wxPoint(200, 40));
```

Итог создания основных элементов, необходимых для программы, выглядит следующим образом:

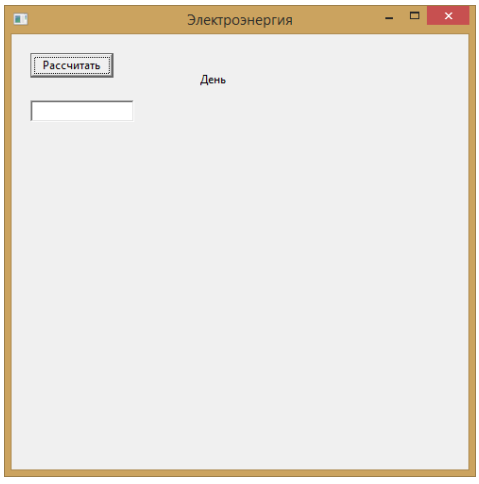

Рис. 9.1. Графическая база для создания программы

Для построения графической части программы мне осталось повторить создание необходимого количества надписей и текстовых полей. Но есть одно, что меня пока не устраивает – цвет панели. Белые поля на белом фоне мне не нравятся.

Казалось бы, в чём проблема? Я просмотрел несколько программ, предлагаемых профессиональными программистами. Та лёгкость, с которой у них всё работает, всё получается, признаться, очень расстроила меня. Но ответа на свой вопрос я, увы, не нашёл. Однако нашёл такое выражение: `SetBackgroundColour()`. Осталось определиться с этим самым colour и тем, как этот фоновый цвет передать панели. Впрочем, для передачи этого фона, я думаю, пригодится оператор стрелки, как это использовалось ранее: `panel -> SetBackgroundColour()`.

То, что цвет определяется в концепции RGB, заданием значений для красной, зелёной и синей составляющих, становится понятно при чтении описания класса wxColour. Но безнадёжные попытки повторить найденные в уроках и советах в Интернете варианты не оставляют мне выбора. Отбросив «приличия», я выбираю: `panel -> SetBackgroundColour(wxColour(0, 255, 0));`

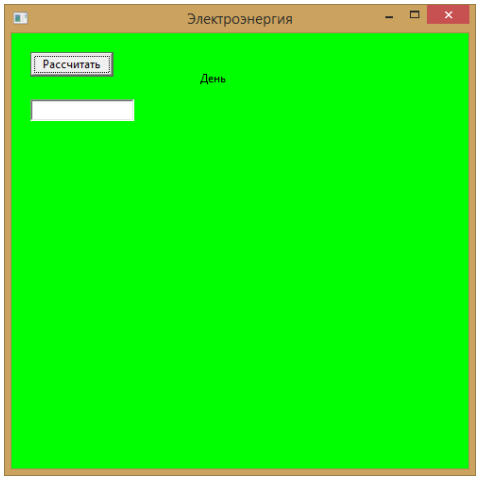

Рис. 9.2. Все базовые компоненты для графической части программы

Не уверен, что поступил правильно, но это работает, в чём можно убедиться, меняя значения параметров и подбирая цвет по своему вкусу. Меня, впрочем, устроит тот, что есть.

Для завершения этой части программы осталось добавить нужное количество окошек для ввода и вывода данных, надписи, и всё это правильно расположить на рабочей панели.

Для добавления текстовых полей и надписей я просто копирую предыдущие удачные строки необходимое количество раз. Затем комментирую всё, оставив текущие строки, где правлю все параметры вставки, проверяя результат запуском сборки. Когда нужный результат получен, я удаляю знак комментария у следующих двух строк. Конечно, программисты, которые привыкли к счёту параметров, например, в пикселях, сделают это сразу, но я не программист, я только учусь.

Итогом этого «учения» становится такой графический интерфейс моей программы:

Рис. 9.3. Итоговая графическая оболочка программы

Следующее, что нужно сделать – это обустроить событие, что я намерен «срисовать» из предыдущих экспериментов с программой Code::Blocks и библиотекой wxWidgets.

Срисовывая событие «нажатие кнопки» из предыдущей своей программы, я сталкиваюсь с тем, что начинаю понимать, почему лучше записать `wxButton *button;` в файле заголовка, а в файле основного кода записать `button = new wxButton (с нужными параметрами);`

Без этого следующее применение кнопки в обработке событий «отправляется за границу», поскольку обработка события происходит в другой функции. Не уверен, что и работа с текстовыми окошками не потребует такого же изменения.

Для тестирования, правильно ли срабатывает механизм события, я оставляю запись, меняющую надпись на кнопке. После трансляции кода и сборки проекта эту надпись я уберу: `button -> SetLabel (wxT("OK"));`

Что ж, пришло время заполнить событие реальными действиями: мне нужно считывать введённые значения, производить расчёт и выводить результаты.

Поскольку я не вписываю код, а копирую, частью из уже написанного кода, который правлю, частью из предыдущих кодов, программа выглядит не лучшим образом, но после сборки показывает, что она работает:

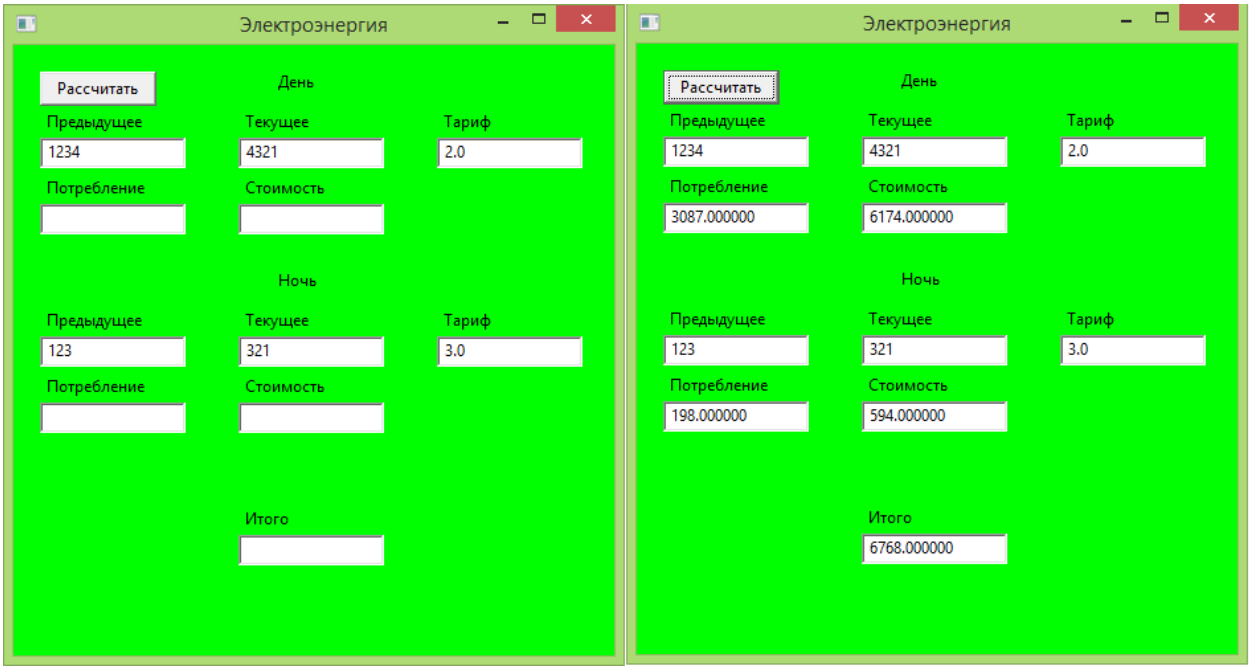

Рис. 9.4. Работающая программа расчётов за электроэнергию

Впрочем, насколько я понимаю, использовать свои наработки – это обычная практика не только для начинающих, но и для опытных специалистов. Другое дело, что и предыдущие мои наработки не отличались элегантностью. Получилась такая программа:

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "elect1.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    Elect1 *elect = new Elect1(wxT("Электроэнергия"));
    elect -> Show(true);

    return true;
}
```

elect1.h

```
#include <wx/wx.h>

class Elect1 : public wxFrame
{
public:
    Elect1 (const wxString &title);

    wxButton *button;

    wxTextCtrl *text1;
    wxTextCtrl *text2;
    wxTextCtrl *text3;
    wxTextCtrl *text4;
    wxTextCtrl *text5;

    wxTextCtrl *text6;
    wxTextCtrl *text7;
    wxTextCtrl *text8;
    wxTextCtrl *text9;
    wxTextCtrl *text10;
    wxTextCtrl *text11;
    void OnButtonClick(wxCommandEvent & event);
};
```

elect1.cpp

```
#include "elect1.h"

Elect1::Elect1 (const wxString &title) : wxFrame (NULL, wxID_ANY, title,
wxDefaultPosition, wxSize(470,500))
{
    wxPanel *panel = new wxPanel(this, wxID_ANY);
    button = new wxButton (panel, wxID_ANY, wxT("Рассчитать"),
wxPoint(20,20));

    wxStaticText *label_d = new wxStaticText(panel, wxID_ANY,wxT("День"),
wxPoint(200, 20));
    panel -> SetBackgroundColour(wxColour(0, 255, 0));

    text1 = new wxTextCtrl (panel, wxID_ANY, wxT(""), wxPoint(20,70));
    wxStaticText *label1 = new wxStaticText(panel,
wxID_ANY,wxT("Предыдущее"), wxPoint(25, 50));
    text2 = new wxTextCtrl (panel, wxID_ANY, wxT(""), wxPoint(170,70));
    wxStaticText *label2 = new wxStaticText(panel, wxID_ANY,wxT("Текущее"),
wxPoint(175, 50));
    text3 = new wxTextCtrl (panel, wxID_ANY, wxT(""), wxPoint(320,70));
    wxStaticText *label3 = new wxStaticText(panel, wxID_ANY,wxT("Тариф"),
wxPoint(325, 50));
    text4 = new wxTextCtrl (panel, wxID_ANY, wxT(""), wxPoint(20,120));
    wxStaticText *label4 = new wxStaticText(panel,
wxID_ANY,wxT("Потребление"), wxPoint(25, 100));
    text5 = new wxTextCtrl (panel, wxID_ANY, wxT(""), wxPoint(170,120));
```

```
    wxStaticText *label5 = new wxStaticText(panel, wxID_ANY,wxT("Стоимость"),
wxPoint(175, 100));

    wxStaticText *label_n = new wxStaticText(panel, wxID_ANY,wxT("Ночь"),
wxPoint(200, 170));

    text6 = new wxTextCtrl (panel, wxID_ANY, wxT(""), wxPoint(20,220));
    wxStaticText *label6 = new wxStaticText(panel,
wxID_ANY,wxT("Предыдущее"), wxPoint(25, 200));
    text7 = new wxTextCtrl (panel, wxID_ANY, wxT(""), wxPoint(170,220));
    wxStaticText *label7 = new wxStaticText(panel, wxID_ANY,wxT("Текущее"),
wxPoint(175, 200));
    text8 = new wxTextCtrl (panel, wxID_ANY, wxT(""), wxPoint(320,220));
    wxStaticText *label8 = new wxStaticText(panel, wxID_ANY,wxT("Тариф"),
wxPoint(325, 200));
    text9 = new wxTextCtrl (panel, wxID_ANY, wxT(""), wxPoint(20,270));
    wxStaticText *label9 = new wxStaticText(panel,
wxID_ANY,wxT("Потребление"), wxPoint(25, 250));
    text10 = new wxTextCtrl (panel, wxID_ANY, wxT(""), wxPoint(170,270));
    wxStaticText *label10 = new wxStaticText(panel,
wxID_ANY,wxT("Стоимость"), wxPoint(175, 250));

    text11 = new wxTextCtrl (panel, wxID_ANY, wxT(""), wxPoint(170,370));
    wxStaticText *label11 = new wxStaticText(panel, wxID_ANY,wxT("Итого"),
wxPoint(175, 350));

    Connect(wxEVT_COMMAND_BUTTON_CLICKED,
wxCommandEventHandler(Elect1::OnButtonClick));

    Centre();
}

    void Elect1::OnButtonClick(wxCommandEvent & event)
    {
    double d_p;
    double d_c;
    double d_tariff;
    double d_cost;
    double n_p;
    double n_c;
    double n_tariff;
    double n_cost;
    double d_all;
    double n_all;
    double d_n_cost;

    wxString f_rez1 = text1 -> GetValue();
    wxString f_rez2 = text2 -> GetValue();
    wxString f_rez3 = text3 -> GetValue();

    wxString f_rez4 = text6 -> GetValue();
    wxString f_rez5 = text7 -> GetValue();
    wxString f_rez6 = text8 -> GetValue();
```

```
    f_rez1.ToDouble(&d_p);
    f_rez2.ToDouble(&d_c);
    f_rez3.ToDouble(&d_tariff);

    f_rez4.ToDouble(&n_p);
    f_rez5.ToDouble(&n_c);
    f_rez6.ToDouble(&n_tariff);

    d_all = d_c - d_p;
    d_cost = d_tariff*d_all;

    n_all = n_c - n_p;
    n_cost = n_tariff*n_all;

    d_n_cost = d_cost + n_cost;

    wxString all_d = wxString::Format(wxT("%f"), d_all);
    text4 -> SetValue(all_d);

    wxString cost_d = wxString::Format(wxT("%f"), d_cost);
    text5 -> SetValue(cost_d);

    wxString all_n = wxString::Format(wxT("%f"), n_all);
    text9 -> SetValue(all_n);

    wxString cost_n = wxString::Format(wxT("%f"), n_cost);
    text10 -> SetValue(cost_n);

    wxString cost_d_n = wxString::Format(wxT("%f"), d_n_cost);
    text11 -> SetValue(cost_d_n);
    };
```

Резюме

Я не знаю, сколько подобных программ нужно создать, чтобы без затруднений делать то, что тебе нужно. Не знаю. Но твёрдо знаю, что если мне потребуется написать программу для компьютера, управляющую микроконтроллером, я это сделаю. И ещё я знаю, что время, потраченное на чтение уроков по программированию на языке C++ с помощью wxWidgets в среде разработки Code::Blocks не прошло бесполезно. Мне стало понятнее то, что на этом языке написано. И мне стало понятнее, что в следующий раз, когда в какой-нибудь программе у меня что-то не будет получаться, я не буду говорить: «дурацкая программа», – поскольку за каждой из них стоит труд программистов, и труд не из лёгких. И этот труд я лучше оценил тогда, когда сам попробовал сделать что-то простенькое, но не бесполезное.