ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра безопасности информационных систем

Вычислительная техника и программирование

Лабораторный практикум

Составитель М.А. Попов

Самара
Издательство «Универс групп»
2007

Печатается по решению Редакционно-издательского совета
Самарского государственного университета

Вычислительная техника и программирование : лабораторный практикум / сост. М.А. Попов. – Самара : Изд-во «Универс групп», 2007. – 60 с.

Лабораторный практикум предназначен для студентов специальностей 090102.65 Компьютерная безопасность и 090103.65 Организация и технология защиты информации. Материалы лабораторного практикума позволяют выполнить 11 лабораторных работ по дисциплине «Вычислительная техника и программирование». К каждой лабораторной работе приведена необходимая теоретическая информация, описание хода выполнения работы и контрольные вопросы для подготовки к защите работы.

СОДЕРЖАНИЕ

Введение

В данном сборнике представлены методические указания для выполнения 11 лабораторных работ по курсу «Вычислительная техника и программирование».

Лабораторные работы рассчитаны на выполнение в компьютерном классе в среде разработки программ *Borland Delphi*. Это позволяет знакомить студентов с современными технологиями программирования и, в то же время, в значительной мере использовать знания языка Паскаль, полученные на первом курсе обучения.

Тематика лабораторных работ выбрана таким образом, чтобы каждая из них была посвящена решению задачи, довольно часто возникающей перед программистом. Выполнив все работы, студент получает набор тех «кирпичиков», из которых, как из конструктора, можно составить самостоятельное приложение.

В отличие от традиционной методики, когда студентам даются небольшие задачи (по вариантам или без таковых), и предлагается самостоятельно их решить, данные методические указания рассчитаны на несколько иную методику обучения. А именно: выдаваемые студентам методические указания содержат не только формулировку задачи, краткие теоретические сведения и контрольные вопросы, но и непосредственно решение задачи. В этом случае основной контроль знаний студентов проводится в форме «защиты» лабораторной работы, т. е. устной беседы студента с преподавателем.

Такая методика исключает ситуацию, когда несколько студентов группы решают задачу не только себе, но и всем остальным.

Лабораторная работа № 1
«Элементы пользовательского интерфейса в программах на языке *Delphi*»

1. Цель работы

Изучение способов создания элементов пользовательского интерфейса программы, таких как меню, диалогового окна выбора файла и др.

2. Теоретическая часть

В данной работе вы создадите программу, окно которой будет содержать в своем составе меню, строку состояния и окно для отображения содержимого выбранного пользователем файла.

Для создания меню используются компоненты *ActionMainMenuBar* и *ActionManager*. Вы познакомитесь с этими компонентами при выполнении работы.

Для отображения содержимого файла будет использоваться компонент *RichEdit*. Данный компонент фактически представляет собой окно для редактирования текста, которое вы размещаете необходимым образом в окне вашей прикладной программы. *RichEdit* предоставляет большое количество разнообразных возможностей форматирования текста, изменения шрифта, его цвета, размера и др. В данной работе мы рассмотрим лишь несколько возможностей, предоставляемых этим компонентом. Оставшиеся вы можете изучить самостоятельно, используя возможности *Помощи*, входящей в состав среды разработки *Delphi*, или изучив соответствующую литературу.

Рассмотрим несколько *Свойств (Properties)* компонента *RichEdit*.

Lines –- текст, содержащийся в окне, с учетом разбиения на строки. Вы можете применять методы *LoadFromFile* и *SaveToFile* для загрузки или сохранения текста (см. ниже).

Text – текст, содержащийся в окне, без учета разбиения на строки.

Enabled – устанавливает, «включено» окно или «выключено»: если данное свойство имеет значение *False*, то пользователь не может прокручивать текст в окне, выделять или изменять его. Однако программа может выводить в это окно текст и производить с ним другие операции.

Font – группа свойств, устанавливающая параметры шрифта в окне *RichEdit*.

Hint – так называемая «подсказка», появляющаяся при наведении курсора мыши на поле *RichEdit* и представляющая собой пояснения, помогающие пользователю понять назначение данного элемента интерфейса вашей программы. Чтобы подсказка отображалась, нужно использовать свойство *ShowHint*.

ShowHint – управление отображением «подсказок». Текст, заданный в свойстве *Hint* отображается, если значение *ShowHint* установлено в *True*.

ReadOnly – разрешение изменения текста пользователем. Если равно *True*, то пользователь не может изменять текст в окне, но может его выделять и прокручивать.

Visible – управление отображением окна ввода текста. Если данной свойство установлено в *False*, то окно не отображается.

Рассмотрим несколько *Методов (Methods)* компонента *RichEdit*.

Clear – полностью очищает поле ввода текста, независимо от наличия или отсутствия выделенного текста.

ClearSelection – удаляет выделенный текст. Если его нет, то ничего не удаляется. Если выделен весь текст в окне, то действует аналогично методу *Clear*.

SelectAll – выделяет весь текст в поле редактирования.

Lines.LoadFromFile(FileName) – метод, позволяющий загрузить текст из файла и отобразить его в окне редактирования.

Lines.SaveToFile (const FileName: string) – метод, позволяющий сохранить текст из окна редактирования в файле.

Для отображения информации, поясняющей работу программы и ее состояния, подсказок для пользователя и прочей дополнительной информации, используется так называемая строка состояния. В программах на *Delphi* используется компонент *StatusBar*. В данной работе вы будете использовать упрощенный порядок работы с данным компонентом.

Рассмотрим свойства компонента *StatusBar*, которые будут использоваться в работе.

SimplePanel – управление видом панели: если *True*, то используется «простая» панель, если *False* – «сложная».

SimpleText – текст, отображаемый в панели состояния, если *SimplePanel* установлено в *True*.

Hint, ShowHint – аналогичны этим же свойствам компонента *RichEdit*.

3. Порядок выполнения работы

Для правильной работы большинства программ (в т. ч. Delphi) под Windows NT необходимо предварительно настроить переменные среды. На рабочем столе найдите значок «Мой компьютер» и кликните по нему правой кнопкой мыши. В меню выберите пункт «Свойства», затем закладку «Переменные среды». Необходимо присвоить переменным Temp и Tmp значение %UserProfile%\Temp.

Внимание! *Не удаляйте символы в фигурных скобках, например, {$R *.res}, находящиеся в шаблоне программы! Это не комментарии, а директивы компилятора, без них ваша программа компилироваться не будет.*

Запустить интегрированную среду разработки программ Delphi. Выполнить команду *File-Close All* для закрытия тех окон проектов, которые были загружены автоматически.

Выполнить команду *File-New-Application.*

Теперь нужно поместить в окне программы строку меню. Для этого в панели компонентов в разделе *Additional* дважды кликнуть мышью на компоненте *ActionMainMenuBar*. Меню будет расположено в верхней части окна вашей программы.

Затем следует найти в панели компонентов в разделе *Additional* компонент *ActionManager* и дважды кликнуть мышью на нем. Его месторасположение на «заготовке» вашей программы значения не имеет.

Следующим этапом создания программы будет размещение поля редактирования *RichEdit*. Его можно найти в разделе *Win32* панели инструментов. По двойному щелчку мыши данный компонент будет помещен в окно вашей программы. Измените его размер и расположение по своему вкусу.

В разделе *Win32* панели компонентов найдите компонент *StatusBar*. Его нужно разместить в нижней части окна вашей программы.

Переходим к созданию меню. Ваша программа будет содержать один пункт меню – *File* и в нем будет два действия: *Open* и *Exit*. Щелкните два раза мышью по компоненту *ActionManager* в окне вашей программы, от-

кроется окно его редактирования. В разделе *Actions* щелкните правой кнопкой мыши в окне *Categories*. В появившемся контекстном меню выберите пункт *New Standard Action…* (так как открытие файла и выход из программы – стандартные действия). В открывшемся окне найдите *TFileOpen* и щелкните ней мышью дважды. В окне редактирования *ActionManager* в окне *Categories* появится строка *File*, а окне *Actions* (находится справа от *Categories*) – пункт (действие) *Open…* Аналогичным образом добавьте в пункт меню *File* еще одно действие – *Exit*.

Теперь нужно поместить только что созданные вами пункт меню и действия непосредственно в строку меню, расположенную в окне вашей программы. Для этого подведите курсор мыши к строке *File* в окне *Categories* и нажмите левую кнопку мыши. Не отпуская ее, «перетащите» пункт на строку меню вашей программы.

Следующим шагом будет задание так называемых фильтров – это возможность для пользователя показывать в окне выбора открываемого файла не все из них, а только файлы определенного типа. Для этого нужно в том же самом окне редактирования *ActionManager* выбрать *Files* в окне *Categories* и затем *Open…* в окне *Actions*. Теперь, не закрывая окна редактирования, перейдите к окну *Object Inspector*. Дважды щелкните мышью на свойстве *Dialog*. Из появившихся свойств выберите *Filter* и нажмите на кнопку с тремя точками справа. В появившемся окне и задаются типы файлов следующим образом: в левом столбце таблицы указывается описание, а в правом – маска файлов, соответствующих данному описанию. Введите в левом столбце описание *Text Files*, а в правом **.txt*. Аналогично заполните следующую строку: *All Files* и **.**. Нажмите кнопку *OK*. Закройте окно редактирования *ActionManager*.

Теперь выберите в окне *Object Inspector* объект *TFileOpen* а затем раздел *Events*. Щелкните мышью дважды на событии *OnAccept*. В окне редактирования текста программы появится заголовок процедуры. Введите следующие строки:

```
RichEdit1.Lines.LoadFromFile(FileOpen1.Dialog.FileName);
StatusBar1.SimpleText := 'Открыт файл: ' + FileOpen1.Dialog.FileName;
```

Основная часть программы готова. Остается ввести текст «подсказок» и задействовать их. Предположим, что при наведении курсора мыши на

окно вывода текста должна появиться подсказка «Здесь отображается содержимое файла». Для этого нужно в окне заготовке программы кликнуть мышью на окне *RichEdit*. Теперь в окне *Object Inspector* найдите свойство (раздел *Properties*) *Hint* и введите текст подсказки. Значение свойства *ShowHint* сделайте равным *True*.

Аналогичным образом снабдите подсказкой строку статуса. Текст подсказки: «Здесь отображается имя открытого файла». Установите значение свойства *SimplePanel* равным *True*.

Теперь откройте текстовый редактор *Блокнот* и создайте текстовый файл. Наберите произвольный текст и сохраните файл с расширением *txt*. Запомните путь к нему.

Запустите свою программу. Откройте созданный вами текстовый файл при помощи меню. Проверьте правильность работы программы.

4. Вопросы для подготовки к защите работы.

1. Опишите своими словами значение понятия *Свойство (Property)* в рамках данной работы.
2. Опишите своими словами значение понятия *Метод (Method)* в рамках данной работы.
3. Перечислите несколько свойств компонента *RichEdit*. Поясните их смысл и назначение.
4. Перечислите несколько методов компонента *RichEdit*. Поясните их смысл и назначение.
5. Перечислите свойства компонента *StatusBar*. Поясните их смысл и назначение.
6. Создать программу с элементами пользовательского интерфейса, указанными преподавателем, без помощи методических указаний. Потренируйтесь самостоятельно.

Лабораторная работа № 2
«Использование процедур рисования в программах на языке *Delphi*»

1. Цель работы

Изучение процедур, позволяющих выводить на экран простейшие графические объекты: линии, окружности и пр.

2. Теоретическая часть

В данной работе вы создадите программу, в окне которой при нажатии кнопки будет выводиться прямоугольник и произвольно расположенные эллипсы.

В операционной системе *Windows* предусмотрено специальное средство для вывода двухмерной графики – *GDI* (*Graphics Device Interface* – интерфейс графического устройства).

GDI является наиболее медленным средством вывода графической информации, поэтому для отображения сложных графических объектов (например, трехмерных) оно практически не используется. Однако, по сравнению с более быстрыми способами, *GDI* имеет неоспоримое преимущество – простоту. Поэтому изучение способов вывода графики целесообразно начать именно с *GDI*.

При отображении графики необходимо, прежде всего, определить то, куда она будет выводиться. Поскольку операционная система *Windows* имеет так называемый «оконный интерфейс», то программа, выводящая на экран графические объекты, не должна, строго говоря, выходить при этом за пределы своего окна, изменяя содержимое окон других программ. В *Delphi* то поле, в котором вы можете рисовать, а также его разнообразные свойства, называется *Canvas*.

Для создания графических объектов на экране в *Delphi* предусмотрено большое число процедур и функций. Следует особо отметить, что они являются не более чем «надстройкой» над аналогичными по назначению функциями *WinAPI*. В данной работе будет рассмотрено несколько из них.

Все функции рисования используют систему координат, которая несколько отличается от привычной. Координаты (0,0) имеет точка в верхнем

левом углу экрана (поля рисования), т. е. координата *Y* возрастает к низу экрана.

Некоторые функции рисования используют заполнение (например, окружности, прямоугольника). Имеется два свойства: *Pen* (*Перо*) и *Brush* (*Кисть*). Первое из них используется при рисовании границ и рамок, а второе – при заполнении фигур. Подробнее вы познакомитесь с этой особенностью в ходе выполнения работы.

FrameRect (Rect: TRect) – процедура, предназначенная для рисования прямоугольника. В качестве параметра передается переменная типа «запись», содержащая координаты вершин прямоугольника. Особенностью данной функции является то, что прямоугольник рисуется без заполнения, однако его границы выводятся цветом, заданным в свойстве *Brush*.

Ellipse (x1, y1, x2, y2: Integer) – процедура для рисования эллипса. В качестве параметров задаются координаты прямоугольника (сам прямоугольник при этом не рисуется), в который данный эллипс можно вписать. Цвет «рамки» эллипса задается свойством *Pen*, цвет заполнения – свойством *Brush*.

FloodFill (x, y: Integer; Color: TColor; FillStile: TFillStyle) – процедура, дающая возможность заполнить («залить») замкнутую область определенным цветом. Параметры:

x, y – координаты точки внутри заполняемой области;

Color – цвет, который будет определять границы заполняемой области.

FillStyle – способ заполнения. Если этот параметр равен *fsBorder*, то заполняться будет область любого цвета, кроме указанного параметром *Color*. В случае *fsSurface* будет заполнена только область с цветом, указанным параметром *Color*.

3. Порядок выполнения работы

Для правильной работы большинства программ (в т. ч. Delphi) под Windows NT необходимо предварительно настроить переменные среды. На рабочем столе найдите значок «Мой компьютер» и кликните по нему правой кнопкой мыши. В меню выберите пункт «Свойства», затем закладку «Переменные среды». Необходимо присвоить переменным Temp и Tmp значение %UserProfile%\Temp.

Внимание! *Не удаляйте символы в фигурных скобках, например, {$R *.res}, находящиеся в шаблоне программы! Это не комментарии, а директивы компилятора, без них ваша программа компилироваться не будет.*

Запустить интегрированную среду разработки программ *Delphi*. Выполнить команду *File-Close All* для закрытия тех окон проектов, которые были загружены автоматически.

Выполнить команду *File-New-Application*.

Затем нужно поместить в окне будущей программы кнопку. Для этого в панели компонентов в разделе *Standard* дважды кликнуть мышью на объекте *Button*. Местоположение и размеры кнопки можно изменить по вашему желанию. В окне *Object Inspector* выбрать объект *Button1*, затем в разделе *Properties* найти свойство *Caption* и присвоить ему значение «Draw», это будет текстом надписи на кнопке. В разделе *Events* найти событие *OnClick* и дважды щелкнуть мышью по нему. В окне исходного текста программы появится заголовок процедуры.

Перед словом *begin* объявите переменные:

```
var
i: Integer;
Rect1: TRect;
```

Затем введите исполняемую часть процедуры:

```
Randomize;
Canvas.Pen.Color := clRed;
Canvas.Brush.Color := clRed;
Rect1 := Rect(100,50,400,300);
Canvas.FrameRect(Rect1);
Canvas.Brush.Color := clWhite;
for i:=0 to 4 do Canvas.Ellipse(100 + 25*i + Random(100), 70 + 25*i + Random(100), 130 + 25*i + Random(100), 90 + 25*i + Random(100));
Canvas.Brush.Color := clRed;
Canvas.FloodFill(101,51,clRed,fsBorder);
```

Для реализации вывода эллипсов случайного расположения и размера используется генератор случайных чисел. Процедура *Randomize* служит

для его инициализации, а функция *Random* позволяет получить случайное число в необходимом диапазоне.

В *Delphi* имеется ряд констант, которые задают основные цвета, например: *clRed*, *clWhite* и т. д. После выполнения работы вы можете попробовать поменять их на другие (*clBlue*, *clYellow*). Обратите внимание, как при этом изменяется результат работы программы.

Запустите свою программу. Проверьте правильность ее работы.

Лабораторная работа № 3
«Использование функций рисования в программах на языке *Delphi* – построение графиков функций»

1. Цель работы

Дальнейшее изучение графических функций и процедур. Знакомство с особенности их использования на примере построения графиков.

2. Теоретическая часть

В данной работе вы создадите программу, в окне которой при нажатии кнопки будет построен график одной из трех функций, приведенных в списке. Также пользователь будет иметь возможность задать максимальное и минимальное значение аргумента.

Само по себе использование стандартных процедур *Delphi* для рисования графиков функций не представляет большой проблемы, однако при решении подобной задачи возникают определенные сложности. Рассмотрим их подробнее.

Все процедуры рисования используют «экранные» координаты, т. е. целые числа, в диапазоне, определяемом размерами окна прикладной программы. Значение функции, график которой нужно вывести на экран, может быть дробным, отрицательным числом самой различной величины, как очень малой, так и большой. Кроме того, значение координаты *Y* на экране растет по направлению к низу экрана, что противоречит традиционному способу построения графиков функций, при котором координата *Y* увеличивается при движении вверх.

Из сказанного выше следует, что для правильного отображения графиков нужно осуществлять масштабирование, т. е. пересчет тех значений *X* и *Y*, которые имели место при вычислении функции непосредственно по формуле, к экранным координатам. Упрощенно можно это объяснить так: нужно «подогнать» размер графика под размеры окна программы. Т. е. точка с максимальной координатой *Y* должна находится в самом верху окна, а точка с минимальной координатой *Y* – в самом низу.

Формулы пересчета могут быть следующими:

$K_x := Width / (x_{max} - x_{min})$,

$K_y := Height / (y_{max} - y_{min})$,

где *Width* – ширина окна прикладной программы, пикселей;

Height – высота этого окна, пикселей;

x_{max}, x_{min} – максимальное и минимальное значение аргумента функции;

y_{max}, y_{min} – максимальное и минимальное значение функции.

Таким образом, если точка имеет координаты (*X*, *Y*), то при выводе на экран ее координаты будут следующими:

$(Round((X–X_{min})*K_x), Height – Round(Y–Y_{min})*K_y))$.

Функция *Round (x: Integer): Integer* позволяет произвести округление числа с плавающей точкой до ближайшего целого. Это необходимо, так как все экранные координаты представляют собой целые числа.

Необходимо обязательно вычитать минимальное значение аргумента или функции из текущей координаты (как это сделано в выражении для экранных координат), так как в общем случае числа *X* и *Y* могут быть меньше нуля, соответственно, и их результат их умножения на коэффициент пересчета будет меньше нуля. Это недопустимо, так как экранная координата – неотрицательное число.

Важно отметить, что минимальные и максимальные значения аргумента станут известны вашей программе лишь на этапе выполнения, когда их введет пользователь.

С минимальным и максимальным значением функции ситуация более сложная. Ясно, что необходимо сначала вычислить все значения функции при заданном диапазоне изменения аргумента и при заданном шаге его изменения, и лишь затем определить максимальное и минимальное значение. В принципе, вычислять значения функции и определять ее максимум и минимум можно в одном цикле.

Алгоритм, применяемый при этом, довольно прост. Объявляется массив с числом элементов, достаточным для построения «плавного» (без изломов) графика функции:

```
y: array [1..1000] of single;
```

Сначала нужно вычислить значение функции для одного из значений аргумента, например, для минимального. Затем нужно присвоить вычисленное значение функции переменным, предназначенным для хранения максимума и минимума функции. Например:

```
y[1] := exp(xmin);
ymin := y[1];
ymax := ymin;
```

Затем нужно в цикле вычислять следующие значения функции, изменяя величину аргумента на значение шага, и записывать их в массив. Одновременно с этим нужно проверять выполнение двух условий:

1) если только что вычисленное значение функции больше того, которое хранится в переменной для максимума, то нужно присвоить его этой переменной;

2) если только что вычисленное значение функции меньше того, которое хранится в переменной для минимума, то нужно присвоить его этой переменной. Например:

```
dx := (xmax–xmin)/1000;
for i:= 2 to 1000 do
begin
y[i] := exp(xmin + i * dx);
if y[i] > ymax then ymax := y[i];
if y[i] < ymin then ymin := y[i];
end;
```

После завершения работы данного цикла максимум и минимум функции для заданного диапазона изменения аргумента гарантированно будут найдены.

В данной работе вы будете использовать две новых для вас процедуры рисования. Рассмотрим их.

MoveTo (x, y: Integer) – процедура предназначена для перемещения так называемого «указателя» в точку с координатами (x, y).

Применительно к графическим процедурам, термин «указатель» обозначает текущую точку рисования. Таким образом, данная процедура просто «перемещает» эту точку в нужное нам место. Самого рисования при этом не происходит.

LineTo (x, y: Integer) – процедура рисует линию, началом которой служит данная текущая точка рисования («указатель»), а концом – точка с координатами (x, y). После вывода линии данная функция устанавливает «указатель» в точку (x, y).

Данные функции обычно используются для рисования последовательности линий, при этом началом следующей линии служит конец предыдущей. Этот случай как раз и имеет место при построении графиков.

3. Порядок выполнения работы

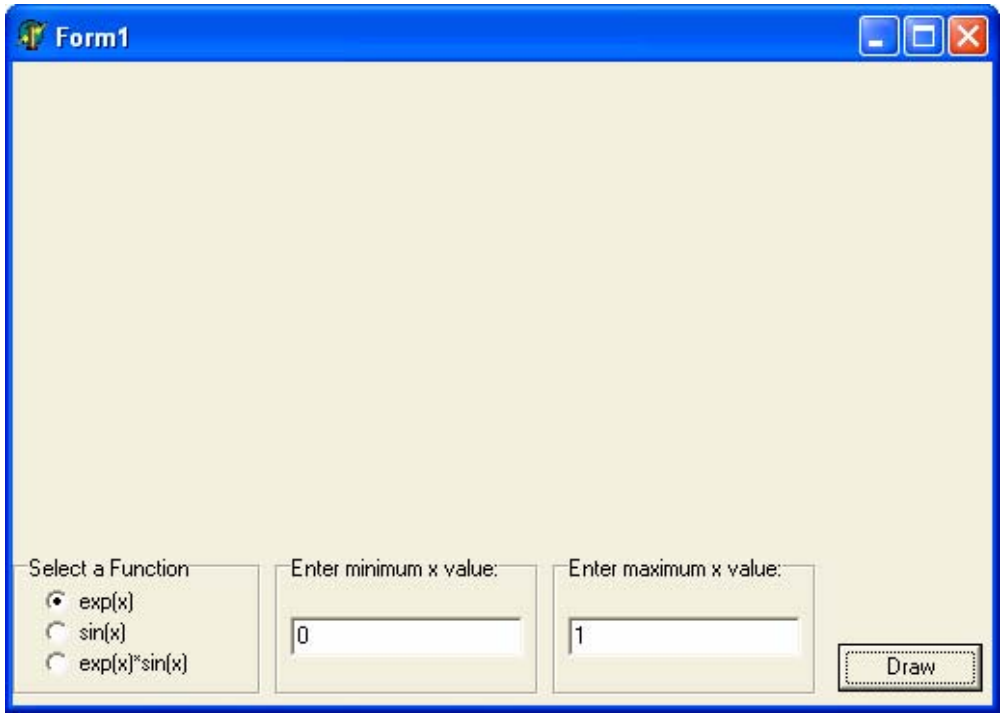
Для правильной работы большинства программ (в т. ч. Delphi) под Windows NT необходимо предварительно настроить переменные среды. На рабочем столе найдите значок «Мой компьютер» и кликните по нему правой кнопкой мыши. В меню выберите пункт «Свойства», затем закладку «Переменные среды». Необходимо присвоить переменным Temp и Tmp значение %UserProfile%\Temp.

Внимание! *Не удаляйте символы в фигурных скобках, например, {$R *.res}, находящиеся в шаблоне программы! Это не комментарии, а директивы компилятора, без них ваша программа компилироваться не будет.*

Запустить интегрированную среду разработки программ *Delphi*. Выполнить команду *File-Close All* для закрытия тех окон проектов, которые были загружены автоматически.

Выполнить команду *File-New-Application*.

Вначале необходимо создать элементы интерфейса программы. Ее окно должно иметь вид, представленный на рисунке.



Здесь используются несколько новых для вас компонентов. Первый из них – так называемая «радиокнопка» (*RadioButton*). В данной программе их будет три, как это видно на рисунке (рядом с радиокнопками находятся названия функций). Радиокнопки используются для выбора взаимоисключающих опций. Т. е. лишь одна радиокнопка из группы может быть отмечена (*Checked*), но не две или более.

Еще один компонент – *Группа Радиокнопок* (*RadioGroup*). Он позволяет, во-первых, отобразить границу группы и показать заголовок (например, «Select a Function», как в данной программе). Во-вторых, можно создать несколько независимых групп радиокнопок (в данной работе эта возможность не используется).

Вам нужно поместить компонент *RadioGroup* так, как показано на рисунке. Обратившись к окну *Object Inspector*, задайте размеры области, ограничивающей группу: *Width* = 129, *Height* = 73. Это важно в данной работе, так как определяет размер области для построения графиков. Свойству *Caption* присвойте значение «Select a Function».

Теперь поместите внутрь группы три радиокнопки как показано на рисунке. Установите подписи к ним при помощи свойства *Caption*. Сделайте функцию *exp(x)* выбранной по умолчанию. Для этого свойство *Checked* кнопки *RadioButton1* сделайте равным *True*.

Теперь нужно создать поля для ввода максимальных и минимальных значений аргумента. Для этого используются компоненты *GroupBox* (они внешне аналогичны *RadioGroup*, однако могут объединять не только радиокнопки). Они должны иметь размер 137x73 точек и размещение, показанное на рисунке. Введите их заголовки как уже было описано выше.

Поместите поля ввода текста (компонент *Edit*) как это показано на рисунке. Установите (при помощи свойства *Text*) минимальное и максимальное значение аргумента, которое будет использовано по умолчанию, например, «0» и «1».

Затем нужно поместить в окне будущей программы кнопку (объект *Button*) как показано на рисунке. В окне *Object Inspector* выбрать объект *Button1*, затем в разделе *Properties* найти свойство *Caption* и присвоить ему значение «*Draw*», это будет текстом надписи на кнопке. В разделе *Events* найти событие *OnClick* и дважды щелкнуть мышью по нему. В окне исходного текста программы появится заголовок процедуры.

Перед словом *begin* введите:

```
procedure Calculate;

begin
dx := (xmax-xmin)/1000;
if (RadioButton1.Checked = true) then
begin
y[1] := exp(xmin);
ymin := y[1];
ymax := ymin;
for i:= 2 to 1000 do
begin
y[i] := exp(xmin + i * dx);
if y[i] > ymax then ymax := y[i];
if y[i] < ymin then ymin := y[i];
end;
end;
if (RadioButton2.Checked = true) then
begin
y[1] := sin(xmin);
ymin := y[1];
ymax := ymin;
for i:= 2 to 1000 do
begin
y[i] := sin(xmin + i * dx);
if y[i] > ymax then ymax := y[i];
if y[i] < ymin then ymin := y[i];
end;
end;
if (RadioButton3.Checked = true) then
begin
y[1] :=  sin(xmin) * exp (xmin);
ymin := y[1];
ymax := ymin;
for i:= 2 to 1000 do
begin
y[i] := sin(xmin + i * dx) * exp(xmin + i * dx);
```

```
if y[i] > ymax then ymax := y[i];
if y[i] < ymin then ymin := y[i];
end;
end;
kx := Width / (xmax – xmin);
ky := (Height – 115) / (ymax – ymin);
end;
```

Это процедура, «вложенная» в обработчик нажатия кнопки. Она будет использована в следующем фрагменте кода.

Затем введите исполняемую часть процедуры *Button1Click*:

```
xmin := StrToFloat(Edit1.Text);
xmax := StrToFloat(Edit2.Text);
Canvas.Pen.Color := clRed;
Canvas.Brush.color := clWhite;
Canvas.Rectangle(0, 0, Width, Height – 113);
Calculate;
Canvas.MoveTo(0, Height – 115 – Round((y[1] – ymin) * ky));
for i := 2 to 1000 do Canvas.LineTo(Round(dx * i * kx), Height – 115 –
Round((y[i] – ymin) * ky));
```

В разделе объявления переменных основной программы введите:

```
x, kx, ky, xmin, xmax, ymin, ymax, dx: single;
y: array [1..1000] of single;
i: Integer;
```

Запустите программу. Задавая различные диапазоны изменения аргумента и выбирая различные функции, проверьте правильность ее работы.

4. Контрольные вопросы для подготовки к защите работы

1. Назовите основные сложности, имеющие место при построении графиков функций.

2. Напишите формулы пересчета координат, полученных при вычислении значений функции, к экранным координатам. Поясните, почему они имеют такой вид.

3. Поясните выражение *(Round((X-X_{min})*K_x), Height – Round(Y-Y_{min})*K_y))*.

4. Зачем нужно знать максимальное и минимальное значение аргумента и функции? Каким образом программа получает эти значения? Пояснить алгоритм нахождения минимума и максимума функции.

5. Пояснить назначение всех переменных, объявленных в разделе *Var* (кроме *Form1*).

6. Почему начальное значение счетчика цикла для нахождения максимума и минимума (см. страницу 4) берется равным *2*, а не *1*?

7. Расскажите о функциях *MoveTo* и *LineTo*. Поясните их использование в данной работе.

8. График функции принято считать набором большого (теоретически – бесконечного) числа точек. Почему при построении графиков в данной работе вы нигде не пользовались функцией рисования точки?

9. Почему в некоторых выражениях, например:

ky := (Height – 115) / (ymax – ymin);

*Canvas.MoveTo(0, Height – 115 – Round((y[1] – ymin) * ky));*

появляется число 115? На что оно влияет?

Лабораторная работа № 4
«Обмен сообщениями между программами в *Windows*»

1. Цель работы

Изучение одного из вариантов организации обмена сообщениями между программами в среде *Windows*.

В ходе работы создаются две программы, одна из которых отправляет сообщение другой, которая, в свою очередь, отображает его на экране.

2. Теоретическая часть

В данной работе рассматривается метод, основанный на использовании функции *SendMessage*. Рассмотрим ее назначение и параметры.

SendMessage(HWnd: Integer; Msg: Integer; WParam: THandle; LParam: Pointer): LResult;

Эта функция отправляет сообщение одному или нескольким окнам. Она имеет следующие параметры:

HWnd – идентификатор окна того приложения, которому предназначено сообщение. Для определения этого идентификатора используется функция *FindWindow* (см. ниже).

Msg – тип отправляемого сообщения. В данной работе отправляется сообщение типа *WM_COPYDATA*.

WParam – дополнительный параметр. Его назначение зависит от типа отправляемого сообщения. При пересылке сообщения типа *WM_COPYDATA*, данный параметр представляет собой дескриптор окна-отправителя.

LParam – еще один дополнительный параметр. В данной работе он используется для передачи указателя на структуру данных сообщения.

Для определения идентификатора окна приложения используется функция *FindWindow*.

function FindWindow (ClassName: PChar; WindowName: PChar): Integer;

ClassName – имя класса, к которому принадлежит разыскиваемое окно. Если это не имеет значения (как в данной работе), указывается значение *nil*.

WindowName – название разыскиваемого окна, т. е. его «заголовок».

Функция возвращает 0, если окно найти не удалось. В противном случае возвращается его идентификатор.

Рассматриваемый способ обмена сообщениями предусматривает передачу указателя на структуру данных сообщения (см. описание функции *SendMessage*). Данная структура должна иметь определенный вид:

```
type
   TCopyDataStruct = record
   dwData: LongInt;
   cbData: LongInt;
   lpData: Pointer;
end;
```

Как видно, структура представляет собой запись, состоящую из трех элементов:

dwData – 32 бита, которые могут быть использованы программистом произвольно, для передачи необходимой информации.

cbData – элемент, показывающий размер (в байтах) той информации, на которую указывает указатель *lpData*.

Помимо этого, необходимо определить, в каком формате будут передаваться данные в сообщении. Для этой цели делается еще одно объявление типа:

```
type
   TRecToPass =  record
   s : string[255];
   i : integer;
end;
```

Этот тип также представляет собой запись, но состоящую из двух элементов:

s – строка, которая будет передаваться в составе сообщения;

i – целое число, также подлежащее передаче.

Следует особо отметить, что такой формат взят для примера, он может быть другим, если это необходимо программисту.

3. Порядок выполнения работы

Внимание! *Не удаляйте символы в фигурных скобках, например, {$R *.res}, находящиеся в шаблоне программы! Это не комментарии, а директивы компилятора, без них ваша программа компилироваться не будет.*

Запустить интегрированную среду разработки программ *Delphi*. Выполнить команду *File-Close All* для закрытия тех окон проектов, которые были загружены автоматически.

Выполнить команду *File-New-Application.* Это будет первая программа – отправитель сообщения.

В разделе *Interface* нужно объявить константу:

```
const WM_COPYDATA = $004A;
```

Затем нужно объявить типы, которые будут использоваться при отправке сообщения:

```
type
   TCopyDataStruct = record
   dwData: LongInt;
   cbData: LongInt;
   lpData: Pointer;
 end;

type
  TRecToPass = record
   s : string[255];
   i : integer;
 end;
```

Теперь нужно присвоить заголовку окна программы значение *Application 1*. Для этого в окне *Object Inspector* следует выбрать объект *Form1*, затем в разделе *Properties* найти свойство *Caption* и присвоить ему указанное значение.

Затем нужно поместить в окне будущей программы кнопку. Для этого в панели компонентов в разделе *Standard* дважды кликнуть мышью на объекте *Button*. Местоположение и размеры кнопки можно изменить по вашему желанию. В окне *Object Inspector* выбрать объект *Button1*, затем в раз-

деле *Properties* найти свойство *Caption* и присвоить ему значение «Отправить!», это будет текстом надписи на кнопке. В разделе *Events* найти событие *OnClick* и поставить ему в соответствие процедуру *Button1Click*.

В окне редактирования текста программы появится заголовок процедуры. Введите следующий текст:

```
var
  h : integer{THandle};
  cd : TCopyDataStruct;
  rec : TRecToPass;
begin
  h := FindWindow(nil, 'Application 2');
  rec.s := 'Message from Application 1';
  rec.i := 32;
  cd.dwData := 3232;
  cd.cbData := sizeof(rec);
  cd.lpData := @rec;
  if h <> 0 then
SendMessage(h, WM_COPYDATA, Form1.Handle, LongInt(@cd));
end;
```

Сохраните программу в отдельном каталоге, выбрав пункт *File--Save Project As*. В качестве каталога укажите *C:\WinNT\Profiles\(Ваш логин) \Личная\Application 1*.

Откомпилируйте программу, нажав *Ctrl-F9*.

Выполнить команду *File-Close All* для закрытия окна написанной вами программы.

Выполнить команду *File-New-Application.* Это будет вторая программа – получатель сообщения.

Теперь нужно присвоить заголовку окна программы значение *Application 2*. Для этого в окне *Object Inspector* следует выбрать объект *Form1*, затем в разделе *Properties* найти свойство *Caption* и присвоить ему указанное значение.

Затем нужно поместить в окне будущей программы поле *Memo*. Для этого в панели компонентов в разделе *Standard* дважды кликнуть мышью на объекте *Memo.*

В разделе *Interface* нужно объявить типы, которые будут использоваться при приеме сообщения:

```
type
PCopyDataStruct = ^TCopyDataStruct;
TCopyDataStruct = record
   dwData: LongInt;
   cbData: LongInt;
   lpData: Pointer;
 end;

type
 PRecToPass = ^TRecToPass;
 TRecToPass =  record
   s : string[255];
   i : integer;
 end;
```

В разделе *Private* нужно поместить:

```
procedure WMCopyData(var m: TMessage); message WM_COPYDATA;
```

Введите текст процедуры (в разделе *Implementation*):

```
procedure TForm1.WMCopyData(var m : TMessage);
begin
Memo1.Lines.Add('TRecToPass.s := ' +
PRecToPass(PCopyDataStruct(m.LParam)^.lpData)^.s);
Memo1.Lines.Add('TRecToPass.i := ' +
IntToStr(PRecToPass(PCopyDataStruct(m.LParam)^.lpData)^.i));
end;
```

В данном фрагменте активно используются указатели, поэтому разберем одну из его строк:

```
Memo1.Lines.Add('TRecToPass.s := ' +
PRecToPass(PCopyDataStruct(m.LParam)^.lpData)^.s);
```

Эта запись означает, что нужно вывести в поле *Memo1*, размещенное в окне приложения, строку, которая содержалась в переданном сообщении. Это делается при помощи метода *Memo1.Lines.Add*, являющегося процедурой, параметром которой и является строка, предназначенная для отображения.

Однако эту строку нужно еще извлечь. При определении типов в данной программе были объявлены указатели: *PrecToPass* и *PCopyDataStruct*.

«Расшифровывать» запись нужно с середины. Запись *m.LParam* означает, что нужно получить указатель на структуру данных сообщения (см. описание функции *SendMessage*). Запись *PCopyDataStruct(m.LParam)^.lpData* означает обращение к структуре передаваемой информации (см. описание типа *TCopyDataStruct*).

Наконец, запись *PRecToPass(PCopyDataStruct(m.LParam)^.lpData)^.s* дает возможность обратиться к элементу *s* записи типа *TrecToPass* (см. выше), который и представляет собой передаваемую в сообщении строку.

Сохраните программу в отдельном каталоге, выбрав пункт *File.Save-Project As*. В качестве каталога укажите *C:\WinNT\Profiles\(Ваш логин)\Личная\Application 2*. Запустите программу, нажав *F9*.

Затем перейдите в каталог (при помощи Проводника, например), в который вы сохранили первую программу. Найдите исполняемый файл, он должен называться *Project1.exe*. Запустите его. Нажмите на кнопку «Отправить!». Переключитесь на вторую программу (ее вы запустили ранее из *Delphi*). В поле *Memo* должно появиться сообщение, которое прислала первая программа – отправитель.

4. Вопросы для подготовки к защите работы.

1. Поясните суть рассматриваемого в работе метода обмена сообщениями.
2. Функция *SendMessage*. Назначение и параметры.
3. Функция *FindWindow*. Назначение и параметры.
4. В программах использовалось несколько типов записей. Поясните назначение каждой из них.
5. Какую операцию выполняет фрагмент:

Memo1.Lines.Add('TRecToPass.s := ' +
PRecToPass(PCopyDataStruct(m.LParam)^.lpData)^.s).

Объясните смысл данного выражения.

6. Какую операцию выполняет фрагмент:

Memo1.Lines.Add('TRecToPass.i := ' +
IntToStr(PRecToPass(PCopyDataStruct(m.LParam)^.lpData)^.i)).

Объясните смысл данного выражения.

7. Пользуясь исходным текстом программы-отправителя и программы-получателя сообщения, поясните принцип их работы и назначение основных операторов, используемых в них.

Лабораторная работа № 5
«Статическая загрузка *DLL*»

1. Цель работы

Требуется составить программу на языке *Delphi*, использующую статическую загрузку *DLL*.

При этом в загружаемой *DLL* должна находиться функция, которая будет использоваться основной программой.

2. Краткая теория

Обозначение *DLL* образовано из словосочетания *Dynamic Load Library*, что в переводе означает «динамически загружаемая библиотека». Таким образом, *DLL* загружаются в оперативную память по мере возникновения необходимости использования содержимого этих библиотек: процедур, функций и прочих ресурсов

Итак, зачем же нужны библиотеки *DLL* и где они используются? Перечислим лишь некоторые из областей их применения:

- **Отдельные библиотеки**, содержащие полезные для программистов дополнительные функции. Например, функции для работы со строками, или же – специальные библиотеки для преобразования изображений;
- **Хранилища ресурсов**. В *DLL* можно хранить не только программы и функции, но и всевозможные ресурсы – иконки, рисунки, строковые массивы, меню, и т.д.;
- **Библиотеки поддержки**. В качестве примера можно привести библиотеки таких известных пакетов, как: *DirectX*, *OpenGL* и т.д.;
- **Части программы**. Например, в *DLL* можно хранить окна программы (формы), и т.п.;
- **Плагины** (Plugins) – дополнения к программе, расширяющие ее возможности.
- **Разделяемый ресурс**. *DLL* может быть использована сразу несколькими программами или процессами. Такое использование *DLL* широко распространено в среде *Windows*.

В данной работе рассматривается статический метод загрузки *DLL* в память компьютера. Это наиболее простой метод для использования функций, импортируемых из *DLL*. При его применении *DLL* загружается пол-

ностью в начале работы основной программы и не может быть «выгружена» в дальнейшем. Однако этот способ имеет очень весомый недостаток – если библиотека, которую использует программа, не будет найдена, то программа просто не запустится, выдавая ошибку и сообщая о том, что ресурс *DLL* не найден.

Создаваемая в этой работе *DLL* будет содержать функцию вычисления значения числа *e* с заданной точностью.

Алгоритм работы функции основан на формуле:

$$e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n .$$

3. Порядок выполнения работы

Запустить интегрированную среду разработки программ Delphi. Выполнить команду *File-Close All* для закрытия тех окон проектов, которые были загружены автоматически.

Выполнить команду *File-New-Other.* Затем в появившемся окне выбрать *DLL Wizard*. Создается заготовка *DLL*.

Введите текст:

```
library mydll;

uses
  SysUtils,
  Classes;

function GetE (eps: extended): extended; stdcall;

var
e, t: extended;
i: integer;

begin

e:=1;
t:=1;
i:=1;
```

```
repeat
t:=t/i;
e:=e+t;
inc(i);
until abs(t)<eps;
Result := e;

end;

exports GetE;

begin
end.
```

Компилировать *DLL*, нажав *Ctrl-F9*. Выполнить команду *File-Close*.

Выполнить команду *File-New-Application.* Это будет основная программа.

Затем нужно поместить в окне будущей программы кнопку. Для этого в панели компонентов в разделе *Standard* дважды кликнуть мышью на объекте *Button*. Местоположение и размеры кнопки можно изменить по вашему желанию. В окне *Object Inspector* выбрать объект *Button1*, затем в закладке *Properties* найти свойство *Caption* и присвоить ему значение «Вычислить!», это текст надписи на кнопке. Дважды кликнуть левой кнопкой мыши по кнопке *Button1* в окне создаваемой программы.

В окне редактирования текста программы появится заголовок процедуры. Введите следующую строку:

```
ShowMessage(FloatToStr(GetE(0. 1))).
```

Число 0.1 определяет точность вычисления значения числа *e*.

В разделе *Implementation* необходимо объявить функцию, импортируемую из *DLL*:

```
function GetE(eps: extended): extended; stdcall; external 'MYDLL.DLL';
```

Запустите программу на выполнение, нажав клавишу *F9*. Задавая вместо 0.1 другие величины (например, 0.0001, 0.000001 и т.д.), проверьте правильность работы программы.

Лабораторная работа № 6
«Динамическая загрузка *DLL*»

1. Цель работы

Требуется составить программу на языке *Delphi*, использующую динамическую загрузку *DLL*.

При этом в загружаемой *DLL* должна находиться функция, которая будет использоваться основной программой.

2. Теоретическая часть

В данной работе рассматривается динамический метод загрузки *DLL* в память компьютера. Этот метод сложнее статического, но и более функциональный. По сути, только динамическая загрузка позволяет реализовать все преимущества, свойственные *DLL*.

При использовании этого метода *DLL* загружаются в оперативную память не при старте основной программы, а по мере необходимости. Как только библиотека больше не нужна, ее можно «выгрузить» из памяти.

Недостаток динамического метода – объем кода, необходимый для его реализации. Кроме того, функция или процедура, импортируемая из *DLL*, доступна лишь тогда, когда эта *DLL* загружена в память.

В данной работе будут использоваться несколько функций *WinAPI* (*Windows Application Program Interface* – набор стандартных процедур и функций, предоставляемых операционной системой *Windows* для работы прикладных программ). Рассмотрим эти функции.

LoadLibrary (LibFileName: PChar) – загрузка указанной библиотеки *LibFileName* в память. При успешном завершении функция возвращает дескриптор (*THandle*) *DLL* в памяти. Тип *PChar* обозначает так называемую «строку, заканчивающуюся нулем». Т. е. в памяти по адресу, указываемому переменной типа *PChar*, находится последовательность символов, причем ее конец обозначается символом с кодом 0. В *Delphi* есть еще один строковый тип – *String*. Его отличие в том, что помимо завершения строки символом с кодом 0, имеется еще и счетчик числа символов в строке.

При загрузке *DLL* могут возникать ошибки (библиотека не найдена, не хватает памяти и др.), функция *LoadLibrary* в таких случаях возвращает значение *nil*.

GetProcAddress (Module: THandle; ProcName: PChar) – считывает адрес экспортированной библиотечной функции. При успешном завершении функция возвращает дескриптор (*TFarProc*) функции в загруженной *DLL*. В случае возникновения ошибки функция возвращает результат *nil*. Это означает, что указатель недействителен («никуда не указывает»). Использовать число 0 для этой цели нельзя, так как нулевой адрес существует.

FreeLibrary(LibModule: THandle) – делает недействительным *LibModule* и освобождает связанную с ним память. Следует заметить, что после вызова этой процедуры функции данной библиотеки больше недоступны до тех пор, пока она вновь не будет загружена.

Создаваемая в этой работе *DLL* будет содержать функцию вычисления значения числа *e* с заданной точностью, аналогичную той, которая использовалась в предыдущей работе.

3. Порядок выполнения работы

Внимание! *Не удаляйте символы в фигурных скобках, например, {$R *.res}, находящиеся в шаблоне программы! Это не комментарии, а директивы компилятора, без них ваша программа компилироваться не будет.*

Запустить интегрированную среду разработки программ *Delphi*. Выполнить команду *File-Close All* для закрытия тех окон проектов, которые были загружены автоматически.

В данной работе используется та же библиотека *MYDLL.DLL*, что и в предыдущей работе. Поэтому если вы сохранили ее, то заново создавать эту библиотеку не нужно, а следует сразу же перейти к написанию основной программы (см. ниже).

Выполнить команду *File-New-Other.* Затем в появившемся окне выбрать *DLL Wizard*. Создается заготовка *DLL*.

Введите текст:

```
library mydll;

uses
  SysUtils,
  Classes;

function GetE (eps: extended): extended; stdcall;
```

```
var
e, t: extended;
i: integer;

begin

e:=1;
t:=1;
i:=1;

repeat
   t:=t/i;
   e:=e+t;
   inc(i);
until abs(t)<eps;
Result := e;

end;

exports GetE;

begin
end.
```

Компилировать *DLL*, нажав *Ctrl-F9*. Выполнить команду *File-Close*.

Выполнить команду *File-New-Application.* Это будет основная программа.

В разделе *Var* нужно объявить:

```
GetE: function (eps: extended): extended;LibHandle: THandle;
```

Теперь нужно поместить в окне основной программы поле для ввода текста. Оно будет использовано для задания пользователем точности вычисления числа *e*. Для этого в панели компонентов в разделе *Standard* дважды кликнуть мышью на объекте *Edit*. В закладке *Properties* (окно *Object Inspector*) следует найти свойство *Text* и сделать его пустым.

Затем нужно поместить в окне будущей программы кнопку. Для этого в панели компонентов в разделе *Standard* дважды кликнуть мышью на объекте *Button*. Местоположение и размеры кнопки можно изменить по ваше-

му желанию. В окне *Object Inspector* выбрать объект *Button1*, затем в разделе *Properties* найти свойство *Caption* и присвоить ему значение «Вычислить», это будет текстом надписи на кнопке. Дважды кликнуть левой кнопкой мыши по кнопке *Button1* в окне создаваемой программы.

В окне редактирования текста программы появится заголовок процедуры. Введите следующий текст:

```
@GetE := nil;
LibHandle := LoadLibrary ('MYDLL.DLL');
if LibHandle <> nil  then begin
 @GetE := GetProcAddress (LibHandle,'GetE');
 if @GetE <> nil then
   ShowMessage (FloatToStr (GetE (StrToFloat (Edit1.Text) ) ) );
end;
FreeLibrary(LibHandle);
```

Запустите программу на выполнение, нажав клавишу *F9*. Задавая различные величины точности (например, 0.0001, 0.000001 и т.д.), проверьте правильность работы программы.

Возможен случай, что программа запускается, но при нажатии на кнопку «Вычислить» ничего не происходит. Следует разместить файл *mydll.dll* в том же каталоге, что и ваш проект. Если программа все же не работает, нужно при вызове функции *LoadLibrary* в тексте программы указывать имя файла библиотеки с полным путем, например:

```
LibHandle := LoadLibrary('D:\Borland\Delphi7\Projects\MYDLL.DLL');
```

4. Вопросы для подготовки к защите работы.

1. Что такое *DLL*? Как расшифровывается это сокращение?
2. Какие преимущества дает использование *DLL*?
3. Каковы особенности динамического метода загрузки *DLL*?
4. В чем состоят преимущества и недостатки динамического метода по сравнению со статическим?
5. Правила языка *Delphi* требуют, чтобы любая функция перед использованием была объявлена. Каким образом объявляется функция, вызываемая из *DLL* при использовании динамического метода? Почему требуется именно такое объявление?

6. Какие действия необходимо выполнить перед тем, как функция, находящаяся в *DLL*, станет доступна для вызова?

7. Что такое *WinAPI*?

8. Опишите назначение, входные параметры и тип возвращаемого результата функции *LoadLibrary*.

9. Опишите назначение, входные параметры и тип возвращаемого результата функции *GetProcAddress*.

10. Опишите назначение, входные параметры и тип возвращаемого результата функции *FreeLibrary*.

11. Что обозначает символ @, стоящий перед именем переменной, процедуры или функции?

12. При написании программы вы использовали зарезервированное слово *nil*. Что оно обозначает и в чем смысл его использования?

Лабораторная работа № 7
«Определение доменного имени по *IP*-адресу»

1. Цель работы

Изучение способа определения доменного имени по уникальному адресу компьютера в сети Интернет. Знакомство с функциями, процедурами и структурами данных, используемыми при работе с сетью.

2. Теоретическая часть

Для правильного понимания принципов создания программ, использующих взаимодействие с Интернетом, необходимо вначале рассмотреть некоторые общие вопросы, касающиеся связи компьютеров в этой сети.

В 1980 гг. ученые одного из университетов Калифорнии разработали концепцию передачи информации по сети от одного компьютера к другому посредством универсальной модели передачи данных, не зависящей от типа платформы и программного обеспечения, установленного на ней. Предложенная ими модель оказалась настолько удачной, что практически все функции, описанные в ней, используются и в последующих реализациях, одной из которых является *WinSocket*. Сокет – абстрактное программное понятие, используемое для обозначения в прикладной программе конечной точки канала связи с коммуникационной средой, образованной компьютерной сетью.

WinSocket, поставляемый со всеми последними версиями *Microsoft Windows*, представляет собой набор функций, оформленных в виде динамически загружаемой библиотеки (*DLL*), и предназначен для осуществления сетевого взаимодействия между компьютерами. Благодаря этому становится возможным применение его встроенных средств в приложениях, создаваемых независимыми разработчиками с помощью любых языков программирования, при условии, что в них имеются встроенные средства для использования функций из внешних динамических библиотек. Такими инструментами могут быть, например, *Borland Delphi*, *Microsoft Visual Basic* или различные реализации *C/C++*.

Создатели *WinSocket* сделали его универсальным средством для построения сетевых приложений: задуманный как интерфейс прикладного программирования для сетей *TCP/IP*, он, тем не менее, может работать с

любыми сетевыми протоколами. Чтобы обеспечить совместимость разрабатываемого программного обеспечения, а также избежать ошибок при попытке вызова функций, отсутствующих в устаревших версиях *WinSocket*, в состав библиотеки были введены две функции – *WSAStartup* и *WSACleanup*. Функция *WSAStartup*, кроме непосредственно инициализации библиотеки, позволяет узнать версию *WinSocket*, установленную на данном компьютере, и вызывается до любой другой функции из числа входящих в состав библиотеки. Вторая функция – *WSACleanup* – должна вызываться последней из функций сокета, например по окончании работы программы. Эта функция очищает программные буферы и переменные, которые были задействованы сокетом. Обе функции могут вызываться в программе неоднократно, но на каждый вызов *WSAStartup* должен приходиться вызов *WSACleanup*.

Function WSAStartup (VersionRequired: Word; var WSData: TWSAData): Integer;

VersionRequired – используется для указания требуемой версии *WinSocket*; в настоящее время используются версии 1.1 (указывается как $101) и 2.0 (указывается как $200);

WSData – структура данных, предопределенная в Delphi следующим образом:

```
TWSAData = record
   wVersion: Word;
   wHighVersion: Word;
   szDescription: array[0..WSADESCRIPTION_LEN] of Char;
   szSystemStatus: array[0..WSASYS_STATUS_LEN] of Char;
   iMaxSockets: Word;
   iMaxUdpDg: Word;
   lpVendorInfo: PChar;
  end;
```

Содержит информацию о параметрах текущей версии *WinSocket*. Ее детальное рассмотрение выходит за рамки данной работы. Обычно не требуется обращаться к этой структуре, она лишь заполняется в результате работы функции *WSAStartup* и не требует вашего дальнейшего участия.

Если инициализация *WinSocket* прошла успешно, функция *WSAStartup* возвращает 0. Любое другое значение свидетельствует об ошибке. Например, имеющаяся на компьютере версия *WinSocket* может быть младше той, которую вы указали параметром *VersionRequired.*

Function WSACleanup: Integer;

Функция не имеет *WSACleanup* параметров. В случае успешного завершения возвращает *0*. Любое другое значение свидетельствует об ошибке.

Теперь кратко рассмотрим информацию, касающуюся так называемых IP-адресов.

Согласно спецификации протокола *TCP/IP*, каждому узлу, подсоединенному к сети, присваивается уникальный номер. Узел может представлять собой компьютер или какое-либо служебное устройство, обеспечивающее функционирование сети. Если один узел имеет несколько физических подключений к сети, то каждому подключению должен быть присвоен свой уникальный номер.

Этот номер, или, по-другому, *IP*-адрес, имеет длину в четыре октета, и состоит из двух частей. Первая часть определяет сеть, к которой принадлежит узел, а вторая – уникальный адрес самого узла внутри сети.

Как известно, *IP*-адрес записывается в виде четырех чисел от 0 до 255, разделенных точкой, например, 213.156.189.012.

Функция *Inet_Addr* предназначена для преобразования такой записи IP-адреса в форму 32-битного числа, пригодного для использования, в частности, для работы с *WinSocket*.

Function Inet_Addr (cp: PChar): DWord;

В качестве параметра в функцию передается строка, содержащая IP-адрес виде четырех октетов, в формате *PChar*. Функция возвращает результат в виде двойного слова.

Для получения информации о сервере (хосте) по его *IP*-адресу используется функция *GetHostByAddr*.

Function GetHostByAddr (Addr: Pointer; Len, Struct: Integer): PHostEnt;

Addr – указатель на сетевой адрес;
Len – длина адреса в байтах;

Struct – тип адреса; для Интернета необходимо указать *AF_INET*.

Результат, возвращаемый данной функцией, имеет тип *PHostEnt*. Данный тип предопределен в *Delphi* следующим образом:

```
type
  PHostEnt = ^THostEnt;
    THostEnt = record
    h_name: PChar;
    h_aliases: ^PChar;
    h_addrtype: SmallInt;
    h_length: SmallInt;
    h_addr: ^PChar;
  end;
```

Как видно, данный тип представляет собой запись. В данной работе мы будем использовать только поле *h_name* – имя домена, которое и требуется узнать.

Кроме того, в данной работе вы будете тип *TSockAddrIn*, также заранее определенный в *Delphi*.

```
TSockAddrIn = record
    sin_family: Byte;
    sin_port: Byte;
    sin_addr: TInAddr;
    sin_zero: array[0..7] of Char);
   end;
```

В данной работе используется лишь поле *sin_addr*, в котором хранится сетевой адрес. Это поле также представляет собой запись, в нашем случае состоящую из одного поля *s_addr*. Чтобы получить к нему доступ, нужно использовать выражение вида:

```
SockAddrIn.sin_addr.s_addr := ...
```

Алгоритм работы создаваемой программы таков.

1. Инициализация *WinSocket*.
2. Запрос у пользователя *IP*-адреса в виде четырех октетов, разделенных точкой.

3. Преобразование этого адреса в сетевой формат.
4. Получение информации о сервере с данным адресом.
5. Извлечение и отображение доменного имени.

3. Порядок выполнения работы

Запустить интегрированную среду разработки программ *Delphi*. Выполнить команду *File-Close All* для закрытия тех окон проектов, которые были загружены автоматически.

Выполнить команду *File-New-Application.* Это будет основная программа.

Поместите в окне программы компоненты *Memo* (поле для отображения текста, в нем будут выводиться доменные имена), *Edit* (поле для ввода текста, в нем пользователь будет вводить *IP*-адрес) и *Button* (управляющая кнопка).

В раздел *Uses*, в котором объявляются используемые модули, добавьте *WinSock*.

В разделе Implementation введите:

```
function IPAddrToName(IPAddr : String): String;
var
  SockAddrIn: TSockAddrIn;
  HostEnt: PHostEnt;
  WSAData: TWSAData;
begin
  WSAStartup($101, WSAData);
  SockAddrIn.Sin_Addr.S_Addr :=  Inet_Addr(PChar(IPAddr));
  HostEnt := GetHostByAddr(@SockAddrIn.Sin_Addr.S_Addr, 4, AF_INET);
  if HostEnt <> nil then result:=StrPas(Hostent^.h_name) else
   result:='Error';
WSACleanup;
end;
```

В теле процедуры обработки нажатия на кнопку укажите:

```
Memo1.Lines.Add (IPAddrToName(Edit1.Text));
```

Запустите программу на выполнение, нажав клавишу *F9*.

Введите следующие IP-адреса (по одному):

64.12.164.65

213.180.194.129

81.19.66.109

Также вы можете указывать другие адреса, если они вам известны.

Проверьте правильность работы программы.

4. Вопросы для подготовки к защите работы

1. Что такое «сокет»?
2. Что представляет собой и для чего предназначена библиотека *Win-Socket*?
3. Для чего используется и что собой представляет *IP*-адрес?
4. Назначение и параметры функции *WSAStartup*.
5. Назначение и параметры функции *WSACleanup*.
6. Назначение и параметры функции *GetHostByAddr.*
7. Поясните алгоритм работы данной программы.
8. «Расшифровать» выражение:

HostEnt := GetHostByAddr(@SockAddrIn.Sin_Addr.S_Addr, 4, AF_INET).

9. Предложить способ определения *IP*-адреса по указанному имени домена.

Лабораторная работа № 8
«Определение *IP*-адреса по доменному имени»

1. Цель работы

Изучение способа определения *IP*-адреса по известному доменному имени. Знакомство с функциями, процедурами и структурами данных, используемыми при работе с сетью.

2. Теоретическая часть

В предыдущей работе вы научились определять доменное имя по заранее известному *IP*-адресу. Вообще говоря, подобная задача является в некоторой степени надуманной. Дело в том, что одной из основных задач системы *DNS* (*Domen Name System*) является избавление рядового пользователя от необходимости запоминать *IP*-адрес, который представляет собой, по сути, ни с чем не соотносимую (в представлении пользователя) последовательность чисел.

В противоположность этому, доменное имя состоит из букв английского алфавита, что дает огромные возможности создания простых для запоминания имен и использования их в повседневном обиходе.

Однако буквенно-цифровые доменные имена не могут непосредственно использоваться в сети Интернет. Они должны быть преобразованы в *IP*-адрес. Естественно, что сама процедура преобразования является незаметной для пользователя, и даже для программиста (до определенного уровня).

Работа программы основана на использовании функции *GetHostByName*, которая позволяет получить информацию о хосте, указав доменное имя.

```
Function GetHostByName (Name: PChar): PHostEnt;
```

Name – параметр, содержащий доменное имя.

Функция возвращает результат типа *PHostEnt*. Данный тип определен следующим образом (уже было рассмотрено в предыдущей работе):

```
type
  PHostEnt = ^THostEnt;
```

```
  THostEnt = record
  h_name: PChar;
  h_aliases: ^PChar;
  h_addrtype: SmallInt;
  h_length: SmallInt;
  h_addr: ^PChar;
 end;
```

В поле *h_addr* и будет храниться нужный нам *IP*-адрес. Однако этот адрес пока находится в сетевом формате, а не в привычном нам виде: четыре числа, разделенных точкой.

Поэтому перед тем, как вывести *IP*-адрес на дисплей, нужно преобразовать его. Это делается при помощи функции *Format*. Она позволяет, например, преобразовать многобайтовое число в строку, которая будет соответствовать заданному шаблону.

Function Format (const Format: string; const Args: array of const): string;

Format – строковая переменная, задающая шаблон; имеет вид, например, *'%d.%d.%d.%d'*, что соответствует *IP*-адресу, буква *d* указывает на то, что число будет записано в десятичной системе;
Args – массив чисел, подлежащих преобразованию в строку, соответствующую шаблону.

Алгоритм работы создаваемой программы таков.

1. Инициализация *WinSocket*.
2. Запрос у пользователя доменного имени.
3. Получение информации о хосте.
4. Извлечение *IP*-адреса.
5. Преобазование *IP*-адреса к «традиционному» виду и вывод его на экран.

3. Порядок выполнения работы

Запустить интегрированную среду разработки программ *Delphi*. Выполнить команду *File-Close All* для закрытия тех окон проектов, которые были загружены автоматически.

Выполнить команду *File-New-Application.* Это будет основная программа.

Поместите в окне программы компоненты *Memo* (поле для отображения текста, в нем будут выводиться *IP*-адреса), *Edit* (поле для ввода текста, в нем пользователь будет вводить доменное имя) и *Button* (управляющая кнопка).

В раздел *Uses*, в котором объявляются используемые модули, добавьте *WinSock*.

В разделе *Implementation* введите:

```
function HostToIP(Name: String): String;
var
  WSData: TWSAData;
  HostEnt: PHostEnt;
  Addr: PChar;
begin
  WSAStartup ($0101, WSData);
    HostEnt := GetHostByName (PChar(Name));
    if (HostEnt<>nil) and (HostEnt^.H_addr<>nil) then
     begin
     Addr := HostEnt^.H_addr^;
     if Addr<>nil then
     Result := Format ('%d.%d.%d.%d', [Byte (Addr [0]),
     Byte (Addr [1]), Byte (Addr [2]), Byte (Addr [3])])
     else
     Result := 'Error';
     end
    else Result := 'Error';
    WSACleanup;
end;
```

В теле процедуры обработки нажатия на кнопку укажите:

```
Memo1.Lines.Add(HostToIP(Edit1.Text));
```

Запустите программу на выполнение, нажав клавишу *F9*.
Вводите любые знакомые вам доменные имена:
yandex.ru,
ssu.samara.ru
и т. д.

Проверьте правильность работы программы. При необходимости дополнительной проверки воспользуйтесь программой, созданной в предыдущей работе.

4. Вопросы для подготовки к защите работы

1. Что представляет собой и для чего предназначена библиотека *WinSocket*?
2. Для чего используется и что собой представляет *IP*-адрес?
3. Назначение и параметры функции *WSAStartup*.
4. Назначение и параметры функции *WSACleanup*.
5. Назначение и параметры функции *GetHostByName.*
6. Поясните алгоритм работы данной программы.
7. «Расшифровать» выражение:

Result := Format ('%d.%d.%d.%d', [Byte (Addr [0]),
Byte (Addr [1]), Byte (Addr [2]), Byte (Addr [3])]).

8. Почему потребовалось дополнительное преобразование *IP*-адреса перед выводом его на экран?
9. В каком виде он хранился до этого?

Лабораторная работа № 9
«Обработка исключительных ситуаций. Оператор *try ... except*»

1. Цель работы

Изучение способа обработки исключительных ситуаций, основанного на использовании оператора *try ... except*.

2. Теоретическая часть

При разработке практически любой программы (исключая простейшие) программист сталкивается с необходимостью предусматривать обработку так называемых «исключительных ситуаций». Примерами таких ситуаций могут быть: деление на нуль, отсутствие запрошенного файла, ввод пользователем букв вместо числа и т. д.

Если специально не предусмотреть реакции программы на эти ошибки, то они будут обрабатываться стандартными средствами *Delphi*. Как правило, это означает, что при, например, неправильном вводе числа, программа будет просто завершена. А ведь можно было просто дать пользователю возможность заново ввести число. Кроме того, разрабатывая законченный программный продукт, программист просто обязан предусматривать обработку исключительных ситуаций, в том числе, обеспечивая вывод окна с подробным описанием ошибки и возможными вариантами решения проблемы.

Возможности обработки исключительных ситуаций в *Delphi* не ограничены лишь реагированием на стандартные ошибки. Однако этот аспект будет изучен в следующих работах.

Используемая в *Delphi* модель обработки исключений является невозобновляемой. Это значит, что при возникновении исключительной ситуации и после ее обработки выполнение программы не будет продолжено с того места, где она возникла.

Группа операторов, для которой вы предусматриваете обработку исключений, называется «защищенной». В *Delphi* есть несколько вариантов защищенных блоков, отличия между которыми заключаются в логике обработки исключений. В данной работе мы будет рассматривать блок *try...except*. Его синтаксис следующий:

```
try
  Statement1;
  Statement2;
  ....
  ....
except
  on Exception1 do Statement;
  on Exception2 do Statement;
  ....
  ....
else
  Statement;
end;
```

Защищенная группа операторов помещается между *try* и *except*. Затем следуют указания о том, как обрабатывать ту или иную ошибку (*on Exception1 do Statement*). *Exception1* представляет собой «название» исключительной ситуации, а *Statement* представляет собой обработчик данной ситуации. Таким образом, вы можете предусмотреть собственный обработчик для разных ошибок. После ключевого слова *else* следует обработчик «по умолчанию», т. е. если произошла ошибка, реакция на которую в явном виде не предусмотрена, выполняется этот обработчик.

В таблице приведены некоторые исключительные ситуации и их обозначения в Delphi.

Обозначение	Описание исключительной ситуации
EConvertError	Преобразование строки в число (или наоборот) невозможно. Пример: преобразуемая в число строка содержит буквы
EInOutError	Ошибка ввода-вывода. Пример: ошибка чтения диска.
EDivByZero	Деление на нуль.
EIntOverflow	Переполнение. Операция привела к получению значения, превышающего верхнюю границу для данного типа переменной.
EInvalidGraphicOperation	Попытка выполнения операций, неприменимых для данного графического формата.

В данной работе вы создадите программу, которая запрашивает два числа и делит первое из них на второе. При этом будет предусмотрена обработка исключительной ситуации *EConvertError*.

3. Порядок выполнения работы

Для правильной работы большинства программ (в т. ч. Delphi) под Windows NT необходимо предварительно настроить переменные среды. На рабочем столе найдите значок «Мой компьютер» и кликните по нему правой кнопкой мыши. В меню выберите пункт «Свойства», затем закладку «Переменные среды». Необходимо присвоить переменным Temp и Tmp значение %UserProfile%\Temp.

Внимание! *Не удаляйте символы в фигурных скобках, например, {$R *.res}, находящиеся в шаблоне программы! Это не комментарии, а директивы компилятора, без них ваша программа компилироваться не будет*.

Запустить интегрированную среду разработки программ *Delphi*. Выполнить команду *File-Close All* для закрытия тех окон проектов, которые были загружены автоматически.

Выполнить команду *File-New-Application*.

Разместите на форме два окошка *Edit*, поле *Memo* и кнопку *Button*.

В разделе *var* добавьте:

```
a, b: Extended;
```

В окошке *Object Inspector* выберите объект *Edit1*, затем закладку *Events* и дважды щелкните по событию *OnExit*. Появится заготовка процедуры. Введите:

```
try
  a:=StrToFloat(Edit1.Text);
except
  on EConvertError do
  begin
  MessageBox (0, 'Правильно введите делимое!', 'Error', MB_OK);
  Edit1.Clear;
  end;
end;
```

Проделайте то же самое для поля *Edit2*. В заготовке процедуры введите:

```
try
  b:=StrToFloat(Edit2.Text);
except
  on EConvertError do
  begin
  MessageBox (0, 'Правильно введите делитель!', 'Error', MB_OK);
  Edit2.Clear;
  end;
end;
```

Обработчик события *OnExit* вызывается в том случае, если вы «покидаете» поле для редактирования. Т. е. если число введено неверно, то при попытки перейти к вводу другого числа, нажать кнопку и т. д. будет произведена попытку преобразовать введенное число в строку. Если происходит исключительная ситуация *EConvertError*, то выводится сообщение.

В обработчике нажатия кнопки укажите:

```
Memo1.Lines.Add(FloatToStr(a/b));
```

Запустите программу, проверьте правильность ее работы.

4. Вопросы для подготовки к защите работы.

1. Что такое «исключительная ситуация»?
2. Для чего нужна обработка исключительных ситуаций?
3. Назовите примеры исключительных ситуаций.
4. Что такое «защищенная группа операторов»?
5. Приведите синтаксис блок *try...except*.
6. Каков алгоритм работы созданной вами программы?
7. Какие недостатки вы обнаружили в ее работе?

Лабораторная работа № 10
«Обработка исключительных ситуаций. Оператор *try ... finally*»

1. Цель работы

Изучение способа обработки исключительных ситуаций, основанного на использовании оператора *try ... finally*.

2. Теоретическая часть

Блок *try ... finally* позволяет обеспечить гарантированное освобождение ресурсов системы, занятых вашим приложением, в любом случае.

Синтаксис блока *try ... finally* следующий:

```
 try
        Statement1;
        Statement2;
...
 finally
        Statements;
  end;
```

Защищенная группа операторов помещается между ключевыми словами *try* и *finally*. Операторы, расположенные после ключевого слова *finally*, выполняются в любом случае, в том числе при возникновении ошибки.

В отличие от *try...except*, блок *try...finally* фактически не обрабатывает исключительные ситуации, а лишь обеспечивает выполнение последовательности операторов. Безусловно, это накладывает определенные ограничения на область применения этого метода обработки исключений. Однако бывают ситуации, когда оправдано применение именно блока *try…finally*.

Например, в одной из предыдущих работ вы создавали программу, определявшую доменное имя по *IP*-адресу. Рассмотрим ее фрагмент.

```
begin
  WSAStartup($101, WSAData);
  SockAddrIn.Sin_Addr.S_Addr :=  Inet_Addr(PChar(IPAddr));
```

```
 HostEnt := GetHostByAddr(@SockAddrIn.Sin_Addr.S_Addr, 4, AF_INET);
 if HostEnt <> nil then result:=StrPas(Hostent^.h_name) else
  result:='Error';
WSACleanup;
end;
```

Эта программа была работоспособна, однако более правильно было бы добавить в нее обработку исключений.

Очевидно, что в этом фрагменте присутствует операция, которая должна быть выполнена в любом случае – это функция завершения работы с сокетами *WSACleanup*. И если в вышеприведенном фрагменте произойдет исключительная ситуация в промежутке между вызовом *WSAStartup* и *WSACleanup*, то последняя может быть не выполнена. Поэтому более правильно было бы оформить фрагмент следующим образом:

```
begin
 WSAStartup($101, WSAData);
  try
  SockAddrIn.Sin_Addr.S_Addr :=  Inet_Addr(PChar(IPAddr));
 HostEnt := GetHostByAddr(@SockAddrIn.Sin_Addr.S_Addr, 4, AF_INET);
 if HostEnt <> nil then result:=StrPas(Hostent^.h_name) else
  result:='Error';
 finally
 WSACleanup;
 end;
end;
```

Эту программу вы и создадите в практической части.

3. Порядок выполнения работы

Для правильной работы большинства программ (в т. ч. Delphi) под Windows NT необходимо предварительно настроить переменные среды. На рабочем столе найдите значок «Мой компьютер» и кликните по нему правой кнопкой мыши. В меню выберите пункт «Свойства», затем закладку «Переменные среды». Необходимо присвоить переменным Temp и Tmp значение %UserProfile%\Temp.

Внимание! *Не удаляйте символы в фигурных скобках, например, {$R *.res}, находящиеся в шаблоне программы! Это не комментарии, а директивы компилятора, без них ваша программа компилироваться не будет.*

Запустить интегрированную среду разработки программ *Delphi*. Выполнить команду *File-Close All* для закрытия тех окон проектов, которые были загружены автоматически.

Выполнить команду *File-New-Application*.

Поместите в окне программы компоненты *Memo* (поле для отображения текста, в нем будут выводиться доменные имена), *Edit* (поле для ввода текста, в нем пользователь будет вводить *IP*-адрес) и *Button* (управляющая кнопка).

В раздел *Uses*, в котором объявляются используемые модули, добавьте *WinSock*.

В разделе *Implementation* введите:

```
function IPAddrToName(IPAddr : String): String;
var
  SockAddrIn: TSockAddrIn;
  HostEnt: PHostEnt;
  WSAData: TWSAData;
begin
  WSAStartup($101, WSAData);
  try
  SockAddrIn.Sin_Addr.S_Addr :=  Inet_Addr(PChar(IPAddr));
  HostEnt := GetHostByAddr(@SockAddrIn.Sin_Addr.S_Addr, 4, AF_INET);
  if HostEnt <> nil then result:=StrPas(Hostent^.h_name) else
   result:='Error';
  finally
  WSACleanup;
 end;
end;
```

В теле процедуры обработки нажатия на кнопку укажите:

```
Memo1.Lines.Add (IPAddrToName(Edit1.Text));
```

Запустите программу на выполнение, нажав клавишу *F9*.

Проверьте правильность работы программы.

4. Вопросы для подготовки к защите работы.

1. Что такое «исключительная ситуация»?
2. Для чего нужна обработка исключительных ситуаций?
3. Что такое «защищенная группа операторов»?
4. Приведите синтаксис блок *try...finally*.
5. В чем его отличие от блока *try...except*?
6. Каков алгоритм работы созданной вами программы?
7. Почему вызов процедуры *WSAStartup* производится за пределами защищенного блока операторов?

Лабораторная работа № 11
«Использование обработки исключительный ситуаций для оптимизации исходного кода программы»

1. Цель работы

Целью данной работы является получение практических навыков использования обработки исключительных ситуаций для более рационального написания программ.

2. Теоретическая часть

Любая, даже не очень сложная, программа должна предусматривать обработку нестандартных ситуаций. Во-первых, это исключительные ситуации, которые были рассмотрены в предыдущих работах. Однако помимо них есть просто ошибки, обычно по вине пользователя, который может, например, неверно ввести какое-то значение. Например, если ввести неверное доменное имя, то функция *GetHostByName* вернет значение *nil*, свидетельствующее об ошибке. Однако это – не исключительная ситуация. Если в программе не выполняется соответствующая проверка, то выполнение следующих команд программы, например, соединение с сервером, вообще не имеет смысла.

Таким образом, нужно проверять, успешно выполнилась функция, или же нет. Например:

```
if WSAStartup($0101, WSData)= 0 then
 begin
 Memo1.Lines.Add ('Winsocket library init... OK');
 HostEnt := GetHostByName(PChar(Edit1.Text));
 end
else Memo1.Lines.Add ('Winsocket library init... Failed');
```

Здесь проверка выполняется при помощи условного оператора *if...else*. И если в небольших программах это еще приемлемо, то в программах большого объема использование такого способа имеет много недостатков: объем кода растет, быстродействие снижается, сильно затрудняется восприятие исходного текста даже самим программистом (например, оператор *if* может оказаться в начале программы, *else* – в середине, а между ними

еще десяток таких же условных операторов), приходится вводить новые переменные, несущие информацию о том, что случилась ошибка (это имеет место, если ошибка происходит внутри процедуры, а реакция на нее выполняется основной программой) и пр.

Рассмотрим для примера знакомый вам исходный текст программы для подключения к серверу.

```
Memo1.Lines.Add ('Trying to connect to ' + Edit1.Text + '...');
if WSAStartup($0101, WSData)= 0 then
 begin
 Memo1.Lines.Add ('Winsocket library init... OK');
 HostEnt := GetHostByName(PChar(Edit1.Text));
  if HostEnt <> nil then
  begin
  Memo1.Lines.Add ('Getting Host Information... OK');
  Addr := HostEnt^.H_addr^;
    if Addr<>nil then
    Memo1.Lines.Add('IP: '+ Format('%d.%d.%d.%d', [Byte (Addr [0]),
    Byte (Addr [1]), Byte (Addr [2]), Byte (Addr [3])]));
  Sock := Socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);
  if Sock <> INVALID_SOCKET then
  begin
  Memo1.Lines.Add('Creating Socket... OK');
  SockAddrIn.Sin_family := AF_INET;
  SockAddrIn.Sin_port := Htons(80);
  SockAddrIn.Sin_Addr.S_Addr := Byte(Addr[3]) shl 24 + Byte(Addr[2])
shl 16 + Byte(Addr[1]) shl 8 + Byte(Addr[0]);
  if Connect(Sock, SockAddrIn, SizeOf(SockAddrIn)) = 0 then
  Memo1.Lines.Add('SUCCESSFULLY CONNECTED')
  else
  begin
  Memo1.Lines.Add('FAILED TO CONNECT');
  Memo1.Lines.Add(IntToStr(WSAGetLastError));
  end;
  end
  else Memo1.Lines.Add ('Creating Socket... Failed');
```

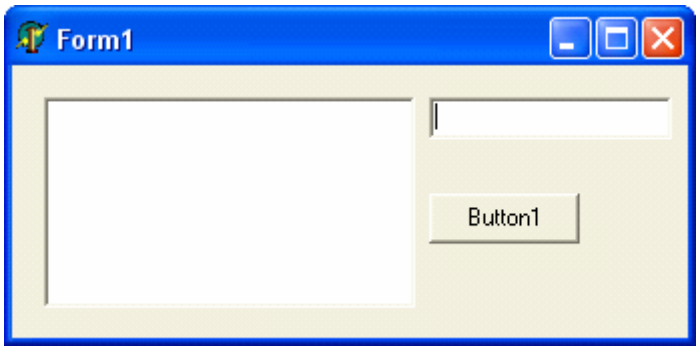
```
 end
 else Memo1.Lines.Add ('Getting Host Information... Failed')
end
 else Memo1.Lines.Add ('Winsocket library init... Failed');
CloseSocket(Sock);
WSACleanup;
end;
```

Даже в такой небольшой программе обилие операторов *if..else* затрудняет анализ текста и отладку программы. Однако долгое время у программиста не было другой альтернативы.

В *Delphi* имеется возможность существенно упростить обработку ошибок, не являющихся по умолчанию исключительными ситуациями. При этом используется блок *try...except*.

В данной работе будет приведен исходный текст программы, по своему назначению совершенно эквивалентный только что рассмотренной. Ваша задача (дополнительно см. раздел 4) – самостоятельно разобраться в методике обработки ошибок, возникающих в процессе работы программы.

3. Исходный текст




```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, WinSock, StdCtrls;

type
  TForm1 = class(TForm)
```

```
    Edit1: TEdit;
    Button1: TButton;
    Memo1: TMemo;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

EInvalidHostError = class (Exception);
EInitSockError = class (Exception);
EInvalidAddrError = class (Exception);
EInvalidSockError = class (Exception);
EConnectFailError = class (Exception);

var
  Form1: TForm1;
  WSData: WSAData;
  HostEnt: PHostEnt;
  Sock: TSocket;
  SockAddrIn: TSockAddrIn;
  Addr: PChar;

  implementation

{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
begin
Memo1.Lines.Add ('Trying to connect to ' + Edit1.Text + '...');
try
try
if WSAStartup($0101, WSData)<> 0 then raise EInitSockError.Create ('');
Memo1.Lines.Add ('Winsocket library init... OK');
HostEnt := GetHostByName(PChar(Edit1.Text));
if HostEnt = nil then raise EInvalidHostError.Create ('');
```

```
Memo1.Lines.Add ('Getting Host Information... OK');
Addr := HostEnt^.H_addr^;
if Addr = nil then  raise EInvalidAddrError.Create ('');
Memo1.Lines.Add('IP: '+ Format('%d.%d.%d.%d', [Byte (Addr [0]),
Byte (Addr [1]), Byte (Addr [2]), Byte (Addr [3])]));
Sock := Socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);
if Sock = INVALID_SOCKET then raise EInvalidSockError.Create ('');
Memo1.Lines.Add('Creating Socket... OK');
SockAddrIn.Sin_family := AF_INET;
SockAddrIn.Sin_port := Htons(80);
SockAddrIn.Sin_Addr.S_Addr := Byte(Addr[3]) shl 24 + Byte(Addr[2]) shl
16 + Byte(Addr[1]) shl 8 + Byte(Addr[0]);
if Connect(Sock, SockAddrIn, SizeOf(SockAddrIn)) <> 0 then raise ECon-
nectFailError.Create ('');
Memo1.Lines.Add('SUCCESSFULLY CONNECTED')
except
on EConnectFailError do
  begin
  Memo1.Lines.Add('FAILED TO CONNECT');
  Memo1.Lines.Add(IntToStr(WSAGetLastError));
  end;
on EInvalidSockError do Memo1.Lines.Add ('Creating Socket... Failed');
on EInvalidHostError do Memo1.Lines.Add ('Getting Host Information...
Failed');
on EInvalidAddrError do Memo1.Lines.Add ('Getting Host Information...
Failed');
on EInitSockError do Memo1.Lines.Add ('Winsocket library init... Failed');
end;
finally
CloseSocket(Sock);
WSACleanup;
end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
Memo1.Lines.Clear;
end;

end.
```

4. Контрольные вопросы.

1. Характеризуйте новый вариант исходного текста по сравнению со старым. Каковы преимущества нового варианта?
2. Попытайтесь сформулировать принцип, лежащий в основе метода обработки ошибок в данной программе.
3. Объясните назначение ключевого слова *raise*.
4. С какой целью в данной программе используется блок *try..finally*?

Печатается в авторской редакции
Компьютерная верстка, макет В.И. Никонов

Подписано в печать 06.03.07
Гарнитура Times New Roman. Формат 60x84/16. Бумага офсетная. Печать оперативная.
Усл.-печ. л. 3,75. Уч.-изд. л. 1,99. Тираж 100 экз. Заказ № 627
Издательство «Универс групп», 443011, Самара, ул. Академика Павлова, 1

Отпечатано ООО «Универс групп»