

Глава 7

Компьютер и Linux

В истории отечественного радилюбительства 1986 год стал годом особенным. В этом году в журнале «Радио» была опубликована конструкция простого мини-компьютера «Радио-86РК», который тут же был изготовлен большим числом радилюбителей.

Примерно в 1987 году и мне удалось стать обладателем этой удивительной по тем временам машины. Тут же стал осваивать язык Ассемблера, созданного для «Радио-86РК» и пробовать разработку собственных радилюбительских программ. В период конца 80-х и начала 90-х годов прошлого столетия мною были сделаны программы для проведения любительских радиосвязей телеграфом и телетайпом.

Отечественных радилюбителей тех времен отличало от радилюбителей всех прочих стран умение работать с паяльником и создавать своими руками такие радилюбительские конструкции, которые зачастую не уступали зарубежным, фирменным. При проведении телетайпных радиосвязей, практически все мои зарубежные корреспонденты не были в состоянии понять, что я работаю на самодельном компьютере и использую при этом компьютерную программу собственной разработки.

Начиная примерно с 1994 года, в нашей стране начали становиться доступными компьютеры IBM PC и появилась возможность приобретать по различным каналам программы для радилюбительских целей, разработанные иностранными программистами. В те годы я старался освоить все попадавшие мне в руки программы по радиосвязи. Восхищался их красочностью, большим разнообразием команд и функций. Переводил файлы документации многих программ на русский язык, делал к этим программам свои вспомогательные файлы, проводил и другую работу, которую можно назвать словом «русификация».

В процесс этой работы невольно пришел к выводу, что российские радилюбители могут и должны сами создавать для себя необходимые компьютерные программы.

Так я стал разрабатывать компьютерные программы для цифровых видов любительской радиосвязи, большинство из которых можно найти в Internet на моем сайте по адресу: r3xb.nm.ru.

Вначале все программы разрабатывались мною в среде программирования Borland Turbo C/C++, с помощью которой создавались программы, работающие под управлением операционной системы MS DOS. В то время операционная система Microsoft Windows 95 еще только начинала осваиваться радилюбителями, но возможности создания красочно оформленных рабочих окон программ привлекла на сторону Windows многих разработчиков программных продуктов.

Если сравнивать между собой процессы разработки радилюбительских программ в операционных системах MS DOS и Microsoft Windows 95/98, то оказывается, что если программы для MS DOS писались на языках программирования Pascal или

C/C++ в виде очень большого числа строк, создаваемых в специальных текстовых редакторах, то для создания Windows-приложений появились среды визуальной разработки Visual Basic фирмы Microsoft и Borland C++ Builder. В этих средах процесс программирования заключается в создании главного рабочего окна программы путем установки (с помощью мыши) на специальную форму таких компонентов рабочего окна, как кнопки, полосы прокрутки, окна редактирования текста и других, при этом исходные коды, описывающие эти компоненты, уже прописаны заранее и вводятся в текст программы автоматически. Программисту остается только в текстовом редакторе написать коды функций для компонента, которые должны выполняться после воздействия на этот компонент мышью.

Интересной особенностью среды быстрого программирования Borland C++ Builder является то, что при соответствующей методике можно начинать изучение процесса создания компьютерных программ без предварительного изучения языка программирования C/C++. Подобный процесс по такой методике приведен в книге «Быстрое программирование на C++» [3].

Операционная система Microsoft Windows 95/98 имеет в своем составе MS DOS 7, поэтому все программы прошлых лет, созданные для работы под управлением операционной системы MS DOS, нормально работают под этими операционными системами. В то же время операционные системы Microsoft Windows 2000/XP созданы на основе ядра операционной системы Microsoft Windows NT. Под управлением этих операционных систем могут работать очень немногие программы MS DOS.

Например, в программах для любительской радиосвязи, созданных для работы под управлением MS DOS, используется нетрадиционное управление COM-портами. Операционные системы Microsoft Windows 2000/XP не поддерживает такое управление. Поэтому разработчики программ для цифровых видов радиосвязи, предназначенных для работы под управлением Windows 2000/XP, вынуждены были в своих программах перейти на использование для приема и передачи радиосигналов используемую в компьютере звуковую карту. При этом качество приема сигналов и их декодирование ухудшилось намного.

В течение последних примерно пяти лет в среде отечественных радиолюбителей стала использоваться операционная система Linux. Эта операционная система сделана по образцу и подобию операционной системы Unix, которая с очень давних пор используется большинством научных организаций и в качестве серверов практически у всех Internet-провайдеров.

По отношению к операционным системам MS DOS и Microsoft Windows, операционная система Linux является совершенно необычной, в ней задействованы совсем иные принципы организации работы самой управляющей системы. Linux — это эффективная и быстрая в работе бесплатная операционная система, в которой реализованы все возможности не бесплатной операционной системы Unix. Авторы некоторых книг считают, что появление Linux привело к небольшой революции в компьютерном мире, после которой бесплатное программное обеспечение стало применяться для решения множества задач — от создания web-сайтов до реализации инфраструктуры корпоративных информационных систем и обучающих программ. О Linux уже издано много интересных книг и статей в разных информационных источниках, в том числе в Internet.

Выражаю благодарность за помощь в освоении программирования в операционной системе Linux Парамонову Р.А. При первой попытке начать создание в системе

Linux компьютерных программ я столкнулся с большими трудностями и вынужден был прервать это занятие. Толчком к возобновлению программирования в среде Linux послужило неожиданное заочное знакомство (по электронной почте) с профессиональным программистом Романом А. Парамоновым из г. Пушкин, Московской области. Роману я благодарен за ценные советы и подсказки, за подборку для меня нужной информации в Internet, а также и за моральную поддержку, благодаря которой я решился снова взяться за освоение программирования в Linux.

Быстрое программирование в среде Kylix

Быстрое программирование в среде Kylix, разработанной известной фирмой Borland казалось мне наиболее интересным и простым для освоения. Дело в том, что в течение нескольких лет мне пришлось заниматься программированием в среде Borland C++ Builder, которая практически, как писалось во многих источниках, основана на тех же принципах, что и Borland Kylix.

Заполучить работоспособную версию Kylix оказалось не так уж и сложно. В Internet, на сайте фирмы Borland, я ввел в окно поисковой системы слово *trial* и получил адреса, по которым можно скачать так называемые trial-версии программных продуктов, созданных фирмой. Эти trial-версии являются экспериментальными версиями и рассчитаны на применение только в течение определенного срока, обычно этот срок равен 30 дням. Однако на сайте рядом с адресом для скачивания экспериментальной версии Kylix находился адрес и так называемой open-версии Kylix3_open. Дополнение «_open» обычно имеет сокращенная версия, которая может свободно распространяться. От полноценной trial-версии Kylix3_open отличается, на мой взгляд, только значительно меньшим количеством задействованных в этой программе компонентов.

Для экспериментов скачал две этих версии, и Kylix-trial и Kylix3_open. В то время на моем компьютере была установлена операционная система Linux Mandrake 9.1. После установки Kylix оказалось, что программа представлена сразу в двух исполнениях. В меню KDE значился Kylix 3 (C++ IDE) и Kylix 3 (Delphi IDE). Но ни то и ни другое исполнение работать под управлением Mandrake 9.1 не стало. В документации к программе было указано, что этот вариант Kylix предназначен для работы под управлением Mandrake 8.2 или Red Hat 7.2, также может работать и под Red Hat 7.3. Тогда пришлось на компьютер устанавливать Linux Mandrake 8.2.

Начинаем программировать

Рабочее окно Kylix3_open (C++ IDE) оказалось очень похожим на рабочее окно интегрированной среды разработки (IDE) программы Borland C++ Builder. На рис. 7.1 показано рабочее окно Kylix3_open.

Обратите внимание на небольшое количество компонентов, расположенных в окне панели компонентов. Можете сравнить рис. 7.1 с рис. П2.1 в приложении 2.

Мною было принято решение создать в среде Kylix3_open программу для расчета индуктивности колебательного контура, аналог которой является составной частью программы INDUKTIW на языке C++ ранее разработанной и опубликованной книге «Быстрое программирование на C++» [3].

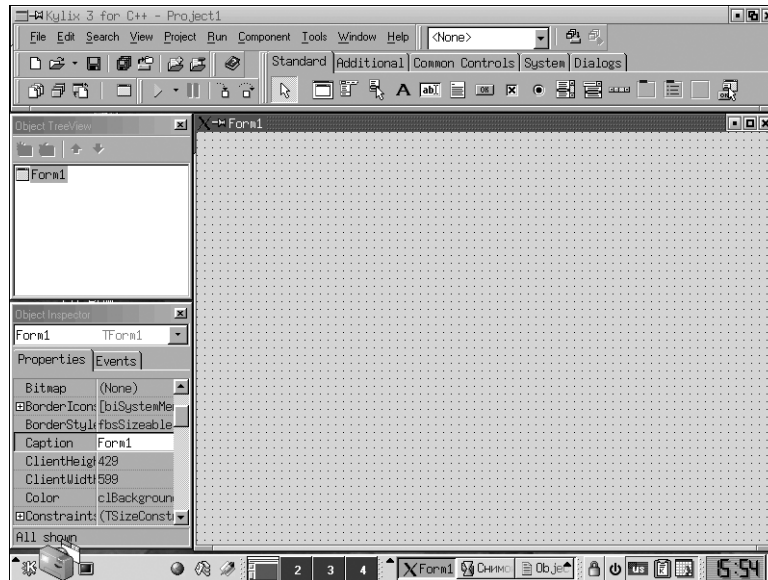


Рис. 7.1. Рабочее окно Kylix3_open

Назначение приложения

Программа предназначена для расчета величины индуктивности простого колебательного контура по заданным величинам рабочей частоты и емкости контура. Для выполнения расчета в верхнее окно редактирования следует ввести величину частоты рабочего диапазона в мегагерцах, а во второе окно ввести величину емкости контура. Затем следует щелкнуть мышью на кнопке **Выполнить расчет**.

Для большей наглядности в рабочем окне программы размещена схема простого колебательного контура.

Создание проекта

Перед началом создания рабочего проекта приложения нужно создать для этого проекта отдельную директорию (папку). Это может быть заранее созданная директория, например, `/usr/home/nick/my_projects/kontur2`. Имеется в виду, что `nick` — это имя пользователя данного компьютера, хозяина этой директории, а `kontur2` — это имя будущего приложения.

Запускаем Kylix (C++ IDE) и выбираем **File⇒New⇒Application**. В заранее созданную директорию сохраняем только что созданные файлы проекта. Для этого выбираем **File⇒Save Projekt As**. Основные файлы проекта сохраняются поочередно. Первым Kylix сохраняет основной функциональный файл проекта и просит ввести имя этого файла. Вводим имя файла «`kontur01.cpp`» и сохраняем файл под этим именем. Затем Kylix предлагает ввести имя файла проекта. Предупреждаю вас о том, что имена сохраняемого первым функционального файла и файла проекта должны различаться. Итак, сохраняем файл проекта под именем «`kontur1.bpr`».

Проект сохранен в определенной директории, приступаем к заполнению формы проекта. Если вы будете разрабатывать собственную программу, следует предвари-

тельно на листе бумаги сделать эскиз формы с размещенными на ней компонентами. В процессе разработки программы количество и размещение компонентов могут значительно меняться, но предварительный эскиз нужен обязательно. На рис. 7.2 показана форма с установленными на ней компонентами.

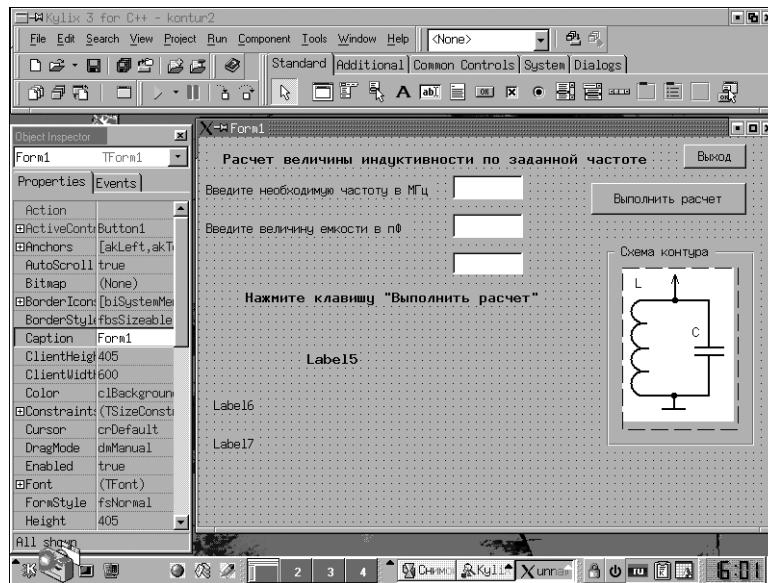


Рис. 7.2. Расположение компонентов на форме

На панели компонентов должна быть установлена страница библиотеки компонентов **Standard** (Стандартная).

Этап 1

Для отображения на форме текстовых строк будем использовать компонент **Label** (Метка). Подводим курсор к символическому изображению самого первого (слева) компонента в странице библиотеки **Standard** и задерживаем на нем курсор. Сразу же рядом возникает небольшое выпадающее окошко подсказки с именем этого компонента. Переместив курсор на символ следующего компонента, читаем его имя, затем, после установки курсора на жирную букву «A» читаем название **Label**. Это как раз то, что нам требуется. Щелкните мышью на символе компонента и переведите курсор в левый верхний угол формы, как раз в то место, откуда должна начинаться первая текстовая строка. Нажмите левую кнопку мыши и немного переместите курсор вправо и вниз. При этом появится окно, размеры которого будут изменяться по мере перемещения курсора. Для нас размеры этого окна не имеют ни малейшего значения, поэтому оставьте окно минимальным и отпустите кнопку мыши. В форме появится название **Label1**, одновременно с ним в окне **Object Inspector** появится перечень всех свойств, принадлежащих этому компоненту. В данном случае для нас необходимы из всего этого перечня только два свойства — свойство для создания текста и свойство для создания размера и цвета шрифта. Первым выберите свойство **Caption** и в принадлежащем этому свойству окне ре-

дактирования удалите имеющееся там название «Label1» и введите текст «Расчет величины индуктивности по заданной частоте». Затем выберите свойство **Font** и щелкните справа в окне редактирования этого свойства на появившейся небольшой кнопке с тремя точками. После этого появится окно выбора шрифта, в котором следует сделать следующие установки:

- ♦ **Font** — Fixed [cronyx];
- ♦ **Script** — Russian Cyrillic (KOI8-R);
- ♦ **Font style** — Bold;
- ♦ **Size** — 20.

На экране установится текст первой строки. Точно таким же образом вводится текст для всех последующих строк. Следует знать, что имена Label1–Label7 так и останутся метками для соответствующих строк. Эти имена находятся в свойствах **Name** каждого компонента. Конечно, эти имена можно изменять, но в данном случае делать это не следует. Новые имена компонентам даются тогда, когда возникает возможность запутаться в цифровых отличиях.

Этап 2

На этом этапе нужно установить компоненты формы, позволяющие вводить в программу исходные данные. Таким компонентом является **Edit**, изображенный справа рядом с жирной буквой «А». Устанавливаем курсор на символ этого компонента и убеждаемся по подсказке в том, что это действительно нужный для нас компонент. Переносим на форму сначала верхний компонент, затем — два нижних. При этом в окне первого компонента имеется текст **Edit1**, а в окне второго — **Edit2**. Это названия компонентов, содержащиеся в свойствах **Name** и в свойствах **Caption** одновременно. Поскольку имена компонентов изменять не будем, то свойство **Name** оставим без изменения, а в окне редактирования свойств **Caption** удалим это название.

Таким образом, на форме установлены три окна редактирования **Edit1**, **Edit2** и **Edit3**, при этом два из них предназначены для ввода пользователем в работающую программу необходимых исходных данных, а третье окно можно и не устанавливать на форму. Это окно мною используется для экспериментальных целей.

Этап 3

Начало вычислений в данном приложении задается кнопками **Button**. Компоненты этих кнопок также располагаются на странице библиотеки **Standard**, символом этих кнопок является изображение миниатюрной кнопочки **ОК**. Установите курсор на этот символ и убедитесь, что подсказка выводит необходимое название **Button**. Затем перенесите этих два компонента в соответствующие места формы, при этом имена компонентов оставьте неизменными, т.е. **Button1**, **Button2**, а в свойствах **Component** введите соответственно тексты «Выход» и «Выполнить расчет».

В результате в форме будут установлены исполнительные компоненты — кнопки **Button**.

Этап 4

Для придания приложению наглядности, следует разместить на форме рисунки с изображением схемы колебательного контура. Разумеется, что рисунок схемы уже должен быть выполнен в графическом редакторе и готов к размещению на форме приложения. Сначала подготовим место для размещения схемы. Чтобы рисунок схемы имел какое-то название, применим компонент `GroupBox`.

Этот компонент служит для создания (в данном случае) рамки с заголовком вокруг будущего рисунка. Вы уже научились по всплывающей подсказке находить нужные компоненты, поэтому самостоятельно устанавливайте этот компонент. При этом нужно сначала задать для рамки наибольшие размеры, чтобы в дальнейшем их можно было легко уменьшить. Обратные действия выполнять сложнее. В свойствах `Caption` следует ввести текст заголовка. В данном случае заголовком является текст «Схема контура». Компонент `GroupBox` служит (в данном случае) чисто декоративным целям и не является обязательным, но вы должны его знать и уметь им пользоваться.

Рисунок схемы размещается в форму с использованием компонента `Image`, который находится в библиотеке `Additional`. Символ этого компонента представляет собой миниатюрную картинку, на которой изображено голубое небо и темно-синие горы. Установите курсор на этот символ и убедитесь по всплывающей подсказке о правильности сделанного выбора. Затем перенесите два компонента на форму и разместите изнутри установленных ранее рамок `GroupBox`.

Для компонента `Image` выбираем свойство `Picture`. Щелкаем мышью на кнопке, расположенной в окне редактирования этого свойства, в результате чего открывается окно **Picture Editor**. В этом окне выбираем **Load**. На экране открывается окно **Load Picture**, предназначенное для поиска графического файла с рисунком. Это стандартное окно выбора файла операционной системы, которое не требует каких-то дополнительных пояснений. Выбираем в этом окне файл, щелкаем мышью на кнопке **ОК** и рисунок из выбранного файла будет вставлен в форму.

Теперь нужно подогнать под размеры рисунка размеры рамки и разработка формы на этом может считаться законченной.

Работа с текстовыми файлами

В процессе вашей работы над установкой на форму необходимых компонентов, `Kylix` непрерывно работал в автоматическом режиме над основными текстовыми файлами проекта `kontur01.cpp` и `kontur01.h`.

Чтобы было удобнее разбираться с названиями компонентов, установленных в форму, в листинге 7.1 приведен текст файла `kontur01.h`.

Листинг 7.1. Текст файла `kontur01.h`

```
//-----
#ifndef kontur01H
#define kontur01H
//-----
#include <Classes.hpp>
#include <QControls.hpp>
```

```

#include <QStdCtrls.hpp>
#include <QForms.hpp>
#include <QExtCtrls.hpp>
#include <QGraphics.hpp>
//-----
class TForm1 : public TForm
{
__published: // IDE-managed Components
    TButton *Button1;
    TButton *Button2;
    TLabel *Label1;
    TLabel *Label2;
    TLabel *Label3;
    TEdit *Edit1;
    TEdit *Edit2;
    TEdit *Edit3;
    TLabel *Label4;
    TLabel *Label5;
    TLabel *Label6;
    TLabel *Label7;
    TGroupBox *GroupBox1;
    TImage *Image1;
    void __fastcall FormCreate(TObject *Sender);
    void __fastcall Button1Click(TObject *Sender);
    void __fastcall Edit2Change(TObject *Sender);
    void __fastcall Button2Click(TObject *Sender);
private: // User declarations
public: // User declarations
    __fastcall TForm1(TComponent* Owner);
};
//-----
extern PACKAGE TForm1 *Form1;
//-----
#endif
//-----

```

Файл создается средой программирования автоматически, вмешательство в него должно быть только очень осмысленным.

Первые две строки совместно с последней строкой относятся к директивам пре-процессора и необходимы для того, чтобы этот файл не устанавливался дважды. Фактически они задают начало и конец подключаемого заголовочного файла `kontur01.h`.

Две наклонных черты «//» являются символом начала строки с пояснениями (комментариями) и средой программирования (компилятором) игнорируются.

Следующие шесть строк, начинающиеся с текста `#include`, равнозначны понятию «подключение» и дают указание компилятору подключить к работе над программой указанные далее в угловых скобках специальные «подключаемые заголовочные файлы», которые содержат описания задействованных в данной программе функций и которые находятся в составе Kylix в директории `INCLUDE`. В названиях этих файлов первая буква «Q» показывает на то, что подключаемые заголовочные файлы относятся к Qt-библиотеке, имеющейся практически во всех версиях операционной системы Linux.

Подробнее о Qt-библиотеке будет рассказано в следующем разделе этой главы.

Если подключаемый файл находится в директории разрабатываемого проекта, то он заключается в двойные кавычки: "...".

Конструкцией `class TForm1 : public TForm` начинается «объявление» класса `TForm1`. Все, что далее находится между фигурными скобками, относится к этому классу. Первыми членами этого класса объявляются установленные нами ранее на форму компоненты типа `Label` и `Button`, затем объявляются функции, вызванные событиями этих компонентов.

Смысл понятия «объявление» заключается в том, что этим действием для объявляемого объекта выделяется необходимый участок памяти и организуется адресный указатель на этот участок.

Функции (члены класса `TForm1`) в этом файле только объявляются, а вот описывать эти функции с помощью программных кодов мы будем самостоятельно. В листинге 7.2 приведен текст соответствующего файла `kontur01.cpp`.

Листинг 7.2. Текст файла *kontur01.cpp*

```
//-----
#include <clx.h>
#pragma hdrstop
#include <math.h>
#include <SysUtils.hpp>
#include <stdlib.h>
#include "kontur01.h"
//-----
#pragma package(smart_init)
#pragma resource "*.xfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)           // 1
    : TForm(Owner)
{
}
//-----
void __fastcall TForm1::FormCreate(TObject *Sender)     // 2
```

```

{
Caption = "Расчет колебательного контура";
Label4->Caption = "";           // Удаляем текст из этой строки
Label1->Font->Color = clBlue;    // Устанавливаем цвет для верхней строки
Label5->Caption = "";           // Удаляем текст этой строки
// Устанавливаем цвет для строки Label5
Label5->Font->Color = clBlue;
Label6->Caption = "";           // Удаляем текст из этой строки
Label7->Caption = "";           // Удаляем текст из этой строки
}
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)    // 3
{
Close();
}
//-----
void __fastcall TForm1::Edit2Change(TObject *Sender)    // 4
{
Label4->Font->Color = clRed;
Label4->Caption = "Нажмите клавишу <Выполнить расчет>";
}
//-----
void __fastcall TForm1::Button2Click(TObject *Sender)    // 5
{
float L;           // Объявляем переменную для индуктивности
float LC;          // Объявляем переменную для вспомогательного числа

if(Edit1->Text == "" || Edit2->Text == "")
    ShowMessage("Введите недостающие данные ");
else
{
    LC = pow((159 / StrToFloat(Edit1->Text.c_bstr()),2);
// Так как в окнах редактирования Kylix строка представлена в виде пе
// ременной WideString, то используется функция Edit1->Text.c_bstr();
    L = LC / StrToFloat(Edit2->Text.c_bstr());
    Label5 -> Caption = "Результаты расчета";
    Label6 -> Caption = "Величина индуктивности = "+
        FloatToStrF(L,ffGeneral,4,2)+" мкГн";
    Label7 ->Caption = "Вспомогательная величина LC =" +
        FloatToStrF(LC,ffGeneral,5,2);
    Edit3->Text = FloatToStrF(L,ffGeneral,4,2);
}
}
//-----

```

Так же, как и в предыдущем листинге, файл начинается с перечисления подключаемых заголовочных файлов. Первым стоит файл `<clx.h>`, который является главным файлом в описаниях всех задействованных в Kylix библиотек компонентов. Далее подключаются файлы `<math.h>` и `<stdlib.h>` — это файлы с описаниями математических функций и файл `kontur01.h`, подключаемый файл собственно данного проекта. Строки, начинающиеся с выражения `#pragma`, являются специфическими для Kylix служебными строками и никаким изменениям или удалениям подвергаться не должны.

В данном случае я также (для удобства обращения) пронумерую все функции.

В «тело» функции 1 никаких дополнительных строк вносить не следует.

Функция 2 создает главное рабочее окно приложения, поэтому в эту функцию следует ввести свойства тех компонентов, которые при первом открытии рабочего окна должны быть выданы на экран. Например, ввести необходимый для компонента начальный текст, показать начальный цвет букв и т.д.

Функция 3 является реакцией на нажатие кнопки **Выход**. Она закрывает все открытые в процессе работы программы файлы и окна, затем прекращает работу программы.

Функция 4 является реакцией на ввод данных в окне редактирования **Edit2** — выводит на экран соответствующий текст.

Функция 5 включается в работу по после нажатия кнопки **Выполнить расчет**. Здесь имеется одна особенность, специфическая для Kylix: вводимая в окна редактирования **Edit1** и **Edit2** информация представлена в виде строки типа `WideString`, поэтому необходимо выполнить преобразование строки `WideString` в число. Такое преобразование выполняется с помощью функции `Edit1->Text.c_bstr()`.

После полного оформления текстового файла `kontur01.cpp` следует сохранить все в соответствующей директории проекта, выполнить компиляцию проекта и создание исполняемого файла. Если все неприятности, встреченные в процессе компиляции, остались позади и успешно создан исполняемый файл, то можно выполнить команду **RUN** и запустить проект на выполнение. На рис. 7.3 показано рабочее окно программы `kontur1` после ввода всех данных и нажатия кнопки **Выполнить расчет**.

На рис. 7.4 показано рабочее окно программы в случае, если кнопка **Выполнить расчет** нажата до ввода информации в окна редактирования **Edit1** и **Edit2**.

Вариант Kylix на базе Delphi

Вариант Kylix3 (Delphi IDE) работает также нормально, как и вариант Kylix (C++ IDE). Подробно на этом останавливаться не будем, приведем только в листинге 7.3 текст файла `Kontur02.pas`.

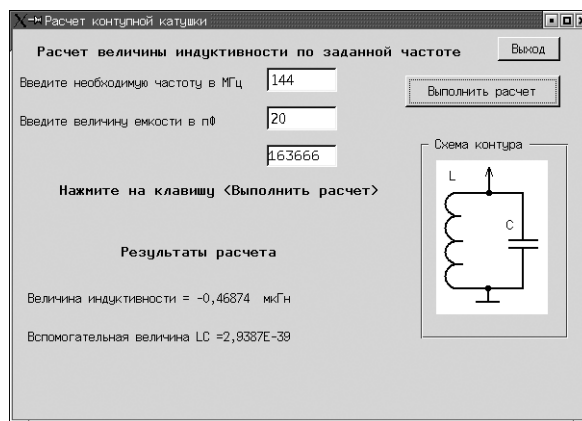


Рис. 7.3. Рабочее окно программы `kontur1`

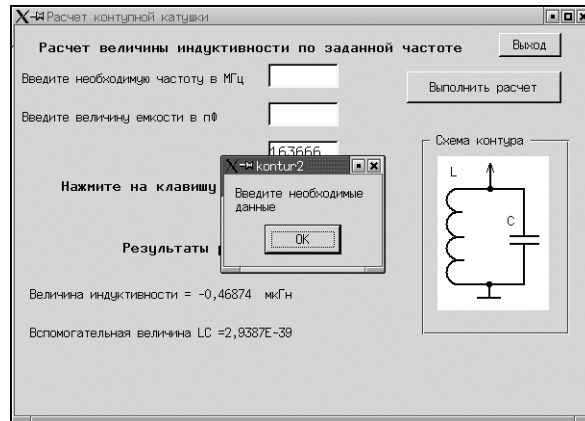


Рис. 7.4. Диалоговое окно на фоне рабочего окна программы

Листинг 7.3. Текст файла *Kontur02.pas*

```
//-----
unit Kontur02;

interface

uses
  SysUtils, Types, Classes, Variants, QTypes, QGraphics, QControls,
  QForms, QDialogs, QStdCtrls, QExtCtrls, Math;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Button1: TButton;
    Label2: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Label3: TLabel;
    Button2: TButton;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    GroupBox1: TGroupBox;
    Image1: TImage;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Edit2Change(Sender: TObject);
  private
```

```
    { Private declarations }
public
    { Public declarations }
end;

var
    Form1: TForm1;

Implementation

{$R *.xfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
    close();
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    Caption := 'Расчет колебательного контура';
    Label4.Caption := '';
    Label11.Font.Color := clBlue;
    Label5.Caption := '';
    Label5.Font.Color := clBlue;
    Label6.Caption := '';
    Label7.Caption := '';
end;

procedure TForm1.Button2Click(Sender: TObject);
var
    L: Double;
    LC: Double;      // WideString
begin
    if (Edit1.Text= '') or (Edit2.Text= '' ) then
        ShowMessage('Введите недостающие данные')
    else
        begin
            LC := (159 / StrToFloat(Edit1.Text))* (159 / StrToFloat(Edit1.Text));
            L := LC / StrToFloat(Edit2.Text);
            Label5.Caption := 'Результаты расчета';
            Label6.Caption := 'Величина индуктивности = '+
                FloatToStrF(L, ffGeneral, 4, 2)+ ' мкГн';
            Label7.Caption := 'Вспомогательная величина LC =' +
                FloatToStrF(LC, ffGeneral, 5, 2);
            Edit3.Text := FloatToStrF(L, ffGeneral, 4, 2);
        end;
end;
```

```

end;

procedure TForm1.Edit2Change(Sender: TObject);
begin
  Label4.Font.Color := clRed;
  Label4.Caption := 'Нажмите на кнопку Выполнить расчет';
end;
end.
//-----

```

Создание проекта в этом варианте проходит также нормально. Исполняемый файл создается и запускается в работу без проблем и из среды программирования и из файлового менеджера **KDE**.

Злоключения

Нет, я не ошибся. Не «Заключение», а именно «Злоключения». Неприятности начались после того, как исполняемый файл программы `konturl`, предварительно отлаженный и протестированный в среде Kylix3 (C++ IDE), не стал запускаться ни из командной строки консоли, ни из файлового менеджера **KDE**. Понять причину мне не удалось, поэтому попытался обратиться с просьбой о помощи к авторам статей по Kylix в Internet. Ни на одно из своих писем ответа я не получил.

Не придумав ничего лучшего, я переустановил Linux Mandrake 8.2. Как ни странно, но после этой переустановки многое изменилось. При первом запуске в работу из консоли созданного ранее исполняемого файла `konturl` система сообщила, что не может найти библиотеку `libborqt-6.9.0-qt2.3.so`. Эта библиотека находилась по адресу `/home/nick/kylix3/bin/` в составе пакета файлов Kylix3. После того, как я скопировал эту библиотеку в директорию `/usr/lib/`, все пришло в норму. Исполняемый файл `konturl` стал запускаться на выполнение и из командной строки консоли, и из файлового менеджера **KDE**.

Вариант исполняемого файла, созданного и отлаженного в среде Kylix3 (Delphi IDE), запускался нормально (с самого начала) только под управлением Linux Mandrake 8.2. Под Linux Mandrake 9.1 и под Linux Mandrake 10.1 он не запускался.

Точно такие же результаты были получены и при работе с Kylix-trial.

Так что, на мой взгляд, среду программирования Kylix3_open и Kylix-trial можно использовать для создания программ, которые будут использоваться на своем компьютере, или программ для нужд других пользователей, на компьютерах которых установлен Linux Mandrake 8.2 или Red Hat 7.2. Следует также учитывать тот факт, что размеры исполняемых файлов даже очень небольших приложений приближаются к 1Мбайт.

Работа с библиотекой Qt

Некоторые понятия

В операционных системах Linux Mandrake и Red Hat изначально закладывается возможность создавать компьютерные программы с помощью среды программирования Qt. Дело в том, что оконная среда (оболочка) **KDE** как раз и создана с помощью компонентов Qt-библиотеки. Напомним, что в Kylix также задействованы подключаемые заголовочные файлы, названия которых начинаются с буквы «Q» — главного признака этой широко применяемой среды программирования, созданной фирмой *Trolltech*.

Существует две версии Qt — коммерческая и бесплатная. Бесплатная версия входит в состав практически всех известных дистрибутивов операционной системы Linux.

Существует два варианта создания компьютерных программ в среде Qt.

- Первый из этих вариантов заключается в создании исходных кодов проекта программы полностью в тестовом редакторе. При этом вы должны знать заранее о том, как написать строку исходного кода для выполнения того или иного действия, т.е. вы должны, приступая к программированию, знать основы Qt. Этот вариант обычно используют профессиональные программисты и советуют всем начинать создание программ именно этим методом.
- Второй метод основан на том, что интерфейсная часть программы создается с помощью входящего в состав **KDE** так называемого Qt-дизайнера — Qt Designer. Интерфейсной частью программы будем называть рабочее окно программы с установленными на нем различными кнопками, окнами редактирования и другими компонентами, необходимыми для управления работой программы, ее «невидимой», основной частью. После создания в графической среде изображения рабочего окна программы, в текстовом редакторе следует написать исходные коды функций на языке программирования C++, они должны служить реакцией на действия того или иного компонента.

Сразу следует запомнить новые названия, специфические для Qt. Все графические примитивы, называемые в Kylix и C++ Builder компонентами, здесь называются *виджетами*. Функции, являющиеся реакциями на действия компонентов называются *слотами*. Также существуют *сигналы*, которые передаются от виджетов к слотам. Если быть точным, то *слоты* — это те программные формы, в которые вставляются функции обработчиков событий.

Чтобы операционная система Linux была настроена на создание компьютерных программ в среде Qt, при установке операционной системы нужно выбрать режим работы компьютера, называемый **Разработка** и в качестве главной оконной среды выбрать среду **KDE**. Однако в версии Linux Mandrake 9.1 такой подход не всегда гарантирует создание необходимых условий работы. Часто в этой версии Linux возникают самые разнообразные проблемы, порой неразрешимые. Например, в версии Mandrake 10.1 таких проблем не было. По-видимому, в Mandrake 9.1 или не полностью укомплектована среда программирования Qt, или она плохо «прописана».

И еще. Начиная с версии qt-x11-free-3.2 в среду программирования Qt внесены большие изменения, которые делают все последующие их версии не совместимыми с более ранними версиями, начиная с версии qt-x11-free-3.1.1. В то же время среда Qt, начиная с версии 3.2, позволяет создавать так называемые кроссплатформенные (или межплатформенные) приложения, которые могут работать в нескольких различных операционных системах. В Internet на сайте фирмы *Trolltech*, а также на многих «зеркальных» сайтах можно найти и скачать различные версии среды Qt. На момент написания этой статьи самой последней была версия qt-x11-free-3.3.4.

Программируем в Qt

В этом разделе рассмотрим процесс создания компьютерной программы с помощью среды Qt Designer под управлением операционной системы Linux Mandrake 10.1, в которую установлена среда qt-x11-free-3.3.3.

На рис. 7.5 показано рабочее окно интегрированной среды разработки Qt Designer.

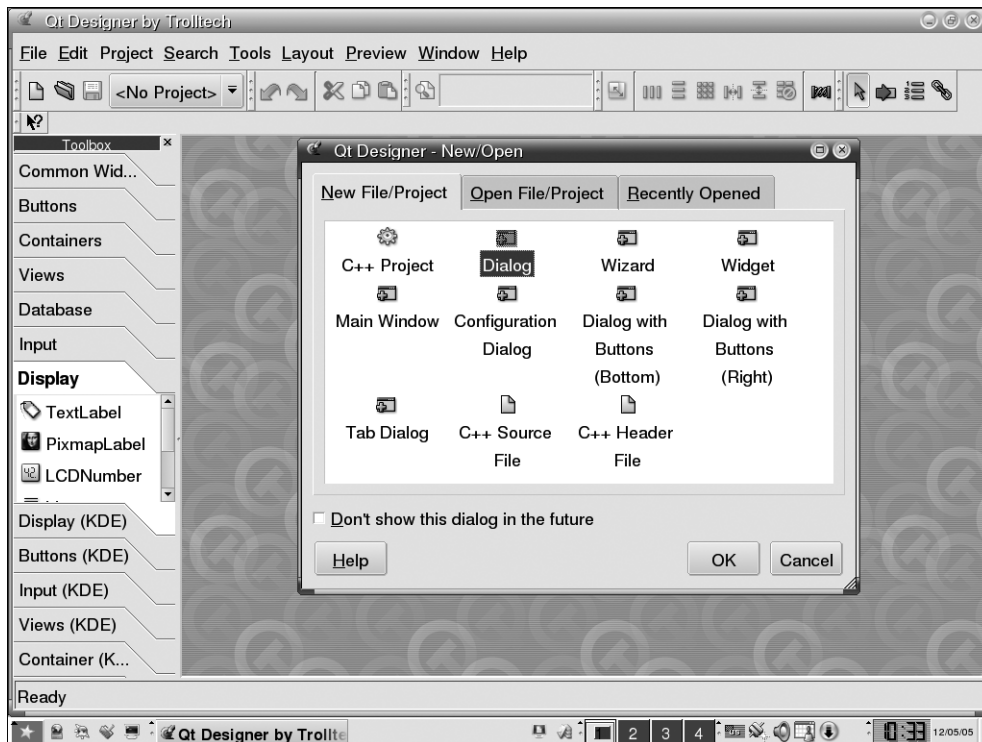


Рис. 7.5. Рабочее окно Qt Designer

По центру рабочего окна размещается диалоговое окно выбора типа создаваемого проекта. Нами будет создаваться проект типа **Dialog**.

Слева на рабочем окне показан перечень разделов библиотеки виджетов (компонентов). В самом верхнем разделе располагаются наиболее часто употребляемые виджеты.

Следующая строка **Buttons** содержит самые разные кнопки.

И так далее. Не будем останавливаться подробно на описании — все это запросто осваивается самостоятельно, а для детального рассмотрения потребуется слишком много места. Желая более подробно разобраться в процессе создания компьютерных программ в среде Qt советую скачать книгу Ж. Бланшет и М. Саммерфилда «Разработка графического интерфейса с помощью библиотеки Qt3» в Internet с сайта www.linuxcenter.ru. В этой книге очень подробно описан весь процесс создания компьютерных программ в среде Qt, в том числе и с использованием Qt Designer.

Перед началом создания проекта Qt-приложения следует создать директорию, в которой будут находиться все файлы проекта. Названием директории (папки) должно быть название нашего будущего приложения. Дадим название для директории `/prob3_qt/`. Под этим именем в дальнейшем автоматически средой программирования будет создан файл проекта и исполняемый файл программы.

Первым делом необходимо дать имя рабочей форме проекта, которое будет служить именем основного класса и файлов создаваемых автоматически средой программирования. В данном случае устанавливаем для формы имя `Proba3`. Затем приступаем к установке в форму виджетов.

На рис. 7.6 показано рабочее окно Qt Designer, с установленными на форме виджетами, которые будут составлять интерфейс будущей программы.

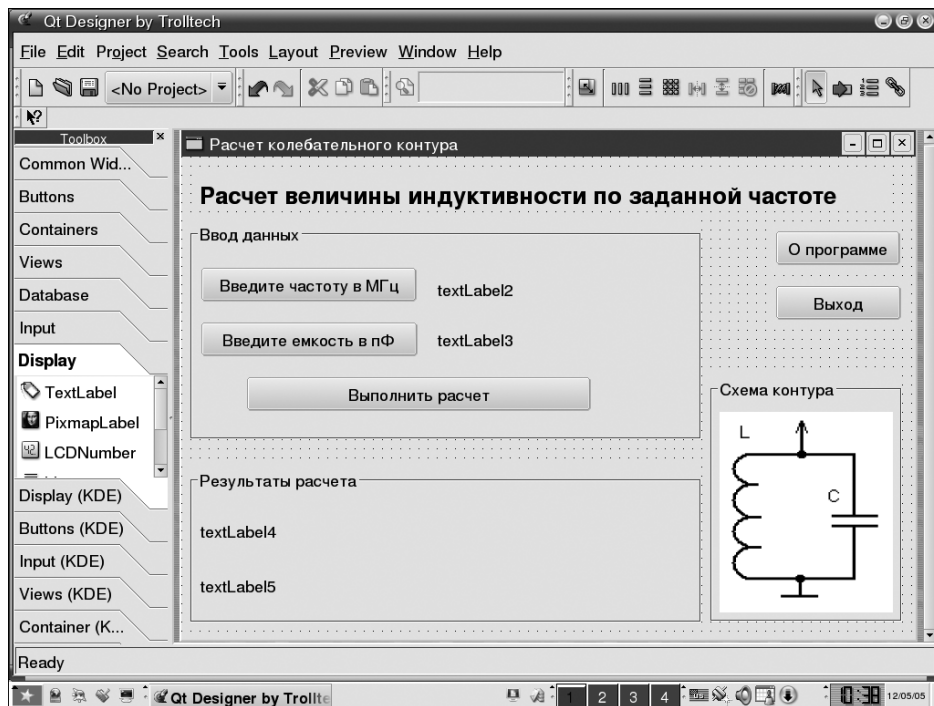


Рис. 7.6. Рабочее окно с заполненной формой

Рассмотрев этот рисунок, вы наверняка заметили много общего с между изображенной на этом рисунке формой и формой рис. 7.2. Дело в том, что мною принято решение показать

вам решение одной и той же задачи различными методами, в двух различных средах программирования.

Для того чтобы установленные на форме виджеты не портили облик рабочего окна программы при изменении его размера, существуют специальные приемы и компоненты, позволяющие стабилизировать размеры и места расположения виджетов на форме вне зависимости от размеров самой формы. Команды для точного расположения виджетов в форме находятся в разделе **Layout** главного меню. Однако в данном приложении, для простоты, мною применяется другой прием для стабилизации размеров и места расположения виджетов на форме — все виджеты устанавливаются внутри специальных *контейнеров*. В данном случае используется контейнер типа `groupBox`, который представляет собой рамку с текстом надписи.

Создаваемая нами в этом разделе программа называется `Proba3` и предназначена для расчета величины индуктивности контурной катушки по заданным величинам частоты и емкости.

Свойства установленных в форму виджетов выбираем в окне **Property Editor**, которое можно открыть с помощью команды **Window**⇒**Views**⇒**Property Editor/Signal handlers**. На рис. 7.7 показано окно **Property Editor/Signal handlers** на фоне формы.

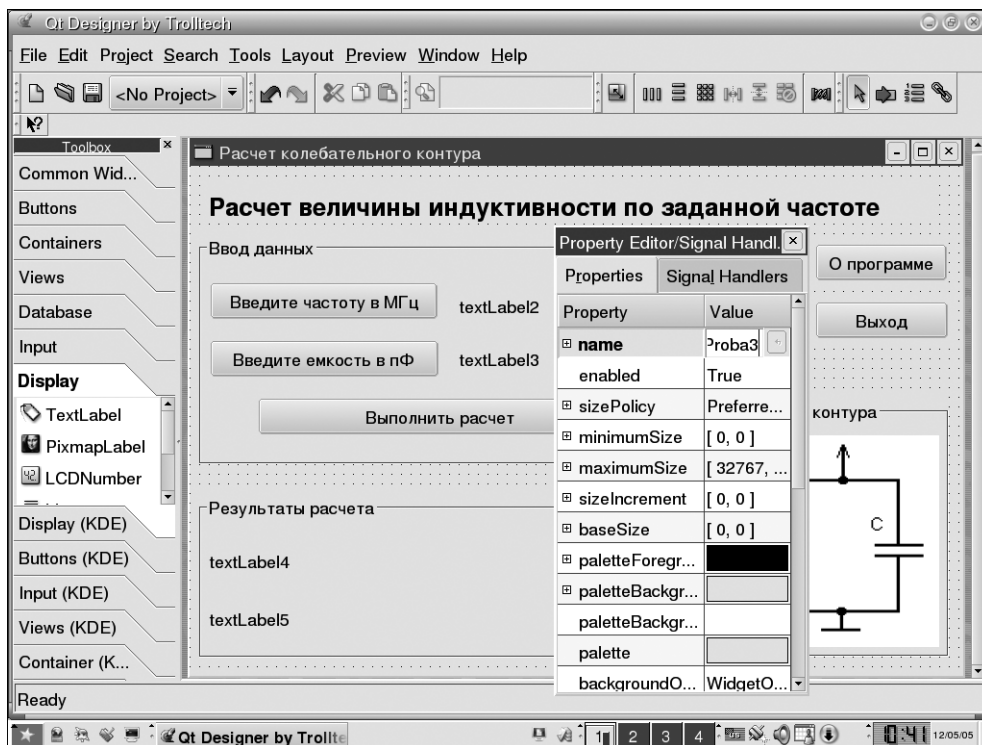


Рис. 7.7. Окно **Property Editor/Signal handlers**

После установки на форму всех виджетов следует назначить примерный порядок включения каждого из виджетов в работу, т.е. установить так называемый «по-

рядок навигации». Для этого выбираем команду **Tools**⇒**Tab Order**, в результате чего на каждом из виджетов, которые могут принимать фокус, появляются цифры в синих кружочках, как это показано на рис. 7.8.

Желаемый порядок очередности срабатывания виджетов устанавливается щелчками мышью и клавишей <Tab>. Для выхода следует нажать клавишу <Esc>.

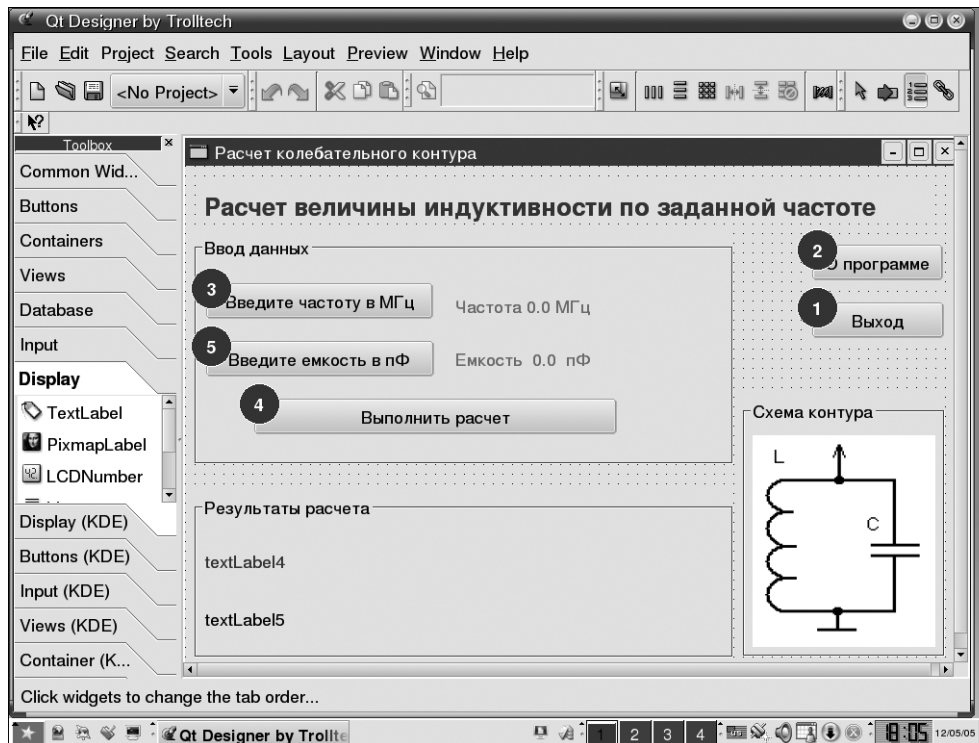


Рис. 7.8. Установка порядка навигации среди виджетов

После этих процедур следует сохранить созданную форму в заданной директории. При этом средой программирования автоматически будет создан файл `proba3.ui`, в котором будет сохранена форма с установленными на ней виджетами.

Теперь наступила очередь создания файла с описанием исходных кодов для каждого из виджетов. Эта работа выполняется в специальном текстовом редакторе, который открывается двойным щелчком мыши в любом месте формы, свободном от виджетов. После этого среда запрашивает о создании файла `proba3.ui.h`, в котором будут находиться исходные коды слотов.

На рис. 7.9 показано рабочее окно IDE с текстовым редактором.

В текстовом редакторе должны быть созданы функции (слоты), которые должны обрабатывать сигналы, поступающие в программу от виджетов. В листинге 7.4 приведен текст исходных кодов создаваемых нами слотов.

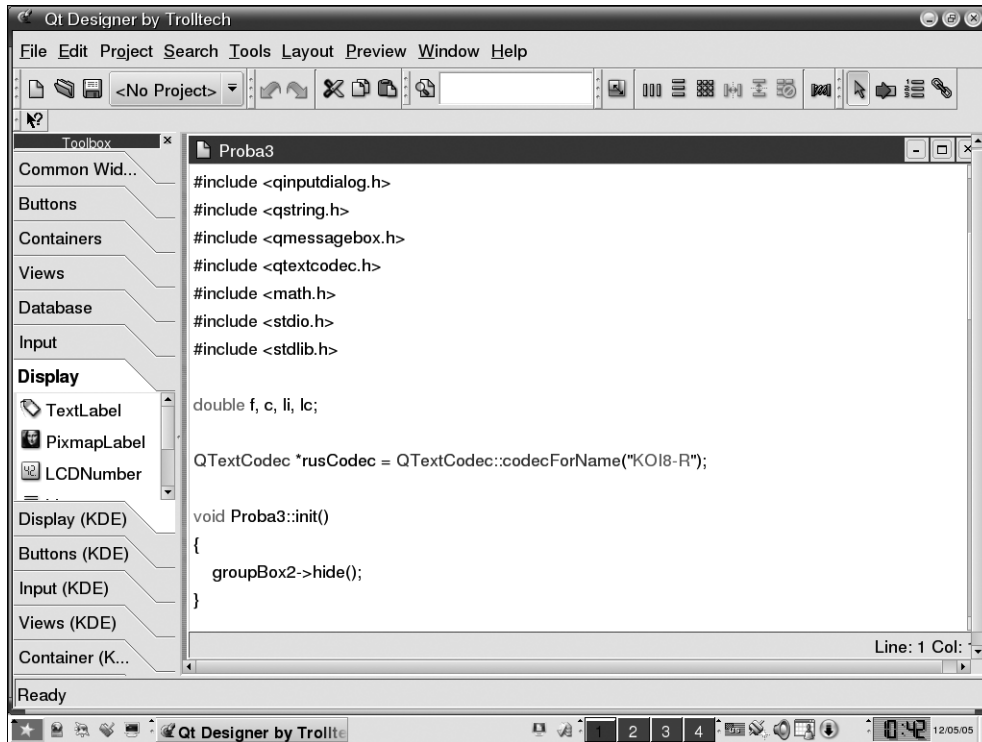


Рис. 7.9. Окно с редактором текста

Листинг 7.4. Текст файла *proba3.ui.h*

```

/*****
**
** ui.h extension file, included from the uic-generated form
** implementation.
**
** If you want to add, delete, or rename functions or slots, use
** Qt Designer to update this file, preserving your code.
**
** You should not define a constructor or destructor in this file.
** Instead, write your code in functions called init() and destroy().
** These will automatically be called by the form's constructor and
** destructor.
*****/

#include <qinputdialog.h>
#include <qstring.h>
#include <qmessagebox.h>
#include <qtextcodec.h>
#include <math.h>

```

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

double f, c, li, lc;
    // Следующая строка разрешает в строках типа QString
    // использовать русские буквы
QTextCodec *rusCodec = QTextCodec::codecForName("KOI8-R");
//-----
void Proba3::init()                                // 1
{
    groupBox2->hide(); // скрывает нижний контейнер groupBox2
}
//-----
void Proba3::about()                                // 2
{
    // Следующая строка разрешает в функциях tr()
    // использовать русские буквы
    QTextCodec::setCodecForTr(QTextCodec::codecForName("KOI8-R"));
    QMessageBox::about(this, tr(" О программе PROBA3"),
        tr(" <p> Программа   <h2>PROBA3</h2>"
            "<p>Автор:  Геннадий – <b>R3XB</b>  &copy;  2005г"
            "<p>http://ra3xb.narod.ru;  mailto:r3xb@kaluga.ru"));
}
//-----
void Proba3::buton1()                                // 3
{
    bool ok;
    QString ss01 = rusCodec->toUnicode("Введите частоту в МГц");
    f=QInputDialog::getDouble("Proba3", ss01, 0,
                                0, 10000, 2, &ok, this);

    QString ss1;
    ss1 = rusCodec->toUnicode("Величина частоты f =
                                %1МГц").arg(f,0, 'g', 10);

    textLabel2->setText(ss1);
}
//-----
void Proba3::buton2()                                // 4
{
    bool ok;
    QString ss02 = rusCodec->toUnicode("Введите емкость в пФ");

    c=QInputDialog::getDouble("Proba3", ss02, 0,
                                0, 10000, 2, &ok, this);

    QString ss2;
```

```

        ss2 = rusCodec->toUnicode("Величина емкости C =
                                   %1пФ").arg(c, 0, 'g', 10);
        textLabel3->setText(ss2);
    }
//-----
void Proba3::buton3()                                     // 5
{
    lc = pow((159/f), 2);
    li = lc/c;
    QString ss3, ss4;
    ss3 = rusCodec->toUnicode("Величина индуктивности L =
                              %1мкГн").arg(li, 0, 'g', 10);
    ss4 = rusCodec->toUnicode("Вспомогательная величина LC =
                              %1").arg(lc, 0, 'g', 10);
    textLabel4->setText(ss3);
    textLabel5->setText(ss4);
    groupBox2->show();
}
//-----

```

В начале листинга 7.4 приведены комментарии на английском языке, которые создаются средой программирования. В них разъясняется, что в этот файл можно включать собственные функции, которые обрабатываются в среде Qt Designer совместно с файлом описания виджетов, установленных на форме. Кроме того, можно создавать функции с названиями `init()` и `destroy()`, которые будут обрабатываться соответственно конструктором и деструктором класса формы.

После этих разъяснений описаны подключаемые заголовочные файлы, в которых описываются функции, задействованные в этом файле.

Первая функция (комментарий `// 1`) `void Proba3::init()` вызывается и используется при создании рабочего окна программы. В тело этой функции запишем команду, которая скрывает контейнер `groupBox2` вместе с размещенными внутри него виджетами сразу при создании рабочего окна программы.

Функция 2 носит название `about()` и служит для вывода на экран диалогового окна с информацией о данной программе по нажатию кнопки **О программе**.

Функция 3 является реакцией на нажатие кнопки **Введите частоту в МГц**, а следующая за ней функция 4 является реакцией на нажатие кнопки **Введите емкость в пФ**.

Функция 5 является реакцией на нажатие кнопки **Выполнить расчет**. Здесь выполняются математические вычисления, а последняя строка делает видимым скрытый в функции 1 контейнер `groupBox2`.

После того, как нами написаны исходные тексты функций, необходимо все эти функции (кроме первой, которая изначально подключена к конструктору) превратить в слоты, т.е. подключить к соответствующим виджетам, реакциями которых они являются. Для выполнения этой процедуры в главном меню выберите команду

Edit⇒Edit Slots (Правка⇒Редактирование слотов), после чего появляется окно **Edit Functions** (Редактирование функций), показанное на рис. 7.10.

Нажимаем в этом окне кнопку **New Function** (Новая функция) и вводим в строку первую функцию — `about()`. Затем точно таким же образом вводим все последующие созданные нами функции, которые являются реакциями на действия имеющихся в форме виджетов. Таким эти функции будут обозначены как слоты.

Далее необходимо соединить сигналами виджеты с соответствующими слотами. Для этого выбираем в главном меню **Edit⇒Connections** (Правка⇒Соединения), в результате чего появляется диалоговое окно **View and Edit Connections** (Просмотр и редактирование соединений), изображенное на рис. 7.11.

Щелкните мышью на кнопке **New** (Новая) и введите сначала в колонку **Sender** (Передачик) название виджита, к которому нужно подключить слот, а во все строки колонки **Signal** (Сигнал) функцию `clicked()`. В колонку **Receiver** (Приемник) введите во все строки имя формы — `Proba3`. В последней колонке **Slot** (Слот) вводите имя функции, соответствующей данному виджету.

Теперь снова сохраните все в выбранной директории `proba3_qt`.

Если посмотреть эту директорию, то можно обнаружить там только два файла — файл `proba3.ui` и файл `proba3.ui.h`.

Для полного комплекта необходимо еще создать в текстовом редакторе файл `main.cpp` (листинг 7.5), и разместить его в этой же директории `proba3_qt`.

Листинг 7.5. Файл `main.cpp`

```
//-----
#include <qapplication.h>
#include "proba3.h"
#include <qtextcodec.h>
```

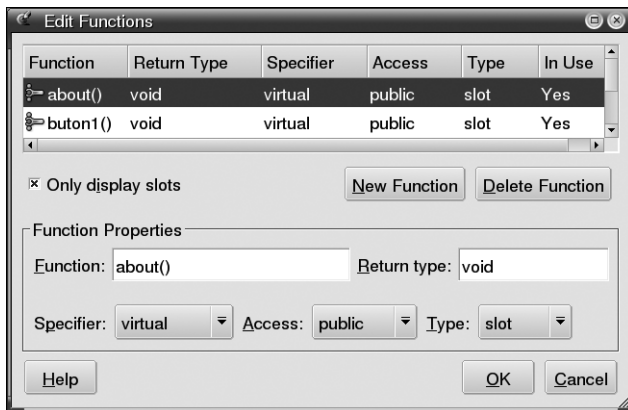


Рис. 7.10. Окно Edit Functions

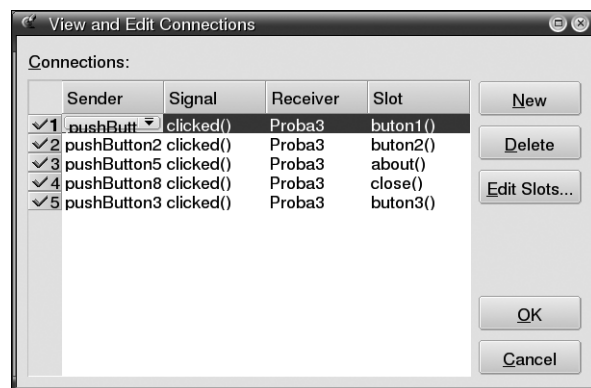


Рис. 7.11. Окно редактора связей

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Proba3 *dialog = new Proba3;
    a.setMainWidget(dialog);
    dialog->show();
    return a.exec();
}
//-----
```

Этот файл объединяет в единую рабочую среду все три файла, находящиеся в заданной директории проекта.

Заключительным этапом программирования является создание исполняемого файла проекта.

Для этого, находясь в директории проекта, вводим в командной строке команду `qmake-project`, которая создает файл проекта `prob3_qt.pro`. Затем вводим команду `qmake prob3_qt.pro`, в результате выполнения которой должен быть создан файл `Makefile`.

После того, как будет создан файл `Makefile`, следует ввести команду `make`, по которой `Makefile` начинает свою работу, результатом которой должно быть создание исполняемого файла программы `prob3_qt`, который создается только в том случае, если не допущено ни единой ошибки.

Исполняемый файл запускается в работу обычным образом, т.е. либо из командной строки, либо из файлового менеджера **KDE**. На рис. 7.12 показано рабочее окно программы сразу после запуска в работу.

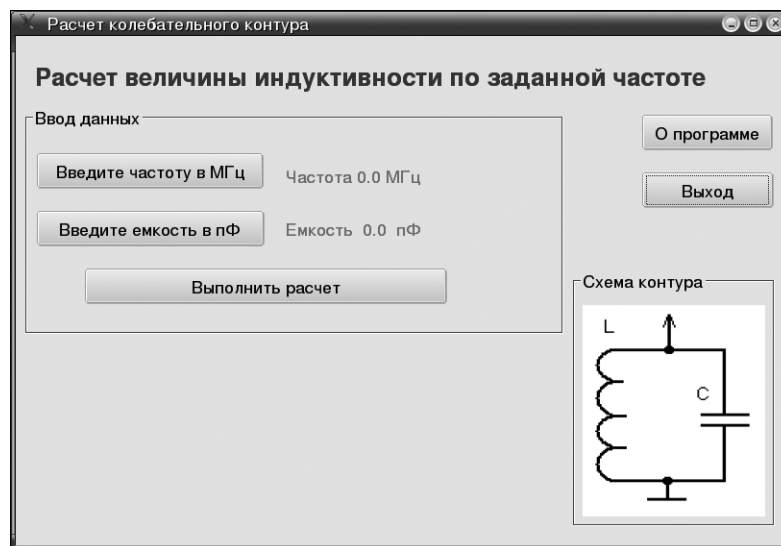


Рис. 7.12. Начальный вид рабочего окна

На рис. 7.13 на фоне главного рабочего окна показано диалоговое окно, предназначенное для ввода в программу частоты.

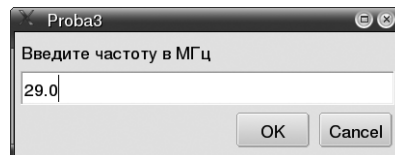


Рис. 7.13. Диалоговое окно для ввода частоты

На рис. 7.14 изображено рабочее окно программы после щелчка мышью на кнопке **Выполнить расчет**.

На рис. 7.15 показано диалоговое окно с информацией о программе и ее авторе.

На этом процесс создания программы в среде программирования Qt закончен. Далее необходимо создать текстовые файлы `INSTALL` и `README`. Если программа довольно сложная, то нужно также создать справочник с описанием всех особенностей программы.

Кроме того, следует знать, что оконная система **KDE** также может служить средой для создания компьютерных программ, в том числе и совместных программ KDE + Qt.

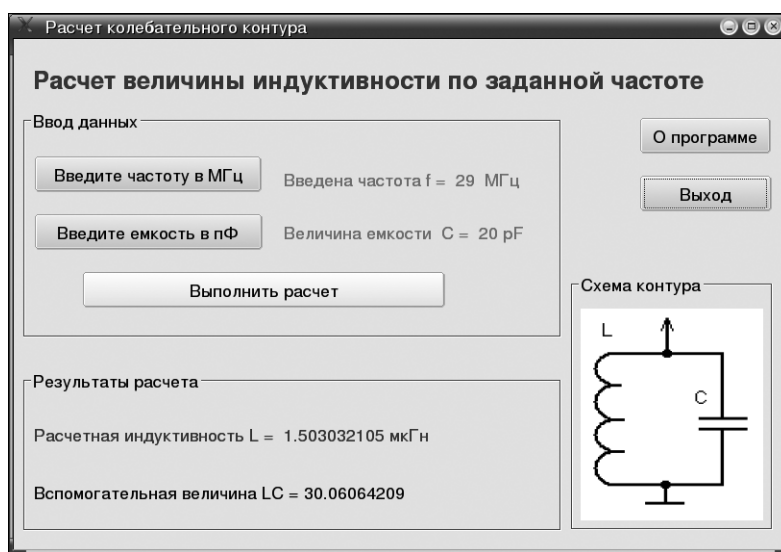


Рис. 7.14. Главное окно программы

Как запустить на выполнение Qt-приложение

Приложение, созданное в предыдущем разделе с помощью среды Qt, согласно заверению разработчиков этой среды, можно запустить на выполнение в самых разных операционных системах. Необходимым условием является только наличие на компьютере работоспособной Qt-библиотеки.

Проверить наличие такой библиотеки можно путем ввода в командную строку команды: `locate qmake | grep bin`.

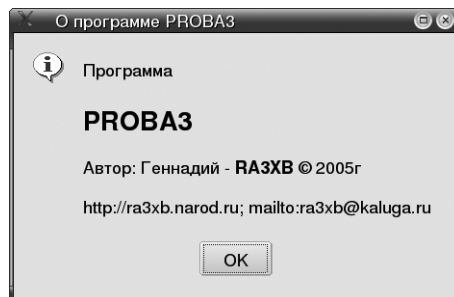


Рис. 7.15. Диалоговое окно О программе

Если библиотека обнаружена, то в ответ система выведет путь к библиотеке:

```
/usr/lib/qt3/bin/qmake
```

или что-то в этом роде, например, Red Hat 9.0 может вывести следующие сообщения:

```
/usr/bin/qmake;  
/usr/bin/qmake3;  
/usr/lib/qt-3.1/bin/qmake.
```

Если система выдала сообщение о том, что эта команда неизвестна, то необходимо обновить (или создать заново) базу данных задействованных в системе команд. Это делается следующим образом. Сначала в командной строке вводится команда: `su`.

В ответ на которую система запрашивает у вас пароль суперпользователя. После ввода пароля вводим команду для создания базы данных: `updatedb`.

Если после создания (или обновления) базы данных система снова отказывается признавать команду `qmake`, то Qt-библиотека на вашем компьютере либо не установлена, либо установлена очень старая версия, в которой `qmake` отсутствует. В этом случае нужно установить на компьютер более новый вариант Linux, или скачать Qt из Internet и установить на свой компьютер. Я думаю, что первый вариант более удобный и надежный.

В лучшем случае, когда все в порядке и система признает команду `qmake`, запустите на выполнение новое Qt-приложение совсем не трудно.

Рассмотрим вариант ваших действий, когда вы скачали с сайта исходные коды программы `proba3`.

1. Создаем директорию под названием `prob3_qt`.
2. В эту директорию скопируем три основных файла, взятых вами из пакета той программы, исполняемый файл которой вы хотите иметь на своем компьютере. Первый из этих файлов — файл `main.cpp`, второй — файл с расширением `ui`, третий — файл с расширением `ui.h`. В рассматриваемом случае это файлы `main.cpp`, `proba3.ui` и `proba3.ui.h`.
3. Находясь в заданной директории, вводим в командной строке команду `qmake-project`, которая создает здесь же файл проекта `prob3_qt.pro`.
4. Затем вводим команду `qmake prob3_qt.pro`, в результате выполнения которой должен быть создан файл `Makefile`.
5. После того, как будет создан файл `Makefile`, следует ввести команду `make`, по которой `Makefile` начинает свою работу, результатом которой должно быть создание исполняемого файла программы `prob3_qt`, который запросто создается, если не было допущено ни единой ошибки при написании текстов исходных кодов функций — слотов.

Полученный исполняемый файл запускается в работу обычным образом, т.е. из командной строки по команде `./prob3_qt` или в файловом менеджере с помощью двойного щелчка мышью на картинке файла.

Коротко о Qt

Qt — мультиплатформенная C++ GUI (Graphical User Interface — графический интерфейс пользователя) прикладная структура. Другими словами можно сказать, что Qt — это среда для создания интерфейса пользователя к компьютерным программам, способная работать на разных компьютерах и под управлением различных операционных систем.

Qt 3.3 вводит новые особенности и много усовершенствований по сравнению с рядом Qt 3.2.x. Приложения Qt ряда версии 3.3 в двоичном исполнении совместимы с приложениями ряда 3.2.x, при этом созданные приложения для 3.2 продолжают работу с Qt 3.3.

Эти утверждения были проверены на следующих платформах:

win32-borland	win32-msvc.net	freebsd-g++
win32-g++	aix-g++	freebsd-icc
win32-icc	aix-xlc	hpux-acc
win32-msvc	aix-xlc-64	hpux-g++
irix-cc	linux-icc	solaris-g++-64
irix-cc-64	solaris-cc	tru64-g++
irix-g++	solaris-cc-64	macx-g++
linux-icc-64	solaris-g++	macx-pbuilder
linux-g++		

Если вы хотите использовать Qt 3 на неподдерживаемой им версии операционной системы Unix, попробуйте войти в контакт с разработчиками по адресу qt-bugs@trolltech.com.

Как получить пакет Qt фирмы *Trolltech AS*.

Qt Open Source Edition. Скачайте архив `.tar.gz` с сайта <ftp.trolltech.com>. Для более быстрого скачивания, используйте, например `ftpsearch`, и найдите `qt-x11-free-3.3.4`.

Qt Professional Edition или **Qt Enterprise Edition.** Чтобы приобрести эти коммерческие пакеты, необходимо сначала получить по электронной почте инструкции о том, как получить новое исполнение Qt. Для контакта обращайтесь на sales@trolltech.com.

О любых проблемах, с которыми вы сталкиваетесь при работе с Qt 3.3, можно сообщить по адресу qt-bugs@trolltech.com.

Как установить Qt на компьютере с операционной системой Linux

Прежде чем начать устанавливать Qt-библиотеку и примеры программ на свой компьютер, необходимо запустить сценарий `configure`, чтобы установить информацию о платформе и выявить другие особенности системы. Это позволит правильно выбрать компилятор для установки и компиляции системы. Перечислим поддерживаемые платформы и компиляторы: Aix-g++ hpux-g++ linux-g++ solaris-cc-64 win32-g++ aix-xlc hpux-g++-64 linux-g++-64 solaris-g++ win32-icc aix-xlc-64 irix-cc linux-icc solaris-g++-

```
64 win32-msvc freebsd-g++ irix-cc-64 macx-g++ tru64-cxx win32-
msvc.net freebsd-icc irix-g++ macx-pbuilder tru64-g++ hpux-acc
linux-ecc-64 solaris-cc win32-borland
```

Если при установке и компиляции Qt 3.x возникнут проблемы, ищите ответы и замечания для разных платформ в Internet на сайте www.trolltech.com/developer/platforms/, где выкладывается информация о всех известных проблемах, о которых поступает информация по электронной почте.

Пример использования сценария `configure`. Введите следующую строку:
`./configure -platform irix-cc-64 -shared -debug.`

Для создания собственной конфигурации системы, можно добавить новые файлы в директорию `mkspecs`. При этом используйте уже существующие конфигурации как «отправные точки» для новой настройки.

Установка Qt/X11 версии 3.3.4

В этом разделе будет описано, как можно установить библиотеку Qt/X11 в директорию `/usr/local/`, которая обычно рекомендуется для установки. Однако установку надо скорректировать в том случае, если выбрана другая директория для установки библиотеки, и вполне возможно, что при этом необходимо зарегистрироваться как `root`. В большинстве случаев этого не требуется.

1. Скачайте файл библиотеки `qt-x11-free-3.3.4.tar.gz` из Internet и скопируйте его в директорию `/usr/local/`. Перейдите в эту директорию с помощью команды: `cd /usr/local`.

Далее следует распаковать файл библиотеки:

```
gunzip qt-x11-free-3.3.4.tar.gz;
tar xvf qt-x11-free-3.3.4.tar.
```

Так создается готовый к работе пакет библиотеки Qt/X11, содержащийся в директории `/usr/local/qt-x11-free-3.3.4`. В этом пакете все файлы находятся в исходных кодах, которые далее необходимо привести в рабочее состояние.

Для удобства дальнейшего обращения к пакету переименуем его (`qt-x11-free-3.3.4`) в более краткое обозначение `qt` (или в `qt3`) с помощью следующей команды:

```
mv qt-x11-free-3.3.4 qt.
```

2. Теперь следует напомнить, что все файлы библиотеки Qt/X11 находятся в директории `/usr/local/qt`.

Вполне возможно, что вам потребуется прописать основные переменные среды окружения Qt/X11 в файле `.profile` (или `.login`, в зависимости от вашей оболочки) в вашей домашней директории. Если такого файла в вашей директории нет, то его следует создать в текстовом редакторе. В новом файле должны быть прописаны следующие пути:

```
QTDIR      — это основная директория Qt;
PATH       — это путь к программе moc и другим инструментам Qt;
MANPATH    — это путь к Qt man страницам;
LD_LIBRARY_PATH — путь к Qt подключаемым библиотекам.
```

Замечание. При работе под IRIX нужно дополнить пути доступа к файлам библиотеки цифрами LD_LIBRARYN32_PATH или LD_LIBRARY64_PATH. Чтобы точно установить переменную, которая соответствует вашей конфигурации, смотрите файл rld (5) man page для получения большего количества информации.

Если используется оболочка — bash, ksh, zsh или sh, то необходимо в файл .profile добавить следующие строки:

```
QTDIR=/usr/local/qt;
PATH=$QTDIR/bin:$PATH
MANPATH=$QTDIR/doc/man:$MANPATH;
LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH;
export QTDIR PATH MANPATH LD_LIBRARY_PATH.
```

Если используется оболочка csh или tcsh, то необходимо в файл .login добавить следующие строки:

```
setenv QTDIR /usr/local/qt;
setenv PATH $QTDIR/bin:$PATH;
setenv MANPATH $QTDIR/doc/man:$MANPATH;
setenv LD_LIBRARY_PATH $QTDIR/lib:$LD_LIBRARY_PATH.
```

После выполнения приведенных выше операций необходимо перезапустить консоль для того, чтобы система могла использовать для установки пути прописанные в переменных ее окружения \$QTDIR и \$PATH. В противном случае установка будет выполнена неправильно, а во время установки могут появиться сообщения об ошибках.

3. Если устанавливается лицензионная копия среды, то следует установить файл с лицензией как \$HOME/.qt-license. Для free-версии файл лицензии не нужен.

4. Создание собственно Qt-библиотеки. На этом этапе создается собственно Qt-библиотека, создаются примеры программ, обучающая программа, и инструменты (например Qt Designer).

Первая команда, ./configure, по этой команде производится конфигурирование Qt-библиотеки для конкретного ПК. Для получения справки по команде ./configure следует записать команду как ./configure -help.

При установке Qt на разные платформы необходимо выполнять ./configure команду с разными параметрами, которые описаны в файле PLATFORMS.

Скорость выполнения команды ./configure зависит от быстродействия компьютера.

Создавать библиотеку, все примеры и обучающую программу следует с помощью команды make.

Если используемая платформа или компилятор не поддерживаются операционной системой, можно обратиться к описанию их работы в Internet на сайте www.trolltech.com/platforms/.

Команда make выполняется довольно долго. Например, на компьютере с процессором Celeron 2700 MHz выполнение этой команды занимает два часа, на компьютере с Celeron 800 — порядка шести часов.

Если выполнение двух предыдущих команд произошло без инцидентов, то следующей командой должна быть:

```
make install.
```

По этой команде выполняется установка всех файлов из пакета Qt именно в те директории на диске вашей машины, для работы в которых эти файлы предназначены.

Команда `make install` выполняется на компьютере с процессором Celeron 2700 MHz в течение примерно 20 минут, на компьютере с Celeron 800 — порядка 70 минут.

5. Если используются разделяемые библиотеки и система выведет следующее сообщение (или подобное ему):

```
Can't load library 'libqt.so.3'
(Не может прочитать библиотеку 'libqt.so.3').
```

Тогда необходимо зарегистрироваться в системе как `root`, чтобы смочь работать с файлом `/sbin/ldconfig`. И не забудьте установить переменную окружения `LD_LIBRARY_PATH` как объяснено выше, в пункте 2.

6. HTML-документация установлена в директорию `/usr/local/qt/doc/html/`, главная страница находится в файле — `/usr/local/qt/doc/html/index.html`, рабочие `man`-страницы установлены в директорию `/usr/local/qt/doc/man/`.

Программирование в GTK

Если программирование в среде Qt выполняется на базе языка программирования C++, то программирование в среде GTK выполняется на базе стандартного языка Си. Программирование в этой среде также может быть выполнено или в режиме ручного написания исходных кодов программы, или с использованием интегрированной среды разработки Glade или Glade2, работающих под управлением оконной среды Gnome.

Некоторые из возможностей сред Qt Designer и Glade похожи.

Немного о GTK

Средствами только одного языка программирования Си нормальный графический интерфейс пользователя (GUI — Graphical User Interface) не построишь, тем более что после красочных интерфейсов, задействованных в Windows-приложениях, пользователь стал очень требователен не только к наличию этого самого GUI, но еще и к дизайну формы (виду окна программы). Поэтому без дополнительных библиотек, позволяющих создавать красочные трехмерные компоненты, не обойтись. Самыми распространенными библиотеками для создания GUI являются библиотеки GTK и Qt. Рекомендуется использовать только эти библиотеки, поскольку велика вероятность того, что они уже будут установлены и на вашем компьютере и у пользователя (GNOME и KDE установлены почти у всех), чтобы не сложилась такая ситуация, что размер программы 300 Кбайт, а для запуска этой программы требуется дополнительная библиотека размером 20 Мбайт.

Скорее всего, GTK уже установлена на компьютер, но если это не так, придется установить пакет `gtk+-devel`, содержащий необходимые файлы для разработки GTK-программ.

Элементы GUI — это кнопки, поля ввода, переключатели и тому подобное, что называется, как и в Qt, виджитами. Если вы знакомы с C++ Builder или работали в Delphi, виджеты подобны визуальным компонентам в C++ Builder или в Delphi.

Как и в Delphi, основным элементом GUI является главное окно (или форма). Виджеты для размещения в окне помещаются в контейнер. В самом окне выравнивать виджеты можно с помощью вертикальных/горизонтальных боксов или же таблиц. Автору приведенного ниже примера больше нравится второй способ, поэтому мы будем использовать именно таблицы.

Виджеты могут реагировать на сигналы, например, щелчок мышью. При этом вызывается функция-обработчик события (сигнала), если вы определили ее.

Если в предыдущем разделе этой главы мною был приведен пример создания программы с помощью интегрированной среды разработки, в этом разделе будет показан процесс создания интерфейса к какой-то программе путем ручного написания исходных кодов. В качестве примера приведу текст исходных кодов программы, опубликованный в Internet Денисом Колисниченко.

В качестве примера Денис приводит исходные коды небольшой программы-конфигуратора, которая вносит изменения в файл `/etc/resolv.conf`. Напомню, что формат этого файла следующий:

```
domain firma.ru;
nameserver 192.168.0.1;
nameserver 192.168.0.2.
```

Директива `domain` определяет имя домена, а две директивы `nameserver` — первый и второй DNS-серверы, соответственно. Помните, что создание файла `resolv.conf` не главное в этом разделе. Главная цель заключается в том, чтобы показать на примере возможности довольно простого метода создания графического интерфейса (GUI) для какой-то программы.

Перейдем к практике

На рис. 7.16 изображено рабочее окно уже готовой программы. Работает она следующим образом. Когда пользователь введет что-нибудь в поле ввода и нажмет клавишу `<Enter>`, программа отобразит введенный им текст на консоли. Когда пользователь щелкнет мышью на кнопке **Ok**, введенная им информация будет еще раз выведена на консоль и записана в файл. Кнопка **Quit** предназначена для завершения работы программы. Программа завершает работу и с помощью кнопки закрытия окна — в GTK программист сам определяет реакции на стандартные кнопки диалогового окна.



Рис. 7.16. Окно программы Resolver

В листинге 7.6 приведен текст файла `resolv.c`. Внимательно прочитайте комментарии в этой программе.

Листинг 7.6. Текст файла *resolv.c*

```

#include <gtk/gtk.h>
#include <stdlib.h>
#include <stdio.h>

gchar *domain, *dns1, *dns2;

/* Массив из трех полей ввода. Первое предназначено для ввода имени
домена, два вторых — [1] и [2] — для ввода IP-адресов серверов DNS */
GtkWidget *edit[3];

/* Наш файл */
FILE *resolv;

/* Функция записи в файл */
void writetofile( GtkWidget *widget,
  gpointer data )
{
  /* С помощью функции gtk_entry_get_text() мы получаем
  введенный пользователем текст из полей ввода */

  domain = gtk_entry_get_text(GTK_ENTRY(edit[0]));
  dns1 = gtk_entry_get_text(GTK_ENTRY(edit[1]));
  dns2 = gtk_entry_get_text(GTK_ENTRY(edit[2]));

  /* Выводим прочитанный текст на консоль */
  g_print ("Domain %s\n", domain);
  g_print ("DNS1 %s\n", dns1);
  g_print ("DNS2 %s\n", dns2);

  /* Перезаписываем файл resolv.conf в текущем каталоге */
  if ((resolv = fopen("resolv.conf", "w")) == NULL)
  {
    /* Наверное, нет места на диске или прав маловато... */
    g_print ("ERR: Cannot to open resolve.conf file\n");
    gtk_main_quit ();
  }

  /* Запись в файл */
  fprintf(resolv, "domain %s\n", domain);
  fprintf(resolv, "nameserver %s\n", dns1);
  fprintf(resolv, "nameserver %s\n", dns2);
  fclose(resolv);
}

/* Эта функция будет запущена, когда пользователь нажмет
кнопку закрытия окна или кнопку Quit */
gint delete_event( GtkWidget *widget,
  GdkEvent *event,
  gpointer data )
{
  /* Функция gtk_main_quit() используется для завершения работы
  GTK-программы. Не нужно для этого использовать функцию exit() */

  gtk_main_quit ();
  return(FALSE);
}

```



```

/* Когда пользователь введет текст и нажмет клавишу <Enter>,
   введенный им текст будет выведен на консоль */

void enter_callback( GtkWidget *widget,
GtkWidget *entry )
{
    domain = gtk_entry_get_text(GTK_ENTRY(entry));
    printf("Domain: %s\n", domain);
}

int main( int argc,
char *argv[] )
{
    GtkWidget *window; /* Окно */
    GtkWidget *button; /* Кнопка */
    GtkWidget *table; /* Таблица для размещения виджетов */
    GtkWidget *label; /* Надпись */
    /* Как видите, все виджеты одного типа — GtkWidget, поэтому мы могли бы
       обойтись даже тремя виджетами — для окна, таблицы и для всех остальных
       элементов GUI*/

    int i;

    /* Инициализация любой GTK-программы */
    gtk_init (&argc, &argv);

    /* Создаем новое окно */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* Устанавливаем заголовок окна */
    gtk_window_set_title (GTK_WINDOW (window), "Resolver");

    /* Устанавливаем реакцию на кнопку закрытия окна. Сигнал — delete_event
       Вызываем функцию delete_event(), которая описана выше */
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
GTK_SIGNAL_FUNC (delete_event), NULL);

    /* Устанавливаем рамку окна */
    gtk_container_set_border_width (GTK_CONTAINER (window), 20);

    /* Создаем таблицу 3x3 */
    table = gtk_table_new (3, 3, TRUE);

    /* Помещаем таблицу в контейнер. Обязательно! */
    gtk_container_add (GTK_CONTAINER (window), table);

    /* Рисуем надписи, помещаем их в ТАБЛИЦУ и отображаем.
       Обратите внимание, что в этом случае нам не нужно
       объявлять отдельную переменную для каждой надписи*/

    label = gtk_label_new("Domain: ");
    /* Координаты ячеек рассматриваются после этого листинга */
    gtk_table_attach_defaults (GTK_TABLE(table), label, 0, 1, 0, 1);
    gtk_widget_show (label);

    label = gtk_label_new("DNS #1: ");
    gtk_table_attach_defaults (GTK_TABLE(table), label, 0, 1, 1, 2);
    gtk_widget_show (label);

    label = gtk_label_new("DNS #2: ");
    gtk_table_attach_defaults (GTK_TABLE(table), label, 0, 1, 2, 3);
    gtk_widget_show (label);
}

```

```

/* Заполняем наш массив полей ввода. По аналогии с Delphi, он был назван
   массив edit[] */
for(i=0; i<3; i++)
{
    /* Новое поле */
    edit[i] = gtk_entry_new();

    /* Если забыть этот оператор, пользователь ничего не сможет ввести */
    gtk_entry_set_editable(GTK_ENTRY(edit[i]), 1);

    /* Определяем одну для всех реакцию на сигнал activate – нажатие Enter*/
    gtk_signal_connect(GTK_OBJECT(edit[i]), "activate",
        GTK_SIGNAL_FUNC(enter_callback),
        edit[i]);

    /* Помещаем edit[i] в таблицу */
    gtk_table_attach_defaults (GTK_TABLE(table), edit[i], 1, 2, i, i+1);

    /* Показываем */
    gtk_widget_show (edit[i]);
}

/* Создаем кнопку "Ok", помещаем в таблицу,
   определяем реакцию на нажатие и показываем */
button = gtk_button_new_with_label ("Ok");
gtk_table_attach_defaults (GTK_TABLE(table), button, 2, 3, 0, 1);
gtk_signal_connect(GTK_OBJECT(button), "clicked",
    GTK_SIGNAL_FUNC(writetofile), NULL);
gtk_widget_show (button);

/* Тоже самое для кнопки Quit */
button = gtk_button_new_with_label ("Quit");
gtk_table_attach_defaults (GTK_TABLE(table), button, 2, 3, 2, 3);
gtk_signal_connect(GTK_OBJECT(button), "clicked",
    GTK_SIGNAL_FUNC(delete_event), NULL);
gtk_widget_show (button);

gtk_widget_show (table); /* Показываем таблицу */
gtk_widget_show (window); /* Показываем окно */

/* Запускаем GTK-программу */
gtk_main ();

return 0;
}

```

Рассмотрим построение таблицы размером 3x3:

0	1	2	3
	Domain	Поле	Ok
1			
	DNS1	Поле	
2			
	DNS2	Поле	Quit
3			

Сначала указываются координаты по оси X, затем — по оси Y. Вот координаты кнопки **Ok**: 2,3; 0,1. Это означает, что кнопка будет расположена в последнем стол-

бике (2,3), но в первом ряду (0,1). Чтобы было понятнее: кнопка **Ок** по оси *X* находится между цифрами 2 и 3, а по оси *Y* — между 0 и 1.

Сохраним созданные исходные коды в заранее подготовленной директории, а затем откомпилируем программу:

```
gcc `gtk-config --cflags` resolv.c -o resolv `gtk-config --libs`.
```

Можно не использовать параметр `-g`, добавляющий отладочную информацию. В результате размер исполняемого файла станет меньше. Программа `gtk-config` передает компилятору всю необходимую информацию о библиотеке `gtk`. Обратите внимание на директиву `#include <gtk/gtk.h>`. Обычно файлы заголовков `gtk` находятся в другом каталоге, например `gtk-1.2`, но это не имеет значения — все необходимые параметры укажет программа `gtk-config`.