НБ МИФИ
004
Ф60

ОСКОВСКИЙ ИНЖЕНЕРНО-ФИЗИЧЕСКИЙ ИНСТИТУТ
(ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ)

Каферда № 46
"Компьютерные медицинские системы"

КОМПЬЮТЕРНЫЕ
МЕДИЦИНСКИЕ
СИСТЕМЫ

К. Г. Финогенов

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

"ПРОГРАММИРОВАНИЕ В СИСТЕМЕ WINDOWS С ПОМОЩЬЮ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ БИБЛИОТЕК

Москва 2008

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

МОСКОВСКИЙ ИНЖЕНЕРНО-ФИЗИЧЕСКИЙ ИНСТИТУТ
(ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ)

К.Г. Финогенов

Программирование в системе Windows с помощью объектно-ориентированных библиотек

Лабораторный практикум

Москва 2008

УДК 004.432.(076.5)
ББК 32.973.202-018я7
Ф60

ИБО 1 экз 724067
ИЗВР 1 экз 724062
АУЛ 62 экз

Финогенов К.Г. Программирование в системе Windows с помощью объектно-ориентированных библиотек. Лабораторный практикум: Учебное пособие. М.: МИФИ, 2008. 104 с.

Первая часть пособия посвящена принципам составления объектно-ориентированных приложений Windows, использующих классы библиотек интерфейсных объектов. Рассматриваются правила составления таких программ, возможности включения в них типичных для приложений Windows интерфейсных средств (меню, диалогов, элементов управления, графических изображений и пр.) в двух средах программирования: Borland C++ (библиотека OWL) и Borland C++ Builder (библиотека VCL).

Вторая часть пособия представляет собой описание лабораторного практикума и комплекта индивидуальных заданий для освоения объектно-ориентированного программирования в системе Windows с помощью библиотек Borland C++ и Borland C++ Builder.

Предназначено для обучения студентов кафедры компьютерных медицинских систем факультета автоматики и электроники МИФИ (специальность "Электроника и автоматика физических установок") по курсам "Языки программирования и операционные системы" и "Компьютерный практикум". Пособие может быть также полезно студентам, аспирантам и преподавателям, знакомым с языком C++ и основами объектно-ориентированного программирования и желающим самостоятельно освоить использование объектно-ориентированных библиотек.

Рецензент В.В. Комаров
Рекомендовано редсоветом МИФИ в качестве учебного пособия

ISBN 978-5-7262-0897-8

© *Московский инженерно-физический институт (государственный университет), 2008*

Редактор Е.Е. Шумакова

Подписано в печать 18.02.2008 г.
Печ. л. 6,5 Уч.-изд. л. 6,5
Изд. № 057-1

Формат 60 × 84 1/16
Тираж 120 экз.
Заказ N 62

Московский инженерно-физический институт (государственный университет)
Типография МИФИ. 115409, Москва, Каширское шоссе, 31

Содержание

Часть 1. Теоретические сведения 5
1. Структура OWL-приложения Windows 5
Простейшая программа с главным окном 6
Структура и ход выполнения программы 7
Заголовочные и другие вспомогательные файлы 12
2. Интерфейс графических устройств GDI 14
Программа с выводом в окно графических объектов 14
Обработка сообщения WM_PAINT 16
Вывод в окно текста и геометрических фигур 19
Задание цвета 20
Вывод растровых изображений 21
3. Организация меню и обработка сообщений Windows 27
Программа с меню в главном окне 27
Таблица откликов и обработка сообщений от пунктов меню. 30
Обработка других сообщений 31
Управление программой с помощью мыши 33
4. Диалоговые окна 34
Стандартный модальный диалог “Открытие файла” 35
Модальный диалог в качестве главного окна приложения 37
Интерфейсные элементы и интерфейсные объекты 40
Взаимодействие пользователя с элементами управления диалогового окна 42
Перекрашивание диалогового окна и его элементов 44
5. Программирование в среде Borland C++ Builder 46
Среда программирования Borland C++ Builder 46
Работа со свойствами объектов 49
Работа с методами объектов 51
Работа с событиями 53
Модификация файлов проекта 53
Вывод в окно графических элементов 55
Работа с совместимой памятью 57
Копирование изображений 60
Обработка цветных цифровых изображений 61
Стандартные модальные диалоги 64

Часть 2. Лабораторный практикум 66
Работы лабораторного практикума 66
Программирование с помощью библиотеки OWL 66
Работа №1. Приложение Windows с выводом в главное окно текста и фигур (индивидуальное задание A) 66
Работа №2. Вывод в главное окно графика математической функции (индивидуальное задание B) 66
Работа №3. Меню и стандартный диалог "Открытие файла" (индивидуальное задание C) 67
Работа №4. Вывод растровых изображений 68
Работа №5. Обработка сообщений от мыши (индивидуальное задание D) 69
Работа №6. Модальный диалог в качестве главного окна приложения 70
Работа №7. Динамическое взаимодействие пользователя с элементами управления диалога 71
Программирование в среде Borland C++ Builder 72
Работа №1. Простейшее приложение с главным окном 72
Работа №2. Работа с элементами формы 73
Работа №3. Вывод в окно графика математической функции (индивидуальное задание E) 74
Работа №4. Синхронизация программных действий с помощью таймера 75
Работа №5. Использование совместимой памяти 77
Работа №6. Меню, стандартные диалоги и растровые изображения 78
Работа №7. Растр в памяти и обработка изображений (индивидуальное задание F) 80
Индивидуальные задания лабораторного практикума 82
Список литературы 104

Часть 1
Теоретические сведения

1. СТРУКТУРА OWL-ПРИЛОЖЕНИЯ WINDOWS

В настоящее время серьезные прикладные программы создаются, как правило, с использованием интегрированных сред разработки, таких, как Visual Studio фирмы Microsoft, Borland C++ или Borland C++ Builder фирмы Borland и др. Эти инструментальные пакеты, помимо обеспечения всего цикла разработки приложения (подготовки исходного текста, компиляции, отладки, пробного исполнения с анализом характеристик создаваемой программы), предоставляют еще и библиотеки классов типичных для Windows интерфейсных средств (окон, диалогов, меню, кнопок, списков, графических инструментов и т. д.), использование которых в программе позволяет существенно сократить время, требуемое для разработки приложения Windows, и повысить его качество. В среде Visual C++ такая библиотека носит название MFC (Microsoft Foundation Classes); в среде Borland C++ – OWL (Object Windows Library); в среде Borland C++ Builder – VCL (Visual Component Library). Все эти библиотеки, как и следует из их названий, являются объектно-ориентированными и требуют от программиста хорошего знания принципов и методов объектно-ориентированного программирования (хотя, между прочим, сама программа, использующая эти библиотеки, не обязательно должна быть объектно-ориентированной).

В настоящем пособии будут рассмотрены основные принципы разработки приложений Windows с использованием двух объектно-ориентированных библиотек: библиотеки OWL 5.0, входящей в состав инструментального пакета Borland C++ 5.0, а также библиотеки VCL из среды Borland C++ Builder. От читателя потребуется знакомство с языком C++, понимание принципов разработки приложений Windows и владение основными методами объектно-ориентированного программирования.

Простейшая программа с главным окном

Пример 1-1 представляет собой простейшее работоспособное OWL-приложение Windows, выводящее на экран главное окно со строкой заголовка и стандартным набором управляющих кнопок (рис. 1.1). Пока это окно будет пустым; заполнять его различными элементами управления и содержательной информацией мы будем в последующих примерах.

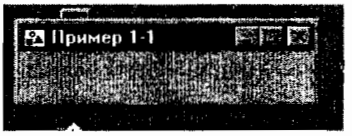

Рис. 1.1. Вывод программы 1-1

```
/*Пример 1-1. Простейшая программа с главным окном*/
#include <owl/framewin.h>//Необходимый заголовочный файл
/*Класс MyApp приложения, производный от TApplication*/
class MyApp:public TApplication{
public:
  void InitMainWindow(void);//Замещаем эту функцию
  };
/*Класс MyWin главного окна, производный от TFrameWindow*/
class MyWin: public TFrameWindow{
public:
  MyWin (TWindow* parent,char* title)://Конструктор
          TFrameWindow(parent,title){//класса MyWin
    Attr.X=10;Attr.Y=10;//Задаем положение
    Attr.W=200;Attr.H=60;//и размеры окна
    SetBkgndColor(TColor(220,220,220));//Задаем цвет окна
    }
  };
/*Замещающая функция InitMainWindow()*/
void MyApp::InitMainWindow(void){
  MyWin* myWin=new MyWin(0,"Пример 1-1");
  SetMainWindow(myWin);//Объявляем окно myWin главным
  }
/*Главная функция приложения OwlMain()*/
int OwlMain(int,char**){
  MyApp* myApp=new MyApp;//Создаем объект приложения
  return myApp->Run();//Вызываем для этого объекта функцию Run()
  }
```

Структура и ход выполнения программы

Библиотека OWL, как и любая другая объектно-ориентированная библиотека для разработки приложений Windows, содержит описание классов для реализации практически всех основных средств Windows. Составление приложения Windows, грубо говоря, заключается в подборе библиотечных классов, реализующих средства, требуемые в конкретном приложении, описании в программе прикладных классов, являющихся производными от библиотечных, и использовании затем наследуемых данных-членов и функций-членов этих классов. При необходимости библиотечные функции-члены модифицируются (путем их замещения прикладными функциями с теми же именами и сигнатурами) с целью придания им требуемых свойств, а данным-членам задаются требуемые значения. Во многих случаях библиотечные функции вполне удовлетворительно решают прикладную задачу и не нуждаются в модификации, и тогда используются не производные от библиотечных, а сами библиотечные классы: создаются их экземпляры (объекты) и вызываются необходимые функции-члены этих классов с передачей им конкретных аргументов.

В приведенной выше программе можно увидеть четыре составных части, присутствующих в любом типичном OWL-приложении:

- класс приложения (**MyApp** в примере), производный от библиотечного класса **TApplication**;
- класс главного окна (**MyWin**), производный от библиотечного класса **TFrameWindow**;
- главная функция приложения **OwlMain()**;
- определения замещаемых функций базовых библиотечных классов. В приведенном примере такая функция одна: это функция **InitMainWindow()**, принадлежащая библиотечному классу **TApplication**.

Таким образом, в рассматриваемом OWL-приложении используются только два библиотечных класса: класс приложения **TApplication** и класс главного окна **TFrameWindow**.

Класс **TApplication**, от которого мы создаем производный класс **MyApp** (имя этого прикладного класса, разумеется, может быть любым), сам является производным от класса **TModule**, и,

таким образом, эти три класса образуют иерархическую структуру, показанную на рис. 1.2.

Рис. 1.2. Иерархия классов обслуживания приложения

Класс **TModule** обеспечивает некоторые общие свойства модуля приложения (под *модулем приложения* понимают обычно его коды, загруженные в память, в отличие от *экземпляра приложения*, содержащего данные), в частности, организует загрузку и выгрузку динамических библиотек, а также предоставляет ряд функций, относящихся к модулю и приложению в целом. Так, в него входят функции-члены загрузки ресурсов **LoadCursor()** и **LoadIcon()**, получения определенной информации о модуле приложения **GetName()** и **GetInstance()**, обработки ошибок **Error()** и ряд других. В примере 1-1 функции базового класса **TModule** не используются.

Класс **TApplication** обеспечивает основные свойства приложения Windows. Функции этого класса, в частности, организуют создание главного окна (именно организуют; собственно создание окна возлагается на функции другого класса – **TWindow**), обрабатывают сообщения, поступающие в окно приложения, и выполняют ряд других важных операций.

Всю работу по инициализации приложения, созданию и отображению главного окна и организации цикла обработки сообщений выполняет открытая функция-член **Run()**, входящая в класс **TApplication**. Поэтому в простых случаях главная функция приложения **OwlMain()** может состоять всего из двух строк: создания экземпляра класса **TApplication** или производного от него и вызова функции **Run()** для этого экземпляра. Именно так выглядит функция **OwlMain()** в рассматриваемом примере. С другой стороны, для задания характеристик и свойств окна приложения приходится вмешиваться в работу этой функции, изменяя ис-

пользуемый по умолчанию алгоритм ее выполнения, что требует понимания этого алгоритма и возможностей его изменения. Рассмотрим процедуру выполнения функции **Run()** более детально.

Основной задачей функции **Run()** можно считать последовательный вызов ряда других функций OWL, принадлежащих как классу **TApplication**, так и классу **TWindow**, что в итоге приводит к нормальному функционированию приложения. Несколько упрощенная последовательность этих вызовов с указанием классов, которым принадлежат вызываемые функции, приведена на рис. 1.3.

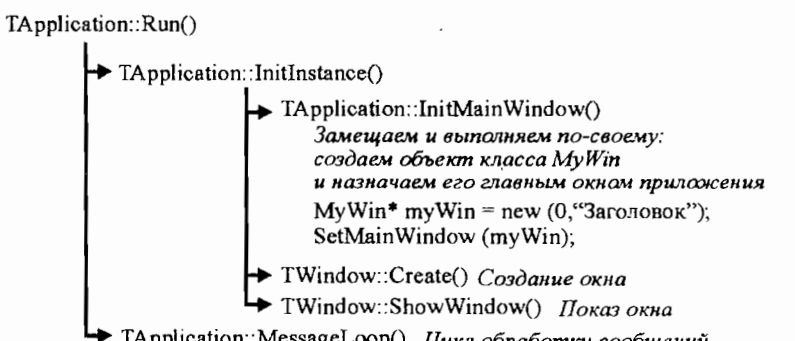

Рис. 1.3. Ход выполнения функции **Run()**

Как видно из рис. 1.3, функция **Run()** последовательно вызывает две другие функции того же класса **TApplication**: **InitInstance()** и **MessageLoop()**.

Функция **InitInstance()** выполняет ряд весьма важных действий. Прежде всего она вызывает функцию **InitMainWindow()**. Эта функция (в своем исходном, незамещенном варианте) создает объект класса **TFrameWindow**, определяющий характеристики главного окна, и вызывает функцию **SetMainWindow()**, которая объявляет новое окно главным. Именно на этом этапе можно изменить, например, расположение, размеры или стиль окна.

После создания объекта класса **TFrameWindow**, характеризующего новое окно, можно обращаться к функциям-членам этого класса, а также и к функциям класса **TWindow**, который является

базовым для **TFrameWindow**. Функция **InitInstance()** это и делает, вызывая последовательно функции класса **TWindow** **Create()** и **ShowWindow()**, выполняющие создание и показ нового окна. Эти действия внутри класса **TWindow** реализуются путем вызова функций интерфейса прикладного программирования (API) **CreateWindow()** и **ShowWindow()**. Как говорят, функции-члены класса **TWindow** **Create()** и **ShowWindow()** *инкапсулируют* в себе обычные функции API Windows.

Образовав и выведя на экран главное окно приложения, функция **Run()** вызовом функции **MessageLoop()** входит в цикл обработки сообщений, в котором приложение и находится до своего завершения.

Для того чтобы можно было изменять характеристики и свойства главного окна, надо создать собственный класс главного окна (в примере 1-1 ему дано произвольное имя **MyWin**), производный от библиотечного класса **TFrameWindow**. Класс **TFrameWindow** сам является производным от класса **TWindow**, и, таким образом, эти три класса образуют иерархическую структуру, показанную на рис. 1.4.

Рис. 1.4. Иерархия классов главного окна

Класс **TWindow** является родоначальником ряда классов, описывающих функционирование разного рода окон (диалогов, органов управления и других), которые, естественно, наследуют возможности своего базового класса **TWindow**. Среди этих производных классов есть и класс окон с рамками **TFrameWindow**.

В класс **TWindow** входит большое число (около 300!) функций-членов, обеспечивающих общие черты поведения окон: **Create()**, **ShowWindow()**, **MoveWindow()**, **Invalidate()**, **GetClientRect()**, **SetBkgndColor()** и т. д. В примере 1-1 ис-

пользовалась функция этого класса **SetBkgndColor()**, устанавливающая цвет фона окна.

Помимо функций в класс **TWindow** входит ряд данных-членов, из которых нас пока будет интересовать только структура типа **TWindowAttr**, которая представлена в классе **TWindow** данным-членом **Attr**. Элементы этой структуры определяют атрибуты окна, устанавливаемые во время его создания: стиль, координаты, размеры и др. В примере 1-1 с помощью элементов этой структуры устанавливаются координаты и размеры окна.

Создав в программе производный от **TApplication** класс **MyApp**, мы получили возможность переопределить в нем функцию класса **TApplication InitMainWindow()**. Зачем это нужно? Дело в том, что в исходном варианте этой функции вызовом функции **SetMainWindow()** создается безымянный экземпляр класса **TFrameWindow**, к которому мы даже не можем обратиться:

```
SetMainWindow(new TFrameWindow(0,GetName());
```

Нам нужно заменить его экземпляром производного от него класса **MyWin**, с функциями и данными которого мы сможем работать. В замещенной функции **InitMainWindow()** выполняется в сущности то же, что было предусмотрено в исходной функции, но в нужном нам варианте:

```
MyWin* myWin=new MyWin(0,"Пример 1-1");//Создаем объект
                    //myWin  прикладного класса MyWin
SetMainWindow(myWin);//Объявляем новое окно главным
```

Как известно, при создании объекта любого класса автоматически вызывается его конструктор. Описывая конструктор нашего класса **MyWin**, мы должны предусмотреть в нем явный вызов конструктора базового класса **TFrameWindow** ради передачи в него инициализирующих параметров. Этот конструктор объявлен в файле BC5\INCLUDE\OWL\WINDOW.H следующим образом:

```
TFrameWindow(TWindow* parent,const char far* title=0,
      TWindow* clientWnd=0,bool shrinkToClient=false,
      TModule* module=0);
```

Нет необходимости изучать все параметры этого конструктора; достаточно заметить, что для всех параметров кроме первого предусмотрены значения по умолчанию, поэтому строго необходимо

передавать в конструктор только один первый параметр. Однако второй параметр позволяет задать заголовок окна, поэтому конструктор **TFrameWindow**, как правило, вызывается с двумя параметрами, как это и сделано в нашем примере.

Рассмотрим определение прикладного класса **MyWin**:

```
class MyWin:public TFrameWindow{
  MyWin (TWindow* parent,char* title)://Конструктор
          TFrameWindow(parent,title){//класса MyWin
    Attr.X=10;Attr.Y=10;//Задаем положение
    Attr.W=200;Attr.H=60;//и размеры окна
    SetBkgndColor(TColor(220,220,220));//Задаем цвет окна
    }
  };
```

В определении класса указывается, что он является производным от **TFrameWindow**, и описывается единственная функция этого класса, именно, его конструктор. В конструкторе **MyWin** предусмотрен вызов конструктора базового класса **TFrameWindow** с передачей ему двух параметров **parent** и **title**. Как мы уже видели, при фактическом вызове этого конструктора ему передаются значения параметров **0** и **"Программа 1-1"**.

Далее в текст конструктора **MyWin** включены действия, которые необходимо выполнить при создании главного окна: изменение действующих по умолчанию значений структурной переменной **Attr**, определяющих координаты и размеры окна, а также вызов открытой функции-члена класса **TWindow SetBkgndColor()** для задания цвета фона окна.

Заголовочные и другие вспомогательные файлы

Исходный текст практически любой программы включает в себя то или иное количество директив **#include**, с помощью которых препроцессор подключает к исходному тексту программы необходимые для ее успешной компиляции заголовочные файлы. В зависимости от средств, используемых в компилируемой программе, состав заголовочных файлов может сильно различаться. В обычных приложениях Windows, как правило, подключаются по меньшей мере заголовочные файлы WINDOWS.H и WINDOWSX.H.

Библиотека OWL содержит большое количество заголовочных файлов, в которых описана вся иерархия OWL-классов. Для ус-

пешной компиляции OWL-программы к ней должны быть подключены заголовочные файлы с описанием всех библиотечных классов, используемых в программе. В примере 1-1 используются библиотечные классы **TApplication** и **TFrameWindow**, описания которых содержатся в заголовочных файлах с достаточно наглядными именами APPLICAT.H и FRAMEWIN.H. Поскольку, однако, заголовочные файлы OWL (как и заголовочные файлы Windows) обычно содержат в себе директивы **#include** подключения дополнительных заголовочных файлов, реально достаточно включить в программу одну лишь директиву

```
#include <owl\framewin.h>
```

Обратите внимание на указание каталога owl в спецификации файла FRAMEWIN.H. Дело в том, что программный пакет системы Borland C++ состоит из нескольких тысяч файлов, организованных в виде сложной иерархической структуры каталогов. Все include-файлы помещены в ветвь этой структуры с именем INCLUDE, однако в этом каталоге имеются еще и нижележащие каталоги. Заголовочные файлы, описывающие классы OWL, помещены в подкаталог с тем же именем (рис. 1.5).

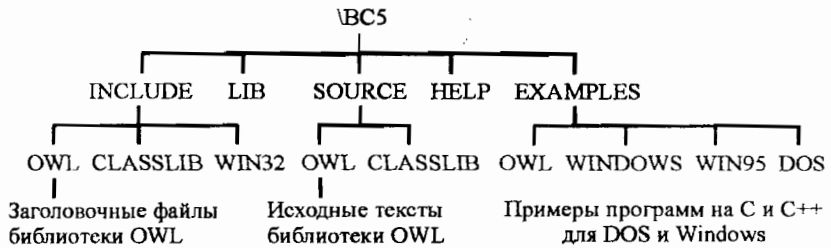

Рис. 1.5. Часть структуры каталогов пакета Borland C++ 5.0

Поскольку в настройках Borland C++ в качестве каталога для включаемых файлов обычно указывается каталог \BC5\INCLUDE, а нужные нам файлы расположены ниже, в подкаталоге OWL, в директивах **#include** необходимо указывать оставшуюся часть пути к включаемым файлам начиная от каталога INCLUDE.

2. ИНТЕРФЕЙС ГРАФИЧЕСКИХ УСТРОЙСТВ GDI

В большинстве программ в окно приложения требуется выводить те или иные графические элементы: тексты, графики, рисунки и пр. Эти операции выполняются с помощью интерфейса графических устройств (Graphics Device Interface, GDI), который включает в себя всю совокупность программных средств (функций и структур данных), обеспечивающих формирование и вывод на различные устройства вывода – экран, принтер и другие – всего многообразия графических объектов.

Программа с выводом в окно графических объектов

Рассмотрим простую программу, иллюстрирующую основные возможности и особенности использования средств GDI в OWL-приложениях. Вывод этой программы приведен на рис. 2.1.

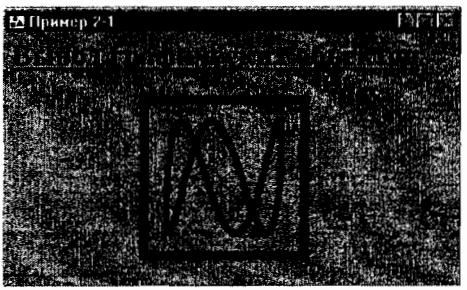

Рис. 2.1. Вывод программы 2-1

```
/*Пример 2-1. Вывод в окно простых графических объектов */
#include <owl/framewin.h>
#include <math.h>
/*Класс MyApp приложения, производный от TApplication*/
class MyApp:public TApplication{
public:
  void InitMainWindow(void); //Замещаем эту функцию
  };
/*Класс MyWin главного окна, производный от TFrameWindow*/
class MyWin: public TFrameWindow{
public:
  MyWin(TWindow* parent,char* title): //Конструктор
        TFrameWindow(parent,title){ //класса MyWin
```

```
    Attr.X=10;Attr.Y=10;Attr.W=400;Attr.H=250;
    SetBkgndColor(TColor::LtGray);//Задаем цвет окна
    }
  void Paint(TDC&,bool,TRect&);//Замещаем функцию Paint()
  };
/*Замещающая функция InitMainWindow()*/
void MyApp::InitMainWindow(void){
  MyWin* myWin=new MyWin(0,"Пример 2-1");
  SetMainWindow(myWin); //Объявляем окно главным
  }
/*Замещающая функция Paint()*/
void MyWin::Paint(TDC& dc,bool,TRect&){
/*Выведем строку текста*/
  TFont font("Times",30,0,0,0,FW_BOLD,0,0,1);
  dc.SelectObject(font);//Выбираем в контекст шрифт
  dc.SetBkMode(TRANSPARENT);//Фон под буквами прозрачный
  dc.TextOut(5,5,"Вывод графических объектов");
/*Выведем график функции*/
  TBrush brush(TColor::LtBlue);//Создаем объект кисти
  dc.SelectObject(brush);//Выбираем кисть в контекст
  dc.SelectStockObject(NULL_PEN);//Выбираем прозрачное перо
  for(float t=0;t<2*M_PI;t+=0.01){//Цикл вывода графика:
    float x=sin(t)*50;//вычисление x-координаты,
    float y=-cos(3*t+M_PI*0.2)*50;// вычисление y-координаты,
    dc.Ellipse(x+187,y+127,x+193,y+133);//вывод точки
    }
/*Выведем рамку вокруг графика*/
  TPen pen(TColor(120,120,240),8);//Создаем объект пера
  dc.SelectObject(pen);//Выбираем в контекст перо
  dc.SelectStockObject(NULL_BRUSH);//Выбираем прозрачную кисть
  dc.Rectangle(120,60,260,200);//Рисуем прямоугольную рамку
  }
/*Главная функция приложения OwlMain()*/
int OwlMain(int,char**){
  MyApp* myApp=new MyApp;//Создаем объект приложения
  return myApp->Run();//Вызываем функцию Run()
  }
```

Сравнивая приведенную выше программу с примером 1-1, легко заметить, что они принципиально различаются лишь одной деталью: в новой программе имеется довольно значительная по объему функция **Paint()**, в которой, очевидно, и выполняются все действия по выводу в окно приложения графических объектов.

Обработка сообщения WM_PAINT

Как известно, вывод в окно приложения изображений, как правило, выполняется в ответ на сообщения **WM_PAINT**. В этом случае при любых манипуляциях с окном – увеличении его размеров, "вытаскивании" его из-под закрывающих окон других выполняемых приложений и др. – содержимое окна автоматически перерисовывается заново средствами Windows, восстанавливая его исходный вид.

Для обработки сообщений Windows в OWL предусмотрен специальный механизм – организация в программе таблицы откликов на обрабатываемые сообщения, которая содержит имена функций обработки сообщений и осуществляет передачу управления на эти функции в случае прихода соответствующего сообщения. Однако сообщение **WM_PAINT**, учитывая его важность, частично обслуживается внутри класса **TWindow**, где имеется своя таблица откликов; программисту предоставляется функция-заглушка с именем **Paint()**, которую он должен заместить в своей программе, заполнив требуемым содержимым.

Рассмотрим вкратце процедуру обслуживания сообщений **WM_PAINT** классом **TWindow**. Это поможет нам разобраться в смысле аргументов функции **Paint()** и правилах составления текста замещающей функции.

Как известно, важнейшим понятием GDI является контекст устройства, содержимое которого в каждый момент определяет характеристики всех доступных инструментов рисования. В действительности в Windows используется не один, а целый ряд контекстов устройств, предоставляющих возможность рисовать в рабочей области окна, во всем окне приложения, на всем экране и т. д. Наиболее общие свойства контекстов устройств описаны в классе **TDC**, являющемся базовым для подклассов, определяющих свойства упомянутых выше контекстов. Из этих производных подклассов нас пока будет интересовать только класс **TPaintDC**, с экземпляром которого **dc** мы сталкиваемся в нашей программе. С другой стороны, сам класс **TDC** является производным от базового для всей графической системы (а не только для контекстов устройств) класса **TGdiBase**. Эти три класса образуют структуру, изображенную на рис. 2.2.

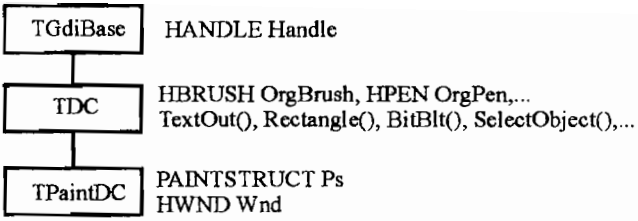

Рис. 2.2. Иерархия классов контекстов устройств

В базовый класс **TGdiBase** входит дескриптор контекста устройства **Handle**. Этот дескриптор имеет обобщенный тип **HANDLE**, который затем преобразуется в более привычный нам тип контекста устройства **HDC**.

Класс **TDC**, являющийся базовым для целого ряда классов, описывающих различные контексты устройств (**TPaintDC**, **TWindowDC**, **TClientDC** и др.), включает дескрипторы исходных графических объектов (инструментов рисования) **OrgBrush**, **OrgPen**, **OrgFont** и **OrgPalette**, а также большое количество (около 200) графических функций, обеспечивающих вывод на экран текстов, фигур и других изображений, создание, выбор и настройку инструментов рисования, получение и изменение режимов работы графической системы и т. д. Большинство этих функций инкапсулирует соответствующие функции API Windows.

Небольшой по объему класс **TPaintDC** содержит в качестве данных-членов дескриптор окна типа **HWND** (данное с именем **Wnd**) и структуру **PAINTSTRUCT** (данное с именем **Ps**), используемую системой Windows при перерисовке изображения. Функций-членов в классе **TPaintDC** всего две: конструктор **TPaintDC()** и деструктор **~TPaintDC()**. В конструкторе вызывается функция API Windows **BeginPaint()**, которая заполняет структуру **Ps** и возвращает контекст устройства; в деструкторе вызывается функция API Windows **EndPaint()**, освобождающая контекст устройства. Таким образом, при создании объекта класса **TPaintDC** автоматически выполняются необходимые инициализирующие действия, а при его уничтожении – необходимые завершающие действия. Стоит еще отметить, что в деструктор **~TPaintDC()** входит также вызов функции **RestoreObjects()**, выполняющей выбор в кон-

текст устройства исходных инструментов. Таким образом, в OWL-программе после окончания работы с созданными и выбранными в контекст устройств инструментами нет необходимости выполнять относительно громоздкую процедуру выбора назад в контекст всех исходных инструментов, так как это делает деструктор класса.

Поступление в окно приложения сообщения **WM_PAINT** приводит к вызову защищенной функции класса **TWindow EvPaint()**. Поскольку сообщение **WM_PAINT** приходит в наше окно, с которым ассоциируется объект **MyWin** (являющийся потомком класса **TWindow** и наследующий его открытые и защищенные члены), то функция **EvPaint()** вызывается именно для этого объекта. Определение функции **EvPaint()** (которое можно найти в файле SOURCE\OWL\WINDOW.CPP), за вычетом некоторых несущественных деталей, выглядит следующим образом:

```
void TWindow::EvPaint(){
  TPaintDC dc(*this);//Создание объекта dc
  TRect& rect=*(TRect*)&dc.Ps.rcPaint;//Получение области вырезки
  Paint(dc,dc.Ps.fErase,rect};//Вызов функции-заглушки Paint()
  }
```

В первом предложении вызывается конструктор класса **TPaintDC**, который создает объект этого класса с именем **dc**. Это имя используется в дальнейших предложениях приведенного фрагмента, а также и в тексте нашей программы. Во втором предложении в переменную **rect** класса **TRect** поступают координаты области вырезки. Наконец, в последнем предложении вызывается упоминавшаяся выше функция-заглушка **Paint()**, в качестве аргументов которой выступают объект **dc**, флаг перерисовки окна **fErase** и прямоугольник с координатами области вырезки **rect**.

Итак, при поступлении в окно приложения сообщения **WM_PAINT** автоматически создается объект класса **TPaintDC** с конкретным именем **dc**, через который мы можем получить доступ ко всему набору графических функций класса **TDC**, и выполняется вызов функции API Windows **BeginPaint()**, которая заполняет структуру **Ps**. Наша задача теперь сводится к вызову любых требуемых функций GDI для объекта **dc** :**SelectObject()**, **TextOut()**, **Elllipse()**, **SetPixel()** и др.

Вывод в окно текста и геометрических фигур

Процедура вывода в окно приложения графических объектов фактически не отличается от используемой в традиционных приложениях Windows, за исключением того, что все функции GDI вызываются для объекта **dc**, а имена этих функций иногда слегка отличаются от имен функций API. Не следует только забывать, что если при традиционном программировании создание различных объектов (перьев, шрифтов и пр.) осуществлялось вызовом функций API (например, **CreatePen()** для пера), то теперь для создания объектов необходимо вызывать конструкторы соответствующих классов OWL (**TPen** для перьев, **TFont** для шрифтов и т. д.). Объектом OWL является также и цвет, так что окраска любого элемента изображения требует создания объекта типа **TColor**.

Приведем несколько фрагментов (предназначенных, естественно, для включения в функцию **Paint()**) вывода в окно приложения графических примитивов:

```
/*Вывод строки текста*/
TFont font("Arial",20,0,900,0,FW_NORMAL,0,1);//Создаем объект
//шрифта Arial размером 20 пикселей, под углом 90°,не жирный, курсив
dc.SelectObject(font);//Выбираем шрифт в контекст
dc.SetBkMode(TRANSPARENT);//Фон под буквами прозрачный
dc.SetTextColor(TColor::LtBlue);//Цвет ярко-синий
dc.TextOut(5,5,"Наша программа");//Собственно вывод строки

/*Вывод прозрачной окружности*/
TPen pen(TColor(200,200,255,5));//Перо бледно-синее, 5 пикселей
dc.SelectObject(pen);//Выбираем его в контекст
dc.SelectStockObject(NULL_BRUSH);//Выбираем со склада в
                                  //контекст прозрачную кисть
dc.Ellipse(10,10,100,100);//Вывод окружности

/*Вывод залитого треугольника */
dc.SelectObject(TBrush(TColor::Ltred)));//Создаем красную
                                  //кисть и выбираем ее в контекст
dc.SelectStockObject(NULL_PEN);//Выбираем со склада в контекст
                                  //прозрачное перо
TPoint pt[3];//Массив неинициализированных объектов TPoint
pt[0]=TPoint(10,10);   //Создаем три объекта типа TPoint с
pt[1]=TPoint(200,50);  //координатами вершин треугольника и
pt[2]=TPoint(150,210);// переносим их в элементы нашего массива
dc.Polygon(pt,3);//Выводим треугольник с заданными вершинами
```

```
/*Вывод прямой линии*/
dc.SelectObject(TPen(TColor::Black,3);//Выбираем в контекст
                                    //черное перо толщиной 3 пикселя
dc.MoveTo(5,100);//Перемещаем начальную точку
dc.LineTo(400,100);//Рисуем линию (горизонтальную)
/*Графическое отображение массива данных точками */
int data[100];//Массив с целочисленными данными
for(int i=0,i<100;i++)
  dc.SetPixel(i,400-data[y],Color::LtBlue);//Точки синие,
                                    //график сместим вниз на 400 пикселей
```

Задание цвета

Выше уже приводились примеры задания цвета инструментов рисования или точек графика различными способами. Общая идея заключается в том, что необходимо создать объект класса **TColor**, указав в конструкторе требуемый цвет. Библиотека OWL предоставляет 9 конструкторов этого класса, позволяющих задавать цвет различными способами. Рассмотрим некоторые из них.

```
TColor(COLORREF value);
```

Этот конструктор требует в качестве параметра значение типа **COLORREF**, которое обычно создается с помощью макроса **RGB()**:

```
TColor clr1 (RGB(255,255,200);//Бледно-желтый цвет
TPen pen (clr1,5);//Бледно-желтое перо толщиной 5 пикселей
```

Однако проще воспользоваться другой формой конструктора:

```
TColor(int r, int g, int b);
```

Здесь компоненты цвета задаются просто в виде трех чисел, а макрос **RGB()** не требуется; очевидно, что преобразование этих чисел в данное типа **COLOREF** осуществляется автоматически где-то в теле конструктора.

Указанными способами можно определить любой из 2^{24} возможных цветов. Однако для задания наиболее распространенных цветов в составе класса **TColor** предусмотрены статические данные-члены с наглядными именами **LtRed**, **LtGreen**, **LtRed**, **LtMagenta** и др. (фактически они являются объектами класса **TColor**, включенными в состав класса в качестве его данных). Имеются также предопределенные члены с именами **White** и

Black. Кроме того, в состав класса включено большое количество членов-данных со стандартными цветами интерфейсных объектов Windows, таких как **SysBtnText**, **SysHighlight**, **SysDesktop** и др. При использовании этих предопределенных объектов необходимо указывать их принадлежность к классу **TColor**:

```
TColor clr2(TColor::LtRed);
TBrush br(clr2);
```

Очевидно, во многих случаях нет необходимости создавать отдельный объект класса **TColor**; вызов соответствующего конструктора этого класса можно включить в состав предложения, в котором требуется указать цвет:

```
TBrush br(TColor::LtRed);//Красная кисть
dc.SetTextColor(TColor::LtBlue);//Синий цвет букв
dc.SetBkColor(TColor(200,200,255));//Бледно-синий фон
```

Вывод растровых изображений

Вспомним основные этапы вывода растровых изображений с использованием средств API Windows:

- изображение, хранящееся в файле с расширением .BMP или непосредственно в загрузочном файле программы, загружается в память функцией **LoadImage()**, образуя совместимую память с изображением;
- создается контекст памяти, совместимый с нашим окном (функцией **CreateCompatibleDC()**);
- дескриптор области памяти с изображением выбирается в совместимый контекст (функцией **SelectBitmap()**);
- функцией **BitBlt()** изображение копируется из совместимой памяти в окно.

При использовании библиотеки OWL надо сделать, в сущности, то же самое, но с применением объектов подходящих классов и их функций-членов. На рис. 2.3 показана схема вывода растрового изображения с указанием участвующих в этой процедуре классов OWL, а также функций API Windows и инкапсулирующих их функций OWL (для функций с большим количеством параметров на рисунке указаны только важнейшие).

Рис. 2.3. Схема вывода растрового изображения

Можно выделить два типичных варианта использования в прикладной программе растровых изображений. В первом варианте определенный набор растровых изображений является неотъемлемой частью приложения. Такая ситуация типична, например, для компьютерных учебных пособий, в которых растровые изображения служат иллюстрациями. В этом случае все изображения, первоначально подготовленные в виде файлов с расширением .BMP, описываются в файле ресурсов, а в процессе компиляции и компоновки включаются в выполнимый файл приложения. Такое приложение является "самодостаточным": все, необходимое для выполнения приложения, находится в выполнимом модуле (файле с расширением .EXE).

Другой вариант предполагает вывод на экран изображений, хранящихся отдельно от выполнимого файла приложения в файлах с расширением .BMP. Так организуются программы, предполагающие просмотр и анализ многих изображений, например, из компьютерного атласа медицинских препаратов. В загрузочном файле такого приложения изображений, естественно, нет; они загружаются в память и отображаются на экране динамически, после ввода пользователем имени требуемого файла.

Рассмотрим сначала пример вывода заранее определенного изображения, хранящегося в загрузочном файле приложения. В этом случае в программный проект необходимо включить файл ресурсов, в котором указано имя файла с изображением.

```
/*Пример 2-2. Вывод изображения посредством файла ресурсов */
/*Файл 2-2.RC*/
pict BITMAP "myimage.bmp"
/*Файл 2-2.CPP*/
#include <owl/framewin.h>
class MyApp:public TApplication{
public:
  void InitMainWindow(void);
  };
class MyWin: public TFrameWindow{
  TBitmap* pBmp;//Указатель на объект совместимой памяти
  TMemoryDC* pmemDC;//Указатель на совместимый контекст
public:
  MyWin(TWindow* p,char* t):TFrameWindow(p,t){
    HINSTANCE hInst=GetModule()->GetInstance();
    pBmp=new TBitmap(hInst,"pict");
    pmemDC=new TMemoryDC(*pBmp);
    }
  void Paint(TDC&,bool,TRect&);
  };
void MyApp::InitMainWindow(void){
  MyWin* myWin=new MyWin(0,"2-2");
  SetMainWindow(myWin);
  }
void MyWin::Paint(TDC& dc,bool,TRect&){
  dc.BitBlt(
    5,5,pBmp->Width(),pBmp->Height(),*pmemDC,0,0,SRCCOPY);
   }
int OwlMain(int,char**){
   MyApp* myApp=new MyApp;
   return myApp->Run();
   }
```

Вся работа по выводу изображения разделена между двумя функциями класса **MyWin**: конструктором этого класса и функцией **Paint()**. Поскольку в данном примере и совместимая память, и ее контекст создаются в конструкторе класса, а использоваться они будут в функции **Paint()**, указатели на оба эти объекта включены в класс **MyWin** в качестве данных-членов, к которым, естественно, открыт доступ из всех функций данного класса:

```
TBitmap* pBmp;//Указатель на объект совместимой памяти
TMemoryDC* pmemDC;//Указатель на совместимый контекст
```

В конструкторе класса **MyWin** выполняются две операции: создания объекта совместимой памяти с загрузкой в нее изображения и создание объекта контекста совместимой памяти с выбором в него самой совместимой памяти.

Для создания объекта совместимой памяти с загрузкой в нее изображения из ресурса (находящегося в загрузочном файле приложения) используется один из конструкторов класса **TBitmap** (всего в этом классе 10 перегруженных конструкторов), у которого в качестве первого параметра указывается дескриптор экземпляра приложения, а в качестве второго – имя ресурса в файле ресурсов (в нашем случае **"pict"**). Для получения дескриптора экземпляра приложения можно воспользоваться функцией-членом базового класса **TModule GetInstance()** (см. рис. 1.2 и текст к нему), однако она должна вызываться для объекта данного модуля, а ни имени модуля, ни указателя на него у нас нет. Поэтому в предложении

```
HINSTANCE hInst=GetModule()->GetInstance();
```

сначала вызывается функция **GetModule()**, которая возвращает указатель на модуль приложения, а затем посредством этого указателя вызывается функция **GetInstance()**, возвращающая в переменную **hInst** нужный нам дескриптор. В следующем предложении конструируется объект совместимой памяти:

```
pBmp=new TBitmap(hInst,"pict");
```

Для создания объекта контекста совместимой памяти (т. е. объекта класса **TMemoryDC**), необходимо выбрать удобный конструктор класса **TMemoryDC**. В этом классе предусмотрено 5 перегруженных конструкторов; если использовать конструктор с параметром-указателем на совместимую память

```
TMemoryDC memDC(*pBmp);
```

то при создании совместимого контекста в него сразу же выбирается дескриптор растра.

Наконец, в функции **Paint()** можно выполнить то, ради чего все это затевалось – вызовом функции **BitBlt()** скопировать изображение из совместимой памяти в окно приложения (из совместимого контекста в контекст окна):

```
dc.BitBlt(5,5,pBmp->Width(),pBmp->Height(),*pmemDC,0,0,SRCCOPY);
```

Как мы видим, размеры копируемого изображения, которые надо указывать при вызове функции **BitBlt()**, определяются очень просто, поскольку в классе **TBitmap** предусмотрены для этого соответствующие функции **Width()** и **Height()**. Обратите внимание на обозначение контекста совместимой памяти: у нас есть указатель на контекст, а для функции **BitBlt()** требуется имя объекта, которое образуется разыменованием указателя (***pmemDC**).

Получив контекст совместимой памяти с изображением, можно использовать его для вывода поверх изображения любых графических объектов (текста, геометрических фигур и др.). Таким образом можно, например, наглядно выделять отдельные элементы изображения. Разумеется, это надо сделать *до* копирования растра в окно:

```
memDC.SelectObject(TPen(TColor::White,2,PS_INSIDEFRAME));
memDC.SelectStockObject(NULL_BRUSH);
memDC.Ellipse(35,25,128,118);
dc.BitBlt(...);
```

На рис. 2.4 приведен пример такого комбинированного изображения.

Рис. 2.4. Фрагмент изображения мазка крови под микроскопом. Сплошной окружностью обведен лейкоцит, пунктирной - эритроцит

Рассмотрим теперь процедуру вывода в окно приложения произвольного растрового изображения, хранящегося в файле с расширением .BMP. Поскольку по сравнению с предыдущим примером изменениям подвергнутся только данные-члены класса **MyWin**, конструктор этого класса и функция **Paint()**, ниже приводятся лишь эти фрагменты программы.

Среди упоминавшихся выше 10 конструкторов класса **TBitmap** нет конструктора, формирующего объект класса из файла .BMP.

Поэтому для загрузки изображения из файла придется воспользоваться в качестве промежуточного элемента объектом класса **TDib**, который описывает изображения в независимом от устройства формате (в противоположность этому объект класса **TBitmap** описывает изображения в зависимом от устройства формате). Для того чтобы указатель **pDib** на объект класса **TDib** был доступен всем функциям класса **TWindow**, он, как и указатель **pBmp**, объявлен в составе данных-членов этого класса:

```
class MyWin: public TFrameWindow{
  TBitmap* pBmp;//Указатель на зависимое от устройства изображение
  TMemoryDC* pmemDC;//Указатель на совместимый контекст
  TDib* pDib;//Указатель на независимое от устройства изображение
```

Объект класса **TDib** естественно создать в конструкторе окна:

```
MyWin(TWindow* p,char* t):TFrameWindow (p,t){
  pDib=new TDib("myimage.bmp");//Загружаем изображение из файла
  TClientDC dc(HWindow);//Создаем объект контекста окна
  pBmp=new TBitmap(dc,*pDib);Создаем совместимую память
  pMemDC=new TMemoryDC(*pBmp);//Создаем совместимый контекст
  }
```

В первой строке этого фрагмента мы создаем объект класса **TDib**, загружая в него изображение из файла, имя которого указывается в качестве параметра конструктора. Вспомним, что создать объект класса **TBitmap** таким же способом нельзя – в классе **TBitmap** нет конструктора с параметром-именем файла.

Далее надо создать совместимую память и совместимый контекст и выбрать память в контекст. Здесь придется воспользоваться другим конструктором совместимой памяти (объекта класса **TBitmap**), который образует блок памяти не из ресурса приложения, а из объекта класса **TDib**. Однако этот конструктор требует в качестве первого параметра обычный контекст окна, который автоматически поступает в приложение при активизации функции **Paint()**, а в том месте программы, где мы находимся, т. е. в конструкторе окна, его нет. Объект контекста окна создается вызовом конструктора класса **TClientDC**; в качестве параметра этого конструктора (дескриптор главного окна типа **HWND**) можно использовать данное-член **HWindow** из базового класса **TWindow**:

```
  TClientDC dc(HWindow);
```

Получив контекст окна, можно создать объект совместимой памяти, а затем объект ее контекста с выбором в него совместимой памяти:

```
pBmp=new TBitmap(dc,*pDib);
pMemDC=new TMemoryDC(*pBmp);
```

В данном случае имя загружаемого файла с изображением однозначно указано в тексте программы. Однако не составляет труда ввести это имя с клавиатуры или сформировать его каким-либо иным способом (например, перебором доступных файлов .BMP).

В функции **Paint()**, как и в предыдущем варианте, мы просто вызываем функцию **BitBlt()** копирования совместимой памяти в окно приложения.

3. ОРГАНИЗАЦИЯ МЕНЮ И ОБРАБОТКА СООБЩЕНИЙ WINDOWS

Программа с меню в главном окне

Обсуждение вопроса об обработке сообщений Windows в OWL-приложениях удобно начать с рассмотрения примера простого приложения с меню в главном окне. Пусть, например, с помощью пунктов меню мы хотим изменять оттенок цвета окна приложения. Приведем полный текст этой программы.

```
/*Пример 3-1. Программа с меню*/
/*Файл 3-1.H*/
#define MI_1 101
#define MI_2 102
/*Файл 3-1.RC*/
#include<owl\window.rh>
#include "03-10.h"
Main MENU {
  POPUP "Цвет окна" {
    MENUITEM "Бледный", MI_1
    MENUITEM "Яркий", MI_2
    MENUITEM "Выход", CM_EXIT
    }
  }
```

```
/*Файл 3-1.CPP*/
#include <owl\framewin.h>
#include "03-10.h"
class MyApp:public TApplication{
public:
  virtual void InitMainWindow(void);
};
class MyWin:public TFrameWindow{
  int sw;//Переключатель цвета окна
  void m1();//Функция отклика на пункт меню MI_1
  void m2();//Функция отклика на пункт меню MI_2
  void Paint(TDC&,bool,TRect&);
public:
  MyWin(TWindow*parent,char* title);
  DECLARE_RESPONSE_TABLE(MyWin);//Объявим таблицу откликов
  };
/*Таблица откликов на выбор пунктов меню*/
DEFINE_RESPONSE_TABLE1(MyWin,TFrameWindow)
  EV_COMMAND(MI_1,m1),//Идентификатор пункта и имя функции отклика
  EV_COMMAND(MI_2,m2),//Другой пункт, другая функция
END_RESPONSE_TABLE;//Конец таблицы откликов
void MyApp::InitMainWindow(void){
  MyWin* myWin=new MyWin(0,"Фон");
  SetMainWindow(myWin);
  }
MyWin::MyWin(TWindow* p,char* t):TFrameWindow(p,t){
  AssignMenu("Main");//Назначение окну myWin меню с именем Main
  }
/*Функция отклика на выбор первого пункта меню*/
void MyWin::m1(){
   sw=1;
   Invalidate();
}
/*Функция отклика на выбор второго пункта меню*/
void MyWin::m2(){
   sw=2;
   Invalidate();
   }
void MyWin::Paint(TDC& dc,bool,TRect&rect){
  TBrush br1(TColor(220,255,255));//Создадим бледную кисть
  TBrush br2(TColor::LtCyan); //Создадим яркую кисть
  if(sw==1){
    dc.FillRect(rect,br1);//Закрасим область rect кистью br1
    }
```

```
  if(sw==2){
     dc.FillRect(rect,br2);//Закрасим область rect кистью br2
     }
  }
int OwlMain(int,char*[]){
  MyApp* pMyApp=new MyApp;
  return pMyApp->Run();
}
```

Меню главного окна приложения описывается, как обычно, в файле ресурсов .RC. Значения идентификаторов пунктов меню **MI_1** и **MI_2** (как имена идентификаторов, так и их значения могут быть любыми) определены нами в файле .H. Для пункта выхода из приложения использован стандартный идентификатор **CM_EXIT** (со значением 24310), описанный в файле WINDOW.RH. Замечательным свойством этого идентификатора является то, что при его использовании отпадает необходимость обрабатывать в программе этот пункт меню – при его выборе завершение приложения выполняется автоматически.

Для того чтобы меню появилось в окне приложения, достаточно вызвать функцию **AssignMenu()** класса **TFrameWindow** с указанием в качестве параметра имени меню из файла ресурсов; этот вызов удобнее всего выполнить в конструкторе класса **MyWin**.

Для обработки сообщений Windows от пунктов меню в программе необходимо предусмотреть следующие элементы:

- объявление в классе главного окна таблицы откликов (макрос **DECLARE_RESPONSE_TABLE**);
- саму таблицу откликов (макросы **DEFINE_RESPONSE_TABLE1** и **END_RESPONSE_TABLE**, а также помещаемые между ними макросы, определяющие конкретные сообщения);
- объявление в классе главного окна прототипов функций откликов, обрабатывающих по требуемым алгоритмам описанные в таблице откликов сообщения Windows;
- определения всех предусмотренных функций откликов.

Рассмотрим эти вопросы более детально.

Таблица откликов и обработка сообщений от пунктов меню

Таблица откликов объявляется как элемент класса главного окна с помощью макроса **DECLARE_RESPONSE_TABLE** с именем нашего конкретного класса в качестве параметра. При расширении этого макроса в класс главного окна включаются объявления некоторых служебных средств, обеспечивающих нахождение в таблице откликов элементов, соответствующих приходящим сообщениям. Поскольку в расширении макроса присутствуют "собственные" спецификаторы доступа (**private** и **public**), его можно располагать в любой секции класса, однако после этого макроса необходимо соответствующим спецификатором восстановить требуемый уровень доступа. Обычно макрос объявления таблицы откликов помещают в конце описания класса, и тогда в восстановлении уровня доступа нет необходимости.

Таблица откликов может располагаться в любом месте программы. Она начинается с макроса **DEFINE_RESPONSE_TABLE1**, в качестве параметров которого указываются имя класса главного окна (**MyWin**), а также родительского класса (**TFrameWindow**).

Формат таблицы откликов зависит от рода обрабатываемых сообщений. Для обработки сообщений от пунктов меню в таблицу откликов включаются макросы **EV_COMMAND** по одному на каждый пункт меню. Эти макросы требуют указания двух параметров – идентификатора соответствующего пункта меню (который закрепляется за пунктом меню в предложении файла ресурсов, описывающим этот пункт меню) и имени функции отклика, т. е. прикладной функции, которая будет вызываться при выборе этого пункта. Таблица откликов завершается макросом **END_RESPONSE_TABLE**, за которым следует знак точки с запятой.

Имена функций откликов программист выбирает произвольно; как видно из приведенного выше текста программы, эти имена неминуемо встречаются в программе в трех местах:

- при объявлении прототипов этих функций в составе класса **MyWin**;
- в макросах таблицы откликов;
- в определениях этих функций.

Так же, как и при процедурном программировании, заранее трудно угадать состав параметров каждой функции отклика. Перечень стандартных сообщений Windows с указанием формата функций откликов можно найти в интерактивном справочнике, входящем в состав пакета Borland C++, в статье Object Windows 5.0 Reference Guide→Object Windows Event Handlers→Standard Windows Messages. Функции откликов на выбор пунктов меню (т. е. в ответ на сообщения **WM_COMMAND**) имеют очень простой прототип

```
void Func();
```

(см. функции **m1** и **m2** примера 3-1); у функций обработки других сообщений прототипы могут отличаться.

Вернемся к тексту примера 3-1. В класс **MyWin** включена целочисленная переменная **sw**, используемая для переключения цвета фона окна. Ее значения (1 или 2) устанавливаются в функциях откликов **m1()** и **m2()**, а анализируются в функции **Paint()**. Для того чтобы при выборе пункта меню окно перекрашивалось, в функции откликов включены вызовы функции **Invalidate()** класса **TWindow** (аналога функции **InvalidateRect()** API Windows), которая заставляет Windows послать в наше окно сообщение **WM_PAINT**. Если функцию **Invalidate()** вызывать без параметра (как в нашем примере), при перекрашивании окна предыдущее изображение затирается; если его требуется сохранять, накладывая новые изображения на предыдущие, функцию **Invalidate()** следует вызывать с параметром **false**.

В функции **Paint()** окно закрашивается требуемой кистью вызовом функции **FillRect()** класса **TDC**. Как уже отмечалось, координаты окна можно получить из последнего параметра функции **Paint()**.

Обработка других сообщений

Как известно, Windows посылает в окна приложения большое количество разнообразных сообщений. Некоторые из них являются прямой реакцией на действия пользователя (например, сообщения **WM_COMMAND**, которые поступают в окно приложения при выборе пунктов меню или щелчках по кнопкам); другие возникают также от действий пользователя, но, можно сказать, неявным образом

(например, сообщение **WM_PAINT** при раскрытии окна); отдельные сообщения вообще не связаны с действиями пользователя (сообщение **WM_TIMER**); наконец, некоторые сообщения приходят в приложение на некотором этапе его выполнения (например, **WM_CREATE** при создании окна или **WM_INITDIALOG** при открытии диалога). С точки зрения способа обработки в OWL-приложении все сообщения можно разбить на три группы.

1. Замещается *имеющаяся* в классе OWL функция отклика, которая автоматически вызывается при поступлении соответствующего сообщения. Наиболее характерной функцией такого рода является функция **Paint()** класса **TWindow** (и ряда других классов), вызываемая системой в ответ на сообщение **WM_PAINT**. Другой пример – функция **EvInitDialog()**, которая вызывается при поступлении сообщения **WM_INITDIALOG** перед выводом диалогового окна на экран.

Поскольку замещающая функция должна в точности соответствовать замещаемой, необходимо по справочнику уточнить имя и список параметров замещаемой функции, например:

```
void TWindow::Paint(TDC&,bool,TRect);
bool TDialog::EvInitDialog(HWND);
```

Для этих сообщений *не нужна таблица откликов*.

2. Для обработки сообщений WM_COMMAND в таблицу откликов (окна или диалога) включается набор макросов вида

```
EV_COMMAND(ID1,Func1),
EV_COMMAND(ID2,Func2),
```

а в соответствующий класс – набор функций отклика с произвольными (но указанными в таблице откликов) именами:

```
void Func1();
void Func2();
```

3. Для обработки всех остальных сообщений в таблицу откликов соответствующего класса включаются макросы вида

EV_WM_*имя-сообщения*

а в класс – функция с *предопределенным* именем. В табл. 3-1 приведены некоторые макросы и соответствующие им функции такого рода.

Таблица 3-1. Макросы и прототипы функций отклика

Макрос	*Функция отклика*
EV_WM_CREATE	`int TWindow::EvCreate(CREATESTRUCT&)`
EV_WM_TIMER	`void TWindow::EvTimer(UINT)`
EV_WM_CTLCOLOR	`HBRUSH EvCtlColor(HDC,HWND,uint)`
EV_WM_PAINT	`void TDialog::EvPaint()`
EV_WM_LBUTTONDOWN	`void EvLButtonDown(uint,TPoint&)`
EV_WM_RBUTTONDOWN	`void EvRButtonDown(uint,TPoint&)`
EV_WM_LBUTTONDBLCLK	`void EvLButtonDblClk(uint,TPoint&)`
EV_WM_RBUTTONDBLCLK	`void EvRButtonDblClk(uint,TPoint&)`

Управление программой с помощью мыши

Для обработки в программе щелчков клавиш мыши предусмотрен целый ряд макросов, включаемых в таблицу откликов, и соответствующих им функций откликов, наиболее важные из которых перечислены в табл. 3-1. Так, при необходимости выполнить в программе некоторые действия при щелчке левой клавишей мыши, в таблицу откликов включается макрос **EV_WM_LBUTTONDOWN**, а в класс окна – замещающая функция **EvLButtonDown()**:

```
DEFINE_RESPONSE_TABLE1(MyWin,TFrameWindow)
  EV_WM_LBUTTONDOWN,
END_RESPONSE_TABLE;

void MyWin::EvLButtonDown(uint,TPoint&){
...//Действия, выполняемые при щелчке левой клавишей мыши
  }
```

Во все функции откликов на операции с мышью через второй параметр типа **TPoint&** поступают текущие координаты мыши, которые при необходимости можно использовать в функции отклика. Через первый параметр (типа **uint**) в функции откликов поступает информация о состоянии управляющих клавиш **Ctrl** и **Shift**, а также и о том, какая клавиша мыши нажата. Эти данные помещаются системой в первые четыре бита параметра, причем каждый из битов имеет определенное символическое обозначение

(рис. 3-1). Таким образом, первый параметр функций откликов на операции с мышью позволяет выделить щелчки клавиш мыши (одиночные или двойные), выполняемые при нажатых клавишах **Ctrl** и **Shift**:

```
/* Функция отклика на щелчок правой клавишей мыши */
void MyWin::EvRButtonDown(uint modKeys,TPoint&){
  if(modKeys == MK_RBUTTON)
    ...//Если ни одна управляющая клавиша клавиатуры не нажата
  if(modKeys & MK_CONTROL) == MK_CONTROL)
    ...//Если щелчок сделан при нажатой клавише Ctrl
  if(modKeys & MK_SHIFT) == MK_SHIFT)
    ...//Если щелчок сделан при нажатой клавише Shift
  }
```

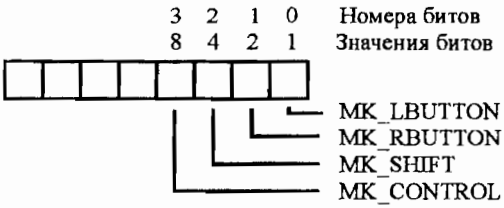

Рис. 3-1. Символические константы, поступающие в функции откликов на щелчки мышью

4. ДИАЛОГОВЫЕ ОКНА

Как известно, диалоговые окна, или просто диалоги, бывают двух видов: модальные и немодальные. *Модальный* диалог характерен тем, что после открытия диалогового окна пользователь может работать только с *его* элементами управления; все приложение блокируется до закрытия диалога. С помощью модального диалога, например, осуществляется поиск на диске файла, без которого программа дальше работать не может. *Немодальные* диалоги не препятствуют пользователю работать с программой и часто используются для изменения масштаба, цвета или способа отображения наблюдаемой на экране информации. Особым видом организации приложения Windows является использование модального диалога в качестве главного окна приложения. Такой режим удобен, в частности, для вывода результатов работы вычислительных программ; он также широко используется в программах управления автомати-

зированными установками. В настоящем пособии будут рассмотрены два аспекта диалоговых окон: работа со стандартным модальным диалогом "Открытие файла" и организация приложения на базе модального диалога.

Стандартный модальный диалог "Открытие файла"

Система Windows предоставляет несколько стандартных модальных диалогов, из которых наибольший интерес представляют диалоги для открытия и сохранения файлов. Рассмотрим фрагмент программы, в котором для поиска требуемого файла используется стандартный диалог "Открытие файла". Будем для определенности считать, что файл содержит числовые данные, которые должны быть выведены в окно приложения в форме графика. Очевидно, что программа должна иметь меню **"Файл"**, в котором имеется пункт **"Открыть"**, и все действия со стандартным диалогом должны выполняться в функции отклика на этот пункт. Только это функция (пусть она имеет имя **Open()**) нас и будет интересовать.

```
/*Пример 4-1 (фрагмент). Стандартный диалог "Открытие файла"*/
void MyWin::Open(){
  TOpenSaveDialog::TData* data;//Объявляем указатель data
/*Создаем структурную переменную типа TData*/
  TData* data=new TOpenSaveDialog::TData(
    OFN_FILEMUSTEXIST,//Флаг инициации диалога
/*Описание шаблонов выводимых имен файлов */
    "Файлы данных (*.dat)|*.DAT|Все файлы (*.*)|*.*|");
/*Создаем диалог "Открытие файла"*/
  TFileOpenDialog* fod=new TFileOpenDialog(this,*data);
  if(fod->DoExecute()==IDOK){//Если файл выбран
/*Открываем выбранный файл для чтения*/
    HANDLE hFile=CreateFile(data->FileName,GENERIC_READ,0,
              NULL,OPEN_EXISTING,0,NULL);
    DWORD dw;
    int d[400];//Пусть в файле 400 данных типа int
    ReadFile(hFile,&d,1600,&dw,NULL);//Читаем 1600 байт
    CloseHandle(hFile);//Файл больше не нужен, закроем его
/*Данные прочитаны. Теперь их можно вывести в окно в виде
графика или обрабатывать любым требуемым способом. */
    }//Конец конструкции if
  }//Конец функции Open()
```

Диалог “Открытие файла” создается как объект класса OWL **TFileOpenDialog**. Конструктор этого класса требует в качестве первого параметра указатель на окно, в которое выводится диалог, а в качестве второго – переменную типа структуры **TData** (описанной в классе **TFileOpenDialog**) с заполнением некоторых элементов этой структуры. Предложением

```
TOpenSaveDialog::TData* data;
```

в программе объявляется указатель **data** на структуру типа **TData**, принадлежащую классу **TOpenSaveDialog**, после чего предложением

```
data=new TOpenSaveDialog::TData(OFN_FILEMUSTEXIST,
    "Файлы данных (*.dat)|*.DAT|Все файлы (*.*)|*.*|");
```

с помощью оператора **new** создается сама структура (опять с указанием принадлежности типа **TData** классу **TOpenSaveDialog**).

В качестве аргументов конструктора **TData** указывается флаг инициации диалога (в данном случае **OFN_FILEMUSTEXIST**, требующий наличия на диске указанного пользователем файла) и символьный фильтр, в который в определенном формате записываются строки, которые будут выведены в окно "**Тип файлов:**" диалога вместе с шаблонами спецификаций отбираемых файлов. В данном примере предусматривается возможность наблюдения в главном окне диалога “Открытие файла” либо всех файлов с расширением **.DAT**, либо вообще всех файлов с любыми расширениями (***.***).

Подготовив структурную переменную типа **TData** с инициализирующими данными, можно создать объект класса **TFileOpenDialog** (назовем его **fod**):

```
TFileOpenDialog* fod=new TFileOpenDialog(this,*data);
```

В качестве первого аргумента выступает указатель на объект окна, в котором открывается диалог, а в качестве второго – инициализирующая структурная переменная типа **TData**. Поскольку мы объявили эту переменную с помощью указателя, а конструктор требует имя переменной, в качестве аргумента указывается указатель со снятой ссылкой.

Создав объект диалога, можно запустить его выполнение. Для этого следует вызвать функцию **DoExecute()** из класса **TFi-**

leOpenDialog. Однако последующие действия (в данном случае открытие файла, чтение из него данных и отображение их в окне приложения) должны выполняться, лишь если пользователь правильно выберет файл. Поэтому вызов функции **DoExecute()** следует поместить в предложение **if** (если файл выбран успешно, функция **DoExecute()** возвращает значение **IDOK**):

```
if(fod->DoExecute()==IDOK){Все действия для выбранного файла}
```

На рис. 4.1 приведен вид диалога "Открытие файла", созданного в приведенном выше фрагменте.

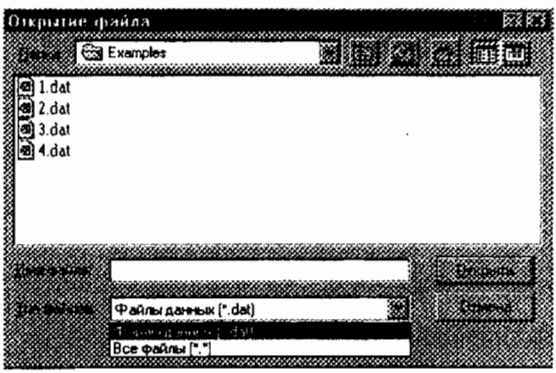

Рис. 4.1. Стандартный диалог "Открытие файла"

Выбрав имя файла, можно его открыть и, прочитав содержащиеся в нем данные, приступить к их обработке.

Модальный диалог в качестве главного окна приложения

В обычном приложении Windows для того чтобы использовать модальный диалог в качестве главного окна приложения, достаточно было в главной функции приложения **WinMain()** вместо создания главного окна вызвать функцию создания модального диалога **DialogBox()**. В OWL-приложении процедура оказывается сложнее. Рассмотрим пример приложения, в котором отсутствует главное окно, а модальный диалог используется для вывода в элемент управления-список содержимого некоторого числового фай-

ла. Заметим кстати, что такая относительно простая организация приложения чрезвычайно удобна, когда назначением приложения является обработка и визуальное отображение больших числовых массивов.

Пример 4-2. Модальный диалог в качестве главного окна приложения

```
/*Файл 4-2.H*/
#define ID_1 101//Идентификатор списка
#define ID_2 102//Идентификатор статического элемента
/*Файл 4-2.RC*/
#include "4-2.h"
Dlg DIALOG 20, 20, 72, 154 {
  CONTROL "",ID_1,"LISTBOX",WS_VSCROLL|WS_BORDER,8,16,56,132
  CONTROL "",ID_2,"STATIC",SS_CENTER,6,4,60,8
  }
/*Файл 4-2.CPP*/
#include <owl/framewin.h>
#include <owl/dialog.h>
#include <owl/listbox.h>
#include <owl/static.h>
#include "4-2.h"
class MyApp:public TApplication{
public:
   void InitMainWindow(void);
   };
class MyDialog:public TDialog{
  TListBox* pLB;//Указатель на объект-список
  TStatic* pSt;//Указатель на объект-надпись
  void SetupWindow();//Замещаем ради инициализации
public:
  MyDialog(TWindow* parent,TResId resId)://Конструктор
             TDialog(parent,resId){//объекта диалога
    pLB=new TListBox(this,ID_1);//Создаем объект-список
    pSt=new TStatic(this,ID_2); //Создаем объект-надпись
    }
  };
void MyDialog::SetupWindow(){
  TDialog::SetupWindow();//Вызываем замещенную функцию
  HANDLE hFile = //Открываем конкретный файл с данными
    CreateFile("1.dat",GENERIC_READ,0,NULL,
    OPEN_EXISTING,0,NULL);
  DWORD dw;  int d[400];
  ReadFile(hFile,&d,1600,&dw,NULL);//Чтение данных
  CloseHandle(hFile);//Закрываем файл
```

```
  char txt[30];
  for(int i=0;i<400;i++){//В цикле заполняем список
    wsprintf(txt,"%03d    %d",i,d[i]);//Числа в символы
    pLB->AddString(txt);//Добавим строку в список
    }
  pSt->SetText("1.dat");//Выводим имя файла в стат. элемент
  }
void MyApp::InitMainWindow(void){
  MyDialog* myDlg=new MyDialog(0,"Dlg");//Создаем диалог
  TFrameWindow* win =//Создаем окно класса TFrameWindow
        new TFrameWindow(0,"Из файла",myDlg,true);
  SetMainWindow(win);//Назначаем его главным окном
  }
int OwlMain(int,char**){
  return new MyApp->Run();
  }
```

Результат работы программы показан на рис. 4.2.

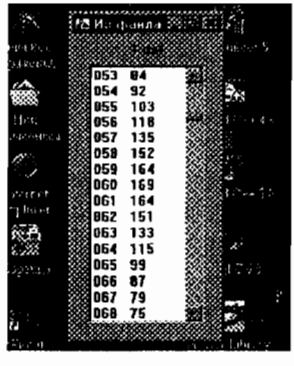

Рис. 4.2. Модальный диалог в качестве главного окна приложения

Рассмотрим структуру этой программы.

Как и всегда, формат диалога описывается в файле ресурсов. В данном примере в составе диалога имеются два интерфейсных элемента – список и статический элемент для вывода поясняющего текста (имени открываемого в программе файла с данными).

Поскольку обычного главного окна у нас не будет, в программе *не объявляется* класс, производный от **TFrameWindow** (в предыдущих примерах его указатель назывался **MyWin**). Вместо него

объявляется прикладной класс **MyDialog**, производный от **TDialog**. Для того чтобы можно было работать с элементами управления диалога (списком и статическим элементом) как с объектами, в класс **MyDialog** включены указатели на объекты классов **TListBox** и **TStatic** (обратите внимание на соответствующие директивы **#include**). Объекты этих классов будут созданы в конструкторе диалога. Кроме того, в классе **MyDialog** объявлена замещающая функция **SetupWindow()**, о которой речь пойдет ниже.

Собственно создание диалога осуществляется в замещающей функции **InitMainWindow()** несколько замысловатым образом. Здесь прежде всего с помощью оператора **new** создается объект **myDlg** нашего прикладного класса **MyDialog**, т. е. объект диалога. Вторым аргументом конструктора служит имя конкретного диалога, заданное в файле ресурсов.

Далее вызовом конструктора **TFrameWindow** создается объект **win** этого класса (а не производного от него класса **MyWin**, как это было во всех предыдущих примерах). Конструктору передаются три параметра. Первый обозначает окно, родительское для создаваемого; если родительского окна нет, значение параметра должно быть равно нулю. Второй параметр (как и ранее) – заголовок создаваемого окна. В качестве третьего параметра передается указатель на окно-клиент. Этот параметр по умолчанию равен 0, что говорит об отсутствии клиента; в предыдущих примерах это умолчание и было использовано. Здесь же клиентом назначается объект диалогового окна **myDlg**. Таким образом, наш модальный диалог фактически является клиентом окна (невидимого) класса **TFrameWindow**. Наконец, вызовом функции **SetMainWindow()** класса **TApplication** созданное окно объявляется главным.

Нам осталось выяснить, как осуществляется заполнение списка и вывод текста в статический элемент. Для этого надо рассмотреть понятия интерфейсных элементов и интерфейсных объектов.

Интерфейсные элементы и интерфейсные объекты

Общий принцип использования стандартной библиотеки объектов, в частности, библиотеки OWL, заключается в том, что для всех используемых в программе *интерфейсных элементов* (главного

окна, дочерних и диалоговых окон, кнопок, списков и пр.) мы создаем *интерфейсные объекты* соответствующих библиотечных (или прикладных, производных от них) классов. Интерфейсный элемент – это то, что мы видим на экране; интерфейсный объект – это экземпляр того или иного класса (обычно присутствующий в программе в виде указателя на класс), посредством которого мы можем обращаться ко всем программным средствам – функциям-членам и данным-членам, включенным в этот класс.

В соответствии с этим принципом в класс **MyDialog** включены указатели **pLB** и **pSt** на классы интерфейсных элементов, а в конструкторе диалога имеются предложения создания интерфейсных объектов этих классов. В качестве первого параметра конструкторам объектов передается указатель на родительский для них класс диалога (указатель **this**), а в качестве второго – идентификатор, закрепленный за интерфейсным элементом в файле ресурсов.

Создание в программе интерфейсных объектов обычно (хотя не всегда) приводит к отображению на экране соответствующих интерфейсных элементов (у нас – списка и статического элемента). Это отображение выполняют программы OWL, которые, разумеется, не знают, чем мы хотим заполнить интерфейсные элементы.

Для того чтобы программист мог вмешаться в процесс создания интерфейсного элемента и заполнить его требуемым содержимым, в состав большинства интерфейсных классов OWL входит специфическая функция, носящая имя **SetupWindow()**. Эта функция вызывается автоматически (не программистом!) сразу же после создания объекта соответствующего класса и выполняет создание дочерних элементов этого класса (для диалога – кнопок, списков и пр.). Замещение этой функции в программе позволяет выполнять необходимые инициализирующие действия над интерфейсными элементами. По этой причине мы и включили в прикладной класс **MyDialog** замещающую функцию **SetupWindow()**.

Однако эта функция прежде всего должна выполнить системные действия по созданию дочерних элементов. Для этого необходимо вызвать ее замещенный вариант из базового класса (в нашем случае – класса **TDialog**):

```
TDialog::SetupWindow();//Вызываем замещенную функцию
```

Текст прикладной функции **SetupWindow()** всегда должен начинаться с этого предложения.

В нашем случае замещающая функция **SetupWindow()** относительно сложна. В ней (см. пример 4.2) прежде всего открывается файл с данными. В примере файл имеет фиксированное имя 1.DAT, хотя не составляет труда получить имя файла с помощью стандартного диалога "Открытие файла", как это было описано выше. Содержимое файла читается в массив с именем **d**, после чего идут действия, ради которых мы заместили функцию **SetupWindow()**: заполнение списка и вывод строки текста в статический элемент.

Для заполнения списка содержимым файла мы в цикле по всем данным, входящим в файл, формируем символьные строки с данными и вызываем для объекта списка (указатель **pLB**) функцию **AddString()**, входящую в состав класса **TListBox**. Именно включение в программу интерфейсного объекта **pLB** и дало нам возможность обращаться ко всему богатому набору функций соответствующего класса OWL. Кроме функции **AddString()** добавления строк в список, в этот класс входят **функции DeleteString()** удаления строк, **GetSelIndex()** получения индекса выбранной строки, **GetString()** получения строки списка по индексу, **ClearList** очистки списка и многие другие (всего их больше трех десятков).

Завершив цикл заполнения списка, мы с помощью указателя **pSt** выводим в статический элемент имя прочитанного файла.

Взаимодействие пользователя с элементами управления диалогового окна

Выше были приведены примеры взаимодействия с элементами диалога, инициируемого программой. В любом месте по ходу выполнения программы можно вызвать функцию **SetText()** для динамического изменения строки текста в диалоговом окне, **Addstring()** или **DeleteString()** для изменения содержимого списка и т. д. Теперь мы рассмотрим реакцию программы на сообщения от элементов управления диалогового окна. Обычно эти сообщения инициируются пользователем, щелкающим по кнопке, выбирающим строку списка и пр. Для их обработки необходимо включить в таблицу откликов соответствующие макросы. При этом надо различать сообщения, инициируемые *самим диалоговым окном*, и сообщения, поступающие от *дочерних элементов* диалога – кнопок, списков и т. д.

В табл. 4.1 приведены наиболее употребительные макросы таблицы откликов диалога, отрабатываемые ими события и предопределенные имена функций откликов.

Таблица 4.1. Некоторые макросы таблицы откликов диалога

Макрос	*Отрабатываемое событие*	*Имя функции отклика*
EV_WM_PAINT	Открывается хотя бы часть диалогового окна	**EvPaint**
EV_COMMAND	Щелчок по кнопке	Любое
EV_CHILD_NOTIFY	Действие над элементом управления	Любое
EV_WM_CTLCOLOR	Windows красит диалог	**EvCtlColor**

Рассмотрим особенности отработки перечисленных событий.

При включении в таблицу откликов макроса **EV_WM_PAINT**, а в состав класса диалога – функции отклика **EvPaint()**, возникает возможность вывода в диалоговое окно графических элементов – текстовых строк, геометрических фигур, графиков и пр. Для получения доступа к функциям GDI необходимо создать объект класса **TPaintDC** для данного окна. Приведем пример вывода в диалоговое окно строки текста:

```
void MyDialog::EvPaint(){
  TPaintDC dc(*this);
  dc.SetTextColor(TColor::LtBlue);
  dc.SetBkMode(TRANSPARENT);
  dc.TextOut(10,10,"График функции");
  }
```

Нетрудно включить в этот фрагмент цикл вывода точек графика, отображающего математическую функцию или другие данные. Используя, например, функцию **Ellipse()** класса TDC, можно отображать каждую точку графика небольшим цветным кружочком.

Макросы **EV_COMMAND** позволяют отрабатывать нажатие кнопок, помещенных в диалоговое окно. На каждую кнопку требуется отдельный макрос с указанием функции отклика, закрепленной за данной кнопкой:

```
EV_COMMAND(ID_BTN1,btn1),
EV_COMMAND(ID_BTN2,btn2),
```

Здесь **ID_BTN1** и **ID_BTN2** – идентификаторы, закрепленные за кнопками, а **btn1** и **btn2** – соответствующие функции отклика. Обработка нажатий кнопок ничем не отличается от обработки сообщений от пунктов меню (см. раздел 3 настоящего пособия).

Макросы **EV_CHILD_NOTIFY** позволяют обрабатывать сообщения как от кнопок, так и от других элементов управления, включенных в диалоговое окно. Поскольку для многих элементов допустимы различные действия (например, строку в списке можно выбрать одиночным или двойным щелчком мыши), макросы **EV_CHILD_NOTIFY** требуют указания в качестве второго параметра так называемого кода оповещения. Так, при использовании для списка с идентификатором **ID_LIST** макроса

```
EV_CHILD_NOTIFY (ID_LIST,LBN_SELCHANGE,Choice)
```

передача управления на функцию **Choice()** произойдет при выборе любой строки списка с помощью мыши или клавиатуры, а при использовании макроса

```
EV_CHILD_NOTIFY (ID_LIST,LBN_DBLCLK,DoubleClick)
```

функция **DoubleClick()** будет вызвана только при двойном щелчке мышью по той или иной строке списка.

Перекрашивание диалогового окна и его элементов

При включении в таблицу откликов макроса **EV_WM_CTLCOLOR**, а в состав класса диалога – функции отклика **EvCtlColor()**, возникает возможность покраски диалогового окна и его элементов. При этом необходимо иметь в виду, что функция **EvCtlColor()** будет вызываться неоднократно, во-первых, при покраске всего диалогового окна, и во-вторых – при покраске каждого из имеющихся в диалоге элементов управления. Функция **EvCtlColor()** имеет следующий прототип:

```
HBRUSH EvCtlColor(HDC hdc, HWND hwndCtl,uint);
```

Здесь **hdc** – дескриптор контекста устройства конкретного элемента управления, а **hwndCtl** – дескриптор самого элемента управления. Функция должна возвращать дескриптор кисти, которой красится весь диалог или окно конкретного элемента в нем.

Рассмотрим случай, когда в диалоговом окне имеется элемент управления-список с идентификатором **ID_LIST**, и мы хотим покрасить оба окна – и диалога, и списка – в некоторый цвет и, кроме того, задать определенный цвет строкам списка. Функция **EvCtlColor()** будет иметь следующий вид:

```
HBRUSH EvCtlColor(HDC hdc,HWND,uint){
  SetTextColor(hdc,RGB(0,0,255))//Синий цвет символов
  SetBkMode(hdc,TRANSPARENT);//Прозрачный фон под символами
  return(CreateSolidBrush(RGB(0,255,255));//Бирюзовые окна
  }
```

Обратите внимание на то, что функция **EvCtlColor()** использует в качестве параметров и возвращаемого значения не объекты OWL, а переменные API Windows. Поэтому весь текст функции **EvCtlColor()** приходится писать с использованием традиционных функций. Правда, в них можно использовать и объекты OWL:

```
HBRUSH EvCtlColor(HDC hdc,HWND,uint){
  SetTextColor(hdc,TColor::LtBlue)//Синий цвет символов
  SetBkMode(hdc,TRANSPARENT);//Прозрачный фон под символами
  return(TBrush(TColor::LtCyan));//Бирюзовые окна
  }
```

Рассмотрим теперь случай, когда мы хотим окно диалога покрасить в один цвет, а окно списка – в другой. Для этого придется анализировать второй параметр функции **EvCtlColor()** и отдельно обрабатывать вызовы этой функции для окна списка и для окна диалога (вспомним, что функция **EvCtlColor()** вызывается системой неоднократно):

```
HBRUSH EvCtlColor(HDC hdc,HWND hwndCtl,uint){
  if(hwndChild = = GetDlgItem(ID_LIST)){//Вызов для списка
    SetBkMode(hdc,TRANSPARENT);//Прозрачный фон
    SetTextColor(hdc,RGB(200,150,0));//Коричневые символы
    return TBrush(TColor::LtYellow));//Желтое окно списка
    }
  return(TBrush(TColor::LtCyan));//Бирюзовое окна диалога
  }
```

Дескриптор элемента управления (в данном случае – списка с идентификатором **ID_LIST**) можно получить с помощью функции **GetDlgItem()** класса **TWindow**.

5. ПРОГРАММИРОВАНИЕ В СРЕДЕ BORLAND C++ BUILDER

Среда программирования Borland C++ Builder

Программный пакет C++ Builder представляет собой интегрированную среду программирования (Integrated Development Environment, IDE), которая обеспечивает весь цикл разработки приложения (подготовку исходного текста, компиляцию, пробное исполнение и отладку) в полноэкранном интерактивном режиме в среде Windows.

В отличие от IDE Borland C++, Borland C++ Builder реализует идеи визуального программирования, при котором часть действий по организации разрабатываемого приложения осуществляется путем перетаскивания мышью на форму-заготовку элементов графического оформления окна (меню, диалоговых окон, окон редактирования, кнопок, списков, надписей и пр.). Тот же визуальный подход используется и для включения в программу таких элементов ее функционирования, как таймеры, стандартные диалоги Windows, проигрыватели звуковых или видеофрагментов и др.

Разумеется, прикладные алгоритмы вычислений или обработки событий программист, как и раньше, должен включать в программу "вручную", в виде программных строк, однако и в этой части среда заметно упрощает процесс разработки программ, предоставляя программисту шаблоны-заготовки для функций отклика на такие события в системе, как, например, выбор пункта меню или срабатывание таймера.

Среда C++ Builder является существенно объектно-ориентированной. Все компоненты среды, включаемые программистом в свою программу, представляют собой экземпляры соответствующих классов. Взаимодействие программиста с этими компонентами в значительной степени заключается либо в изменении значений данных-членов, либо в вызове функций-членов (методов) их классов. Таким образом, разработка приложений в среде C++ Builder требует основательного знакомства с методами и техникой объектно-ориентированного программирования.

При запуске IDE C++ Builder на экран выводится начальный кадр, содержащий большое количество различных элементов (рис. 5.1).

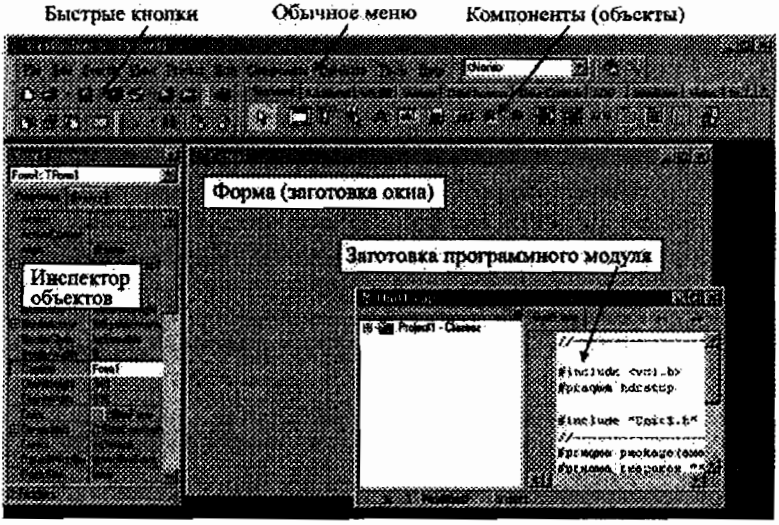

Рис. 5.1 Общий вид начального кадра IDE Borland C++ Builder

Общая идея программирования в C++ Builder заключается в перетаскивании мышью на форму требуемых компонентов (объектов) из панелей компонентов и включение в предоставляемые средой заготовки программных модулей (файлов с расширениями .CPP и .H) программных строк, реализующих прикладные алгоритмы.

Каждый объект C++ Builder обладает определенным набором *свойств* (Properties), предоставляет специфические для этого объекта *методы* (Methods) и генерирует определенные *события* (Events).

Свойства объекта можно отождествить с данными-членами класса, характеризующими внешний вид и состояние экземпляров этого класса. Например, для формы, интерфейсного объекта класса **TForm**, образующего главное окно приложения, а при необходимости и дочерние окна, характерны такие свойства как **Caption** (заголовок окна), **Color** (цвет фона), **Height** (высота) и **Width** (ширина), **Cursor** (курсор) и ряд других (всего более сотни свойств). Для таймера, объекта класса **TTimer**, позволяющего синхронизовать выполнение отдельных фрагментов приложения, характерны другие свойства: **Name** (имя), **Interval** (период срабатывания) и **Enable** (флаг активности).

Следует иметь в виду, что для многих объектов в окне инспектора объектов отображаются не все их свойства, а лишь те, которым присвоен атрибут **__published** (опубликованные). Полный перечень свойств объекта можно получить с помощью интерактивного справочника.

Методы характеризуют действия, которые можно выполнять над объектом. Так, в классе **TForm** предусмотрены методы **Tile()** и **Cascade()**, позволяющие расположить дочерние окна MDI-приложения мозаикой (первый метод) или каскадом (второй), **Close()**, закрывающий окно и все приложение, если окно является главным, **Hide()** и **Show()**, позволяющие скрыть или, наоборот, показать окно и многие другие. В состав методов класса входят также его конструкторы и деструктор.

Некоторые методы перекрываются со свойствами объекта. Так, упомянутые выше методы **Hide()** и **Show()** изменяют значение свойства **Visible**. Таким образом, временно скрыть окно можно двумя способами: вызвать метод **Hide()** или присвоить свойству **Visible** значение **false**.

Получить список методов конкретного класса можно с помощью интерактивного справочника, обратившись к статье, посвященной интересующему нас классу (левый кадр на рис. 5.2) и перейдя в ней на вкладку **Methods** (правый кадр того же рисунка).

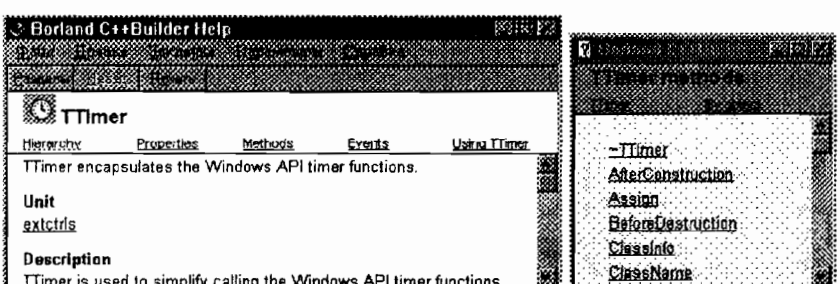

Рис. 5.2. Статья интерактивного справочника, посвященная классу TTimer(слева), и вкладка списка методов (функций) того же класса

События в C++ Builder эквивалентны сообщениям Windows. При включении в приложение того или иного объекта для него определяется список возможных событий в соответствии с набором сообщений, формируемых системой для данного объекта. Перечень

событий можно увидеть в окне инспектора объектов, где для событий предусмотрена отдельная вкладка (рис. 5.3). Для некоторых объектов может быть предусмотрено всего одно событие; для других число событий может составлять несколько десятков. Так, таймер может генерировать единственное сообщение **WM_TIMER**, которому отвечает событие **OnTimer**; в окно приложения, напротив, может поступать свыше 30 сообщений, каждому из которых сопоставлено соответствующее событие (**WM_CREATE** – **OnCreate**, **WM_PAINT** – **OnPaint**, **WM_MOUSEMOVE** – **OnMouseMove** и т. д.).

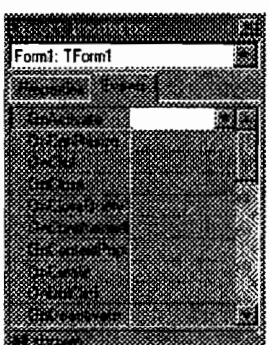

Рис. 5.3. Список событий для класса TForm

Для того чтобы добавить в приложение обработчик конкретного события (функцию обработки сообщения), достаточно в окне инспектора объектов дважды щелкнуть мышью по пустому полю справа от имени события. Это заставит среду включить в программный модуль заготовку для соответствующей функции. Текст этой функции программист, разумеется, должен написать сам.

Работа со свойствами объектов

Свойства объектов можно подразделить на те, которые отражают непосредственные характеристики объекта, и те, которые являются указателями на другие объекты. Например, свойство **Width** класса **TForm** определяет ширину окна. Сократить ширину окна в два раза можно таким образом:

```
Width = Width/2;
```

В правой части этого предложения указание имени свойства **Width** позволяет *получить* текущее значение ширины окна; указание того же имени в левой части позволяет, наоборот, *задать* ширине окна новое значение. Таким образом, свойство **Width** доступно как для чтения, так и для записи. Некоторые свойства доступны только для чтения. Так, свойство **ClientRect** позволяет только определить, но не задать координаты рабочей области:

```
TRect r = ClientRect;//В структуре r координаты рабочей области
r.bottom /=2;//Сократили высоту в r в два раза
ClientRect=r;//Недопустимо! Будет ошибка компиляции
```

Свойства другого рода следует рассматривать как указатели на те или иные объекты. Например, в классе **TForm** имеется свойство **Font**, которое фактически является указателем на объект класса **TFont**, в свою очередь содержащий в себе ряд свойств – характеристик используемого шрифта: **Name**, **Color**, **Size** и др. Обращение к этим свойствам осуществляется посредством указателя **Font**:

```
Font->Name="Times New Roman";//Назначается гарнитура шрифта
Font->Color=clBlue;//Назначается цвет шрифта
Font->Size=24;//Назначается размер шрифта
```

В этом примере **clBlue** – предопределенная константа, соответствующая светло-синему цвету.

Часто для обращения к конкретной характеристике объекта приходится использовать цепочку указателей, иногда довольно длинную. Так, среди свойств класса окна **TForm** имеется свойство **Canvas** (холст) – контекст окна. Объект **Canvas**, в свою очередь, содержит свойства **Pen** и **Brush**, которые также являются указателями на соответствующие объекты. В объект **Pen** входят свойства **Width** и **Color**, которые уже являются обычными данными. Чтобы создать, скажем, красное перо толщиной 4 пикселя, необходимо включить в программу следующие предложения:

```
Canvas->Pen->Color=clRed;
Canvas->Pen->Width=4;
```

Еще сложнее задать характеристики пера в объекте класса **TImage**, служащего для загрузки растрового изображения:

```
myImage->Picture->Bitmap->Canvas->Pen->Color=clGreen;
```

В классе **TImage**, которому принадлежит созданный в программе объект **myImage**, имеется свойство **Picture**, а в нем – свойство **Bitmap**, куда, собственно, и попадает загружаемое изображение. Для рисования поверх этого изображения следует обратиться к его холсту – свойству **Canvas**, в котором хранятся (так же в виде свойств) текущие инструменты рисования, в частности, перо **Pen** со свойствами **Color**, **Width** и др. Все перечисленные свойства, кроме последних, являются указателями на объекты вложенных классов, и для обращения к свойству **Color** необходимо перечислить все вышележащие указатели.

Работа с методами объектов

Вызов методов, входящих в классы C++ Builder, осуществляется в общем так же, как и обращение к функциям любых классов в объектно-ориентированной программе. В тех случаях, когда некоторый метод вызывается в функции того же класса, его имя указывается без дополнительной квалификации. Если же вызываемый метод принадлежит другому классу, необходимо указывать объект, в который входит метод. Чаще всего таким объектом является свойство-указатель, и если это свойство входит в цепочку вложенных классов, приходится указывать все вышележащие указатели. Рассмотрим несколько фрагментарных примеров.

Пусть в окне приложения (т. е. в объекте класса **TForm**) имеется меню, и при выборе пункта **"Выход"** мы хотим закрыть приложение. Для этого надо вызвать метод **Close()**, принадлежащий классу **TForm**. Поскольку функция отклика на выбор пункта меню (пусть она имеет имя **N1Click**) входит в тот же класс **TForm**, вызов метода **Close()** осуществляется непосредственно:

```
void __fastcall MyForm::TForm(TObject *Sender){
  Close();
  }
```

Заготовку функции отклика предоставляет среда C++ Builder, поэтому в тонкости определения этой функции (тип функции, тип возвращаемого значения, тип и назначение параметра) в данном случае можно не вдаваться.

Рассмотрим теперь вывод в окно приложения (объект класса **TForm**) каких-либо графических элементов, в частности, строки

текста. Для того чтобы содержимое окна не терялось при его свертывании, все рисование необходимо осуществлять в функции отклика на сообщение **WM_PAINT**. Пусть эта функция называется **FormPaint**. Для вывода в окно текста надо вызвать метод **TextOut**, который входит в класс **TCanvas**, олицетворяющий собой контекст окна. Класс **TCanvas** представлен в классе **TForm** свойством **Canvas** – указателем на контекст. Поэтому метод **TextOut** (как и любой другой метод, связанный с рисованием в окне) следует вызывать посредством этого указателя:

```
void __fastcall MyForm::FormPaint(TObject *Sender){
  Canvas->TextOut(10,25,"График функции");
  }
```

Свойство **Canvas** принадлежит объекту нашего окна, и в тот же объект входит функция **FormPaint()**. Как известно, в любую функцию класса при ее вызове неявно поступает указатель **this** на текущий объект. Таким образом, полная квалификация метода **TextOut()** выглядит следующим образом:

```
this->Canvas->TextOut(10,25,"График функции");
```

Указатель **this** на *текущий* объект можно опустить, однако в тех случаях, когда используется метод или свойство из *другого* класса, необходимо явно указывать объект этого класса.

Пусть в той же функции **FormPaint()** мы хотим вывести текст в совместимую память и затем скопировать эту память в окно приложения. Будем считать, что в программе имеется указатель **myBitmap** на созданный ранее объект совместимой памяти класса **TBitmap**. Функция **FormPaint()** будет выглядеть так:

```
void __fastcall TMyForm::FormPaint(TObject *Sender){
  myBitmap->Canvas->TextOut(10,25,"График функции");
  Canvas->Draw(0,0,myBitmap);
  }
```

В первом предложении мы вызываем метод **TextOut()** для холста совместимой памяти, в который входит свое свойство **Canvas**. Во втором предложении метод копирования **Draw()** вызывается уже для холста окна, и имя **Canvas** не предваряется указателем (неявно же используется указатель **this**).

Работа с событиями

Как уже отмечалось, среда C++ Builder предоставляет заготовку функции отклика на любое выбранное пользователем событие. Для этого достаточно дважды щелкнуть мышью по пустому полю справа от имени события в инспекторе объектов для конкретного компонента. Прикладная обработка событий в C++ Builder в принципе не отличается от обработки сообщений при традиционном программировании или программировании с помощью библиотеки OWL.

Модификация файлов проекта

Как уже отмечалось, при активизации среды Boland C++ Builder автоматически создается комплект файлов нового приложения, из которых программиста интересуют главным образом два: заголовочный файл с именем по умолчанию UNIT1.H и программный файл (по умолчанию UNIT1.CPP), содержащий заготовки функций, содержательный текст которых определяется программистом.

Заголовочный файл содержит определение прикладного класса, производного от библиотечного класса **TForm**, описывающего главное окно приложения. Имя этого класса определятся именем формы. По умолчанию при создании первого проекта в данном сеансе работы со средой среда дает форме имя **Form1**, а прикладному классу окна – **TForm1**; если свойству **Name** формы главного окна присвоить, например, значение **MyForm**, прикладной класс автоматически получит имя **TMyForm**. Обратите внимание на то, что свойство **Name** определяет программное имя создаваемого средой прикладного класса; свойство **Caption** определяет заголовок окна и имеет чисто декоративный смысл.

Новый класс (в файле UNIT1.H) поначалу будет содержать только описатели доступа и конструктор **TMyForm**:

```
class TMyForm : public TForm{
__published://Компоненты, управляемые средой
  void __fastcall FormPaint(TObject *Sender);
private: //Объявления пользователя
public: //Объявления пользователя
  __fastcall TMyForm(TComponent* Owner);
};
```

В программный файл UNIT1.CPP, помимо нескольких директив препроцессора, будут включены только указатель на новый класс окна и конструктор этого класса (пустой):

```
TMyForm *MyForm;
__fastcall TMyForm::TMyForm(TComponent* Owner)
        : TForm(Owner) {}
```

По мере того как программист с помощью инспектора объектов "заказывает" обработку тех или иных событий, среда включает в определение класса **TMyForm** прототипы соответствующих функций отклика, а в программный файл – заготовки этих функций. При переносе на форму каких-либо компонентов (меток, диалогов, таймеров, меню и т. д.) среда дополняет секции данных прикладного класса указателями на создаваемые объекты:

```
class TMyForm : public TForm{
__published:// Компоненты, управляемые средой
  TTimer *Timer1;//Программист добавил компонент-таймер
  TLabel *Label1;//Программист добавил компонент-метку
  void __fastcall FormCreate(TObject *Sender);//Заказали
                                    //обрабатку события OnCreate
  void __fastcall FormPaint(TObject *Sender);//Заказали
                                    //Обработку события OnPaint
private://Объявления пользователя
public://Объявления пользователя
  __fastcall TMyForm(TComponent* Owner);//Прототип конструктора
  };
```

Соответственно в программном файле появятся заготовки (пустые) функций откликов:

```
TMyForm *MyForm;
__fastcall TMyForm::TMyForm(TComponent* Owner)
        : TForm(Owner){}//Конструктор
void __fastcall TMyForm::FormCreate(TObject *Sender){}
void __fastcall TMyForm::FormPaint(TObject *Sender){}
```

Обычно по мере разработки приложения программист сталкивается с необходимостью объявления прикладных переменных разного рода: массивов данных, указателей на блоки совместимой памяти и пр. В этом случае новые переменные включаются в состав членов прикладного класса не средой, а программистом.

Вывод в окно графических элементов

Вывод в окно приложения текста, геометрических фигур и других графических элементов осуществляется с помощью методов класса **TCanvas** (холст), инкапсулирующего в себе контекст устройства. Указатель на холст в виде свойства **Canvas** входит в класс **TForm** окна, предоставляя доступ ко всем графическим методам холста, которые практически не отличаются от соответствующих графических функций API Windows.

Собственно вывод в окно осуществляется, как правило, в функции **FormPaint()** отклика на событие **OnPaint**; изменение характеристик инструментов рисования (размера и гарнитуры шрифта, цвета и толщины пера и пр.) можно выполнить в той же функции **FormPaint()**, особенно если по ходу рисования эти характеристики требуется изменять. В тех случаях, когда все рисование будет осуществляться одним набором инструментов, их характеристики можно задать в функции отклика на событие **OnCreate** или даже в конструкторе окна. Характеристики шрифта для текста, выводимого в окно, можно также задать "вручную" в окне инспектора объектов для объекта формы.

Приведем пример вывода в окно приложения нескольких графических элементов. В этом примере свойству **Name** формы главного окна было присвоено, как и ранее, значение **MyForm**; в класс **TMyForm** была включена функция отклика на событие **OnPaint**, которая по умолчанию получила имя **FormPaint**.

```
void __fastcall TMyForm::FormPaint(TObject *Sender){
  Canvas->Font->Name="Times New Roman";//Задаем имя шрифта
  Canvas->Font->Size=16;//Задаем размер шрифта
  Canvas->Font->Color=clYellow;//Задаем цвет шрифта
  Canvas->Brush->Style=bsClear;//Фон под шрифтом прозрачный
  Canvas->TextOut(10,10,"Фигура: Ellipse");//Вывод текста
  Canvas->Pen->Style=psClear;//Прозрачное перо
  Canvas->Brush->Color=(TColor)0x80A0;//Коричневая кисть
  Canvas->Ellipse(10,45,150,100);//Рисуем эллипс
  }
```

В предпоследнем предложении фрагмента коричневый цвет кисти (отсутствующий среди стандартных цветов) задан с помощью шестнадцатеричного числа, преобразованного в тип **TColor**.

С таким же успехом можно воспользоваться макросом **RGB**:

```
Canvas->Brush->Color=RGB(0xA0,0x80,0);
```

Обратите внимание на различающийся порядок перечисления компонентов цвета в макросе RGB и при представлении цвета в виде одного целого числа.

Результат выполнения приведенного фрагмента показан на рис. 5.4. В проекте приложения с помощью инспектора объектов были заданы свойства формы **Caption**, **Height** и **Width**, которые получили значения **Пример**, **135** и **175**. Цвет формы (т. е. цвет главного окна) оставлен серым по умолчанию.

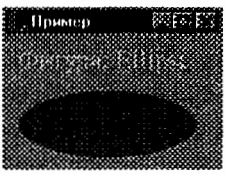

Рис. 5.4. Вывод в окно приложения графических элементов

Из приведенного примера видно, что графические методы класса **TCanvas** практически не отличаются от соответствующих функций GDI Windows. В некоторых случаях, однако, отличие оказывается существенным. Это относится, в частности, к свойству **Pixels** рисования точками:

```
for(float t=0;t<2*M_PI;t+=0.01){
   int x=sin(t)*100;
   int y=-cos(t)*100;
   Canvas->Pixels[x+120][y+120]=clBlue;
     }
```

Приведенный фрагмент выведет в окно приложения фигуру Лиссажу, показанную на рис. 2.1

Мы видим, что свойство **Pixels** представляется именем двумерного массива, индексы которого определяют *x*- и *y*-координаты отображаемого пикселя, а значение каждого элемента массива задает его цвет.

Цвет конкретного пикселя можно не только задать (записать), но и определить (прочитать). Установить этот факт можно, обратившись к статье “Pixels” интерактивного справочника, где это свойство определяется следующим образом:

__**property** TColor Pixels[**int** X][**int** Y]={**read**=GetPixel, **write**=SetPixel};

Видно, что для свойства **Pixels** определены обе операции: как чтение (**read**), так и запись (**write**). Программное предложение

```
TColor col=Canvas->Pixels[100][100];
```

поместит в переменную **col** (которая, естественно, должна иметь тип **TColor**) код, соответствующий цвету пикселя, расположенного в точке с координатами 100,100.

Работа с совместимой памятью

Как известно, совместимая память может использоваться в двух основных вариантах: для загрузки в нее готового растрового изображения из файла (или из выполнимого модуля приложения при использовании на этапе разработки программы файла ресурсов) и для формирования в пустой памяти требуемого изображения (например, графика) программным образом, с помощью графических функций, в нашем случае – с помощью методов класса **TCanvas**. Рассмотрим сначала первый вариант использования совместимой памяти.

Самый простой способ отображения рисунка, хранящегося в файле с расширением .BMP, заключается в перетаскивании на форму компонента **Image** (значок , находящийся на вкладке **Additional** палитры компонентов). При этом в заголовочном файле появляется указатель на объект класса **TImage**, по умолчанию принимающий имя **Image1**:

```
TImage* Image1;
```

В классе **TImage** имеется свойство **Picture** – указатель на объект класса **TPicture**. В этот объект можно загрузить растр из файла, для чего в классе **TPicture** имеется метод **LoadFromFile()**:

```
Image1->Picture->LoadFromFile("1234.bmp");
```

Фактически загружаемый растр попадает при этом в свойство **Bitmap** – объект класса **TBitmap**, входящий в класс **TPicture** и представляющий собой собственно совместимую память (точнее, указатель на объект совместимой памяти). Среди свойств класса **TBitmap** имеется свой холст – объект **Canvas** класса **TCanvas**,

а в нем, естественно, инструменты рисования – шрифт, перо и кисть. Взаимоотношение всех этих объектов (свойств вышележащих классов) схематически показано на рис. 5.5.

Предложение загрузки растра из файла (вызовом метода **LoadFromFile()**) можно включить в функцию отклика на событие **OnCreate** или в конструктор класса окна. Больше ничего делать не надо; при запуске приложения рисунок появится в том месте окна, куда был перетащен компонент **Image**. Надо только проследить за тем, чтобы размер этого компонента, отображенный в свойствах **Height** и **Width**, был не меньше размеров рисунка.

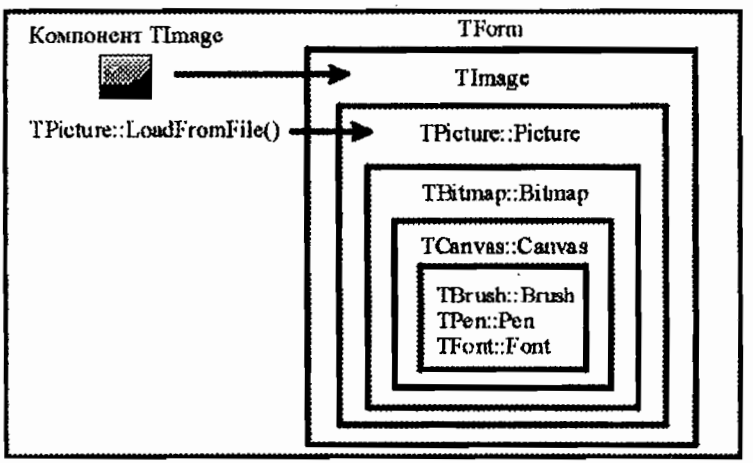

Рис. 5.5. Цепочка объектов, вложенных в компонент класса TImage

Подгонку размеров компонента **Image** под размер рисунка можно выполнить несколькими способами. Проще всего с помощью инспектора объектов присвоить свойству **AutoSize** объекта **Image1** значение **true**. Тогда независимо от установленных размеров компонента, рисунок будет отображаться в окне полностью.

Другой способ заключается в программном определении размеров загруженного рисунка и программном же задании размеров объекта **Image1**:

```
Image1->Width = Image1->Picture->Width;
Image1->Height = Image1->Picture->Height;
```

В этом фрагменте переменной **Image1->Width**, т. е. свойству **Width** объекта **Image1**, присваивается значение переменной **Image1->Picture->Width**, обозначающей такое же по имени свойство, но другого объекта – вложенного в **Image1** свойства **Picture**, отображающего реальный размер растра. Любопытно отметить, что в отличие от свойства **TImage->Width**, которое допускает как запись, так и чтение, свойство **TPicture->Width** доступно только для чтения; задать ему значение нельзя.

Наличие в классе **TBitmap** свойства **Canvas** (см. рис. 5.5) дает возможность выводить с помощью методов класса **TCanvas** поверх загруженного изображения любые графические объекты – строки текста, рамки и пр.

Рассмотрим теперь другой подход к использованию совместимой памяти – создание пустого блока памяти с последующим заполнением его требуемыми графическими объектами.

Поскольку в этом случае совместимую память мы будем создавать "своими руками", в состав класса окна в файле .H необходимо включить указатель на объект памяти класса **TBitmap**:

```
Graphics::TBitmap* bmp;//Указатель на совместимую память
```

Оператор разрешения видимости :: здесь нужен для того, чтобы определить принадлежность класса **TBitmap** модулю **Graphics**.

Где-либо в программе, например, в функции отклика на событие **OnCreate**, следует создать объект памяти, задать его размер и после этого помещать в него требуемые графические элементы:

```
bmp=new Graphics::TBitmap;//Выделяем память под объект
bmp->Height=256;//С помощью свойств Height и Width
bmp->Width=256;//класса TBitmap задаем размеры растра
bmp->Canvas->Brush->Color=clYellow;//Кисть для заливки памяти
bmp->Canvas->FillRect(TRect(0,0,256,256));//Закрашиваем
                                          //память, иначе там будет мусор
bmp->Canvas->Brush->Style=bsClear;//Прозрачный фон под буквами
bmp->Canvas->TextOut(0,15,"График");//Выводим текст в память
for (int i=0;i<400;i++){//В цикле выводим график функции
  int y=f(i)//Вычисляем значение отображаемой функции f()
  bmp->Canvas->Pixels[i][-y+200]=clBlue;//Выводим точку
  }
```

По завершению цикла в памяти будет находиться текстовая строка и график из синих точек, однако окно останется пустым.

Чтобы изображение появилось в окне, его надо туда скопировать из памяти (как правило, в функции **FormPaint()**):

```
Canvas->Draw(5,5,bmp);//Копирование памяти в окно с точки (5,5)
```

Первые два параметра метода **Draw()** позволяют задать координаты окна, начиная с которых в окне отображается содержимое совместимой памяти (объекта класса **TBitmap**).

При таком методе создания совместимой памяти цепочка интересующих нас объектов, связанных с рисованием, упрощается (рис. 5.6).

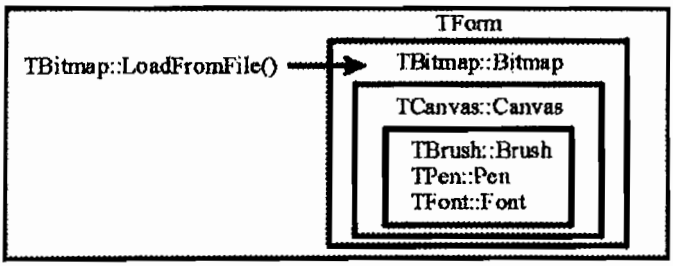

Рис. 5.6. Цепочка объектов, вложенных в объект совместимой памяти

Поскольку в классе **TBitmap** имеется "свой" метод **LoadFromFile()** (унаследованный от базового класса **TGraphic**), и при таком способе использования совместимой памяти имеется возможность загрузить в нее рисунок из файла:

```
bmp->LoadFromFile("abcd.bmp");
```

Ясно, что в этом случае для загрузки растра используется метод **TBitmap::LoadFromFile** (ср. с рис. 5.5).

Копирование изображений

При обработке растровых изображений возникает необходимость копирования изображений из одного блока совместимой памяти в другой. Например, загрузив с диска исходное изображение в объект **TImage**, мы можем скопировать это изображение в блок совместимой памяти и, применяя к этой копии различные алгоритмы обработки, сравнивать исходное (неизменное) изображение с результатами текущей обработки. Казалось бы, для этого достаточно выполнить программный фрагмент такого рода:

```
Image1->Picture->LoadFromFile("1.bmp");//Загружаем из файла
Graphics::TBitmap* bmp;//Указатель на совместимую память
bmp=new Graphics::TBitmap;//Выделяем память под объект
bmp= Image1->Picture->Bitmap;//Копируем из Bitmap в bmp
```

Приведенный фрагмент будет работать, однако в нем осуществляется копирование не самой памяти, а лишь указателей на нее (так называемое *мелкое копирование*). В результате указатели **Bitmap** и **bmp** будут указывать на один и тот же блок памяти (с загруженным с диска изображением), и при обработке изображения в памяти **bmp** оно будет точно так же изменяться при обращении к нему через указатель **Bitmap**.

Для того чтобы осуществить так называемое *глубокое копирование*, т. е. скопировать не указатели, а саму память, следует обратиться к потоковым операциям. В класс **TBitmap** (как и в ряд других классов среды Builder), включены методы записи и чтения памяти через специально предусмотренную разновидность потоков – потоки памяти **TMemoryStream**. Для копирования изображения следует создать объект этого класса и сначала записать в него методом **SaveToStream()** копируемый блок памяти, а затем методом **LoadFromStream()** прочитать содержимое этого потока в другой блок совместимой памяти:

```
/* Объявляем указатель и создаем объект потока */
TMemoryStream* strm=new TMemoryStream();
/* Выводим в поток изображение, загруженное в объект Image1 */
Image1->Picture->Bitmap->SaveToStream(strm);
/* Сбрасываем в начало текущую позицию в потоке */
strm->Position=0;
/* Вводим изображение из потока в объект bmp */
bmp->LoadFromStream(strm);
```

Теперь оба блока памяти содержат одно и то же изображение, но, в то же время, допускают его независимую обработку.

Обработка цветных цифровых изображений

В зависимости от количества используемых цветов и метода их кодирования, форматы растровых изображений могут различаться. В настоящем подразделе описывается принцип обработки изображений, записанных в широко распространенном 24-битовом формате RGB (red, green, blue – красный, зеленый, синий).

Цветное изображение в формате RGB хранится в виде последовательности из троек байтов, причем в младшем байте каждой тройки (с индексом [0]) записывается интенсивность синего компонента цвета, в байте с индексом [1] – интенсивность зеленого компонента, а в байте с индексом [2] – красного (рис. 5.7).

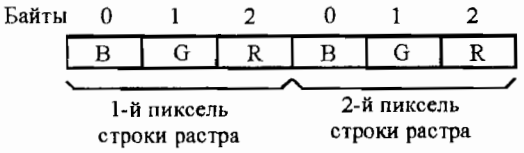

Рис. 5.7. Размещение цветовых компонентов пикселей в 24-битовом растровом изображении

Таким образом, в каждой строке прямоугольного растра число байтов в три раза больше ширины изображения в пикселях, а число таких строк равно высоте растра в пикселях (рис. 5.8).

Рис. 5.8. Схема памяти с цветным изображением размером 100 × 50 пикселей

В системе Borland C++ Builder для доступа к отдельным пикселям изображения предусмотрено свойство **ScanLine[]**, входящее в класс **TBitmap**. Это свойство возвращает указатель на строку растра, номер которой указывается в качестве аргумента при обращении к **ScanLine[]**:

```
Byte* ptr=(Byte*)bmp->ScanLine[j];
```

Здесь **bmp** – указатель на совместимую память с цветным изображением, полученный тем или иным из описанных выше спосо-

бов. Поскольку свойство **ScanLine[]** возвращает обобщенный указатель типа **void***, а реально этот указатель содержит адрес байта, его необходимо привести к типу **Byte***. (Обратите внимание на необычное обозначение байтовой переменной. В языке C++ такие переменные имеют тип **unsigned char**, в системе Windows определен тип **BYTE**, а в Borland C++ Builder используются обозначения типов переменных из языка Delphi: **Byte**, **Word**, **Integer**, **Boolean**, **Real** (вместо **double**) и т. д.)

Индексируя далее указатель **ptr**, можно получить доступ к любому байту строки. Для обработки всего изображения следует цикл просмотра байтов строки поместить в цикл просмотра строк. Например, в приведенном ниже фрагменте отдельно анализируется значение каждого цветового компонента пикселя, и, если это значение превышает середину диапазона яркости (число 128), интенсивность компонента увеличивается до максимума; в противном случае она снижается до 0. В результате изображение преобразуется в 8-цветное (три чистых основных цвета максимальной интенсивности и их сочетания).

```
for(int j=0;j<bmp->Height;j++){//Цикл по строкам
  Byte* ptr=(Byte*)bmp->ScanLine[j];
  for( int i=0;i<bmp->Width*3;i+=3){//Цикл по пикселям
                                    //(тройкам байтов строки пикселей)
    if(ptr[i]>128) ptr[i]=255;//Анализируем синий компонент
      else ptr[i]=0;
    if(ptr[i+1])>128 ptr[i+1]=255; //Анализируем зеленый
      else ptr[i+1]=0;
    if(ptr[i+2]>128) ptr[i+2]=255; //Анализируем красный
      else ptr[i+2]=0;
    }//Конец цикла по тройкам байтов строки
  }//Конец цикла по строкам
```

Внешний цикл выполняется по всем строкам изображения от 0 до **bitmap->Height**, а во внутреннем цикле обрабатываются тройки байтов (приращение осуществляется через три байта). Заметьте, что длина строки в пикселях равна **bitmap->Width**, но в числе байтов – **bitmap->Width*3**.

Стандартные модальные диалоги

На вкладке **Dialogs** панели объектов предусмотрен ряд значков для включения в программу стандартных модальных диалогов Windows – для выбора открываемых и сохраняемых файлов, вывода файлов на печать, поиска и замены элементов текста и др. Рассмотрим особенности создания и использования стандартных диалогов для работы с цифровыми изображениями.

При перетаскивании на форму компонента **OpenPictureDialog** (значок , находящийся на вкладке **Dialogs** палитры компонентов) в заголовочном файле появляется указатель на объект **TOpenPictureDialog** с именем по умолчанию **OpenPictureDialog1**:

```
TOpenPictureDialog *OpenPictureDialog1;
```

В классе **TOpenPictureDialog** предусмотрен метод **Execute()** для создания интерфейсного элемента диалога, а также свойство **FileName**, в которое поступает имя выбранного файла. Метод **Execute()** возвращает **true**, если пользователь выбрал существующий файл, и **false** в случае выбора клавиши **Cancel**.

Обычно изображение, хранящееся в выбранном файле, должно быть отображено в окне приложения, для чего на форме должен иметься компонент класса **TImage**, а в заголовочном файле – указатель на объект этого класса:

```
TImage* Image1;
```

Тогда для выбора файла с отображением его в объекте **Image1** достаточно выполнить следующий фрагмент:

```
if(OpenPictureDialog1->Execute()) //Если файл выбран
  Image1->Picture->LoadFromFile(//Загрузить файл с именем,
    OpenPictureDialog1->FileName);//поступившим в FileName
```

Если метод **Execute()** возвращает **true**, для объекта **Picture**, входящего в компонент **Image1**, вызывается имеющийся в **Picture** метод загрузки файла **LoadFromFile()** (см. рис. 5.5), в качестве аргумента которого указывается свойство **FileName** из объекта **OpenPictureDialog1**, в которое, как уже отмечалось, поступает имя выбранного пользователем файла.

Приведенный выше фрагмент наиболее естественно включить в функцию отклика на выбор соответствующего пункта меню.

Столь же просто работать с диалогом **SavePictureDialog**, предназначенным для указания имени файла, в который будет записан объект **TImage**. При перетаскивании на форму компонента **SavePictureDialog** (значок) в заголовочном файле появляется указатель на объект **TSavePictureDialog**:

```
TSavePictureDialog *SavePictureDialog1;
```

В классе **TSavePictureDialog**, как и в классе **TOpenPictureDialog**, предусмотрен метод **Execute()** для активизации диалога, свойство **FileName**, в которое поступает имя заданного пользователем файла, а также свойство **DefaultExt**, позволяющее задать расширение, которое будет присвоено имени файла, если пользователь не задал расширение явным образом. В классе же **TPicture**, кроме метода **LoadFromFile()**, загружающего указанный файл в объект изображения, имеется и метод **SaveToFile()**, позволяющий записать изображение в указанный файл. Таким образом, фрагмент программы, сохраняющий в выбранном пользователем файле изображение из объекта **Image1**, будет выглядеть следующим образом:

```
SavePictureDialog1->DefaultExt="BMP";//Расширение по умолчанию
if(SavePictureDialog1->Execute())//Если имя файла указано
  Image1->Picture->SaveToFile(      //Сохранить файл с именем,
    SavePictureDialog1->FileName);//поступившим в FileName
```

Разумеется, для сохранения изображения в файле совсем не обязательно использовать диалог **TSavePictureDialog**. Если имя файла заранее определено, то для вывода в файл изображения, хранящегося в объекте **TImage**, достаточно вызвать метод **SaveToFile()** для свойства **Picture** объекта класса **TImage** с указанием имени записываемого файла:

```
Image1->Picture->SaveToFile("temp.bmp");
```

Метод **SaveToFile()** имеется и в свойстве **Picture** совместимой памяти класса **TBitmap** (см. рис. 5.6). Таким образом, если мы работаем не с компонентом **TImage**, а с блоком совместимой памяти, созданным программным образом:

```
Graphics::TBitmap* bmp = new Graphics::TBitmap;
```

то для записи этого блока памяти в файл достаточно строки

```
bmp->SaveToFile("myfile.bmp");
```

Часть 2
Лабораторный практикум

Работы лабораторного практикума

Программирование с помощью библиотеки OWL

Работа №1. Приложение Windows с выводом в главное окно текста и фигур (индивидуальное задание А)

На основе примера 2-1 из настоящего пособия разработайте программу в соответствии с индивидуальным заданием. Цвет окна задается в конструкторе главного окна с помощью функции **`SetBkgndColor()`**. Создать необходимые инструменты (перья, кисти и шрифты) можно в функции **`Paint()`**. Объекты перьев, кистей и шрифтов создаются с помощью конструкторов классов **`TPen`**, **`TBrush`** и **`TFont`**; цвет задается с помощью объектов класса **`TColor`**; выбор инструментов в контекст окна осуществляется с помощью функций **`SelectObject()`** и **`SelectStockObject()`** из класса **`TDC`**; в том же классе можно найти многочисленные функции, выполняющие настройку контекста (**`SetTextColor()`**, **`SetBkMode()`** и др.) и рисование в окне (**`TextOut()`**, **`Rectangle()`** и др.).

Работа №2. Вывод в главное окно графика математической функции (индивидуальное задание В)

В середине главного окна достаточно большого размера нарисуйте оси координат и выведите вокруг этих осей график указанной в задании математической функции. Абсциссы и ординаты точек графика вычисляйте в цикле, определив число шагов в соответствии с заданием. Каждую точку графика рисуйте небольшим цветным кружочком размером в 3-4 пикселя. Не забудьте сдвинуть график так, чтобы он правильно изображался относительно начала координат, и перевернуть его относительно горизонтальной оси. Вычисление координат выводимых точек можно выполнить в функции **`Paint()`** главного окна.

Работа №3. Меню и стандартный диалог "Открытие файла" (индивидуальное задание С)

С помощью программы Блокнот создайте файл данных с произвольным именем и запишите в него информацию (в символьной форме) в соответствии с индивидуальным заданием.

Составьте программу с меню "**Файл**", состоящим из двух пунктов "**Открыть**" и "**Выход**". При выборе пункта "**Открыть**" на экран выводится стандартный диалог Windows "**Открытие файла**"; при выборе пункта "**Выход**" приложение завершается.

В файле ресурсов опишите меню из двух пунктов (с разделительной линией между ними), присвоив пункту "**Открыть**" произвольный символический идентификатор, а пункту "**Выход**" – стандартный идентификатор **CM_EXIT**.

В таблицу откликов класса главного окна включите макрос **EV_COMMAND** с указанием идентификатора первого пункта меню и имени закрепленной за ним прикладной функции отклика. Прототип этой функции включите в качестве члена в класс главного окна (производного от класса **TFrameWindow**); саму эту функцию опишите в любом месте программы.

В функции отклика на выбор пункта "**Открыть**" подготовьте все данные, необходимые для работы стандартного диалога Windows "**Открытие файла**", откройте этот диалог и после выбора вашего файла данных выполните следующие действия:

- откройте выбранный файл;
- прочитайте из файла все его содержимое;
- с помощью функции **sscanf()** перенесите данные, прочитанные из файла, в отдельные переменные соответствующих типов;
- выполните указанные в задании действия над данными;
- с помощью функции C++ **wsprintf()** (или, при необходимости, **sprintf()**) преобразуйте эти данные в символьную форму, а с помощью функции OWL **TextOut()** выведите в окно приложения информацию в соответствии с заданием;
- закройте файл.

Для получения контекста рабочей области окна воспользуйтесь конструктором класса **TClientDC**; в качестве параметра этого конструктора (дескриптор главного окна типа **HWND**) можно использовать данное-член **HWindow** из базового класса **TWindow**.

Работа №4. Вывод растровых изображений

Создайте собственный цветной рисунок с помощью программы Windows Paint. Далее воспользуйтесь в качестве основы программой из предыдущей работы, где в ответ на выбор пункта меню выводился стандартный диалог Windows “Открытие файла”.

а. Разработайте приложение, позволяющее выводить в главное окно приложения растровое изображение из файла, выбранного с помощью этого диалога.

Включите в класс **MyWin** три указателя:

- указатель **pBmp** на класс растровых изображений **TBitmap**;
- указатель **pMemDC** на класс контекстов памяти **TMemoryDC**;
- указатель **pDib** на класс независимых от устройства растров **TDib**.

В конструкторе главного окна обнулите эти указатели.

В функции отклика на выбор пункта **"Открыть"** подготовьте все данные, необходимые для работы стандартного диалога Windows **"Открытие файла"**, откройте этот диалог и после выбора файла с рисунком выполните следующие действия:

- создайте объект независимого от устройства растра (указатель **pDib**), загрузив в него выбранный файл;
- вызовом конструктора **TClientDC** получите контекст **dc** главного окна;
- Вызовом конструктора **TBitmap** создайте объект в зависимом формате (указатель **pBmp**), переместив в него рисунок из объекта **pDib**;
- создайте объект контекста памяти (указатель **pMemDC**), совместимой с контекстом **dc** главного окна;
- выберите совместимую память в совместимый контекст;
- вызовом функции **Invalidate()** инициируйте перерисовку главного окна.

В функции **Paint()** главного окна вызовом функции **BitBlt()** скопируйте контекст совместимой памяти в контекст главного окна. Копирование осуществляйте, только если объект совместимой памяти существует (указатель **pBmp** не равен нулю).

Функция **BitBlt()** требует указания размеров растрового изображения. Для определения этих размеров в классе **TBitmap** предусмотрены удобные функции **Width()** и **Height()**.

b. Модифицируйте предыдущий вариант программы, заключив рисунок в совместимой памяти в толстую цветную рамку и поместив в какой-либо угол рисунка свою фамилию в качестве подписи. Выведите также в главное окно приложения под рисунком имя файла, выбираемого с помощью диалога.

Работа №5. Обработка сообщений от мыши (индивидуальное задание D)

Дополните предыдущую программу обработкой сообщений от мыши и анализом цветов отдельных пикселей изображения в соответствии с индивидуальным заданием.

a. Добавьте в число данных-членов класса **`MyWin`** следующие переменные:

- флаг, сигнализирующий о щелчке мышью (булева переменная);
- переменную **`pixel`** типа **`TColor`** для помещения в нее цвета выбранного мышью пикселя.

Включите в таблицу откликов главного окна приложения один из макросов обработки сообщений от мыши (в соответствии с индивидуальным заданием). Соответствующую этому макросу функцию отклика включите в состав функций класса главного окна.

В функции отклика на щелчок мышью выполните следующие действия:

- проанализируйте (если указано в индивидуальном задании) состояние клавиш **`Shift`** или **`Ctrl`**;
- установите флаг щелчка;
- вызовом конструктора **`TClientDC`** получите контекст **`dc`** главного окна;
- вызовом функции **`GetPixel()`** получите в переменную **`pixel`** цвет пикселя в совместимой памяти, на который указывает курсор мыши;
- вызовом функции **`Invalidate()`** инициируйте перерисовку главного окна.

В функции **`Paint()`** главного окна, помимо копирования совместимой памяти в окно, предусмотрите вывод в окно (за пределами растрового изображения) текстовой или графической информации согласно индивидуальному заданию. Значения компонентов

цвета пикселя можно получить с помощью функций **Red()**, **Green()** и **Blue()** класса **TColor**.

Работа №6. Модальный диалог в качестве главного окна приложения

Разработайте программу, в которой в качестве главного окна приложения выступает модальный диалог достаточно большого размера с элементом управления – списком. Закрасьте окно диалога каким-либо бледным цветом и выведите в него оси координат и график функции $f(t)$, нарисованный небольшими цветными кружочками в соответствии с индивидуальным заданием В (см. работу №2). Предусмотрите в окне диалога элемент управления – список, заполненный вычисленными значениями аргумента (t) заданной функции и самой функции (f) в виде двух колонок. Для графиков математических функций, задаваемых параметрически, в виде двух функций $x(t)$ и $y(t)$, выведите в строки списка три числа – аргумент t и значения x и y.

В программе, помимо всегда имеющегося класса приложения, объявляется еще лишь один класс диалога с именем, например, **MyDialog**, производный от **TDialog**. Предусмотрите следующий состав членов этого класса:

- указатель на интерфейсный объект класса **TListBox**;
- числовые массивы соответствующих типов для хранения вычисленных значений аргументов и функции;
- конструктор, явно вызывающий конструктор базового класса **TDialog** с передачей ему двух первых параметров;
- объявление таблицы откликов диалогового окна;
- объявления двух замещающих функций откликов: **EvCtlColor()** для перекрашивания окна диалога и **EvPaint()** для рисования в нем, а также замещающей функции **SetupWindow()** для вычисления координат графика и заполнения ими списка в диалоговом окне.

Объявляя массивы для хранения данных, выберите для них тип данных и длину, исходя из индивидуального задания. Для графиков математических функций, задаваемых в форме $f(t)$, потребуется два массива; для функций, задаваемых параметрически – три.

В конструкторе диалогового окна создайте объекта списка.

В таблицу откликов включите макросы **EV_WM_CTLCOLOR** и **EV_WM_PAINT**.

В функции **SetupWindow()** (после вызова исходной, замещенной функции) следует, во-первых, вычислить в цикле аргументы и значения указанной в задании функции с сохранением их в предусмотренных для этого массивах и, во-вторых, заполнить этими значениями список в диалоговом окне.

В функции **EvPaint()** следует прежде всего создать объект контекста устройства вызовом конструктора производного от **TDC** класса **TPaintDC**:

```
TPaintDC dc(*this);
```

Используя объект **dc**, следует выбрать в контекст устройства объекты пера и кисти и вывести в окно диалога требуемый график, получая данные для него из заполненных ранее массивов данных.

Выводя в цикле точки графика, не забудьте преобразовывать числа, хранящиеся в массивах данных, в координаты графика в виде номеров пикселей.

Модальный диалог, служащий в данной работе главным окном приложения, создается в функции **InitMainWindow()**.

Работа №7. Динамическое взаимодействие пользователя с элементами управления диалога

Модифицируйте предыдущую программу, дополнив ее возможностью выбора щелчком мыши любой пары координат из списка с отображением выбранной точки на графике в виде черного квадратика относительного большого размера (например, 5x5 пикселей).

Добавьте к стилям списка в файле ресурсов стиль **LBS_NOTIFY**; в этом случае при выборе строки списка в таблице откликов диалога будет активизироваться макрос **EV_CHILD_NOTIFY**.

Добавьте в состав класса **MyDialog** объявление функции отклика с произвольным именем для макроса **EV_CHILD_NOTIFY**, а в состав таблицы откликов включите макрос **EV_CHILD_NOTIFY**:

```
EV_CHILD_NOTIFY(ID_1,LBN_SELCHANGE,Choice);
```

Все действия по выбору строк списка и отображению выбранных точек на графике осуществляются в функции отклика **Choice()**, которая вызывается каждый раз, когда пользователь выбирает новую строку списка, и в приложение, соответственно, направляется сообщение **EV_CHILD_NOTIFY** с кодом оповещения **LBN_SELCHANGE**.

Для определения номера выбранной строки следует воспользоваться функцией **GetSelIndex()** из класса **TListBox**; получить из списка строку с заданным номером можно с помощью функции **GetString()** из того же класса.

Для преобразования символьной строки списка, содержащей два или три числа, в соответствующие числовые данные, следует воспользоваться функцией C++ **sscanf()**.

Для рисования на графике квадратика с полученными из списка координатами понадобится контекст устройства рабочей области диалогового окна. Его можно получить с помощью конструктора класса **TClientDC**, как это делалось в работе №4.

Программирование в среде Borland C++ Builder

Работа №1. Простейшее приложение с главным окном

1. Откройте интегрированную среду программирования Borland C++. На рабочем столе появится исходный кадр среды с открытым по умолчанию проектом с именем **Project1**.

2. Выбором пункта меню **File➢SaveProject As...** сохраните последовательно файлы, входящие в проект, под именами, например, **U1** (вместо **Unit1**) и **P1** (вместо **Project1**), проследив за тем, чтобы они попали в ваш рабочий каталог. Откройте программу Norton Commander и просмотрите состав вашего рабочего каталога. В нем должны появиться файлы:

P1.BPR – служебный файл проекта;

P1.CPP – фрагмент исходного текста программы с функцией WinMain();

P1.RES – файл ресурсов;

U1.CPP – фрагмент исходного текста программы с функциями обработки сообщений;

U1.DFM – файл определения формы;

U1.H – заголовочный файл.

Просмотрите с помощью программы Norton Commander образованные средой файлы и попытайтесь разобраться в их содержимом.

3. Откомпилируйте созданную автоматически программу, выбрав пункт меню **Project➢Build P1**.

4. Запустите программу на выполнение, щелкнув по значку на панели инструментов в верхней части кадра. На экран будет вы-

ведено главное окно приложения стандартного вида. Завершите программу, щелкнув по кнопке .

5. Закройте среду программирования Borland C++ Builder.

Работа №2. Работа с элементами формы

1. Откройте среду программирования Borland C++ Builder. Выбором пункта меню **File➢OpenProject** откройте созданный в предыдущей работе проект **P1.BPR**. По верхнему краю кадра располагаются главное меню, инструментальная панель (слева) и палитра компонентов (справа). Под этими элементами в правой части кадра находится изображение "формы" (заготовки для главного окна приложения), а в левой части – окно инспектора объектов (Object Inspector). Окно инспектора содержит две вкладки – свойств (Properties) и событий (Events) для отображаемой формы. Найдите эти элементы среды программирования.

2. Перед тем как приступать к модификации программы, сохраните ее компоненты в своем рабочем каталоге под именами **P2** и **U2**. Для этого придется воспользоваться двумя пунктами меню: **File➢SaveProject As...** и **File➢Save As...**

3. Найдя в окне инспектора свойство **Caption**, задайте для окна новый заголовок (например, "**Программа 2**").

4. При создании проекта среда дает новой форме стандартное имя **Form1**. Можно оставить это имя, однако удобнее давать формам имена, соответствующие номерам разрабатываемых приложений. Поэтому в поле свойства **Name** окна инспектора задайте имя **Form2**.

5. С помощью свойства **Color** задайте окну какой-либо цвет из отображаемой палитры цветов.

6. С помощью свойств **ClientWidth** и **ClientHeight** задайте новый размер рабочей области окна (например, 500 × 400 пикселей). Запустив программу (значок), удостоверьтесь в изменении размеров окна. То же действие можно выполнить проще, изменяя размер формы с помощью мыши.

7. Щелкните по кнопке на панели инструментов, а затем по какому-либо месту формы. В форме появится изображение объекта – метки (**Label**), а в левой части кадра будет отображено окно инспектора объектов для этого объекта. Найдя свойство **Caption**, укажите в качестве заголовка метки текст "**Мой график**". Найдя

свойство **Font** и открыв перечень характеристик шрифта, задайте шрифту увеличенный размер (12 – 14 пикселей), какой-либо цвет и другое имя (например, Times New Roman). Если текст метки выйдет за пределы окна, с помощью мыши переместите метку. Запустите программу и убедитесь в ее правильной работе.

8. Сохраните все файлы модифицированной программы, выбрав пункт меню File➢Save All или щелкнув по кнопке на панели инструментов. Закройте среду Borland C++ Builder.

Работа №3. Вывод в окно графика математической функции (индивидуальное задание Е)

1. Откройте созданный в предыдущей работе проект, воспользовавшись пунктом меню File➢OpenProject или щелкнув по кнопке на панели инструментов. Перед тем как приступить к модификации программы, сохраните компоненты проекта под именами **P3** и **U3** (см. пункт 2 предыдущей работы). Задайте для главного окна заголовок "**Программа 3**" и какой-либо светлый цвет. Задайте для формы имя **Form3**.

2. Щелкните по какому-либо месту формы. В левой части кадра будет отображено окно инспектора объектов для формы. (Следите за тем, окно *какого объекта* отображает инспектор объектов!) Перейдите на вкладку событий (**Events**). Щелкните по событию перекраски окна **OnPaint**; в правом столбце инспектора объектов откроется окно для ввода имени обработчика выбранного события. Щелкните дважды по этому окну, чтобы среда создала заготовку для функции обработчика и присвоила ей стандартное имя **FormPaint** (функции можно присвоить произвольное имя, хотя в данном случае это вряд ли целесообразно). В правую часть кадра будет выведен текст программного модуля U3.CPP с пустой пока функцией **FormPaint()**.

3. Любое рисование в окне осуществляется с помощью объекта **Canvas** (холст) – указателя на класс **TCanvas**, который создается для каждой формы автоматически и является эквивалентом контекста устройства. Посредством объекта **Canvas** можно изменять свойства графических инструментов (кистей, перьев, шрифтов) и вызывать функции вывода графических объектов (точек, линий, геометрических фигур, строк текста). Все программные предложения, связанные с рисованием, включаются в созданную ранее функцию **FormPaint()**.

Задайте для холста цветное перо толщиной в 4-5 пикселей и кисть другого цвета. Вызовом для объекта **Canvas** соответствующих методов выведите в середину окна какую-либо геометрическую фигуру (круг, эллипс, квадрат). Запустите программу и убедитесь в ее правильном отображении. Удалите эту фигуру.

4. Освоив основы работы с холстом, приступите к выполнению индивидуального задания. В середине главного окна указанного в задании цвета нарисуйте тонкими черными линиями горизонтальную и вертикальную оси координат. Организуйте цикл из требуемого числа шагов, в котором выполняются вычисления указанной в задании математической функции и вывод точек в окно с помощью подходящих методов класса **TCanvas**. Каждую точку графика отображайте в соответствии с заданием. Установив с помощью инспектора объектов увеличенный размер шрифта и назначив ему какой-либо яркий цвет, выведите под графиком строку текста с описанием отображаемой функции. Проверьте работу программы при сворачивании и разворачивании окна. При обнаружении недостатков устраните их. Сохраните файлы проекта щелчком по кнопке на панели инструментов.

Работа №4. Синхронизация программных действий с помощью таймера

1. Создайте из предыдущей работы новый проект с именем **P4**. Дайте форме имя **Form4**. Исправьте заголовок окна. Удалите из формы объект-метку (надпись «Мой график»).

2. На палитре компонентов в верхней части кадра перейдите на вкладку **System** и перенесите на форму (в любое ее место) объект таймера **Timer** (значок). В левой части кадра появится окно диспетчера объектов для объекта таймера с именем **Timer1**. Удостоверьтесь, что свойству **Enable** присвоено значение **true**, т. е. таймер включен. С помощью свойства **Interval** задайте разумную частоту работы таймера (5...20 Гц). Таймер будет использоваться в работе для анимации графика – последовательного, с частотой таймера, вывода его точек. Перейдя в инспекторе объектов на вкладку **Events**, выберите событие **OnTimer** и дважды щелкните по пустому полю справа, чтобы среда создала заготовку для функции обработчика с присвоением ей стандартного имени **Timer1Timer**. В правую часть кадра будет выведен текст программ-

ного модуля U4.CPP с пустой пока функцией **Timer1Timer()**. Включите в эту функцию вызов функции **Invalidate()** перерисовки окна по событиям таймера. В дальнейшем текст функции **Timer1Timer()** будет расширен.

3. Перейдите в файл заголовков U4.H. Если его нет на экране, откройте окно U4.CPP, щелкните по нему правой клавишей мыши и выберите пункт Open Source/Header File. Включите в секцию закрытых членов класса **TForm4** массивы для сохранения вычисленных значений точек отображаемого графика. Если математическая кривая задана в функциональной форме, достаточно одного массива для значений ординат графика; при задании кривой в параметрической форме потребуются два массива – для абсцисс и ординат точек. Внимательно отнеситесь к определению типа и числа элементов в массивах.

Предусмотрите также в секции закрытых членов класса **TForm4** целочисленную переменную, которая будет служить индексом элементов в массивах. Эта переменная при запуске приложения должна принимать начальное значение, равное 0, однако при объявлении переменной в составе класса присвоить ей значение нельзя; придется сделать это в ответ на сообщение о создании окна приложения. В окне инспектора объектов для формы **Form4** перейдите на вкладку **Events**, выберите событие **OnCreate** и создайте заготовку для функции обработчика этого события со стандартным именем **FormCreate**. Включите в эту функцию предложение инициализации объявленной ранее индексной переменной.

5. Перенесите предложения вычисления значений точек кривой в функцию **Timer1Timer()**. При каждом срабатывании таймера должны вычисляться значения *для одной точки*, соответствующей текущему значению индексной переменной. Вычислив значение функции f (или два значения x и y при параметрическом задании кривой), сохраните результат в предусмотренном массиве (массивах) и выполните инкремент индексной переменной.

Включите в функцию **Timer1Timer()** анализ индексной переменной и останов таймера после достижения ею максимального значения. Для останова таймера достаточно присвоить свойству **Enabled** объекта **Timer1** значение **false**.

6. Модифицируйте функцию **FormPaint()**, чтобы при выводе точек в окно приложения (в цикле по всем точкам) значения точек не вычислялись, а извлекались из массивов (возможно, с преобра-

зованием значений функции в координаты). Откомпилируйте и запустите приложение. Убедитесь в том, что форма графика осталась такой же, как и в предыдущей работе, но точки теперь выводятся на экран последовательно, с частотой работы таймера. Обратите внимание на неприятное мерцание экрана при перерисовке изображения. Устранить мерцание можно, формируя график не в окне приложения, а в совместимой памяти. Этот метод будет рассмотрен в следующей работе.

Работа №5. Использование совместимой памяти

1. Создайте из предыдущей работы новый проект с именем **P5**. Дайте форме имя **Form5**. Исправьте заголовок окна. Включите в закрытую секцию класса **TForm5** указатель на объект класса **TBitmap** (из модуля **Graphics**), образующий совместимую память. Удалите объявления массивов для значений точек графика.

2. В функции **FormCreate()** выполните ряд действий по созданию совместимой памяти (памяти растра) и подготовке ее содержимого.

- С помощью оператора **new** выделите память под объект совместимой памяти.
- Свойствам этого объекта **Height** и **Width** присвойте такие значения, чтобы график поместился в выделенную память.
- Задайте для холста растра (не для холста формы!) цветную кисть и закрасьте память растра этим цветом посредством вызова метода **FillRect()**. Цвет кисти выберите несовпадающим с цветом окна, чтобы в процессе отладки программы можно было видеть область окна, соответствующую объекту совместимой памяти.
- Перенесите из метода **FormPaint** все предложения, относящиеся к выводу осей координат и текстовой строки, заменив в них холст формы холстом растра. Задайте для шрифта холста растра цвет, размер и гарнитуру. Определите для холста растра требуемые по заданию перо и кисть для рисования точек графика.

В функции **Timer1Timer()**, вычислив очередное значение отображаемой математической функции, выведите точку графика в память растра и вызовом метода **Draw()** для холста окна (не растра!) инициируйте перерисовку окна.

Функция **FormPaint()** в данном варианте приложения будет

состоять из единственного (и необязательного) предложения перерисовки окна, которое нужно лишь для того, чтобы при изменении размеров окна изображение в окне на затиралось.

Выполните прогон программы и убедитесь, что график выводится так же, как и в предыдущей работе, но мерцание экрана исчезло.

Работа №6. Меню, стандартные диалоги и растровые изображения

1. Создайте новый проект с именем **P6** (открыв заново среду Borland C++ Builder или, если среда уже открыта для другого проекта, выбрав пункт меню **File>New Application**). Дайте форме имя **Form6**, задайте цвет главного окна и его заголовок.

2. Включите в приложение меню "**Файл**" с пунктами "**Открыть**" и "**Выход**". Для этого щелкните по значку меню в палитре компонентов, после чего щелкните по любому месту формы. На форме появится изображение значка меню, а на панели инспектора объектов будут отображены свойства и события нового объекта – меню с именем по умолчанию **MainMenu1**.

Дважды щелкните по значку меню в окне формы. На экране появится окно диспетчера меню, а на панели инспектора объектов будет выделено свойство **Caption**. Задайте в качестве заголовка меню слово "**Файл**" и нажмите клавишу Enter. В окне диспетчера меню появится указанный вами заголовок. Щелкните по нему. Под ним появится поле для первого пункта меню. Щелкните по этому полю, введите заголовок "**Открыть**" в поле **Caption** на панели диспетчера объектов и нажмите клавишу Enter. В окне диспетчера меню появится поле для следующего пункта. Щелкните по нему и задайте для него в качестве заголовка знак минус (–). В меню появится разделитель.

Создайте третий пункт меню, указав для него заголовок "**Выход**". Запустите программу и убедитесь в появлении главного меню в окне приложения.

3. Вернувшись в окно диспетчера меню, щелкните по первому пункту меню. В инспекторе объектов будет отображено окно свойств и событий для этого пункта меню. Перейдите на вкладку **Events**, выберите событие **OnClick** и дважды щелкните по пустому полю справа, чтобы среда создала заготовку для функции обработчика с присвоением ей стандартного имени (например,

N2Click). В правую часть кадра будет выведен текст программного модуля U6.CPP с пустой пока функцией **N2Click()**. Включите в эту функцию какое-либо невыполнимое предложение (знак комментария или пустой оператор – точку с запятой), иначе при дальнейших операциях пустая функция **N2Click()** исчезнет и ее придется создавать заново.

Повторите эту операцию для второго пункта меню ("**Выход**"). Если окно диспетчера меню оказалось закрытым другими окнами, то для вывода его на передний план можно воспользоваться пунктом главного меню среды **View➢Window List➢Form6➢MainMenu1**. Убедитесь, что в тексте программного модуля U6.CPP появилась вторая функция обработчика с именем, например, **N4Click()**. Включите в эту функцию вызов функции **Close()** для завершения приложения.

Закройте окно диспетчера меню. Если в дальнейшем оно понадобится снова, дважды щелкните по значку меню в окне формы. Запустите приложение и убедитесь, что его теперь можно закрыть, выбрав пункт меню "**Выход**".

4. На палитре компонентов в верхней части кадра перейдите на вкладку **Additional** и перенесите на форму объект изображения **Image** (значок). Расположите его в левом верхнем углу формы. В левой части кадра появится окно диспетчера объектов для объекта изображения с именем **Image1**. Найдя в окне диспетчера соответствующие свойства, задайте размеры изображения 200×200 пикселей, поскольку изображения медицинских препаратов, подготовленные для этой работы, имеют именно такой размер.

5. Найдите на палитре компонентов вкладку **Dialogs** и перенесите на форму объект **OpenPictureDialog**, предназначенный для открытия файлов с изображениями (значок). Определите для диалога свойство начального каталога (**InitialDir**), в качестве которого задайте свой рабочий каталог (можно указать знак текущего каталога – точку (.).

6. В функции отклика на пункт меню "**Открыть**" вызовом метода **Execute()** активизируйте объект диалога и вызовом метода **LoadFromFile()** загрузите выбранный файл в объект **Image1**.

7. Запустите программу, выберите в окне диалога "Открытие файла" файл с изображением медицинского препарата, соответствующий номеру вашего задания, и убедитесь в его правильном

отображении в окне приложения. Не забудьте сохранить все файлы созданного проекта щелчком по кнопке на панели инструментов, поскольку они понадобятся для выполнения следующей работы.

Работа №7. Растр в памяти и обработка изображений (индивидуальное задание F)

1. Создайте из предыдущей работы новый проект с именем **P7**. Дайте форме имя **Form7**. Исправьте заголовок окна.

2. Включите в закрытую секцию класса **TForm7** (в файле заголовков **U7.H**) указатель на объект класса **TBitmap** из модуля **Graphics**, образующий совместимую память, как это делалось в работе №5.

3. Откройте окно инспектора формы **Form7**. Перейдя на вкладку событий **Events**, создайте заготовку функции обработки события создания главного окна **FormCreate**. В этой функции оператором **new** выделите память под объект совместимой памяти.

4. Найдите в файле U7.CPP функцию отклика на пункт меню "**Открыть**". В предыдущей работе в этой функции открывался стандартный диалог Windows "Открытие файла", и для свойства **Picture** объекта **Image1** вызывался метод **LoadFromFile()**. В результате в интерфейсном объекте **Image1** отображалось загруженное растровое изображение. В настоящей работе следует выполнить, кроме указанных выше, еще две операции: скопировать то же изображение в совместимую память и отобразить содержимое совместимой памяти в окне приложения.

Для копирования изображения создайте (в той же функции отклика на пункт меню) объект класса **TMemoryStream**, вызовом метода **SaveToStream** запишите в поток содержимое объекта **Image1**, а затем вызовом метода **LoadFromStream** прочитайте данные из потока в созданный вами ранее объект совместимой памяти.

Вызовом для контекста формы метода **Draw()** скопируйте изображение из совместимой памяти в окно, расположив его правее объекта **Image 1**.

В результате после запуска программы и выбора рабочего файла в окне приложения должно дважды отобразиться одно и то же изображение.

5. Включите в меню дополнительный пункт "**Обработать**". Для этого, открыв диспетчер меню, щелкните *правой* клавишей мыши по "пункту" меню – разделителю и в появившемся окне выберите пункт **Insert**. В окне инспектора объектов щелкните дважды по окну события **OnClick**, в результате чего попадете в заготовку функции отклика на выбор этого пункта меню. Поместите в эту функцию алгоритм обработки изображения, соответствующий индивидуальному заданию. Обработка осуществляется в объекте совместимой памяти; завершив обработку, следует снова вызовом метода **Draw()** для контекста формы скопировать эту память (с измененным содержимым) в окно приложения.

Последним действием сохраните обработанное изображение в файле с заданным именем (например, TEMP.BMP), вызвав метод **SaveToFile()** для объекта совместимой памяти.

6. Выполните прогон программы: откройте файл с изображением медицинского препарата и выполните обработку этого изображения. Проанализируйте получившийся результат. Если в индивидуальном задании указан уровень отсечки того или иного цветового компонента, измените этот уровень и наблюдайте изменение вида обработанного изображения. Откройте файл с обработанным изображением TEMP.BMP. Убедитесь в том, что в этот файл действительно записано обработанное изображение.

Индивидуальные задания лабораторного практикума

Задание А1. В окно бледно-зеленого цвета выведите три красных круга диаметром 200 пикселей, расположенные пирамидой (один круг в центре над двумя другими). Выведите поверх изображения свою фамилию шрифтом Times, размером 80 пикселей, перечеркнутым, синего цвета. Надпись расположите горизонтально.

Задание В1. Выведите график функции $f = \sin t + \cos^2 t$, изменяя значение t от – 10 до 10 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 100.

Задание С1. Запишите в файл два целых числа. Получите в программе их сумму. Выведите в окно приложения исходные числа и результат вычисления с поясняющими надписями.

Задание D1. В ответ на щелчок левой клавишей мыши выведите круг диаметром 50 пикселей, залитый цветом исследуемого пикселя, а под кругом – значение цвета в шестнадцатеричной форме.

Задание Е1. В окно бледно-синего цвета выведите график декартова листа, уравнение которого в параметрической форме имеет вид $x = 3\,a\,t\,/\,(1+t^3)$, $y = 3\,a\,t^2\,/\,(1+t^3)$. Примите $a = -1.5$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от –50 до 50 с шагом 0.01, и умножайте их на 50. Точки графика отображайте в виде залитых цветом `clRed` треугольников размером 4-5 пикселей с вершиной, направленной вниз, используя метод `Polygon` класса `TCanvas`.

Задание F1. Преобразуйте цветное изображение в серое, сохранив градации яркости.

Задание А2. В окно голубого цвета выведите три концентрических квадрата с размерами сторон 300, 200 и 100 пикселей. Внешний квадрат нарисуйте толстым зеленым пером, средний – фиолетовым, внутренний – красным. Квадраты должны быть прозрачными. Выведите поверх изображения свою фамилию шрифтом Arial, размером 80 пикселей, жирным, желтого цвета. Надпись расположите под углом 15° к горизонтали.

Задание В2. Выведите график функции $f = e^{-t}\sin^2(6\,t)$, изменяя значение t от – 2 до 2 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 50.

Задание С2. Запишите в файл два дробных числа в экспоненциальном формате. Получите в программе их произведение и округ-

лите до целого. Выведите в окно приложения имя выбранного вами файла и результат вычислений.

Задание D2. В ответ на щелчок правой клавишей мыши выведите значение цвета исследуемого пикселя в шестнадцатеричной форме, а под байтами этого числа – условные обозначения компонентов цвета (КЗС).

Задание E2. В окно бледно-желтого цвета выведите фигуру Лиссажу, уравнение которой в параметрической форме имеет вид $x = \sin t$, $y = \cos (a\,t+b)$. Примите значение $a = 2$, $b = 0.8\pi$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от 0 до 2π с шагом 0.05, и умножайте их на 100. Точки графика отображайте в виде залитых цветом `clFuchsia` кружочков размером 4-5 пикселей.

Задание F2. Преобразуйте цветное изображение в черно-белое.

Задание A3. В окно желтого цвета выведите три соприкасающихся круга диаметром 200 пикселей каждый, расположенные горизонтально и залитые красным, зеленым и синим цветами. Выведите поверх изображения свою фамилию шрифтом Courier размером 80 пикселей, курсивом, черного цвета. Надпись расположите под углом 45° (снизу вверх).

Задание B3. В окно бледно-зеленого цвета выведите график функции $f = 2\,t^4 - t^5$, изменяя значение t от – 2 до 3 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 5.

Задание C3. Запишите в файл два дробных числа в формате с десятичной точкой. Получите в программе их сумму и разность. Выведите в окно приложения исходные числа и результаты вычислений с поясняющими надписями.

Задание D3. В ответ на двойной щелчок левой клавишей мыши выведите три соприкасающихся круга диаметром 50 пикселей, залитые компонентами цвета исследуемого пикселя. Под кругами выведите условные обозначения компонентов цвета (КЗС).

Задание E3. Выведите график улитки Паскаля, уравнение которой в параметрической форме имеет вид $x = a\cos^2 t + \cos t$, $y = a\cos t \sin t + \sin t$. Примите $a = -1.3$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от 0 до 2π с шагом 0.05, и умножайте их на 100. Точки графика отображайте в виде залитых цветом `clYellow` квадратиков размером 4-5 пикселей.

Задание F3. Переведите изображение в красный цвет, сохранив градации яркости.

Задание A4. В окно красного цвета выведите залитый синим цветом квадрат, ориентированный под углом 45° к осям координат. Такой наклонный квадрат рисуется с помощью функции `Polygon()` класса `TDC`. Выведите поверх изображения свою фамилию шрифтом Arial размером 80 пикселей, жирным, желтого цвета. Надпись расположите вертикально (снизу вверх).

Задание B4. Выведите график функции $f = \sin^3 t + b \cos^4 t$, приняв $b = -0.5$. Изменяйте значение t от – 10 до 10 и расположите в этом диапазоне 400 точек. Ординаты графика умножайте на 100.

Задание C4. Запишите в файл свою фамилию и год рождения. Выведите в окно приложения прочитанную фамилию и ваш возраст от 1900 г.

Задание D4. В ответ на двойной щелчок правой клавишей мыши выведите значения компонентов цвета в десятичной форме с соответствующими комментариями (`Значение синего:` и т. д.).

Задание E4. В окно цвета `clBackground` выведите график улитки Паскаля, уравнение которой в параметрической форме имеет вид $x = a \cos^2 t + \cos t$, $y = a \cos t \sin t + \sin t$. Примите $a = -0.7$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от 0 до 2π с шагом 0.05 и умножайте их на 100. Точки графика отображайте символом ^ фиолетового цвета.

Задание F4. Преобразуйте цветное изображение в инверсное серое с сохранением градаций яркости.

Задание A5. В окно желтого цвета выведите три концентрические окружности с диаметрами 400, 300 и 200 пикселей. Внешнюю окружность нарисуйте толстым (6-8 пикселей) зеленым пером, среднюю – синим, а внутреннюю – коричневым. Образованные круги должны быть прозрачными. Выведите поверх изображения свою фамилию шрифтом Times размером 80 пикселей, курсивом, синего цвета. Надпись расположите вертикально (сверху вниз).

Задание B5. Выведите график функции $f = t^2 \sin^2 t$, изменяя значение t от – 10 до 10 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 5.

Задание C5. Запишите в файл три целых числа. Получите в программе их среднее значение. Выведите в окно приложения исходные числа и результат вычисления с поясняющими надписями.

Задание D5. В ответ на щелчок левой клавишей мыши при нажатой клавише Shift выведите значение цвета исследуемого пикселя в шестнадцатеричной и десятичной формах. Под байтами шест-

надцатеричного числа укажите условные обозначения компонентов цвета (КЗС).

Задание E5. В окно красного цвета выведите график циклоиды, уравнение которой в параметрической форме имеет вид $x = a\,(t - \sin t)$, $y = a\,(1 - \cos t)$. Примите $a = 1.5$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от –20 до +20 с шагом 0.1. Для получения наглядного графика значения x умножайте на 10, а y – на 50. Точки графика отображайте в виде залитых цветом **`clSilver`** треугольников размером 4-5 пикселей с вершиной, направленной влево, используя метод **`Polygon`** класса **`TCanvas`**.

Задание F5. Разделите изображение на 3 слоя по яркости, покрасив самый светлый слой в ярко-синий цвет, средний – в синий, а темный слой – в темно-синий цвет.

Задание A6. В окно серого цвета выведите круг розового цвета диаметром 300 пикселей, у которого правая средняя четверть (сектор размером 90°) покрашена в синий цвет. Выведите поверх изображения свою фамилию шрифтом Courier размером 80 пикселей, жирным, красного цвета. Надпись расположите под углом 45° (сверху вниз).

Задание B6. Выведите график функции $f = \mathrm{arctg}\ t$, изменяя значение t от – 10 до 10 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 100.

Задание C6. Запишите в файл два числа: целое и дробное в экспоненциальном формате. Получите в программе их произведение и округлите до целого. Выведите в окно приложения имя выбранного вами файла и результат вычисления.

Задание D6. В ответ на щелчок левой клавишей мыши при нажатой клавише Ctrl выведите три квадрата размером 50 пикселей, залитые компонентами цвета исследуемого пикселя. Под квадратами выведите условные обозначения компонентов цвета (КЗС).

Задание E6. В окно синего цвета выведите график фигуры Лиссажу, уравнение которой в параметрической форме имеет вид $x = \sin t$, $y = \cos\,(a\,t+b)$. Примите значение $a = 2$, $b = -\,0.55\pi$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от 0 до 2π с шагом 0.05, и умножайте их на 100. График нарисуйте желтыми точками, используя метод **`Pixels`**.

Задание F6. Преобразуйте цветное изображение в черно-белое инверсное.

Задание А7. В окно голубого цвета выведите три концентрических круга с диаметрами 300, 200 и 100 пикселей. Закрасьте внешний круг зеленым цветом, средний – желтым, а внутренний – белым. Выведите поверх изображения свою фамилию шрифтом Times размером 80 пикселей, жирным, курсивом, темно-синего цвета. Надпись расположите горизонтально.

Задание В7. Выведите график функции $f = e^{-t} \sin 10\, t$, изменяя значение t от – 2 до 2 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 50.

Задание С7. Запишите в файл два числа: целое и дробное в формате с десятичной точкой. Получите в программе их сумму и разность. Выведите в окно приложения исходные числа и результаты вычислений с поясняющими надписями.

Задание D7. В ответ на щелчок правой клавишей мыши при нажатой клавише Shift выведите три круга диаметром 50 пикселей, залитые компонентами цвета пикселя. Под кругами выведите условные обозначения компонентов цвета (КЗС).

Задание Е7. В окно бледно-синего цвета выведите график гиперболической спирали, уравнение которой в параметрической форме имеет вид $x = a \cos t / t$, $y = a \sin t / t$. Примите $a = 2$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от –50 до +50 с шагом 0.01; полученные значения x и y умножайте на 100. График нарисуйте красными точками, используя метод **`Pixels`** класса **`TCanvas`**.

Задание F7. Выделите в изображении темные области, оставив их цвет; светлые области закрасьте белым.

Задание А8. В окно желтого цвета выведите круг диаметром 400 пикселей, верхняя половина которого закрашена синим, а нижняя – красным цветом. Части окружности рисуются с помощью функций **`Pie()`** или **`Chord()`** класса **`TDC`**. Выведите поверх изображения свою фамилию вертикально снизу вверх шрифтом Courier размером 80 пикселей, жирным, перечеркнутым, белого цвета.

Задание В8. Выведите график многочлена $f = a\, t^3 + b\, t^2 + c\, t + d$ при значениях коэффициентов $a = 1.2$, $b = -3$, $c = 0.6$, $d = 1.5$. Изменяйте значение t от – 1.5 до 2.5 и расположите в этом диапазоне 400 точек. Ординаты графика умножайте на 20.

Задание С8. Запишите в файл целое число и название учебной группы. Выведите в окно приложения прочитанное название группы и квадрат исходного числа.

Задание D8. В ответ на щелчок правой клавишей мыши при нажатой клавише Ctrl выведите три соприкасающихся круга, залитые компонентами цвета исследуемого пикселя. Ниже выведите еще три круга, залитые чистыми цветами максимальной яркости.

Задание E8. В окно цвета **clSilver** выведите график декартова листа, уравнение которого в параметрической форме имеет вид $x = 3\ a\ t\ /\ (1+t^3)$, $y = 3\ a\ t^2\ /\ (1+t^3)$. Примите $a = -1.5$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от – 20 до 20 с шагом 0.05, и умножайте их на 50. Точки графика отображайте в виде залитых синим цветом половинок круга размером 5-6 пикселей, обращенных округлой частью вниз. Используйте метод **Pie** класса **TCanvas**.

Задание F8. Выделите в изображении светлые области, оставив их цвет; темные области закрасьте фиолетовым цветом.

Задание A9. В окно черного цвета выведите три концентрических квадрата с размерами сторон 400, 300 и 200 пикселей. Внешний квадрат закрасьте желтым цветом, средний – синим, а внутренний – красным. Выведите поверх изображения свою фамилию шрифтом Times размером 80 пикселей, жирным, подчеркнутым, светло-серого цвета. Надпись расположите горизонтально, "вверх ногами".

Задание B9. В окно ярко-синего цвета выведите график функции $f = t^2 \sin t$, изменяя значение t от – 4 до 4 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 20.

Задание C9. Запишите в файл свою фамилию и возраст. Получите в программе число прожитых вами дней. Выведите в окно приложения результат вычисления и прочитанную фамилию.

Задание D9. В ответ на двойной щелчок левой клавишей мыши при нажатой клавише Shift выведите значения компонентов цвета в десятичной форме. Ниже выведите те же значения в шестнадцатеричной форме. Еще ниже выведите условные обозначения компонентов цвета (КЗС).

Задание E9. Выведите график кардиоиды, уравнение которой в параметрической форме имеет вид $x = \cos t\ (1+\cos t)$, $y = \sin t\ (1+\cos t)$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от 0 до 2π с шагом 0.01, и умножайте их на 100. Точки графика отображайте в виде залитых цветом **clWhite** половинок круга размером 5-6 пикселей, обращенных округлой частью вправо. Используйте метод **Pie** класса **TCanvas**.

Задание F9. Переведите изображение в синий цвет, сохранив градации яркости.

Задание A10. В окно светло-зеленого цвета выведите круг диаметром 400 пикселей, залитый желтым цветом, верхняя четверть которого (до соответствующей хорды) закрашена темно-зеленым цветом. Фигуры, ограниченные частью окружности и хордой, рисуются с помощью функции **`Chord()`** класса **`TDC`**. Выведите поверх изображения свою фамилию шрифтом Arial размером 80 пикселей, жирным, курсивом, синего цвета. Надпись расположите под углом 30° к горизонтали.

Задание B10. Выведите график функции $f = \sin^3 t + \cos^4 t$, изменяя значение t от – 10 до 10 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 100.

Задание C10. Запишите в файл год своего рождения. Получите в программе число прожитых вами месяцев. Выведите в окно приложения свою фамилию и результат вычисления.

Задание D10. В ответ на двойной щелчок левой клавишей мыши при нажатой клавише Ctrl выведите три соприкасающихся квадрата, залитые компонентами цвета исследуемого пикселя. В квадратах укажите условные обозначения компонентов цвета (КЗС).

Задание E10. В окно цвета **`clFuchsia`** выведите график улитки Паскаля, уравнение которой в параметрической форме имеет вид $x = a \cos^2 t + \cos t$, $y = a \cos t \sin t + \sin t$. Примите $a = -1.3$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от 0 до 2π с шагом 0.05, и умножайте их на 100. Точки графика отображайте символом буквы **`v`** цвета **`clAqua`**.

Задание F10. Разделите изображение на 3 слоя по яркости, покрасив самый светлый слой в желтый цвет, средний – в коричневый, а темный слой – в темно-коричневый цвет.

Задание A11. В окно светло-коричневого цвета выведите толстую (6-8 пикселей) фиолетовую окружность диаметром 400 пикселей, левая половина которой заштрихована горизонтально, а правая – вертикально тем же фиолетовым цветом. Части окружности рисуются с помощью функций **`Pie()`** или **`Chord()`** класса **`TDC`**. Кисть для штриховки замкнутой фигуры создается конструктором **`TBrush`** с указанием стиля, значения которого можно найти в описании функции **`CreateHatchBrush()`**. Выведите поверх изображения свою фамилию шрифтом Times размером 80 пикселей,

курсивом, синего цвета. Надпись расположите вертикально сверху вниз.

Задание B11. Выведите график функции $f = e^{-(t-a)(t-a)}$ при значении параметра $a = 5$. Изменяйте значение t от – 10 до 10 и расположите в этом диапазоне 400 точек. Ординаты графика умножайте на 100.

Задание C11. Запишите в файл два целых числа. Получите в программе их геометрическое среднее. Выведите в окно приложения исходные числа и результат вычислений.

Задание D11. В ответ на двойной щелчок правой клавишей мыши при нажатой клавише Shift выведите три круга, залитые компонентами цвета исследуемого пикселя. В кругах укажите численные значения компонентов в десятичной форме.

Задание E11. В окно коричневого цвета выведите график гиперболической спирали, уравнение которой в параметрической форме имеет вид $x = a \cos t / t$, $y = a \sin t / t$. Примите $a = 2$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от –50 до +50 с шагом 0.1; полученные значения x и y умножайте на 100. Точки графика отображайте в виде залитых цветом **`clLime`** квадратиков размером 4-5 пикселей.

Задание F11. Переведите изображение в зеленый цвет, сохранив градации яркости.

Задание A12. В окно светло-голубого цвета выведите желтый круг диаметром 400 пикселей, с "выеденной" с правой стороны частью (как у надкусанного яблока). Выведите поверх изображения свою фамилию шрифтом Times размером 80 пикселей, подчеркнутым, жирным, курсивом, красного цвета. Надпись расположите под углом 75° (снизу вверх).

Задание B12. В окно цвета **`clLtGray`** выведите график многочлена $f = a\,t^4 + b\,t^3$ при значениях коэффициентов $a = 1.5$, $b = -5$. Изменяйте значение t от – 1.5 до 3.5 и расположите в этом диапазоне 400 точек. Ординаты графика умножайте на 10.

Задание C12. Запишите в файл два дробных числа в формате с десятичной точкой. Получите в программе их частное и округлите его до целого. Выведите в окно приложения имя выбранного вами файла и результат вычисления.

Задание D12. В ответ на двойной щелчок правой клавишей мыши при нажатой клавише Ctrl выведите три квадрата, залитые компонентами цвета исследуемого пикселя. Поверх этих квадратов вы-

ведите небольшие круги, залитые чистыми цветами максимальной яркости.

Задание E12. Выведите фигуру Лиссажу, уравнение которой в параметрической форме имеет вид $x = \sin t$, $y = \cos (a\,t+b)$. Примите значение $a = 2$, $b = 0.8\pi$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от 0 до 2π с шагом 0.01, и умножайте их на 100. График нарисуйте зелеными точками, используя метод **`Pixels`** класса **`TCanvas`**.

Задание F12. Преобразуйте цветное изображение в серое с 256 градациями яркости.

Задание A13. В окно коричневого цвета выведите три соприкасающихся круга диаметром 150 пикселей каждый, расположенные вертикально и залитые серым, красным и белым цветами. Выведите поверх изображения свою фамилию шрифтом Times размером 80 пикселей, подчеркнутым и перечеркнутым, черного цвета. Надпись расположите вертикально снизу вверх.

Задание B13. Выведите график функции $f = t^4 - t^5$, изменяя значение t от – 1 до 1.5 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 100.

Задание C13. Запишите в файл два числа: дробное в формате с фиксированной точкой и целое. Получите в программе их частное и округлите до целого. Выведите в окно приложения имя выбранного вами файла и результат вычисления с поясняющей надписью.

Задание D13. В ответ на щелчок левой клавишей мыши выведите круг, залитый цветом исследуемого пикселя, а под ним – еще три круга, залитые компонентами цвета пикселя.

Задание E13. Выведите график улитки Паскаля, уравнение которой в параметрической форме имеет вид $x = a \cos^2 t + \cos t$, $y = a \cos t \sin t + \sin t$. Примите $a = -0.7$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от 0 до 2π с шагом 0.01 и умножайте их на 100. Точки графика отображайте в виде залитых цветом **`clPurple`** квадратиков размером 4-5 пикселей.

Задание F13. Закрасьте темно-красным темные области изображения.

Задание A14. В окно серого цвета выведите три вложенных друг в друга квадрата с размерами сторон 300, 200 и 100 пикселей, соприкасающиеся левыми верхними углами. Внешний квадрат закрасьте фиолетовым цветом, средний – темно-синим, внутренний –

светло-синим. Выведите поверх изображения свою фамилию шрифтом Courier размером 80 пикселей, жирным, желтого цвета. Надпись расположите под углом 45° (снизу вверх).

Задание B14. Выведите график функции $f = e^{-t} \cos 5\,t$, изменяя значение t от – 2 до 2 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 50.

Задание C14. Запишите в файл два дробных числа: первое в формате с десятичной точкой, а второе в экспоненциальном формате. Получите в программе их произведение. Выведите в окно приложения исходные числа и результат вычисления с поясняющими надписями.

Задание D14. В ответ на щелчок правой клавишей мыши выведите значения компонентов цвета в десятичной форме с соответствующими комментариями (**`Синий:`** и т.д.), а под ними – квадраты, залитые компонентами цвета исследуемого пикселя.

Задание E14. Выведите график фигуры Лиссажу, уравнение которой в параметрической форме имеет вид $x = \sin t$, $y = \cos (a\,t+b)$. Примите значение $a = 2$, $b = -\,0.55\pi$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от 0 до 2π с шагом 0.01, и умножайте их на 100. Точки графика отображайте в виде залитых цветом **`clPurple`** кружочков размером 4-5 пикселей.

Задание F14. Закрасьте желтым светлые области изображения.

Задание A15. В окно синего цвета выведите круг диаметром 400 пикселей желтого цвета, у которого верхняя левая четверть (сектор размером 90°) прозрачна. Сектор рисуется с помощью функции **`Pie()`** класса **`TDC`**. Выведите поверх изображения свою фамилию шрифтом Arial размером 80 пикселей, перечеркнутым, жирным, красного цвета. Надпись расположите под углом 75° (снизу вверх).

Задание B15. В окно цвета **`clMaroon`** выведите график функции $f = a \cos t + b \sin^3 t$ при значениях параметров $a = 0.8$, $b = -1.6$. Изменяйте значение t от – 10 до 10 и расположите в этом диапазоне 400 точек. Ординаты графика умножайте на 50.

Задание C15. Запишите в файл число в формате с десятичной точкой и свою фамилию. Получите в программе корень из числа. Выведите в окно приложения результат вычисления и прочитанную фамилию, добавив к ней инициалы.

Задание D15. В ответ на двойной щелчок левой клавишей мыши выведите значение синего компонента цвета исследуемого пикселя в десятичной форме с соответствующим комментарием. Рядом вы-

ведите круг, залитый цветом этого компонента, а под ним – второй круг, залитый синим цветом максимальной интенсивности.

Задание E15. Выведите график циклоиды, уравнение которой в параметрической форме имеет вид $x = a\ (t - \sin t)$, $y = a\ (1 - \cos t)$. Примите $a = 1.5$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от –20 до +20 с шагом 0.01. Для получения наглядного графика значения x умножайте на 10, а y на 30. График нарисуйте желтыми точками, используя метод **`Pixels`** класса **`TCanvas`**.

Задание F15. Переведите изображение в фиолетовый цвет, сохранив градации яркости.

Задание A16. В окно желтого цвета выведите два эллипса синего цвета с общим центром, расположенные один горизонтально, а другой вертикально. Выведите поверх изображения свою фамилию шрифтом Times размером 80 пикселей, узким (шириной 10 пикселей), жирным, красного цвета. Надпись расположите горизонтально.

Задание B16. Выведите график функции $f = \sin t + \cos t$, изменяя значение t от – 10 до 10 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 100.

Задание C16. Запишите в файл дробное число в формате с десятичной точкой. Получите в программе целую и дробную части числа. Выведите в окно приложения исходное число и результаты вычислений с поясняющими надписями.

Задание D16. В ответ на двойной щелчок правой клавишей мыши выведите значение красного компонента цвета исследуемого пикселя в десятичной форме с соответствующим комментарием. Рядом выведите круг, залитый цветом этого компонента, а под ним – второй круг, залитый красным цветом максимальной интенсивности.

Задание E16. В окно цвета **`clLime`** выведите график кардиоиды, уравнение которой в параметрической форме имеет вид $x = \cos t\ (1 + \cos t)$, $y = \sin t\ (1 + \cos t)$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от 0 до 2π с шагом 0.05, и умножайте их на 100. Точки графика нарисуйте залитыми черным ромбиками размером 4-5 пикселей, используя метод **`Polygon`** класса **`TCanvas`**.

Задание F16. Разделите изображение на 2 слоя по яркости, покрасив светлый слой в красный цвет, а темный – в синий.

Задание A17. В окно бледно-зеленого цвета выведите круг диаметром 400 пикселей, левая половина которого закрашена темно-красным, а правая – ярко-красным цветом. Полукруги рисуются с помощью функций **`Pie()`** или **`Chord()`** класса **`TDC`**. Выведите поверх изображения свою фамилию шрифтом Courier размером 80 пикселей, подчеркнутым, жирным, светло-серого цвета. Надпись расположите под углом 15° сверху вниз.

Задание B17. Выведите график функции $f = \sin^2 t + \cos^3 t$, изменяя значение t от – 10 до 10 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 100.

Задание C17. Запишите в файл название учебного предмета и целое число. Получите в программе квадрат числа. Выведите в окно приложения результат вычисления с поясняющей надписью и прочитанное название предмета.

Задание D17. В ответ на щелчок левой клавишей мыши при нажатой клавише Shift выведите значение зеленого компонента цвета исследуемого пикселя в десятичной форме с соответствующим комментарием. Рядом выведите круг, залитый цветом этого компонента, а под ним – второй круг, залитый зеленым цветом максимальной интенсивности.

Задание E17. В окно очень бледно-зеленого цвета выведите график декартова листа, уравнение которого в параметрической форме имеет вид $x = 3\ a\ t\ /\ (1 + t^3)$, $y = 3\ a\ t^2\ /\ (1 + t^3)$. Примите $a = -1.5$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от –50 до 50 с шагом 0.01, и умножайте их на 50. Точки графика отображайте в виде залитых цветом **`clMaroon`** треугольников размером 4-5 пикселей с вершиной, направленной вниз, используя метод **`Polygon`** класса **`TCanvas`**.

Задание F17. Переведите изображение в оттенки голубого цвета, сохранив градации яркости.

Задание A18. В окно светло-бирюзового цвета выведите круг диаметром 400 пикселей, разделенный пополам диагональю, проходящей вправо и вверх. Левую половину круга закрасьте красным цветом, правую – синим. Полукруги рисуются с помощью функций **`Pie()`** или **`Chord()`** класса **`TDC`**. Выведите поверх изображения свою фамилию шрифтом Times размером 80 пикселей, широким (шириной 80 пикселей), курсивом, черного цвета. Надпись расположите горизонтально.

Задание B18. Выведите график функции $f = e^{-t} \sin 5\ t$, изменяя значение t от – 4 до 4 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 20.

Задание C18. Запишите в файл свое имя и дробное число в экспоненциальном формате. Получите в программе квадратный корень из числа. Выведите в окно приложения прочитанное имя, исходное число и результат вычисления.

Задание D18. В ответ на щелчок левой клавишей мыши при нажатой клавише Ctrl выведите круг, залитый цветом исследуемого пикселя, а под ним – строку со значениями цветовых компонентов в десятичной форме.

Задание E18. В окно цвета **`clBackground`** выведите график улитки Паскаля, уравнение которой в параметрической форме имеет вид $x = a \cos^2 t + \cos t$, $y = a \cos t \sin t + \sin t$. Примите $a = -0.7$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от 0 до 2π с шагом 0.05 и умножайте их на 100. Точки графика отображайте символом ^ фиолетового цвета.

Задание F18. Преобразуйте цветное изображение в инверсное серое с сохранением градаций яркости.

Задание A19. В окно светло-серого цвета выведите прямоугольный треугольник синего цвета с длиной катетов 400 пикселей. Треугольник рисуется с помощью функции **`Polygon()`** класса **`TDC`**. Выведите поверх изображения свою фамилию шрифтом Arial размером 80 пикселей, курсивом, белого цвета. Надпись расположите под углом 75°.

Задание B19. Выведите график функции $f = t^2 \sin t$, изменяя значение t от – 4 до 4 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 40.

Задание C19. Запишите в файл число в экспоненциальном формате и свою фамилию. Получите в программе корень из числа. Получите в программе целую и дробную части числа. Выведите в окно приложения прочитанную фамилию и результаты вычислений.

Задание D19. В ответ на щелчок правой клавишей мыши при нажатой клавише Shift выведите значения трех компонентов цвета исследуемого пикселя в шестнадцатеричной и десятичной формах.

Задание E19. В окно очень светло-серого цвета выведите график фигуры Лиссажу, образующейся при сложении двух гармонических колебаний с разными частотами и фазами. Уравнение этой кривой в параметрической форме имеет вид $x = \sin t$, $y = \cos (a\ t+b)$.

Примите значение $a = 2$, $b = -0.55\pi$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от 0 до 2π с шагом 0.05, и умножайте их на 100. График нарисуйте темно-синими точками, используя метод **`Pixels`** класса **`TCanvas`**.

Задание F19. Разделите изображение на 2 слоя по яркости и инвертируйте его, покрасив светлый слой в синий цвет, а темный – в светло-голубой.

Задание A20. В окно бледно-розового цвета выведите три концентрических эллипса с размерами 500x300, 400x200 и 200x100 пикселей. Внешний эллипс нарисуйте толстым (6-8 пикселей) черным пером, средний – синим, а внутренний – желтым. Эллипсы должны быть прозрачными. Выведите поверх изображения свою фамилию шрифтом Times размером 80 пикселей, перечеркнутым, узким (шириной 10 пикселей), красного цвета. Надпись расположите горизонтально.

Задание B20. Выведите график функции $f = e^{-s(t-a)(t-a)}$ при значении параметров $s = 0.5$, $a = -2$. Изменяйте значение t от – 10 до 10 и расположите в этом диапазоне 400 точек. Ординаты графика умножайте на 100.

Задание C20. Запишите в файл свою фамилию и год рождения. Выведите в окно приложения прочитанную фамилию и оставшееся вам "время жизни" до 3000 г.

Задание D20. В ответ на щелчок правой клавишей мыши при нажатой клавише Ctrl выведите квадрат, залитый цветом исследуемого пикселя, а под ним две строки: значение цвета в шестнадцатеричной форме и значения компонентов цвета в десятичной форме.

Задание E20. В окно бледно-фиолетового цвета выведите график циссоиды, уравнение которой в параметрической форме имеет вид $x = a\ t^2 / (1 + t^2)$, $y = a\ t^3 / (1 + t^2)$. Примите $a = 1$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от $-\pi$ до π с шагом 0.01, и умножайте их на 50. Точки графика отображайте в виде залитых цветом **`clPurple`** треугольников размером 4-5 пикселей с вершиной, направленной вверх, используя метод **`Polygon`** класса **`TCanvas`**.

Задание F20. Переведите изображение в желтый цвет, сохранив градации яркости.

Задание A21. В окно голубого цвета выведите изображение Солнца в виде желтого круга, эллиптической орбиты Земли вокруг него и самой Земли в виде коричневого кружка в верхней точке этой орбиты. Выведите поверх изображения горизонтально свою фамилию шрифтом Courier размером 20 пикселей, подчеркнутым, белого цвета. Надпись расположите горизонтально.

Задание B21. Выведите график функции арктангенса $f = \mathrm{arctg}\ t$, изменяя значение t от – 10 до 10 и расположив в этом диапазоне 400 точек. Ординаты функции умножайте на 100.

Задание C21. Запишите в файл два числа: дробное в экспоненциальном формате и целое. Возведите первое число в степень второго. Выведите в окно приложения исходные числа и результат вычисления с поясняющими надписями.

Задание D21. В ответ на двойной щелчок левой клавишей мыши при нажатой клавише Shift выведите круг, залитый цветом исследуемого пикселя, а рядом – числовое значение цвета в десятичной и шестнадцатеричной формах.

Задание E21. В окно цвета `clInactiveCaption` выведите график гиперболической спирали, уравнение которой в параметрической форме имеет вид $x = a \cos t / t$, $y = a \sin t / t$. Примите $a = 2$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от –50 до +50 с шагом 0.01; полученные значения x и y умножайте на 100.Точки графика отображайте символом звездочки (*) красного цвета.

Задание F21. Разделите изображение на 2 слоя по яркости, покрасив светлый слой в светло-серый цвет, а темный – в темно-серый.

Задание A22. В окно синего цвета выведите большой желтый треугольник. Треугольник рисуется с помощью функции `Polygon()` класса `TDC`. Выведите поверх изображения свою фамилию шрифтом Arial размером 80 пикселей, узким (шириной 20 пикселей), черного цвета. Надпись расположите под углом 45° сверху вниз.

Задание B22. Выведите график многочлена $f = a\ t^4 + b\ t^3$ при значениях коэффициентов $a = 1.5$, $b = -5$. Изменяйте значение t от – 4 до 6 и расположите в этом диапазоне 400 точек.

Задание C22. Запишите в файл два дробных числа: первое в экспоненциальном формате, а второе – в формате с десятичной

точкой. Получите в программе их произведение. Выведите в окно приложения результат вычисления и имя выбранного вами файла.

Задание D22. В ответ на двойной щелчок левой клавишей мыши при нажатой клавише Ctrl выведите значение цвета исследуемого пикселя в шестнадцатеричной форме, а под байтами этого числа – условные обозначения компонентов цвета (КЗС).

Задание E22. В окно очень бледно-синего цвета выведите график строфоиды, уравнение которой в параметрической форме имеет вид $x = a\ (t^2 - 1)$, $y = a\ t\,(t^2 - 1) / (t^2 + 1)$. Примите $a = 1$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от $-\pi/2$ до $\pi/2$ с шагом 0.01 и умножайте их на 100. Точки графика отображайте символом плюс (+) увеличенной высоты (свойство **`Height`** класса **`TFont`**). Не забудьте после вывода графика установить в контексте исходную высоту шрифта.

Задание F22. Преобразуйте цветное изображение в монохромное синее.

Задание A23. В окно синего цвета выведите изображение красного полукруглого солнца (как бы заходящего за горизонт) с расходящимися от него несколькими лучами в виде толстых красных линий. Линии рисуются функцией **`LineTo()`**, начальная точка перемещается функцией **`MoveTo()`**. Половины круга рисуются функциями **`Pie()`** или **`Chord()`** класса **`TDC`**. Выведите поверх изображения свою фамилию шрифтом Times размером 80 пикселей, жирным, курсивом, светло-серого цвета. Надпись расположите горизонтально вверх ногами.

Задание B23. Выведите график функции $f = 1/\mathrm{arctg}\ t$, изменяя значение t от – 2 до 2 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 20.

Задание C23. Запишите в файл число в экспоненциальном формате и свою фамилию. Получите в программе корень из числа. Выведите в окно приложения результат вычисления и прочитанную фамилию, предварив ее инициалами.

Задание D23. В ответ на двойной щелчок правой клавишей мыши при нажатой клавише Shift выведите три круга диаметром 50 пикселей, залитые компонентами цвета исследуемого пикселя. Под кругами выведите названия цветов компонентов.

Задание E23. В окно цвета **`clFuchsia`** выведите график улитки Паскаля, уравнение которой в параметрической форме имеет вид $x = a \cos^2 t + \cos t$, $y = a \cos t \sin t + \sin t$. Примите $a = -1.3$. Зна-

чения x и y вычисляйте в цикле по переменной t, принимающей значения от 0 до 2π с шагом 0.05, и умножайте их на 100. Точки графика отображайте символом буквы **v** цвета **clAqua**.

Задание F23. Разделите изображение на 3 слоя по яркости, покрасив самый светлый слой в салатный цвет, средний – в зеленый, а темный слой – в темно-зеленый цвет.

Задание A24. В окно зеленого цвета выведите друг над другом два полукруга, соприкасающихся своими округлыми частями. Верхний полукруг залейте синим цветом, нижний – красным. Половины круга рисуются функциями **Pie()** или **Chord()** класса **TDC**. Выведите поверх изображения свою фамилию шрифтом Courier размером 80 пикселей, подчеркнутым и перечеркнутым, белого цвета. Надпись расположите горизонтально.

Задание B24. Выведите график функции $f = \cos t + \sin^2 2t$, изменяя значение t от – 4 до 4 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 100.

Задание C24. Запишите в файл свою фамилию и год рождения. Получите в программе свой возраст. Выведите в окно приложения прочитанную фамилию и результат вычисления.

Задание D24. В ответ на двойной щелчок правой клавишей мыши при нажатой клавише Ctrl выведите значения компонентов цвета в шестнадцатеричной и десятичной формах с соответствующими комментариями (**Значение красного:** и т.д.).

Задание E24. В окно желтого цвета выведите график строфоиды, уравнение которой в параметрической форме имеет вид $x = a\ (t^2 - 1)$, $y = a\ t\,(t^2 - 1) / (t^2 + 1)$. Примите $a = -1$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от $-\pi/2$ до $\pi/2$ с шагом 0.01, и умножайте их на 100. График нарисуйте черными точками, используя метод **Pixels** класса **TCanvas**.

Задание F24. Разделите изображение на 2 слоя по яркости, покрасив светлый слой в желтый цвет, а темный – в коричневый.

Задание A25. В окно светло-голубого цвета выведите желтое изображение солнца с расходящимися от него несколькими толстыми желтыми лучами. Линии рисуются функцией **LineTo()**, начальная точка перемещается функцией **MoveTo()** класса **TDC**. Выведите поверх изображения свою фамилию шрифтом Arial размером 80 пикселей, перечеркнутым, жирным, курсивом, красного цвета. Надпись расположите под углом 30° снизу вверх.

Задание B25. Выведите график функции $f = a \cos t + b \sin^3 t$ при значениях параметров $a = 0.8$, $b = -1.6$. Изменяйте значение t от -10 до 10 и расположите в этом диапазоне 400 точек. Ординаты графика умножайте на 50.

Задание C25. Запишите в файл два числа: дробное в формате с десятичной точкой и целое. Получите в программе их произведение и округлите его до целого. Выведите в окно приложения имя выбранного вами файла и результат вычисления.

Задание D25. В ответ на щелчок левой клавишей мыши при нажатой клавише Shift выведите три круга диаметром 50 пикселей, залитые компонентами цвета исследуемого пикселя. Под фигурами укажите названия цветов компонентов.

Задание E25. В окно цвета **`clAqua`** выведите график конхоиды Никомеда, уравнение которой в параметрической форме имеет вид $x = a + b \cos t$, $y = a \operatorname{tg} t + b \sin t$. Примите $a = 1$, $b = 1$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от $-\pi/2$ до $3\pi/2$ с шагом 0.01 и умножайте x на 50, а y на 20. Точки графика отображайте символом буквы **x** красного цвета.

Задание F25. Преобразуйте цветное изображение в монохромное зеленое.

Задание A26. В окно голубого цвета выведите удлиненный ромб, расположенный горизонтально и залитый темно-красным цветом. Ромб рисуется с помощью функции **`Polygon()`** класса **TDC**. Выведите поверх изображения свою фамилию шрифтом Times размером 150 пикселей, белого цвета. Надпись расположите вертикально (сверху вниз).

Задание B26. Выведите график функции $f = \sin t / \cos t$, изменяя значение t от -4 до 4 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 10.

Задание C26. Запишите в файл два дробных числа: первое в формате с десятичной точкой, а второе в экспоненциальном формате. Получите в программе их разность. Выведите в окно приложения результат вычисления в обоих форматах (экспоненциальном и с десятичной точкой).

Задание D26. В ответ на щелчок левой клавишей мыши при нажатой клавише Ctrl выведите круг диаметром 50 пикселей, залитый цветом исследуемого пикселя, а под ним – еще три таких же круга, залитые компонентами цвета пикселя.

Задание E26. В окно темно-коричневого цвета выведите график эпициклоиды, уравнение которой в параметрической форме имеет вид $x=(a+b)\cos t - a\cos((a+b)\,t/a)$, $y=(a+b)\sin t - a\sin((a+b)\,t/a)$. Примите $a = 1$, $b = 2$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от $-\pi$ до π с шагом 0.01 и умножайте на 20. График нарисуйте белыми точками, используя метод **`Pixels`** класса **`TCanvas`**.

Задание F26. Выделите в изображении темные области, оставив их цвет; светлые области закрасьте бледно-желтым.

Задание A27. В окно бледно-зеленого цвета выведите фигуру наподобие песочных часов – два узких сектора, соединенные вершинами и нарисованные толстым белым пером. Секторы должны казаться прозрачными. Секторы рисуются вызовом функции **`Pie()`** класса **`TDC`**. Насыпьте на дно песочных часов немного желтого песка (воспользуйтесь для этого функцией **`Chord()`** класса **`TDC`**). Выведите поверх изображения свою фамилию шрифтом Times размером 80 пикселей, узким (шириной 15 пикселей), красного цвета. Надпись расположите горизонтально вверх ногами.

Задание B27. Выведите график функции $f = e^{-t}\sin 5\,t$, изменяя значение t от – 4 до 4 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 20.

Задание C27. Запишите в файл дробное число в формате с десятичной точкой и свою фамилию. Получите в программе куб числа. Выведите в окно приложения прочитанную фамилию и результат вычисления.

Задание D27. В ответ на щелчок правой клавишей мыши при нажатой клавише Shift выведите три круга диаметром 50 пикселей, залитые компонентами цвета исследуемого пикселя. Под ними выведите еще три круга, залитые чистыми цветами максимальной яркости.

Задание E27. В окно цвета **`clOlive`** выведите график конхоиды Никомеда, уравнение которой в параметрической форме имеет вид $x = a + b\cos t$, $y = a\,\mathrm{tg}\,t + b\sin t$. Примите $a = 1$, $b = 3$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от $-\pi/2$ до $3\pi/2$ с шагом 0.01 и умножайте x на 50, а y на 20. Точки графика нарисуйте желтыми ромбиками размером 4-5 пикселей, используя метод **`Polygon`** класса **`TCanvas`**.

Задание F27. Преобразуйте цветное изображение в монохромное желто-коричневое.

Задание A28. В окно коричневого цвета выведите два соприкасающихся круга диаметром 200 пикселей каждый, расположенные по диагонали и залитые зеленым и синим цветами. Выведите поверх изображения свою фамилию шрифтом Courier небольшого размера (30 пикселей), но большой ширины (80 пикселей), жирным, красного цвета. Надпись расположите под углом 30° снизу вверх.

Задание B28. Выведите график многочлена $f = a\,t^3 + b\,t^2 + c\,t + d$ при значениях коэффициентов $a = 0.8$, $b = -3$, $c = 0.6$, $d = 1.5$. Изменяйте значение t от – 3 до 5 и расположите в этом диапазоне 400 точек. Ординаты графика умножайте на 10.

Задание C28. Запишите в файл свой возраст. Получите в программе число прожитых вами месяцев. Выведите в окно приложения свою фамилию и результат вычислений.

Задание D28. В ответ на щелчок правой клавишей мыши при нажатой клавише Ctrl выведите три круга диаметром 50 пикселей, залитые компонентами цвета исследуемого пикселя. В середину кругов выведите значения компонентов цвета.

Задание E28. В окно темно-серого цвета выведите график улитки Паскаля, уравнение которой в параметрической форме имеет вид $x = a\cos^2 t + \cos t$, $y = a\cos t\sin t + \sin t$. Примите $a = -0.7$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от 0 до $2\pi = 6.28$ с шагом 0.01, и умножайте их на 100. График нарисуйте белыми точками, используя метод **`Pixels`** класса **`TCanvas`**.

Задание F28. Выделите в изображении темные области, оставив их цвет; светлые области закрасьте желтым.

Задание A29. В окно светло-голубого цвета выведите три круга, расположенные на одной горизонтальной прямой так, что каждый следующий круг сдвинут относительно предыдущего на величину своего радиуса. Первый круг покрасьте в красный цвет, второй – в белый, а третий – в зеленый. Выведите поверх изображения свою фамилию шрифтом Times размером 80 пикселей, жирным, черного цвета. Надпись расположите под углом 5° сверху вниз.

Задание B29. Выведите график функции $f = \sin t\ \cos 2\,t$. Изменяйте значение t от – 10 до 10, расположив в этом диапазоне 400 точек. Ординаты функции умножайте на 100.

Задание C29. Запишите в файл два числа: дробное в экспоненциальном формате и целое. Получите в программе их сумму. Вы-

ведите в окно приложения исходные числа и результат вычисления с поясняющими надписями.

Задание D29. В ответ на двойной щелчок левой клавишей мыши при нажатой клавише Shift выведите три квадрата размером 50 пикселей, залитые компонентами цвета исследуемого пикселя. Поверх фигур выведите значения компонентов в десятичной форме.

Задание E29. В окно цвета **`clLime`** выведите график кардиоиды, уравнение которой в параметрической форме имеет вид $x = \cos t\ (1+\cos t)$, $y = \sin t\ (1+\cos t)$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от 0 до 2π с шагом 0.05, и умножайте их на 100. Точки графика нарисуйте залитыми черным ромбиками размером 4-5 пикселей, используя метод **`Polygon`** класса **`TCanvas`**.

Задание F29. Преобразуйте цветное изображение в монохромное красное.

Задание A30. В окно желтого цвета выведите неправильный выпуклый пятиугольник темно-синего цвета. Такая фигура рисуется с помощью функции **`Polygon()`** класса **`TDC`**. Выведите поверх изображения свою фамилию шрифтом Arial размером 10 пикселей, красного цвета. Надпись расположите горизонтально "вверх ногами".

Задание B30. Выведите график параболы $f = t^2 + 2\ t$, изменяя значение t от – 4 до 4 и расположив в этом диапазоне 400 точек. Ординаты функции умножайте на 20.

Задание C30. Запишите в файл два дробных числа: первое в экспоненциальном формате, а второе в формате с десятичной точкой. Получите в программе их произведение. Выведите в окно приложения исходные числа и результат вычисления с поясняющими надписями.

Задание D30. В ответ на двойной щелчок левой клавишей мыши при нажатой клавише Ctrl выведите значения трех компонентов цвета исследуемого пикселя в шестнадцатеричной форме, а ниже – условные обозначения компонентов (КЗС).

Задание E30. В окно бледно-красного цвета выведите график фигуры Лиссажу, уравнение которой в параметрической форме имеет вид $x = \sin t$, $y = \cos\ (a\ t+b)$. Примите значение $a = 2$, $b = -\ 0\ .55\pi$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от 0 до 2π с шагом 0.01, и умножайте их на

100. Точки графика нарисуйте прозрачными черными ромбиками размером 4-5 пикселей, используя метод **Polygon** класса **TCanvas**.

Задание F30. Разделите изображение на 3 слоя по яркости, покрасив самый светлый слой в желтый цвет, средний – в красный, а темный – в зеленый.

Задание A31. В окно светло-голубого цвета выведите два равнобедренных треугольника, нарисованные толстым (10-15 пикселей) зеленым пером, расположенные вертикально и соединенные вершинами. Треугольники должны быть прозрачными. Такую фигуру можно нарисовать с помощью одного вызова функции **Polygon ()** класса **TDC**. Выведите поверх изображения свою фамилию шрифтом Times размером 80 пикселей, перечеркнутым, курсивом, красного цвета. Надпись расположите горизонтально.

Задание B31. Выведите график функции арктангенса $f = \mathrm{arctg}\ t$, изменяя значение t от – 10 до 10 и расположив в этом диапазоне 400 точек. Ординаты функции умножайте на 100.

Задание C31. Запишите в файл число в экспоненциальном формате и свою фамилию. Получите в программе корень из числа. Выведите в окно приложения результат вычисления и прочитанную фамилию.

Задание D31. В ответ на двойной щелчок правой клавишей мыши при нажатой клавише Shift выведите три круга, залитые компонентами цвета исследуемого пикселя. Укажите, какие круги соответствуют красному, зеленому и синему компонентам цвета.

Задание E31. В окно черного цвета выведите график эпициклоиды, уравнение которой в параметрической форме имеет вид $x = (a + b) \cos t - a \cos((a + b)\, t / a)$, $y = (a + b) \sin t - a \sin((a + b)\, t / a)$. Примите $a = 1$, $b = 5$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от $-\pi$ до π с шагом 0.01, и умножайте на 20. Точки графика отображайте в виде залитых цветом **clYellow** половинок круга размером 5-6 пикселей, обращенных округлой частью влево. Используйте метод **Pie** класса **TCanvas**.

Задание F31. Покрасьте изображение в фиолетовый цвет, сохранив градации яркости.

Задание A32. Составьте программу, в которой главное окно раскрашено цветами российского флага (широкие полосы белого, синего и красного цветов). Выведите поверх изображения свою

фамилию шрифтом Times размером 80 пикселей, подчеркнутым, жирным, черного цвета. Надпись расположите под углом 20° (снизу вверх).

Задание B32. Выведите график функции $f = e^{-t} \cos(10\,t)$, изменяя значение t от – 2 до 2 и расположив в этом диапазоне 400 точек. Ординаты графика умножайте на 50.

Задание C32. Запишите в файл три дробных числа в экспоненциальном формате. Получите в программе их среднее. Выведите в окно приложения имя выбранного вами файла и результат вычисления.

Задание D32. В ответ на двойной щелчок правой клавишей мыши при нажатой клавише Ctrl выведите значение цвета исследуемого пикселя в шестнадцатеричной и десятичной формах. Выведите также значения отдельных компонентов цвета пикселя.

Задание E32. В окно цвета `clSilver` выведите график конхоиды Никомеда, уравнение которой в параметрической форме имеет вид $x = a + b \cos t$, $y = a \operatorname{tg} t + b \sin t$. Примите $a = 1$, $b = 2$. Значения x и y вычисляйте в цикле по переменной t, принимающей значения от $-\pi/2$ до $3\pi/2$ с шагом 0.01 и умножайте x на 50, а y на 20. Точки графика отображайте символом буквы **z** зеленого цвета.

Задание F32. Преобразуйте цветное изображение в монохромное инверсное.

Список литературы

1. Финогенов К.Г. Лабораторный практикум “Основы программирования на языке C++ ”. – М.: МИФИ, 2004.

2. Финогенов К.Г. Лабораторный практикум “Основы разработки приложений Windows”. Кн. 1-2. М.: – МИФИ, 2004 - 2005.

3. Финогенов К.Г. Основы объектно-ориентированного программирования. Лабораторный практикум. – М.: МИФИ, 2008.