Предисловие

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 Интернете, 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 ОС **Linux Mandrake 9.1**. После инсталляции **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 написано, 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**. Пришлось на компьютер устанавливать **Linux Mandrake 8.2**.

Начинаем программирование

Рабочее окно **Kylix3_open (C++ IDE)** оказалось очень похожим на рабочее окно интегрированной среды разработки (IDE) программы **Borland C++ Builder**. На рис. 1 показано рабочее окно **Kylix3_open**.

Обратите внимание на небольшое количество компонентов, расположенных в окне панели компонентов.

Не долго думая, мною принято решение создать в среде **Kylix3_open** программу для расчета индуктивности колебательного контура, аналог которой является составной частью разработанной мною ранее программы **INDUKTIW**, опубликованной в Интернете на сайте по адресу http://r3xb.by.ru/ .

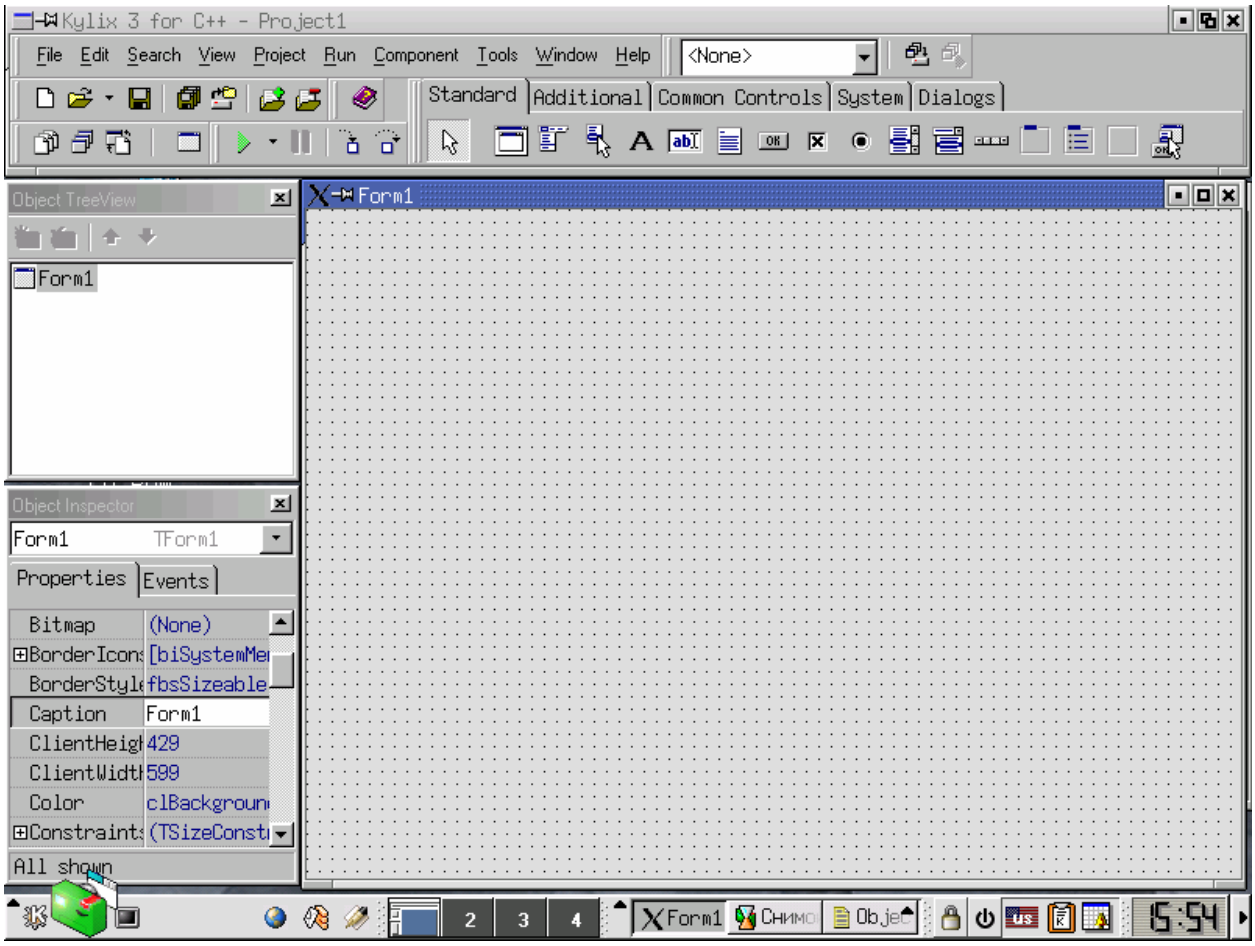

Рис. 1. Рабочее окно Kylix3_open

Что делает приложение

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 а мегагерцах, а во второе окно ввести величину емкости контура. Затем следует нажать клавишу <**Выполнить расчет**>. Под сочетанием слов «нажать клавишу» я имею ввиду, что нужно установить курсор мышки на изображение клавиши и нажать левую кнопку мышки. Иногда будет употребляться выражение «щелкнуть левой кнопкой мышки».

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

Начинаем создавать проект

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 предварительно 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. 2. показана форма с установленными на ней компонентами.

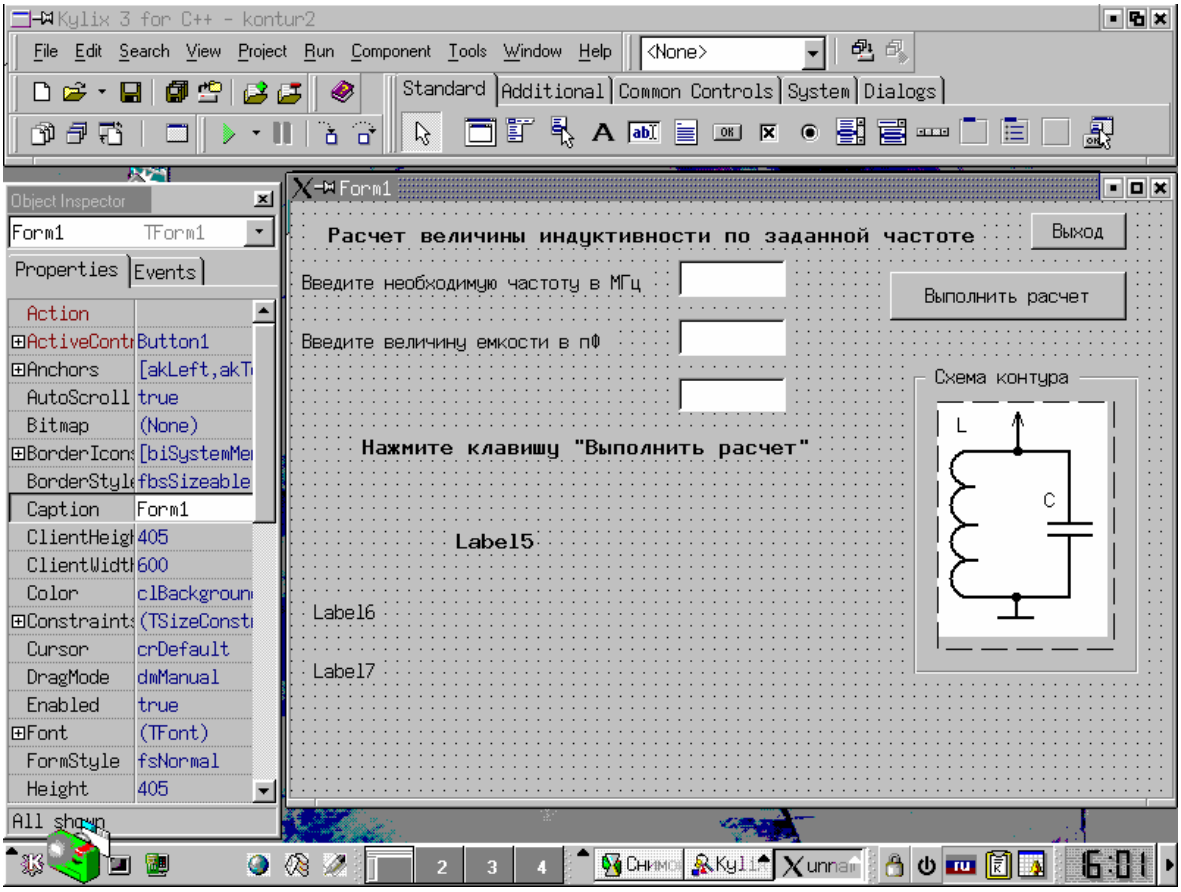

Рис. 2. Расположение компонентов на форме

На панели компонентов должна быть установлена страница библиотеки компонентов **Standard.**

Этап 1

Для отображения на форме текстовых строк будем использовать компонент **Label (**переводится как «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. Щелкаем мышкой на символе компонента и переводим курсор в левый верхний угол формы, как раз в то место, откуда должна начинаться первая текстовая строка. Нажимаем левую клавишу мышки и немного перемещаем курсор вправо и вниз. При этом наблюдаем появления некоторого окна, размеры которого изменяются по мере перемещения курсора. Для нас размеры этого окна не имеют ни малейшего значения, поэтому оставляем окно минимальным и отпускаем клавишу мышки. На форме появляется название **Label1,** одновременно с ним в окне **Object Inspector** появляется 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 выбираем свойство **Caption** и принадлежащее этому свойству окне редактирования стираем имеющееся там название "Label1" и вводим текст «Расчет величины индуктивности по заданной частоте». Затем выбираем свойство **Font** и щелкаем, на появившейся в окне редактирования этого свойства справа, небольшой кнопке с тремя точками. После этого появляется окно выбора шрифта, в котором следует выбрать:

- Font – Fixed [cronyx];
- Script – Russian Cyrillic (KOI8-R);
- Font style – Bold;
- Size – 20;

На экране установился текст первой строки.

Точно таким же образом задаем тексты для всех последующих срок. Следует знать, что имена **Label1** … **Label7** так и останутся за соответствующими строками – метками. 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 на форму компоненты, позволяющие вводить в программу исходные данные. Таким компонентом является **Edit**, изображенный справа рядом с «жирной» буквой '**A**'. 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 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 вычислительных действий в данном приложении задается кнопками **Button**. Компоненты этих кнопок также располагаются на странице библиотеки **Standard,** символом этих кнопок является изображение миниатюрной кнопочки с буквами **OK**. Устанавливаем курсор на этот символ и убеждаемся, что подсказка выдает необходимое название **Button**. Затем переносим два этих компонента на форму в соответствующие места, при этом имена компонентов оставляем неизменными, т.е. **Button1, Button2,** а свойствах **Component** пишем соответственно тексты «Выход» и «Выполнить расчет».

На форму 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.

Собственно рисунок схемы размещается на форму с использованием компонента **Image,** который размещен 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 из внутри установленных ранее рамок **GroupBox**.

Для компонента **Image** выбираем свойство **Picture**. Щелкаем мышкой по кнопочке, расположенной в окне редактирования этого свойства, в результате чего всплывает окно **Picture Editor**. В этом окне выбираем **Load** и всплывает новое окно, окно **Load Picture**, предназначенное для нахождения пути к нужному рисунку. Это окно типовое и каких – то специальных пояснений не требует. Выбираете путь к нужному рисунку, нажимаете клавишу **OK** и этот нужный рисунок моментально устанавливается на форму.

Теперь нужно под размеры рисунка подогнать 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`.

Для того, чтобы вам было удобнее разбираться с названиями установленных на форме компонентов, ниже, в листинге 1, привожу текст файла `kontur01.h`.

Листинг 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 препроцессора 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, начинающихся 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 названия этих файлов первая буква ‘**Q**’ показывает на то, что подключаемые заголовочные файлы относятся к **Qt** библиотеке, имеющейся практически во всех вариантах ОС **Linux**.

Подробнее о библиотеке **Qt** будет рассказано в следующих статьях этой серии.

Если подключаемый файл находится в директории разрабатываемого проекта, то он вместо угловых скобок заключается в двойные кавычки “…”.

Словами `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, т.е. наделять их соответствующими кодами будем делать сами уже в файле `kontur01.cpp`, текст которого приведен в листинге 2.

Листинг 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 = "";          // убираем текст этой строки
 Label1->Font->Color = clBlue;  // задаем цвет для верхней строки
 Label5->Caption = "";          // убираем текст этой строки
 Label5->Font->Color = clBlue;  // задаем цвет для строки Label5
 Label6->Caption = "";          // убираем текст этой строки
 Label7->Caption = "";          // убираем текст 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);
// т.к. в Kylix в окнах редактирования строка представлена в виде
//  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 окно редактирования **Edit2** – выдает на экран соответствующий текст.

Функция 5 включается в работу по после нажатия на клавишу <Выполнить расчет>. Здесь имеется одна особенность, специфическая для **Kylix**: введенная вами в окна редактирования **Edit1** и **Edit2** информация представлена в виде строки типа **WideString**, поэтому для перевода этих данных в число необходимо выполнить преобразование строки WideString в число. Такое преобразование выполняется функцией `Edit1->Text.c_bstr()`.

После полного оформления тскстового файла `kontur01.cpp` следует сохранить все в соответствующей директории проекта, провести компиляцию и создание исполняемого файла проекта.

Если все неприятности, встреченные в процессе компиляции, остались позади и успешно создан исполняемый файл, то можно выполнить команду **RUN** и запустипь проект в работу. На рис. 3 показано рабочее окно программы **kontur1** после ввода всех данных и нажатия клавиши <Выполнить расчет>.

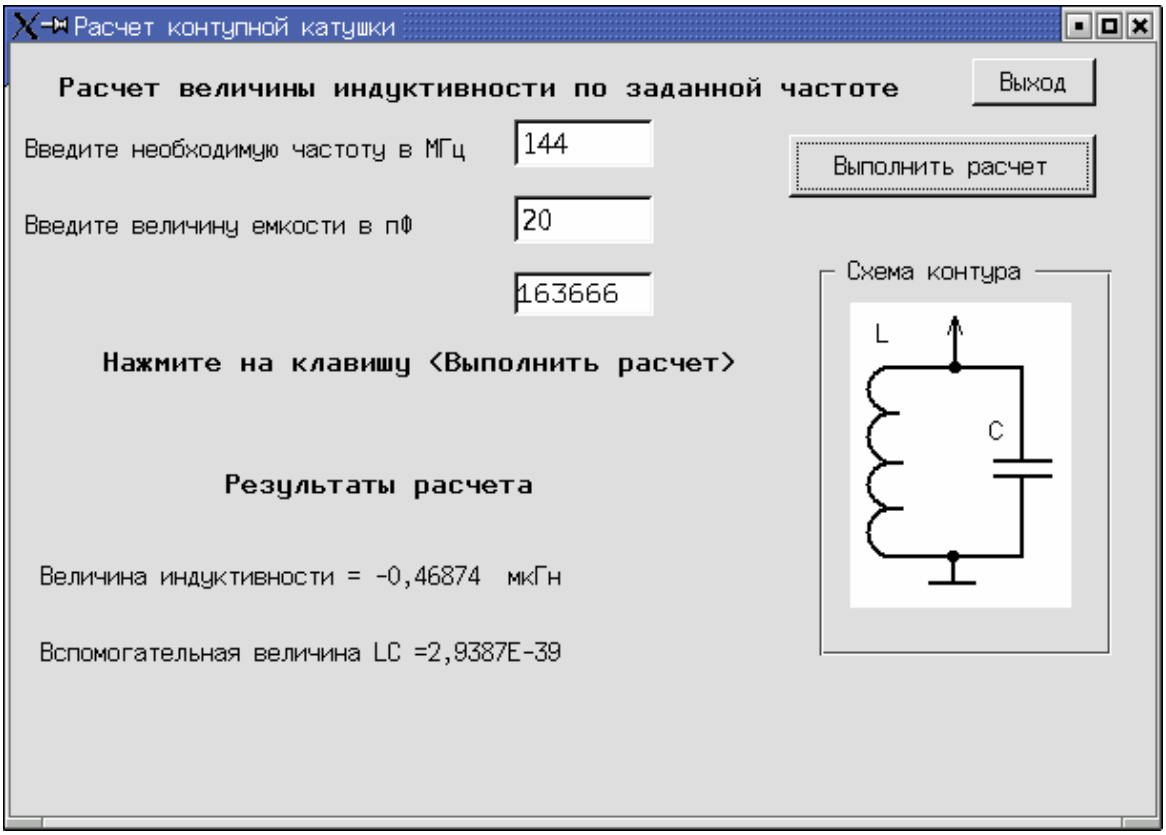

Рис. 3. Рабочее окно программы kontur1

На рис. 4 показано рабочее окно программы в случае, если клавиша <Выполнить расчет> нажата до ввода информации в окна редактирования **Edit1** и **Edit2**.

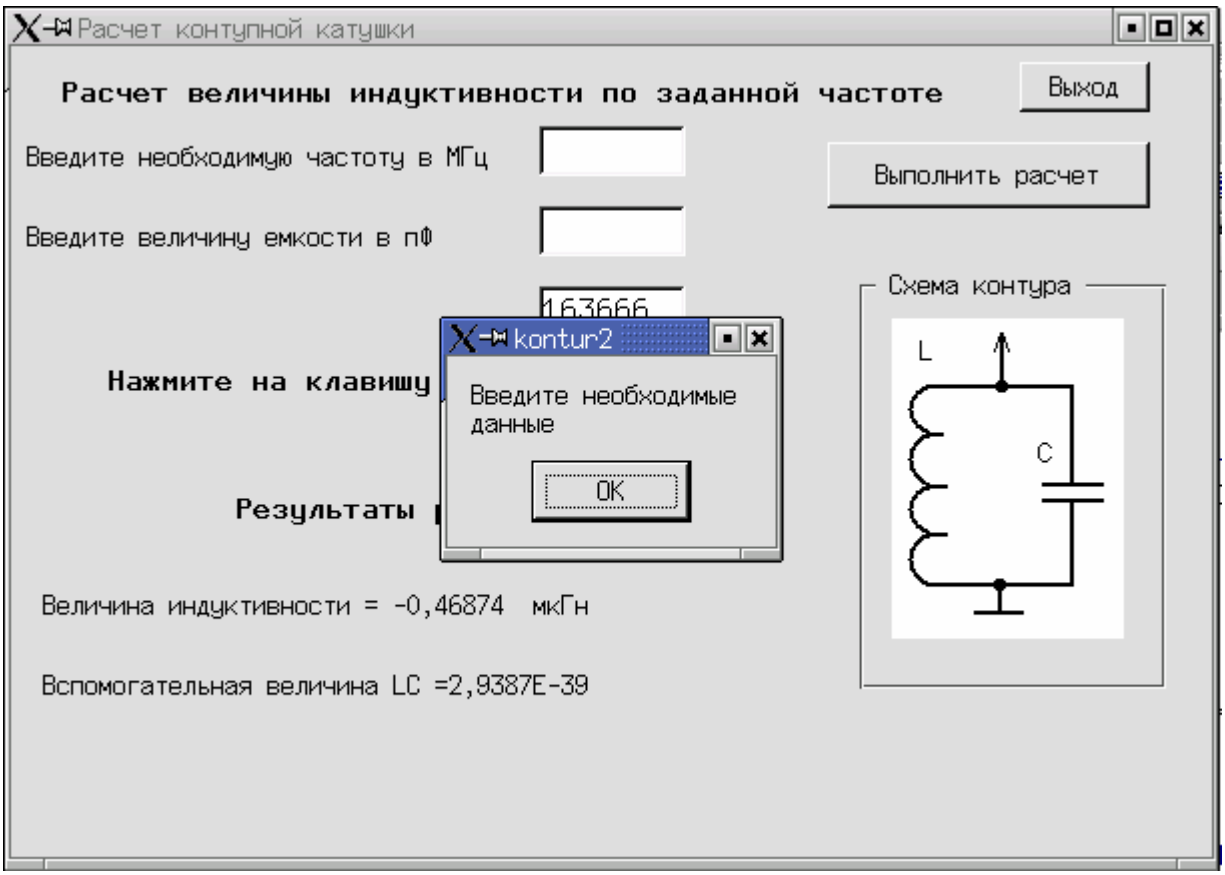

Рис. 4. Диалоговое окно на фоне рабочего окна программы

Вариант Kylix на базе Delphi

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

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 := '';
 Label1.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;

procedure TForm1.Edit2Change(Sender: TObject);
begin
 Label4.Font.Color := clRed;
 Label4.Caption := 'Нажмите на клавишу <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**.

Заключение

В этом разделе хочу довести для вашего сведения о некоторых неприятных моментах, связанных с программами, созданными в среде **Kylix**.

Неприятности начались после того, как исполняемый файл программы **kontur1**, предварительно отлаженный и опробованный в среде **Kylix3 (C++ IDE),** не стал запускаться в работу 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, которые я смог найти в Интернете. 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 **kontur1** 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 **kontur1** стал запускаться в работу и из командной строки консоли, и из файлового менеджера **KDE**.

Вариант исполнительного файла, созданный и отлаженный в среде **Kylix3 (Delphi IDE),** запускался в работу нормально с самого начала, но только под управлением **Linux Mandrake 8.2**. Под **Linux Mandrake 9.1** и под **Linux Mandrake 10.1** он также не запускался в работу.

Точно такие же результаты были получены 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 1Мб.

Литература

Г.Тяпичев «Начальный курс быстрого программирования на C++», опубликована в Интернете по адресу http://r3xb-tga.narod.ru .