РКФ

Радиоконструкторский факультет

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ»

УТВЕРЖДАЮ
Заведующий кафедрой РЭТЭМ
_____**В.И.Туев**
“___” _____2018 г.

Основы работы в открытой среде разработки программного обеспечения Lazarus

Лабораторный практикум
по дисциплине «Информатика. ГИС в экологии и природопользовании» для студентов направления 050306 «Экология» (бакалавриат)

Разработчик:
Доцент кафедры РЭТЭМ
_____Д.В.Озеркин

Томск 2018

СОДЕРЖАНИЕ

1. Лабораторная работа №1 – Создание шаблона для разрабатываемых программ

1.1 Цель работы

1. Приобретение начальных навыков работы в интегрированной среде объектно-ориентированного программирования (ООП) *Lazarus*.

2. Освоение работы с компонентами ***TForm*** (Форма), ***TLabel*** (Надпись) и ***TButton*** (Командная кнопка).

В ходе выполнения работы следует научиться:

- работать в интегрированной среде *Lazarus*;
- усвоить правила использования основных типов объектов в *Lazarus*;
- освоить приёмы создания простейших графических интерфейсов в среде *Lazarus*.

1.2 Порядок выполнения лабораторной работы

Перед выполнением этой работы следует:

1. Ознакомиться с теоретической частью «Основы объектно-ориентированного программирования (ООП)». В качестве источника знаний использовать [7, 8].
2. Изучить правила работы с интегрированной средой программирования IDE *Lazarus*, а также ознакомиться со структурой программ в системе *Lazarus*.
3. Войти в свой личный каталог, загрузить и настроить систему программирования *Lazarus*.
4. Создать шаблон для собственных разрабатываемых приложений и сохранить его в личном каталоге на сервере (имя каталога – фамилия студента).
5. Добиться, чтобы при компиляции приложения-шаблона не было сообщений об ошибках.
6. Ответить на контрольные вопросы.
7. Оформить отчёт и защитить его у преподавателя.

1.3 Отчетность

Отчёт должен быть выполнен в соответствии с [9] и состоять из следующих разделов:

1. Цель работы.
2. Откомпилированный текст разработанного приложения (в электронном виде).
3. Ответы на контрольные вопросы.
4. Выводы.

Для получения зачёта студент должен:

– уметь отвечать на контрольные вопросы;
– показать, что разработанное приложение работает правильно;
– уметь пояснить назначение элементов разработанного приложения;
– продемонстрировать навыки работы в интегрированной среде *Lazarus*.

1.4 Практическое задание

1.4.1 Назначение шаблона разрабатываемых программ

Шаблон проектирования (англ. *Design pattern*) в разработке программного обеспечения – повторимая конструкция, позволяющая снизить сложность разработки за счёт использования готовых абстракций для решения некоторых повторяющихся задач. С подобными элементарными шаблонами (заготовками программ) сталкиваются в своей повседневной деятельности практически все разработчики программного обеспечения. Хороший правильный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова, снижая количество ошибок и время разработки программ. Разумно постоянно модернизировать, улучшать шаблон, используя приобретаемый опыт.

Начиная разработку нового программного проекта, следует загрузить шаблон и сохранить его под новым именем. Модифицировать всегда легче и быстрее, чем создавать всё заново!

1.4.2 Запуск интегрированной среды *Lazarus*

Откройте *Lazarus*, если он у вас закрыт, или закройте старый проект и начните новый. Для запуска интегрированной среды (IDE) *Lazarus* сделайте двойной щелчок ЛК мышки на значке  . На экране появятся несколько окон *Windows* (рис. 1.1).

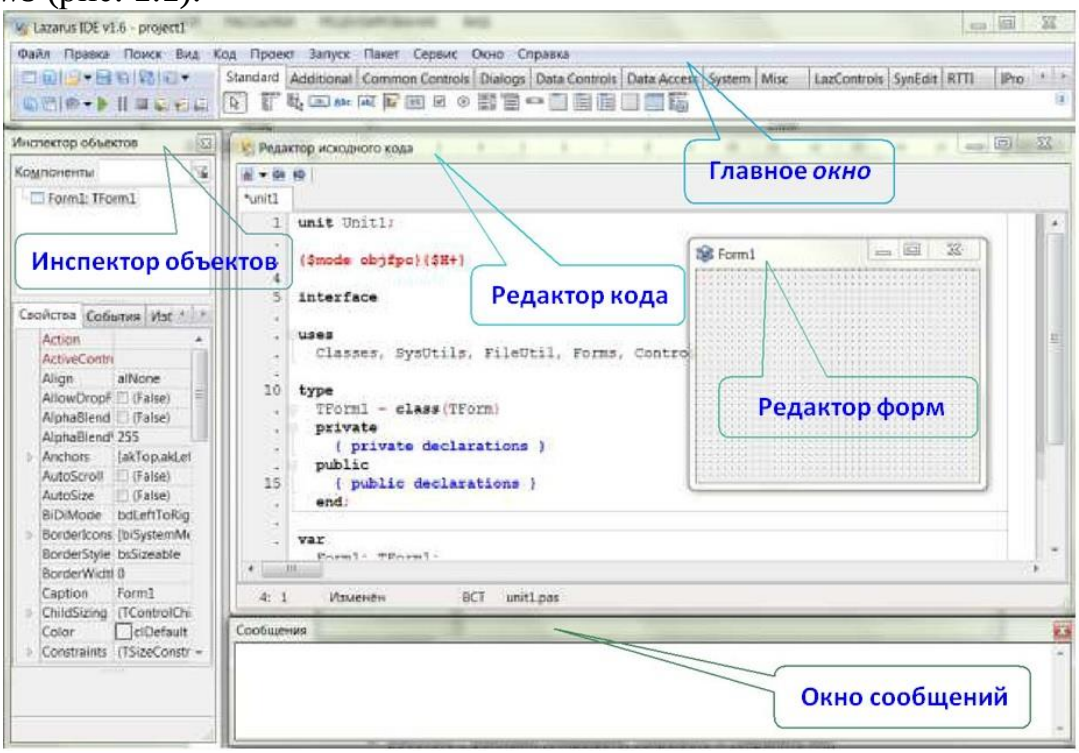


Рис. 1.1 - Окна интегрированной среды разработки программ ***Lazarus***

Главное окно *Lazarus* управляет проектом создаваемой программы. Оно размещено вверху и содержит:

1. **Главное меню,** позволяющее активизировать **выпадающие меню** с перечнем сгруппированных по их предназначению команд (работа с файлами, отладка, компиляция, справка, запуск различных вспомогательных утилит и т.п.).

2. **Панель инструментов** с наиболее часто используемыми командами **Главного меню** (команды сохранения и открытия, запуска, останова, трассировки и т.п.).

3. **Палитру (панель) компонентов,** содержащую множество вкладок, позволяющих выбрать стандартные компоненты, используемые при конструировании формы будущего окна приложения.

Инспектор Объектов (*Object Inspector*), расположенный слева, содержит:

1. В верхней части – **Дерево объектов**, в котором в иерархической виде располагаются все объекты, используемые в текущей Форме.

2. В нижней части **Инспектора Объектов – вкладки** *Свойства, События, Избранное, Ограничения*, в которых настраивают различные параметры текущего компонента.

В окне **Редактора Форм** (окно будущего приложения с именем ***Form1*** по умолчанию) осуществляется редактирование формы – положения и размеров компонентов, размещённых на этой форме.

Занимающий большую часть экрана **Редактор исходного кода** *Lazarus* (*Lazarus Source Editor*) содержит исходный код программы, который мы должны создать.

В **Окно сообщений** (Messages) выводятся различные сообщения: о найденных ошибках, о завершении компиляции, о наличии объявленных, но неиспользуемых переменных и т.п.

1.4.3 Создание нового проекта

Новый проект создаётся с помощью последовательности команд **Проект / Создать проект**. В появившемся диалоговом окне (рис. 1.2) следует выбрать опцию **Приложение** и нажать кнопку **ОК**.

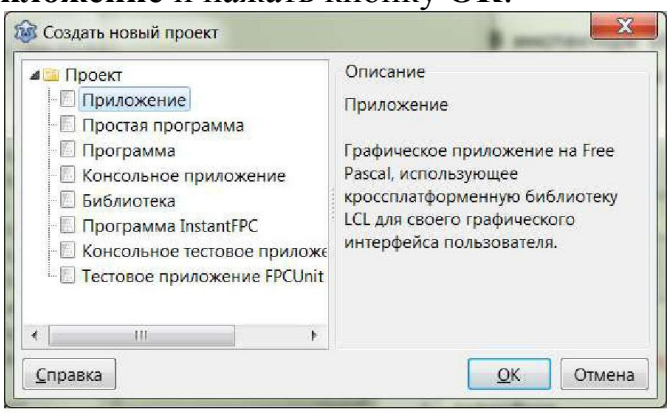


Рис. 1.2 - Диалоговое окно **Создать новый проект**

1.4.4 Сохранение созданного проекта

Сохранить созданный проект можно с помощью последовательности команд **Проект / Сохранить проект как…** В открывшемся окне **Сохранить проект** (рис. 1.3) необходимо создать новую папку с именем ***Шаблон Lazarus*** для шаблона файлов Ваших проектов (проект будет содержать несколько файлов), открыть её, набрать в строке **Имя файла**, например, ***Template*** и щёлкнуть по кнопке **Сохранить**. В результате мы сохраним файл ***Template***, содержащий сведения о проекте. Заметим, что в Lazarus (как и в Pascal) строчные и заглавные буквы не различаются. Тем не менее, для удобочитаемости кода лучше приучиться использовать заглавные буквы, чтобы выделять названия.

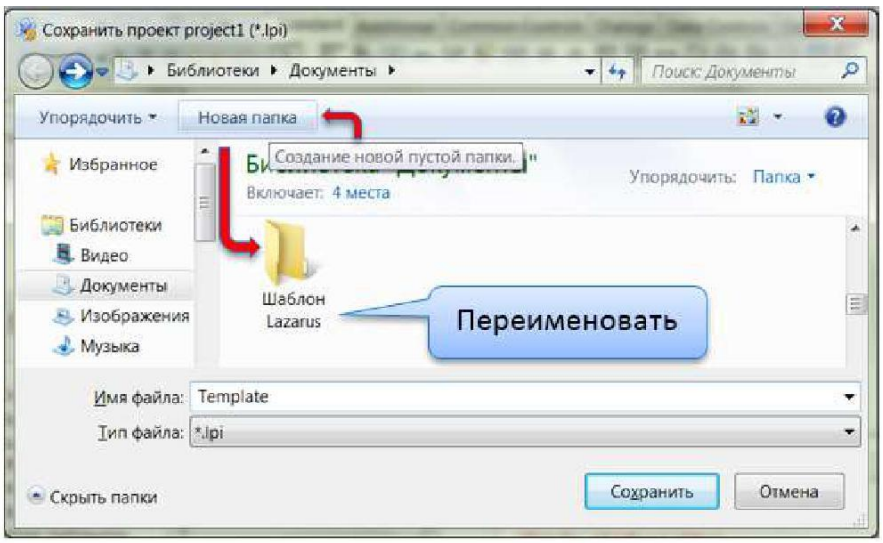


Рис. 1.3 - Диалоговое окно **Сохранить проект**

Автоматически откроется диалоговое окно **Сохранить** для сохранения программного кода проекта, которое по умолчанию имеет заголовок ***Unit1***. Дадим ему имя, например, ***Main*** или ***unitivanov*** (фамилию, конечно, свою!) (файл ***unitivanov.pas***), в котором также необходимо щёлкнуть по кнопке **Сохранить**.

Кроме этих двух файлов в папке проекта создаётся автоматически ещё несколько файлов, в том числе – ***unitivanov.lfm***, который представляет собой файл с полными данными о проектировщике формы: о позициях, размерах и т.п. размещённых в форме компонентов. В папке проекта теперь должны содержаться следующие файлы (рис. 1.4).

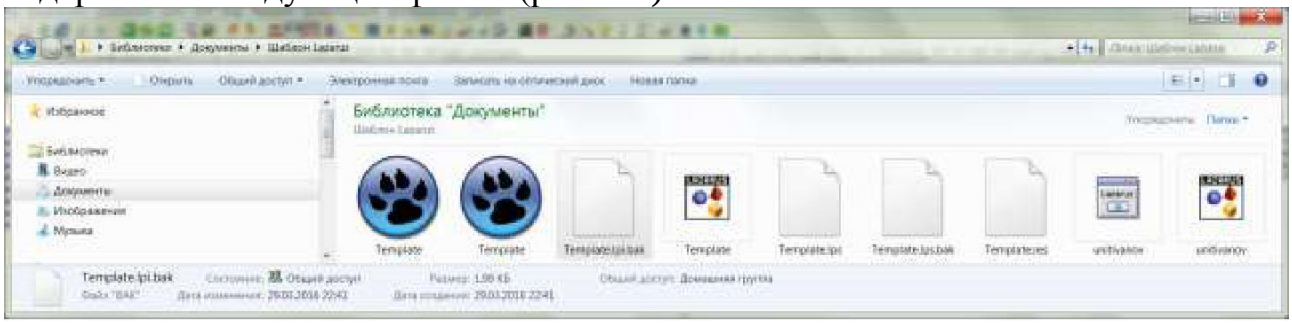


Рис. 1.4 - Содержание папки **Шаблон Lazarus** после сохранения проекта

Примечание. Откуда появляются файлы с расширением **.bak*?

Это страховочные копии (англ. ***backup***) предыдущих редакций исходных файлов до момента их последнего сохранения.

Если нужно вернуться к предыдущей редакции файла (например, если сохранённая версия содержит ошибки и требуется вернуть то, что было), то переименуйте страховочную копию – вместо *bak* запишите расширение исходного файла.

1.4.5 Создание Формы

Каждая программа *Lazarus* с графическим интерфейсом содержит как минимум одно окно, внутри которого отображаются остальные элементы интерфейса. Окно является объектом класса TForm (Форма) и обладает всеми свойствами стандартных графических окон. Оно, как правило, имеет заголовок. Свойства Формы определяют вид окна Вашей программы. Его можно свернуть, можно развернуть во весь экран, можно менять размеры. Окно можно перемещать в любое место экрана. Таблица 1.1 представляет назначение основных свойств компонента TForm**,** таблица 1.2 – его методов, а Таблица 1.3 – обрабатываемых событий.

Таблица 1.1 – Некоторые часто используемые свойства компонента класса ***TForm*** (Форма)

Свойство	Тип	Описание
Name	Строка	Имя формы. В программе могут быть несколько форм. Имя формы в программе используется для управления соответствующей формой и доступа к компонентам формы. Имя не должно содержать недопустимые символы, пробелы и т.д.
Caption	Строка	Текст заголовка окна. Обычно здесь выводят название программы и имя документа, связанного с этой программой или краткое содержание программы, например «Мои расчеты»
Top	Целое число	Расстояние от верхней границы формы до верхней границы экрана.
Left	Целое число	Расстояние от левой границы формы до левой границы экрана.
Width, Height	Целое число	Ширина, высота формы. Размеры задаются в пикселях.
Icon		Значок в заголовке диалогового окна, обозначающий кнопку вывода системного меню.
Color		Цвет фона.
Font	Объект *TFont*	Шрифт элементов интерфейса. Шрифт, используемый по «умолчанию» для компонентов, находящихся на поверхности формы.
Canvas		Поверхность, на которую можно вывести графику.
Position		Положение окна при запуске: *poDesigned* – положение окна и его размеры остаются такими же, что и при проектировании;

		poDefault – положение окна и его размеры определяется автоматически операционной системой; *poDefaultPosOnly* – положение окна определяется автоматически операционной системой, а размеры соответствуют установкам при проектировании; *poDefaultSizeOnly* – размеры окна определяется автоматически операционной системой, а положение соответствуют установкам при проектировании; *poScreenCenter* или *poDesktopCenter* – окно выводится в центре экрана, размер определяется при проектировании; *poMainFormCenter* – форма отображается в центре главной формы, размер определяется при проектировании, если имеется только одна главная форма, то этот параметр соответствует; *poScreenCenter*; *poOwnerFormCenter*– форма отображается в центре той формы, которая является владельцем данной формы.
Align		Управление положением формы на экране: *alNone* – положение формы и его размеры не меняются; *alBottom* – форма располагается внизу экрана; *alLeft* – форма располагается в левой части экрана; *alRight* – форма располагается в правой части экрана; *alTop* – форма располагается вверху экрана; *alClient* – форма занимает весь экран.

Таблица 1.2 – Назначение некоторых часто используемых методов компонента класса

Метод	Аргументы	Возвращаемое значение	Описание
Show	Нет	Нет	Показывает окно на экране
ShowModal	Нет	Целое число	Показывает окно как модальное
Close	Нет	Нет	Закрывает окно

Примечание. Форма может быть **модальной** и **немодальной**.

Модальная форма это такая форма, которая не позволяет открывать другие формы, пока она сама не будет закрыта.

К модальным формам обычно относятся диалоговые окна. Чтобы отобразить форму в модальном режиме необходимо вызвать метод.

По умолчанию главная форма приложения открывается в *немодальном режиме*.

Таблица 1.3 – Назначение некоторых обрабатываемых событий компонента класса ***TForm***

Событие	Описание
OnResize	Происходит при изменении размеров окна
OnShow	Происходит при появлении окна на экране
OnHide	Происходит при исчезновении окна
OnActive	Происходит при активации окна
OnDeactive	Происходит при деактивации окна

1.4.6 Уточнение заголовка формы

В инспекторе объектов (компонент ***Form1: TForm1***) исправить значение параметра ***Caption*** – заменить, например, текст ***Form1*** на ***Шаблон Иванов И.И. гр. 218-1***, после чего нажать Enter (рис.1.5).

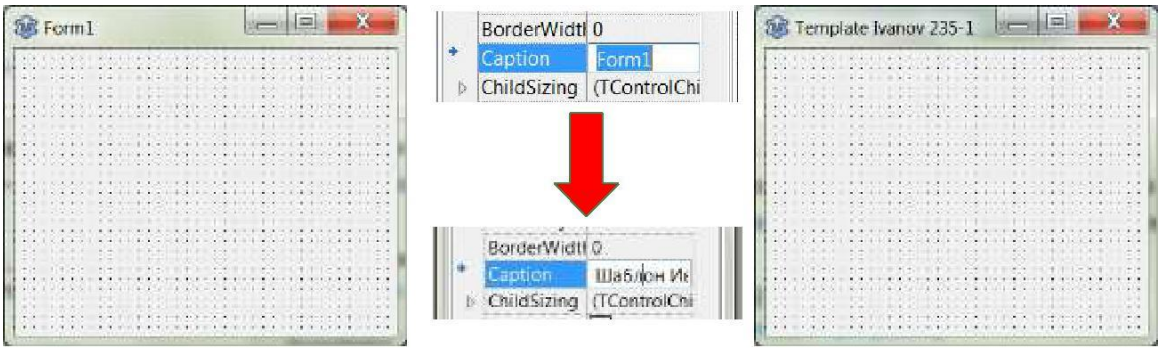


Рис. 1.5 - Уточнение заголовка формы

Примечание. Изменение размеров **Формы** выполняется стандартными действиями с помощью мышки, как это принято в *Windows*.

Чтобы изменить размеры окна формы, щёлкните ЛК на его любой угол, верхнюю, нижнюю, левую или правую границу. Если указатель превратится в соответствующую двунаправленную стрелку, показывающую, что окно формы готово к изменению размеров, то перетащите указатель в нужном направлении.

Нельзя изменить размер окна, если оно свёрнуто или развёрнуто во весь экран.

1.4.7 Формирование информационных строк

Для вывода на Форму текста, который пользователь не может изменить во время выполнения программы применяют компонент ***TLabel*** (Надпись, Метка) (рис. 1.6).

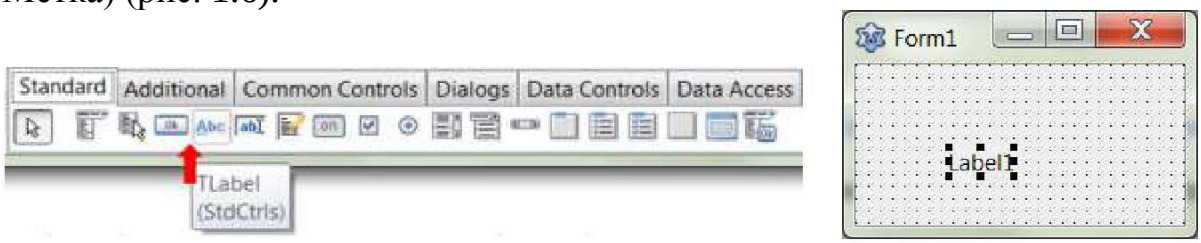


Рис. 1.6 - Положение компонента во вкладке «Standard» палитры и его вид

Таблица 1.4 знакомит с часто используемыми параметрами компонента ***TLabel.***

Таблица 1.4 – Часто используемые параметры компонента ***Label*** (Надпись)

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к этому компоненту и его свойствам.
Caption	Отображаемый текст в поле надписи.
Left	Расстояние от левой границы поля вывода до левой границы формы.
Top	Расстояние от верхней границы поля вывода до верхней границы формы.
Width, Height	Ширина и высота поля вывода (в пикселях).
AutoSize	Признак того, что размер поля определяется его содержимым. *True* – вертикальный и горизонтальный размер надписи зависит от длины текста, *False* – размеры компонента устанавливаются вручную, выравнивание текста производится свойством *Alignment*.
WordWrap	Признак того, что слова, которые не помещаются в текущей строке, автоматически переносятся на следующую строку (значение свойства *AutoSize* должно быть *False*). Если *WordWrap* имеет значение *True*, то производится перенос текста по словам, т.е. текст надписи отображается в несколько строк, иначе текст выводится в одну строку.
Alignment	Способ выравнивания текста внутри поля: *taLeftJustify* – выравнивание по левому краю; *taCenter* – выравнивание по центру; *taRightJustify* – Выравнивание по правому краю
Font	Параметры шрифта, используемые для отображения текста: *Font.Name* – вид шрифта; *Font.Size* – размер шрифта; *Font.Color* – цвет шрифта.
ParentFont	Признак наследования компонентом характеристик шрифта формы, на которой находиться компонент. Если значение свойства равно *True*, то текст выводиться шрифтом, установленным для формы.
Color	Цвет фона области вывода текста.
Transparent	Управляет отображением фона области вывода текста. Значение *True* делает область вывода текста прозрачной (область не закрашивается цветом, заданным свойством *Color*).
Visible	Позволяет скрыть текст (*False*) или сделать его видимым (*True*) во время выполнения приложения.

Примечание. Разместить нужный компонент на **Форме** можно следующими способами:

1. Дважды щёлкнуть левой кнопкой мыши (ЛК) на значке компонента, расположенного на палитре компонентов. При этом компонент попадёт в левый верхний угол Формы, а не в то место, куда Вы хотите. Но его затем можно перетащить в желаемое место с помощью мышки (как это принято в *Windows*).

2. Щёлкнуть на значке компонента (при этом он выделяется) и щёлкнуть в любое желаемое место на Форме. Таким образом, компонент можно поместить, куда было задумано.

Мы будем использовать компоненты ***TLabel*** для формирования информационных строк с назначением программы, фамилией её автора, номером варианта и условиями индивидуального задания (рис. 1.7).

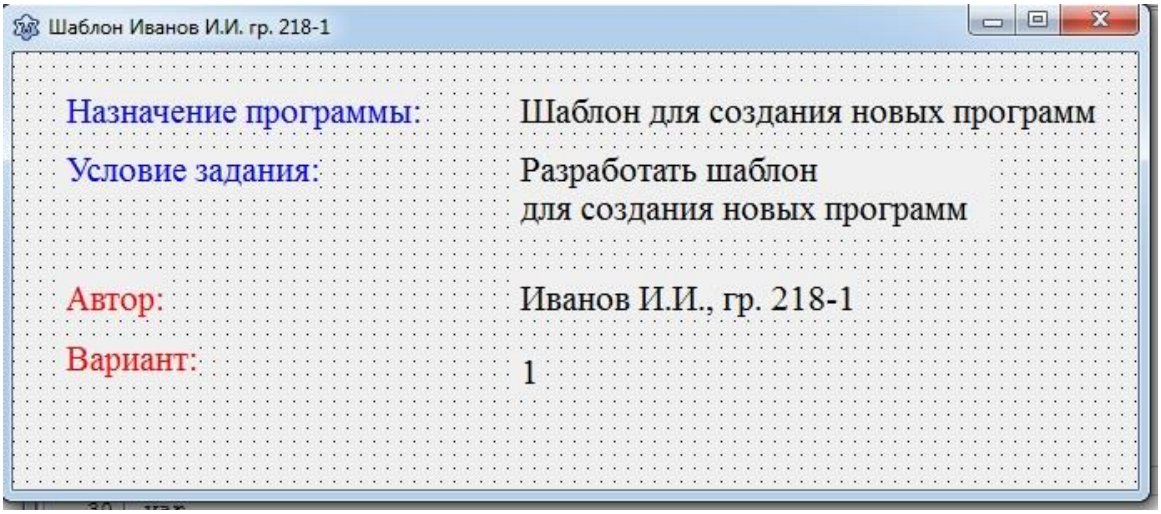


Рис. 1.7 - Создание информационных строк в Форме

Так как в создаваемой Форме использованы тексты с разным оформлением (цвет текста, начертание), то потребовалось добавить 8 компонентов ***TLabel***. Обратите внимание – как изменяется дерево компонентов в **Инспекторе объектов,** а также программный код программы в окне **Редактора исходного кода** при добавлении компонентов ***TLabel***. С помощью мышки размещаем компоненты ***TLabel*** в нужных местах Формы. Используйте вспомогательные линии, появляющиеся при перемещении этих компонентов для выравнивания надписей по вертикали и горизонтали.

Для каждого компонента вводим требуемый текст в качестве параметра свойства ***Caption***. Нажимаем кнопку ... у свойства ***Font*** и в появившемся диалоговом окне уточняем параметры шрифта отображаемого текста (рис. 1.8).

Чтобы длинный текст в условии задания мог отображаться в нескольких строках, установим значение свойства ***AutoSize*** в ***False***, а свойству ***WordWrap*** присвоим значение ***True.*** Нажмём кнопку ... свойства ***Caption*** и в открывшемся окне ***Диалог ввода строк*** отредактируем многострочный текст.

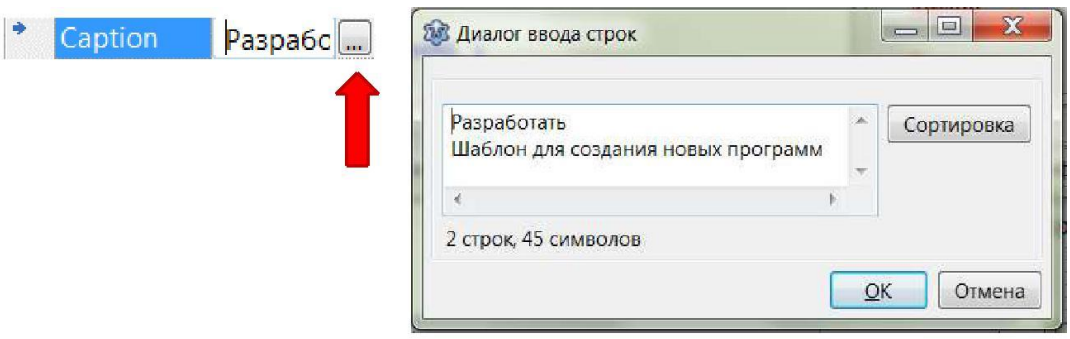


Рис. 1.8 - Редактирование многострочной надписи

Уточняем окончательно размеры Формы и надписей.

1.4.8 Создание командной кнопки

Компонент ***TButton*** (Кнопка) – командная кнопка, с помощью которой пользователь может вызывать выполнение какого-либо действия (рис. 1.9).

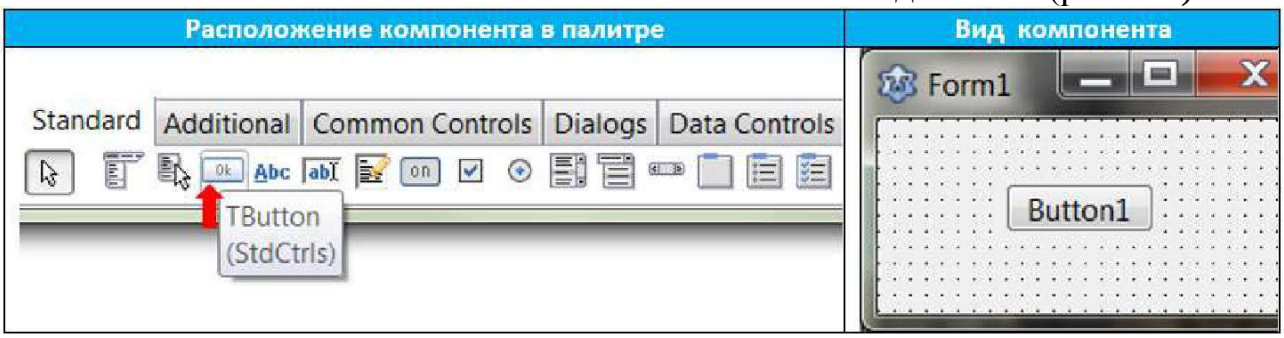


Рис. 1.9 - Положение компонента ***TButton*** (Кнопка) во вкладке «Standard» палитры и его вид

Таблица 1.5 представляет назначение основных свойств компонента ***TButton***.

Таблица 1.5 - Основные свойства компонента ***TButton*** (Кнопка)

Свойство	Описание
Name	**Имя компонента.** Используется в программе для доступа к компоненту и его свойствам.
Caption	**Текст на кнопке.**
Left	Расстояние от левой границы кнопки до левой границы формы.
Top	Расстояние от верхней границы кнопки до верхней границы формы.
idth, Height	**Ширина, высота кнопки.**
Enabled	**Признак доступности кнопки.** $True$ -кнопка доступна, $False$ – кнопка недоступна (надпись на кнопке имеет бледный вид, нажатие на кнопку не приводит ни к каким действиям, даже если имеется обработчик события $OnClick$).
Visible	**Позволяет скрыть текст.** $False$ – текст видим, $True$ – текст невидим.
Hint	**Контекстная подсказка** – текст, который появляется рядом с указателем мыши при наведении указателя (для того чтобы текст появился, надо чтобы значение свойства $ShowHint$ было $True$).
ShowHint	Разрешает ($True$) или запрещает ($False$) отображение подсказки при наведении указателя на кнопку.
Default	Если установлено значение $True$, то нажатие клавиши $Enter$ будет эквивалентно щелчку по кнопке мышью.
Cancel	Если установлено значение $True$, то нажатие клавиши Esc будет эквивалентно щелчку по кнопке мышью.

Таблица 1.6 представляет обрабатываемые события этого компонента.

Таблица 1.6 - События компонента ***TButton***

Событие	Описание
OnClick	Щелчок на кнопке
OnFocus	Получение фокуса

Щелчок по кнопке мышью вызывает событие ***OnClick***, в обработчике которого программист инициирует выполнение каких-либо действий, команд и процедур.

Вставьте в Форму ещё один компонент ***TLabel***. В окне ***Инспектора объектов*** появится новый объект ***Label9:TLabel***. В ***Инспекторе объектов*** уберите в свойстве ***Caption*** значение ***Label9*** (создаём пустую надпись). Используя диалоговое окно свойства ***Font*** для компонента ***Label9***, подберите параметры выводимого текста – шрифт, начертание, размер, цвет (рис. 1.10).

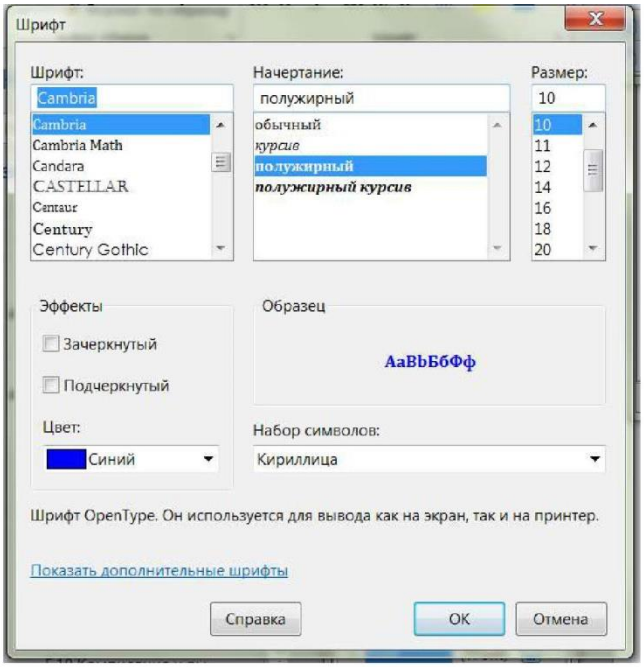


Рис.1.10 - Диалоговое окно Шрифт. Установка параметров выводимого текста

Разместите в Форме кнопку (***TButton***). Определите значение свойства ***Caption*** как **Важное сообщение**. Высоту, ширину и положение кнопки на форме отрегулируйте с помощью мышки.

Напишите программный код для процедуры обработчика события **Щелчок на кнопке.** Пока мы этого не сделаем, кнопка работать не будет - при нажатии на кнопку ничего происходить не будет.

Для этого выполните двойной щелчок по кнопке. Откроется **Редактор исходного кода,** в котором после кода, созданного автоматически, добавится новая подпрограмма-процедура (рис. 1.11) – ***TForm1.Button1Click***. Это обработчик события **Щелчок на кнопке** (англ. *Button* – кнопка, *Click* – щелчок). Пока это лишь пустая заготовка процедуры обработчика события, не содержащая ни одного оператора. Конечно, при нажатии кнопки она ничего не будет делать.

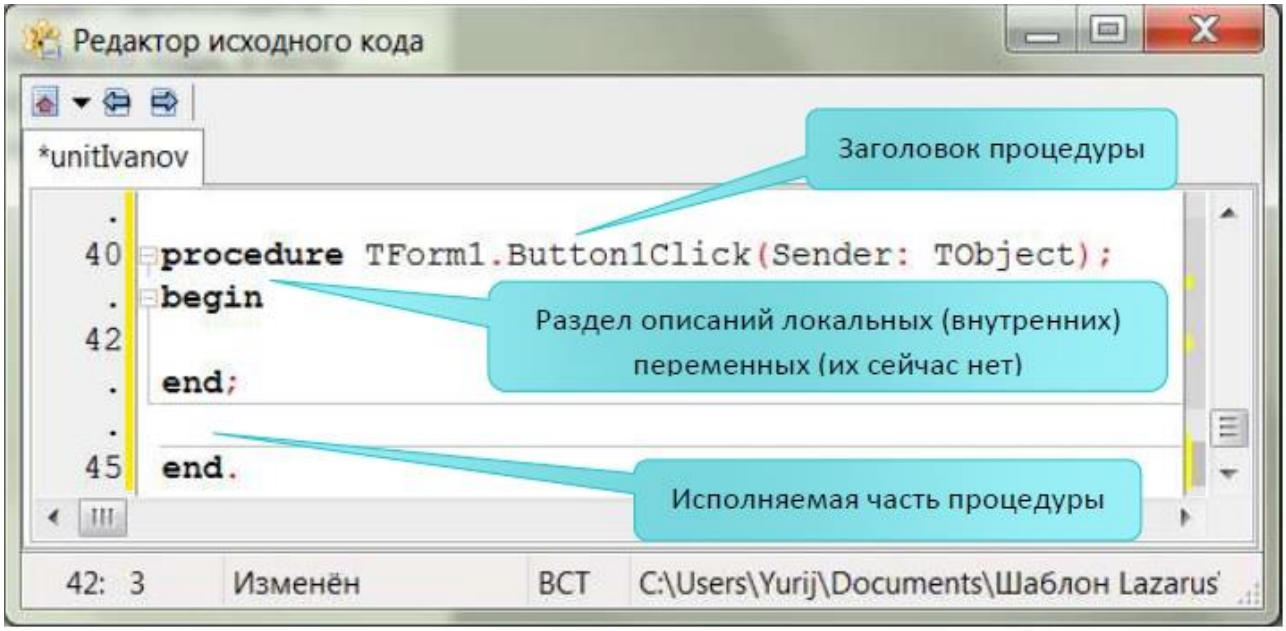


Рис. 1.11 – Пустой шаблон процедуры обработчика события **Щелчок на кнопке**

Структура подпрограммы-процедуры содержит **Заголовок, Раздел описаний** и **Исполняемую часть**:

```
Procedure <имя процедуры>(список формальных параметров);
// Раздел локальных описаний процедуры (действуют только внутри процедуры):
// констант, переменных, типов, меток, процедур, функций
begin
//<Операторы исполняемой части процедуры>
...
end;
```

Формальные параметры в заголовке процедур записываются в виде:

var *имя параметра: имя типа*

и отделяются друг от друга точкой с запятой. Если параметр не возвращает результат расчёта, то ключевое слово ***var*** может отсутствовать. Если параметры однотипны, то их имена можно перечислять через запятую, указывая общее для них имя типа. При описании параметров можно использовать только стандартные (встроенные) имена типов, либо имена пользовательских типов, определённые с помощью оператора ***type*****.**

Список формальных параметров может отсутствовать.

Процедура вызывается по имени:

< имя процедуры > (список фактических параметров);

Список фактических параметров – это их перечисление через запятую. Значение каждого фактического параметра при вызове процедуры как бы подставляется вместо формального параметра, стоящего на том же месте в заголовке.

После передачи входных параметров, управление временно передаётся процедуре – выполняются операторы исполняемой части. Возврат значений выходных параметров (только для тех, которые помечены ***var***) в вызывающую программную единицу происходит непосредственно после работы исполняемой части процедуры. После завершения работы процедуры управление возвращается в вызывающую программную единицу и её работа продолжается.

Каждый формальный параметр указывается вместе со своим типом. Соответствующий ему фактический параметр указывается без типа. Между формальными и фактическими параметрами должно быть соответствие по количеству параметров, по их типу и порядку следования.

Чтобы процедура выполнила необходимые действия, напишите соответствующий код между операторными скобками ***begin*** и ***end***. В нашем случае это оператор присваивания, который в случае щелчка ЛК мышью по кнопке **Важное сообщение** изменяет свойство ***Caption*** объекта ***Label9*** на новое значение «**Я, Иванов И.И., – студент ТУСУРа!**» вместо отсутствующего текста.

Вводя код, обратите внимание на подсказку, появляющуюся после ввода точки (нужно немного подождать), следующей за ***Label9*** (рис. 1.12). Подсказка представляет собой всплывающее меню, в котором перечислены допустимые свойства и методы компонента ***Label9***.

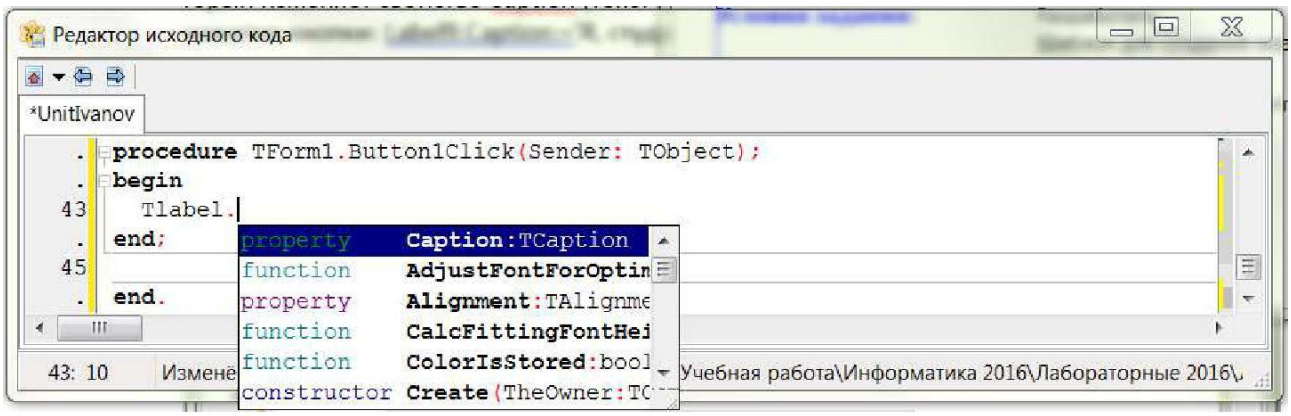


Рис. 1.12 - Всплывающее меню, в котором перечислены допустимые свойства и методы компонента ***Label9***

С помощью мыши Вы при необходимости можете выбрать из списка нужное свойство (англ. ***property***) – в нашем случае property ***Caption***, или метод (подпрограммы типов *procedure*, *function*, *constructor*). Вы также можете начать вводить имя нужного свойства или метода, при этом *Lazarus* автоматически прокручивает список и находит имена, первые буквы которых совпадают с вводимыми буквами. Это поможет Вам, если Вы забыли точное имя свойства или метода. Если теперь нажать <*Пробел*> или <*Enter*>, то *Lazarus* вместо Вас автоматически завершит ввод имени.

Окончательный вид процедуры обработчика события кнопки «**Важное сообщение**» приведён на рис. 1.13.

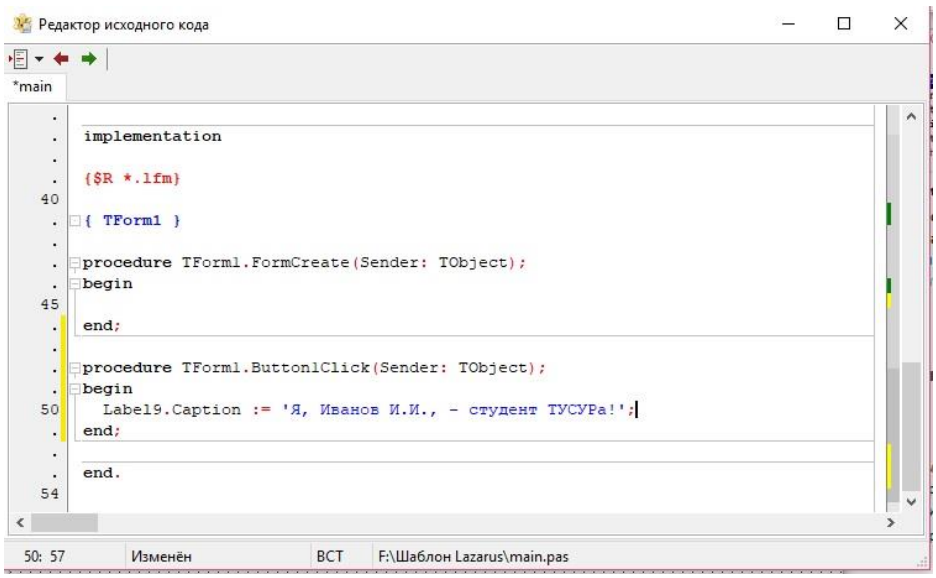


Рис. 1.13 – Окончательный вид процедуры обработчика события кнопки «Важное сообщение»

1.4.9 Создание кнопки «Завершить работу»

Добавьте из палитры на Форму ещё одну кнопку ***Button2:TButton***. Установите значение свойства ***Caption*** для этой кнопки – **Завершить работу**.

Напишите программный код для процедуры обработчика события **Щелчок на кнопке** для кнопки **Завершить работу**. Откройте **Редактор исходного кода** с помощью двойного щелчка по кнопке **Завершить работу** на Форме, в котором добавится заготовка новой процедуры ***TForm1.Button2Click*** для нашей Формы – обработчика события **Щелчок на кнопке *Button2***.

Чтобы процедура выполнила необходимые действия по закрытию окна, напишите между операторными скобками ***begin*** и ***end*** оператор вызова метода закрытия окна ***Close*** (см. таблицу 1.2). На рис. 1.14 представлен вид процедуры обработчика события кнопки **Завершить работу**.

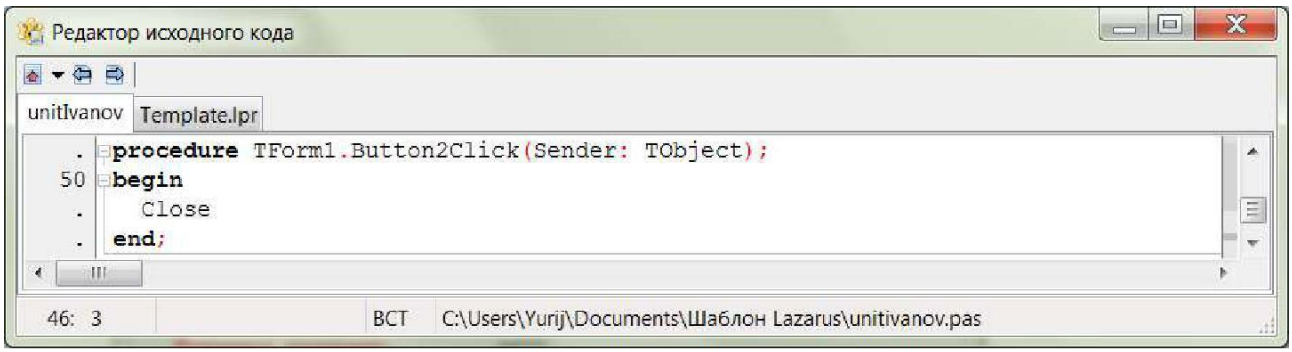
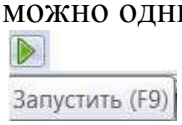


Рис. 1.14 - Вид процедуры обработчика события кнопки «Завершить работу»

1.4.10 Компиляция и выполнение программы

Выполнить программу можно одним из способов:

1. Щёлкнуть по кнопке 

на панели инструментов.
2. Выбрать команду **Запуск (Run) / Запустить (Run)** в главном меню.
3. Нажать клавишу <***F9***>.

Происходит процесс компиляции, в результате которого в папке проекта создаётся *exe*-файл. В **Окне сообщений** выводится протокол сборки проекта (рис. 1.15).

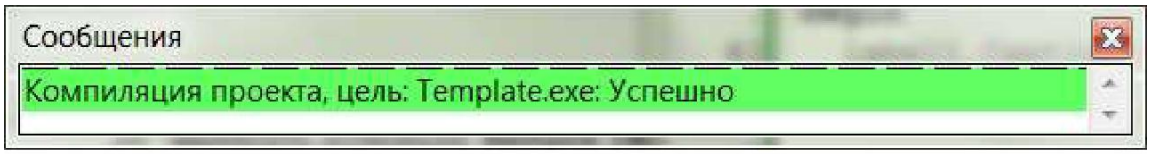


Рис. 1.15 - Сообщение об успешной компиляции

В случае, если были допущены ошибки, соответствующее сообщение об этом появится в окне сообщений.

Если компиляция прошла успешно, то на экране появится Окно с кнопками, однако пока что без надписи. Если теперь щёлкнуть ЛК по кнопке **Важное сообщение**, то в окне появится надпись «**Я, Иванов И.И., – студент ТУСУРа!**» (рис. 1.16).

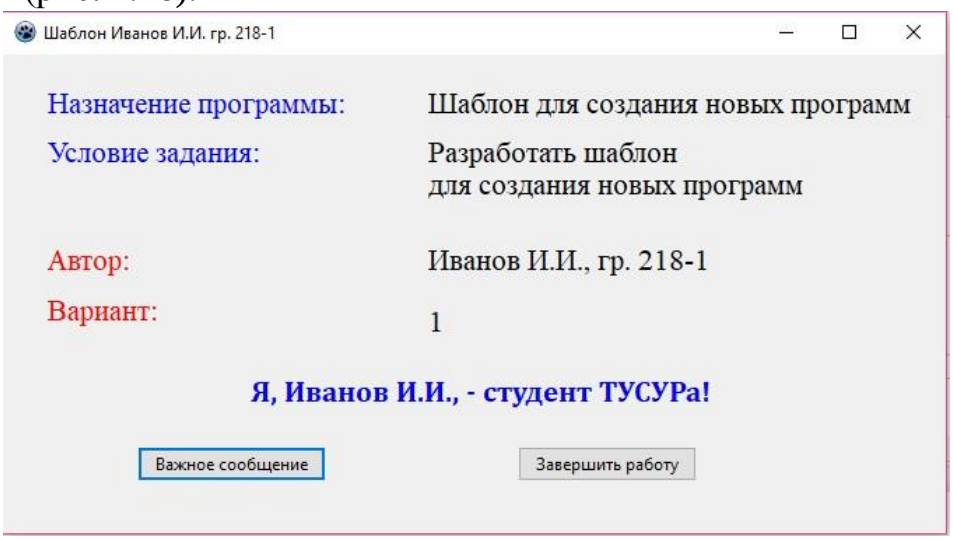


Рис. 1.16 - Результаты работы программы

Если щёлкнуть ЛК мыши по кнопке **Завершить работу**, то работа созданного Вами приложения завершится.

Таким образом, вы создали приложение, реагирующее на действия пользователя. Скомпилированная программа сохранится в папке проекта в виде файла с расширением *exe*. Он может быть выполнен на компьютере без среды разработки *Lazarus*.

1.4.11 Сохранение файлов проекта

Для этого выполните команду ***Проект / Сохранить*** или ***Файл / Сохранить***.

1.5 Контрольные вопросы

1. Дайте определение объектно-ориентированному программированию. Какими достоинствами и недостатками обладает данная технология?
2. Что собой представляют классы, объекты, интерфейсы?
3. В чем заключается сущность принципов инкапсуляции, полиморфизма, наследования?
4. Какие принципы положены в основу технологии ООП?
5. Что собой представляет визуальное проектирование интерфейса?
6. Перечислите основные элементы пользовательского интерфейса *Lazarus*.
7. Какие функции выполняет палитра компонентов в *Lazarus*?
8. Для чего используется ***Инспектор объектов***?
9. Разъясните назначение основных файлов, входящих в проект *Lazarus*.
10. Из каких компонентов состоит IDE *Lazarus*?
11. В чем отличие невизуальных от визуальных компонентов?
12. С помощью какого свойства меняется заголовок у компонента?
13. Проекты сохраняются в одном файле или нет?
14. Какие команды текстового редактора Вы знаете?
15. Что такое блок текста программы и как его выделить? Какие операции с блоками вы знаете?
16. Как осуществить контекстный поиск и замену?
17. Что представляет собой строка комментариев?
18. Объясните назначение интерфейсной секции (*interface*), секции реализации (*implementation*) и секции инициализации модуля *unit*.
19. Верно ли, что в модуле *unit*:

а) количество подпрограмм в интерфейсной секции должно совпадать с количеством подпрограмм в секции реализации;

б) количество подпрограмм в интерфейсной секции может быть меньше количества подпрограмм в секции реализации;

в) количество подпрограмм в интерфейсной секции может быть больше количества подпрограмм в секции реализации.

20. Покажите на примерах, как осуществляется:

а) запуск и выход из интегрированной среды;

б) загрузка и сохранение файлов проекта;

в) компиляция и запуск программы.

2 Лабораторная работа №2 – Консольные программы на Free Pascal

2.1 Цель работы

Знакомство с основными элементами языка программирования *Free Pascal/Lazarus*: переменными, их типами, основными операциями и функциями.

В ходе выполнения работы следует усвоить:

- структуру программ на *Free Pascal/Lazarus*;
- назначение основных объектов на *Free Pascal/Lazarus*.

Научиться:

- выбирать и объявлять основные типы данных в программах *Free Pascal*;
- усвоить правила использования констант, переменных и операторов в *Free Pascal/Lazarus*;
- освоить правила формирования выражений в *Free Pascal/Lazarus*;
- работать в консольной версии *Free Pascal*.

2.2 Порядок выполнения работы

Перед выполнением этой работы следует:

1. Ознакомиться с теоретической частью (раздел 2.5). В качестве дополнительного источника знаний использовать [7, 8].
2. Выполнить индивидуальное задание по своему варианту.
3. Загрузить систему программирования *Free Pascal/Lazarus*.
4. Войти в режим редактирования и набрать текст программы.
5. Запустить программу на трансляцию и выполнение.
6. Исправить ошибки, получить и проанализировать результаты.
7. Ответить на контрольные вопросы.
8. Оформить отчёт и защитить его у преподавателя.

2.3 Отчетность

Отчёт должен быть выполнен в соответствии с [9] и состоять из следующих разделов:

1. Тема и цель работы.
2. Индивидуальное задание.
3. Откомпилированный текст программы (в электронном виде).
4. Ответы на контрольные вопросы.
5. Выводы.

Защита лабораторной состоит в ответах на контрольные вопросы по теоретическому и практическому разделам работы. При защите отчёта по работе для получения зачёта студент должен:

– уметь отвечать на контрольные вопросы;
– показать, как работает разработанное приложение;
– продемонстрировать навыки работы в консольной версии *Free Pascal*.

2.4 Контрольные вопросы

1. Что такое консольное приложение? Чем оно отличается от визуального?
2. Перечислите стандартные типы языка *Free Pascal*. Каковы диапазоны допустимых значений для целых и вещественных типов данных?
3. Какова структура консольной программы *Free Pascal*?
4. Как описать именованные константы?
5. Для какой цели используются типизированные константы?
6. В каком порядке выполняются операции в выражениях?
7. Как работает оператор присваивания?
8. Как ввести в консольную программу данные с клавиатуры?
9. Как записываются операторы вывода на экран в языке *Free Pascal?*
10. С какой целью осуществляется форматирование результатов вывода?

2.5 Краткие теоретические сведения

Кроме визуальных приложений (англ. *Application*), имеющих графический интерфейс под Windows, *Lazarus* позволяет разрабатывать и обычные *консольные приложения* (англ. *Custom Program*), имитирующие работу под операционной системой MS-DOS с текстовым пользовательским интерфейсом (без графики), которые также могут быть созданы в оболочке *Free Pascal*.

Запуск среды программирования консольного приложения *Free Pascal* в *Lazarus* можно осуществить с помощью меню *Файл / Создать…* В появившемся диалоговом окне (рис. 2.1) выбрать *Проект / Программа* и нажать кнопку *ОК*.

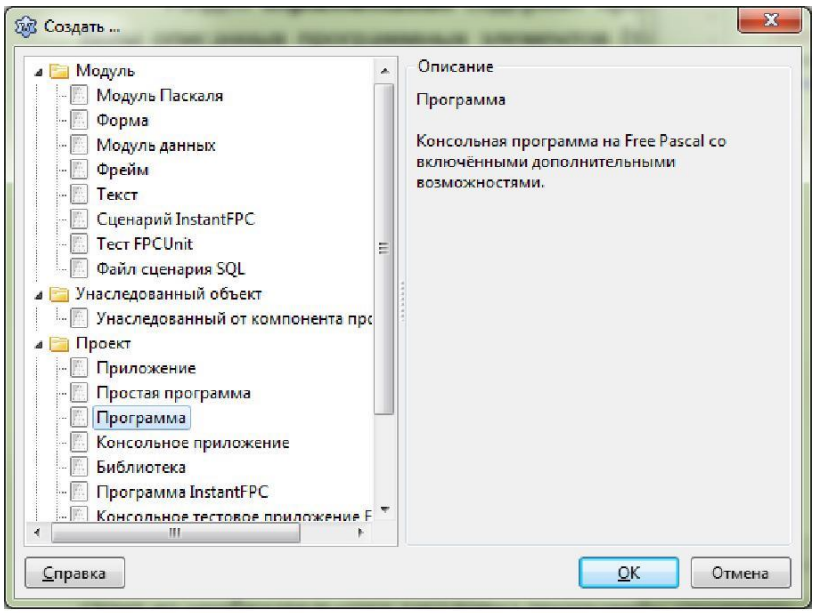


Рис. 2.1 - Диалоговое окно "Создать…"

На экране появится окно редактора исходного кода (рис. 2.2), в котором представлена заготовка программы на языке *Free Pascal*.

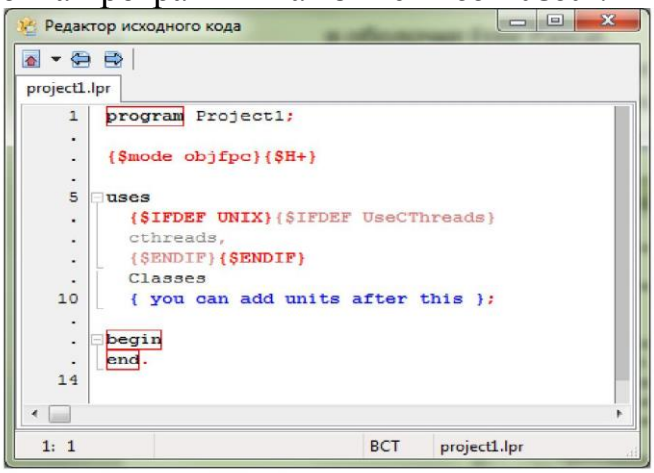


Рис. 2.2 – Заготовка программы в **Редакторе исходного кода**

Структурно консольная программа на *Free Pascal* состоит из необязательного заголовка программы (**program**), который называет программу, и основного программного блока, в котором между ключевыми словами **begin** и **end** находятся операторы, описывающие исполняемые программой действия (таблица 2.1).

Таблица 2.1 - Структура консольной программы

Оператор начала раздела	Назначение	Примечание
Program Имя_программы;	Заголовок	Имя_программы (по умолчанию Project1) при желании можно изменить.
Uses Имя_модуля_1, ..., Имя_модуля_N;	Подключение модулей, библиотек	Разделы описаний программного блока, в котором должны быть описаны все объекты, используемые в программе (в любом порядке).
Label Имя_метки_1, ..., Имя_метки_N;	Описание меток	
Const Описание_константы_1, ..., Описание_константы _N;	Описание констант	
Type Описание_типа_1, ..., Описание_типа _N;	Описание типов данных	
Var Описание_переменной_1, ..., Описание_переменной _N;	Описание переменных	
Procedure Заголовок_процедуры; ... *begin* ... {Тело процедуры} *end.*	Описание процедур, используемых в программе	
Function Заголовок_функции; ... *begin* ... {Тело функции} *end.*	Описание функций, используемых в программе	
begin ... *end.*	Тело программы	Раздел исполняемых операторов

2.6 Работа в редакторе исходного текста Free Pascal

Редактор *Free Pascal* обладает возможностями, характерными для большинства текстовых редакторов. С его помощью можно создавать и редактировать тексты программ. Кроме этого, редактор обладает возможностями подсветки синтаксиса, а также рядом других удобств.

Текст в редакторе можно выделять, копировать, вырезать, вставлять. Кроме того, в редакторе можно осуществлять поиск заданного фрагмента текста, выполнять вставку и замену.

Все допустимые операции в редакторе собраны в меню **Правка** и **Поиск** главного меню *Lazarus* (рис. 2.3). Там же приведены и «Горячие клавиши», соответствующие клавишам меню.

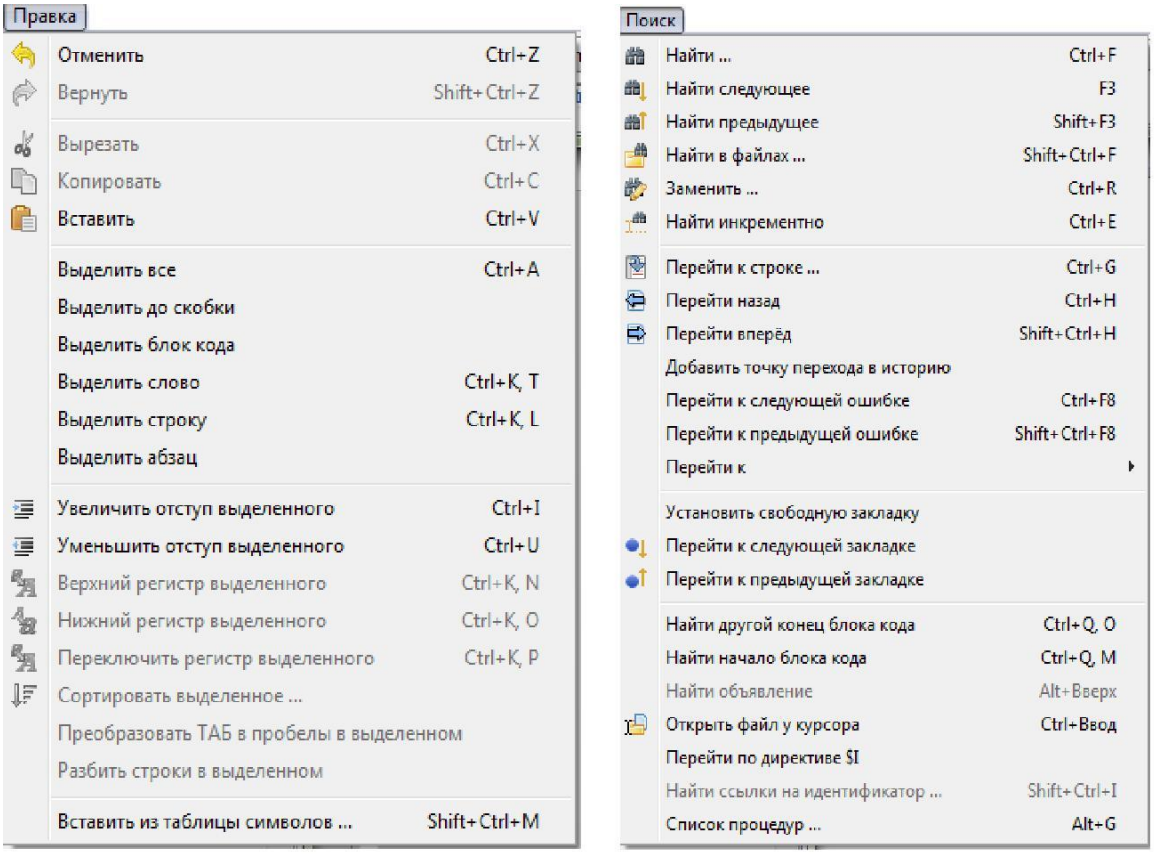


Рис. 2.3 - Меню **Правка** и **Поиск** Редактора исходного текста

Для доступа к меню можно:

1. использовать для выбора пункта меню мышь;
2. нажать **F10**, чтобы переключить фокус на меню. Затем можно использовать клавиши со стрелками для навигации по меню. Для выбора пункта меню используется клавиша *Enter*.

Контекстное меню можно вызвать щелчком правой кнопки мыши (рис. 2.4).

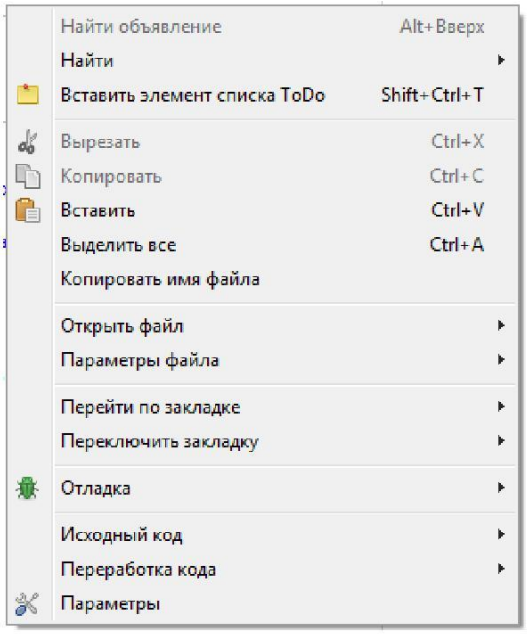


Рис. 2.4 - Контекстное меню Редактора

Для выхода из меню без выполнения каких-либо действий следует нажать клавишу *Esc*.

2.7 Пример разработки программы

```
Program Ivanov; //Заголовок программы;
{Назначение программы – Освоение работы с Free Pascal в консольном режиме}
{Автор: Иванов И.И., студент группы 218-1}
{Вариант задания:  N° 1}
{Условия задания – Рассчитать по формуле значение функции Y = f((x)}
{$mode objfpc} {$H+} //директива компилятора
Uses //Используемые модули
{$IFDEF UNIX} {$IFDEF UseCThreads} //директива компилятора
{$ENFIF} {$ENDIF} //директива компилятора
Classes
{ you can add units after this}
{Разделы описаний данных}
Const //Используемые константы
        a = 2.5; b = Pi;
Var //Используемые переменные
        Y, //Результат
        x : real; //Аргумент функции
```

```
begin {Начало основного блока программы Ivanov}
{Вывод на экран заголовка программы}
writeln('Расчёт функции Y := sqrt(abs(sin(x)/(a - b))');
writeln; //Пропуск строки
write('Введите аргумент функции x ='); //Вывод на экран запроса
Readln(x); //Ввод с клавиатуры аргумента функции
Y := sqrt(abs(sin(x)/(a - b));  //Расчёт по формуле
Writeln; //Пропуск строки
Writeln('Результат Y = ', Y : 12 : 5);
Writeln; //Пропуск строки
Writeln('Для завершения программы нажмите Enter…');
Readln;
end. {Конец основного блока программы Ivanov}
```

Примечание. По умолчанию в среде *Lazarus* используется кодировка UTF-8. Однако консольные приложения в ОС Windows используют старую кодировку MS DOS – CP866. Чтобы в консольных приложениях, создаваемых с помощью среды разработки *Lazarus*, правильно выводились русские символы, нужно:

1. Щёлкнуть правой кнопкой мыши в окне редактора исходного кода.
2. Выбрать ***Параметры файла / Кодировка / CP866***.
3. В появившемся окне нажать кнопку с надписью ***Изменить файл***, после чего не забыть сохранить изменённый файл.

2.8 Рекомендации по составлению программы

1. Составление программы целесообразно начать с создания файла консольной программы и сохранения её в предварительно подготовленном каталоге проекта. Обязательно предусмотреть:

– разумный выбор идентификаторов;

– многократный ввод данных при исполнении программы, т.е. возможность повторного счета при других исходных данных;

– простейший диалог типа «запрос-ответ» при вводе данных;

– необходимые комментарии в тексте программы;

– вывод результатов в удобном для пользователя виде.

2. Так как исходные данные, промежуточные и окончательные результаты арифметических вычислений обыкновенно содержат и дробную часть, то разумно использовать данные вещественного типа.

3. Для возведения в степень ***b*** неотрицательного и не равного нулю выражения (не существует логарифмов от отрицательных чисел) во *Free Pascal* можно воспользоваться следующим преобразованием:

$$a^b = exp(b*ln(a)).$$

4. Извлечь корень степени ***n*** из числа ***a*** (при ***a*** > 0) означает возвести число ***a*** в степень 1/***n*** . Для извлечения корня ***n*** из неотрицательного и не равного нулю выражения можно воспользоваться во *Free Pascal* следующим преобразованием:

$$\sqrt[n]{a} = \exp(\ln(a)/n).$$

5. Выразить десятичный логарифм через натуральный можно преобразованием:

$$lg(a) = ln(a)/ln(10).$$

2.9 Индивидуальные задания

1. Для своего варианта (таблица 2.2) написать консольную программу на языке *Free Pascal* для вычисления функции, значение аргумента *x* для которой вводят с клавиатуры.

2. Определить порядок выполнения операций в формуле, пронумеровав их.

3. Произвести тестирование программы, созданной в консольной версии *Free Pascal* при заданных значениях констант и двух значениях переменной.

Таблица 2.2 - Варианты индивидуальных заданий

№ варианта	Формула	Константы		Переменная	
		a	*b*	x_1	x_2
1.	$Y := \sqrt[5]{\frac{lg(a+x)}{b^{a+x}+b}} + e^{5\cdot a}$	6	10	0.5	0
2.	$Y := \frac{sin(ln(a+b\cdot x)) + (b+x)^3}{a\cdot x^2 + b\cdot x - 1}$	1.7	0.5	1.5	1.8
3.	$Y := a - \frac{lg(a+x)\cdot x^b}{cos(\pi\cdot x)}$	3.0	-0.5	1.3	2.0
4.	$Y := \frac{b^5 - tg\,x}{\ln x}$		1.5	0.3	2.0
5.	$Y := \frac{\left(\frac{a}{b} - 3^{a\cdot x}\right)\cdot \cos x}{\sqrt{x+2}}$	1.5	-100	0.8	1.3
6.	$Y := \frac{\left(a + \sqrt[5]{x+3}\right)\cdot (x+b)}{\ln b}$	1.0	2.0	-0.5	0.5
7.	$Y := \frac{b\cdot e^{(x-b)}}{tg(5\cdot x)} + \frac{a}{b}$	π	5.0	6.0	6.5
8.	$Y := 3\cdot \frac{x^b - a^x}{b^2 + a^x}$	π/2	-0.9	1.0	2.0
9.	$Y := \frac{b\cdot ln(\lvert a+x\rvert)}{\sqrt{a^b} - \sin b}$	π/3	4.0	-3.0	3.0
10.	$Y := \frac{sin^2 x - a^x}{\sqrt{\pi\cdot x - 4} - a}$	2π/3		2.0	2.5
11.	$Y := \frac{\sqrt[5]{x} + \frac{1}{a}}{lg\left(\left\lvert\sqrt[5]{x} + \frac{1}{a}\right\rvert\right)}$	π		1.5	3.5
12.	$Y := \frac{\frac{1}{\sqrt{\lvert x+1\rvert + a}} + \frac{1}{a}}{arctg(a-x)}$	4.0		-6.0	-5.0
13.	$Y := \frac{lg^2(x+a) - \lvert x+a\rvert}{cos^2(x+a)}$	5.0		10.0	0
14.	$Y := \frac{\frac{1}{\cos x} + \frac{1}{cos^3(x^2)} + b}{e^{2\cdot b\cdot x}}$	0.5	-3.0	0.9	1.2

Окончание таблицы 2.2

№ варианта	Формула	Константы a	Константы b	Переменная x_1	Переменная x_2
15.	$Y:=\frac{sin(a\cdot x)+\lvert x^{10}\rvert}{cos(b\cdot x)+\frac{1}{5}}$	-π	π	1.5	1.7
16.	$Y:=\frac{ln(x^{2.5}+a)-\sqrt{lg(x^2)}}{arctg\frac{x}{b}}$	14.0	200.0	10.0	5.0
17.	$Y:=\frac{e^{0.01\cdot a}-cos(b\cdot x)\cdot\lvert x\rvert}{lg^5\lvert x\rvert+\frac{1}{x}}$	50.0	π/3	5.5	-4.0
18.	$Y:=\frac{e^{sin(a\cdot x)}+cos(b\cdot x)}{5\cdot cos^2(a\cdot x)-a}+\frac{1}{x}$	π	2π	0.8	0.6
19.	$Y:=\frac{\lg x+\lvert x\rvert}{a\cdot x^2}\cdot b+x^{2.5}$	-2.0	3.0	4.0	3.0
20.	$Y:=\left\lvert\frac{cos(2\cdot x)}{a+\sin x}\right\rvert+x^2\cdot\sqrt{\lvert\sin x\rvert}$	-1.0	2.3	8.0	5.0
21.	$Y:=\frac{x^3}{cos(a\cdot x)+sin(b\cdot x)}+\lvert x\cdot e^x\rvert$	-π	π	5.0	4.0
22.	$Y:=sin(x)\cdot a+\frac{lg(b+x)}{\sqrt{b}}$	-1.0	1.5	0	1.0
23.	$Y:=\frac{\frac{tg\,x}{\ln x}+\frac{\lg x}{a}}{\lvert b\cdot x\rvert}$	2	-2.5	2.0	3.0
24.	$Y:=tg(x)\cdot\frac{lg(\lvert b+x\rvert)}{\sqrt{a\cdot x+b}}$	10.0	-9.0	0	2.2
25.	$Y:=\frac{\sqrt{(a\cdot x)^2+b}}{a^{0.1\cdot b}-lg(\lvert b\cdot x\rvert)}$	5.0	6.0	-4.0	3.0
26.	$Y:=\frac{x^b\cdot\cos x}{\sqrt{b-x}+a}$	-1.5	3.0	0.5	1.0
27.	$Y:=\frac{arctg(b\cdot x)+\lvert x\rvert}{\sqrt{b-x}+a}$	-3.0	5π	1.8	2.0
28.	$Y:=\frac{\frac{1}{a+x}+e^{\lvert\sin x\rvert}}{lg(b-x)}$	-8.0	7.0	0.1	0.3
29.	$Y:=\frac{tg(a+x)-\frac{1}{lg(a\cdot x)}}{\sqrt[5]{b\cdot x}}$	1.0	2.0	3.0	4.0

3 Лабораторная работа №3 – Операторы присваивания

3.1 Цель работы

Цель работы: освоение основных элементов интегрированной среды визуального программирования *Lazarus* и разработка с её помощью простейших приложений.

В ходе выполнения работы следует усвоить:

- структуру программ на *Lazarus*;
- назначение базовых объектов на *Lazarus*.

Научиться:

- работать в интегрированной среде *Lazarus*;
- усвоить правила использования основных типов данных (констант, переменных) и операторов в *Lazarus*;
- освоить приёмы программирования и отладки простейших линейных алгоритмов.

3.2 Порядок выполнения работы

Перед выполнением этой работы следует:

1. Ознакомиться с теоретической частью (раздел 3.5). В качестве дополнительного источника знаний можно использовать [7, 8].
2. Выполнить индивидуальное задание по своему варианту.
3. Войти в свой личный каталог, загрузить и настроить систему программирования *Lazarus*.
4. Войти в режим редактирования и набрать текст программы.
5. Запустить программу на трансляцию и выполнение.
6. Исправить ошибки с помощью отладки, чтобы программа давала правильные результаты.
7. Ответить на контрольные вопросы.
8. Оформить отчёт и защитить его у преподавателя.

3.3 Отчетность

Отчёт должен быть выполнен в соответствии с [9] и состоять из следующих разделов:

1. Тема и цель работы.
2. Индивидуальное задание.
3. Блок-схема алгоритма.
4. Откомпилированный текст программы (в электронном виде).
5. Ответы на контрольные вопросы.
6. Результаты выполнения программы.
7. Выводы.

При защите отчёта по работе для получения зачёта студент должен:

– уметь отвечать на контрольные вопросы;

– обосновать структуру выбранного алгоритма и доказать, что разработанное приложение работает правильно;

– уметь пояснить назначение элементов разработанного приложения;

– продемонстрировать навыки работы в интегрированной среде *Lazarus*.

3.4 Контрольные вопросы

1. Дайте определение объектно-ориентированному программированию. Какими достоинствами и недостатками обладает данная технология?
2. Что собой представляют классы, объекты, интерфейсы?
3. В чем заключается сущность принципов инкапсуляции, полиморфизма, наследования?
4. Какие принципы положены в основу технологии ООП?
5. Что собой представляет визуальное проектирование интерфейса?
6. Перечислите основные элементы пользовательского интерфейса *Lazarus*.
7. Какие функции выполняет палитра компонентов в *Lazarus*?
8. Для чего используется **Инспектор объектов**?
9. Разъясните назначение основных файлов, входящих в проект *Lazarus*.
10. Какой алгоритм называется линейным? Из каких операторов он состоит?

3.5 Линейные программы

Решение любой задачи достигается обработкой информации или данных. Поэтому, как программисту, Вам необходимо знать, как:

- осуществить ввод данных – т.е. ввести информацию (данные) в программу с клавиатуры, диска или из порта ввода/вывода;
- сохранить данные (информацию) во внутренней (оперативной) или внешней (экран, магнитные и оптические диски и другие устройства ввода-вывода) памяти. Это достигается использованием констант, переменных и структур, содержащих числа (целые и вещественные), текст (символы и строки) или указатели – адреса переменных и структур;
- правильно задать команды обработки данных (операторы, инструкции). С их помощью осуществляют присваивание значений, вычисление выражений, сравнение значений (равно, не равно, больше и т.д.);
- получить данные из программы (вывод данных) на экран, на диск или в порт ввода/вывода.

Вы можете написать и упорядочить операторы так, чтобы:

- некоторые из них выполнялись при выполнении определённого условия или ряда условий (условное выполнение);
- другие выполнялись определённое число раз (циклы), пока истинно некоторое условие, или пока условие не станет истинным;

- другие собирались в отдельные части, объединённые именем (подпрограммы), которые могут быть выполнены в нескольких местах программы, где есть вызов их по имени.

При любых значениях исходных данных в линейном алгоритме все действия ***A1, A2, …. AN*** выполняются однократно, строго последовательно, одно за другим (рис. 3.1). Такой порядок выполнения действий называется *естественным*.

Рис. 3.1 – Линейный алгоритм

Программы с линейной структурой являются простейшими и используются в чистом виде на практике достаточно редко: при расчёте обычных арифметических и алгебраических выражений и решении ряда простейших задач.

Алгоритм линейной структуры представляет собой последовательность действий и не содержит каких-либо условий.

В линейных программах могут применяться только:

- операторы (процедуры) ввода,
- операторы (процедуры) вывода,
- операторы присваивания (изменения значения переменных),
- операторы обращения к подпрограммам.

Рассмотрим важнейшие элементы программы на языке *Lazarus*, используемые при создании линейных программ.

Оператор присваивания является основным вычислительным оператором. Если в программе надо выполнить вычисление, то нужно использовать оператор присваивания.

В результате выполнения оператора присваивания значение переменной меняется, ей присваивается новое значение.

В общем виде инструкция присваивания выглядит так:

< *Имя* > := < *выражение* >

где *Имя* – имя переменной, значение которой изменяется в результате выполнения оператора присваивания; : = символ присваивания; *выражение* – выражение, значение которого присваивается переменной, имя которой указано слева от символа оператора присваивания.

Оператор присваивания выполняется следующим образом:

- сначала вычисляется значение выражения, которое находится справа от символа присваивания (***: =***);

- затем вычисленное значение записывается в переменную, имя которой стоит слева от символа присваивания.

Например, в результате выполнения операторов:

i:= 0; {значение переменной ***i*** становится равным нулю}

a:=*b*+*c*; {значением переменной ***a*** будет суммой значений переменных ***b*** и ***c***}

j:=*j* +1; {значение переменной ***j*** увеличивается на единицу}

Выражение должно быть совместимо по присваиванию с типом переменной. Оператор присваивания считается верным, если тип выражения соответствует или может быть приведён к типу переменной, получающей значение. Например, переменной типа ***real*** можно присвоить значение выражения, тип которого ***real*** или ***integer***, а переменной типа ***integer*** можно присвоить значение выражения только типа ***integer***.

Так, например, если переменные ***i*** и ***n*** имеют тип ***integer***, а переменная ***d*** – тип ***real***, то

i := n / 10; i := 1.0; {операторы неправильные}

d := i +1; {операторы записаны правильно}

Любая программа в своей работе использует какие-то исходные данные. Для организации ввода можно использовать компонент ***TEdit*** (Поле ввода), для вывода результатов – компонент ***TLabe***l (Поле вывода). Компонент ***TEdit*** представляет из себя поле ввода (редактирования) строки символов (рис. 3.2).

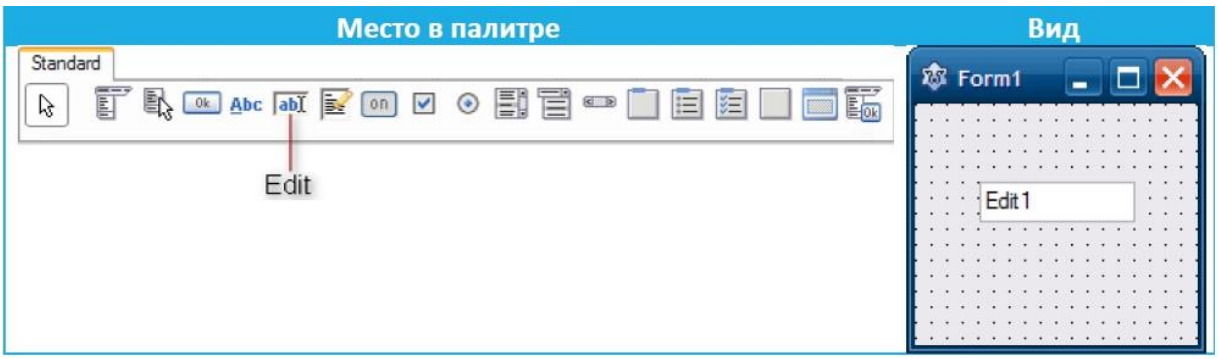


Рис.3.2 – Компонент ***TEdit*** и его место в Палитре инструсентов

Таблица 3.1 представляет основные свойства компонента ***TEdit.***

Таблица 3.1 - Основные свойства компонента ***TEdit*** (поле ввода строки символов)

Свойство	Описание
Name	**Имя компонента.** Используется в программе для доступа к компоненту и его свойствам, в частности для доступа к тексту, введённому в поле редактирования.
Text	**Текст**, находящийся в поле ввода и редактирования.
Left	Расстояние от левой границы компонента до левой границы формы.
Top	Расстояние от верхней границы компонента до верхней границы формы.
Width, Height	**Ширина, высота поля.**
Font	**Шрифт**, используемый для отображения вводимого текста.
ParentFont	Признак наследования компонентом характеристик шрифта формы, на которой находиться компонент. Если значение свойства равно *True*, то при изменении свойства *Font* формы автоматически меняется значение свойства *Font* компонента.
Enabled	Используется для ограничения возможности изменить текст в поле редактирования. Если значение свойства равно *False*, то текст в поле редактирования изменить нельзя.
Visible	Позволяет скрыть текст (*False*) или сделать его видимым (*True*).

Чтобы ввести численные данные с помощью компонент ***TEdit*** (Поле ввода), а для вывода численных результатов – компонент ***TLabel*** (Поле вывода) – следует подключить специальные функции перевода из символьного представления в численное (таблица 3.2) и наоборот (таблица 3.3).

Таблица 3.2 - Преобразование строк в другие типы

Обозначение	Тип аргументов	Тип результата	Действие
StrToDateTame(S)	Строка	Дата и время	Преобразует символы из строки ***S*** в дату и время
StrToFloat(S)	Строка	Вещественное число	Преобразует символы из строки ***S*** в вещественное число
StrToInt(S)	Строка	Целое число	Преобразует символы из строки ***S*** в вещественное число
Val(S, X, Kod)	Строка	Число	Преобразует троку символов ***S*** во внутреннее представление числовой переменной X. Если преобразование прошло успешно, ***Kod*** = 0.

Таблица 3.3 - Обратное преобразование

Обозначение	Тип аргументов	Тип результата	Действие
DataTimeToStr(V)	Дата и время	Строка	Преобразует дату и время в строку ***S***
FloatToStr(V)	Вещественное число	Строка	Преобразует вещественное число в строку ***S***
IntToStr(V)	Целое число	Строка	Преобразует целое число в строку ***S***
FloatToStrF(V, F, P, D)	Вещественное число	Строка	Преобразует вещественное число ***V*** в строку символов с учётом формата ***F*** и параметров ***P, D***.

3.6 Пример разработки программы

Последовательность стандартных шагов: *ввести, сохранить, отредактировать* и *выполнить* – определяет общий сценарий разработки практически любой программы. Рассмотрим, как можно выполнить любой из этих шагов.

ЗАДАЧА. Известны длины сторон треугольника a,b и c. Вычислить площадь S, периметр P и величины углов α,β и γ треугольника (рис. 3.3).

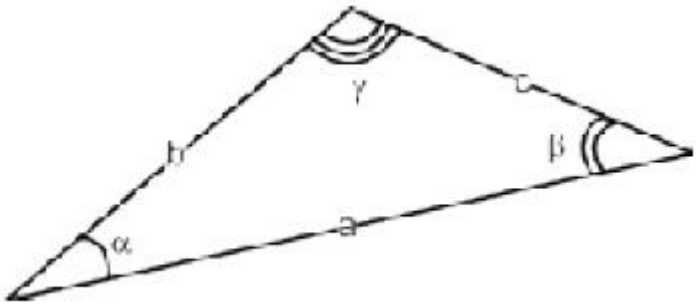


Рис. 3.3 – Треугольник

Прежде чем приступить к написанию программы, вспомним математические формулы, необходимые для решения задачи.

Для вычисления площади треугольника применим теорему Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

где $p = \frac{a+b+c}{2}$ – полупериметр.

Один из углов найдём по теореме косинусов:

$$\cos(\propto) = \frac{b^2 + c^2 - a^2}{2bc}$$

второй — по теореме синусов:

$$\sin(\beta) = \frac{b}{a}\sin(\alpha)$$

третий — по формуле:

$$\gamma = \pi - (\alpha + \beta).$$

Решение задачи можно разбить на следующие этапы:

1. Определение значений a,b и c (ввод величин a, b и c в память компьютера).

2. Расчёт значений S, P, α, β и γ по приведённым формулам.

3. Вывод значений S, P, α, β и γ.

Внешний вид программы каждый может разработать самостоятельно. Например, разместить на форме десять меток, три поля ввода и одну кнопку (рис. 3.4).

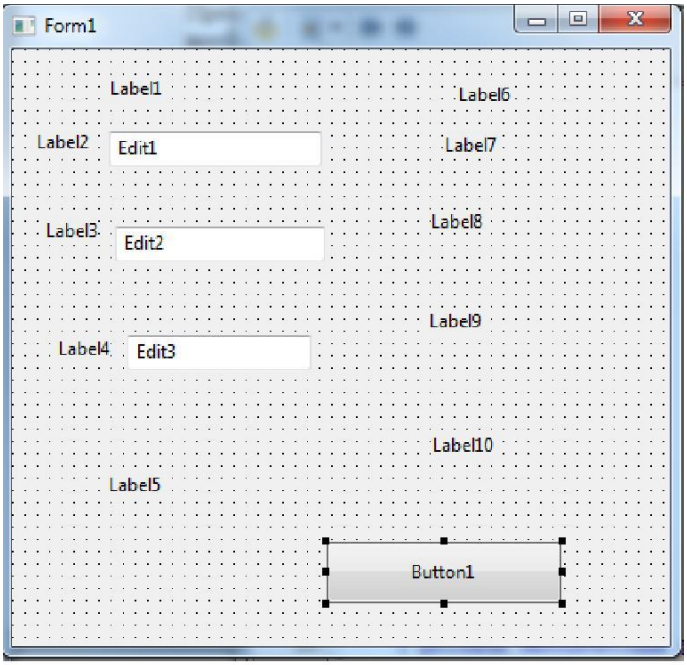


Рис. 3.4 - Вид формы с компонентами

Потребуется изменить заголовки компонентов (свойство ***Caption***) в соответствии с рис. 3.5. Как это делать – мы уже знаем из лабораторной работы №1.

Рис. 3.5 - Интерфейс программы

Двойной щелчок по кнопке **Вычислить** приведёт к созданию процедуры ***TForm1.Button1Click***.

Задача программиста заполнить шаблон описаниями и операторами. Все команды, указанные в процедуре между словами ***begin*** и ***end***, будут выполнены при щелчке по кнопке **Выполнить.**

В нашем случае процедура ***TForm1.Button1Click*** может иметь вид:

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
//Описание переменных (все переменные вещественного типа):
var
   a, b, c, {стороны треугольника}
   alfa, beta, gamma, {углы треугольника}
   S,  {площадь треугольника}
   p :  real; {полупериметр треугольника}

begin
//Из полей ввода Edit1, Edit2, Edit3 считываются введенные строки,
//с помощью функции StrToFloat(x) преобразовываются в вещественные числа
//и записываются в переменные a, b, c.
   a := StrToFloat(Edit1.Text);
   b := StrToFloat(Edit2.Text);
   c := StrToFloat(Edit3.Text);
   p := (a + b + c) / 2; //Вычисление значения полупериметра
//При вычислении значения площади применяется
// функция sqrt(x) – корень квадратный из x.
   S := sqrt(p * (p-a) * (p-b) * (p-c));
//При вычислении значения угла alfa в радианах применяем функции:
// arccos(x) - арккосинус x и sqr(x) – возведение x в квадрат
   alfa := arccos((sqr(b) + sqr(c) - sqr(a)) / 2 / b / c);
//При вычислении значения угла beta в радианах применяем функцию:
// arcsin(x) - арксинус x;
   beta := arcsin(b / a * sin(alfa));
//Вычисление значения угла gamma в радианах.
//Математическая постоянная pi – встроена в язык программирования.
   gamma := pi - (alfa + beta);
//Перевод радиан в градусы
  alfa := alfa * 180 / pi;
  beta := beta * 180 / pi;
   gamma := gamma *180 / pi;
//Для вывода результатов вычислений используем
//операцию слияния строк ≪+≫ и функцию FloatToStrF(x), которая
//преобразовывает вещественную переменную x в строку
//и выводит ее в указанном формате.
```

```
//В нашем случае под переменную отводится три позиции,
//включая точку и ноль позиций после точки.
//Величины углов в градусах выводятся на форму
//в соответствующие объекты типа надпись.
  Label6.Caption := 'alfa = ' + FloatToStrF(alfa, ffFixed, 3,0);
  Label7.Caption := 'betta = ' + FloatToStrF(beta,ffFixed,3,0);
  Label8.Caption := 'gamma=' + FloatToStrF(gamma,ffFixed,3,0);
//Используем функцию FloatToStrF(x) для форматированного вывода.
//В нашем случае под все число отводится пять позиций,
//включая точку, и две позиций после точки.
//Значения площади и периметра выводятся на форму.
  Label9.Caption := 'Периметр P = ' + FloatToStrF(2 * p, ffFixed, 5, 2);
  Label10.Caption := 'Площадь S = ' + FloatToStrF(S, ffFixed, 5, 2);
end;
```

Обратите внимание, что было написано всего десять команд, предназначенных для решения поставленной задачи, все остальное – комментарий, который писать необязательно.

Примечание. В списке встроенных функций среды *Lazarus* не хватает таких функций как: возведение числа в произвольную степень; извлечение произвольного корня из числа; обратные тригонометрические функции и т.д. Для расширения математических возможностей программ есть подключаемый модуль Math. Перед началом работы следует подключить этот модуль в разделе uses:

```
uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls, Math;
```

3.7 Варианты заданий для составления линейных программ

После запуска IDE *Lazarus* создайте подкаталог для Вашего нового проекта. Затем загрузите ранее сделанный Вами шаблон программы. Возможно у Вас появятся новые мысли по модернизации программ-шаблона. Откорректируйте его, проверьте и сохраните под тем же именем.

Чтобы не испортить шаблон программы, переименуйте его и запишите в созданный подкаталог под новым именем, которое Вы пожелаете дать разрабатываемой программе.

При составлении программы обязательно предусмотреть:

- разумный выбор идентификаторов;
- многократный ввод данных при исполнении программы, т.е. возможность повторного счета при других исходных данных;
- простейший диалог типа «запрос-ответ» при вводе данных;
- необходимые комментарии в тексте программы;

- вывод результатов в удобном для пользователя виде (отформатированные результаты, размерность, цвет и т.п.);

- подготовку тестового примера, позволяющего доказать правильность работы Вашей программы.

Вариант 1. Вычислить медианы треугольника со сторонами *a, b, c* по формулам:

$$m_a = 0.5\sqrt{2b^2 + 2c^2 - a^2}$$
$$m_b = 0.5\sqrt{2a^2 + 2c^2 - b^2}$$
$$m_c = 0.5\sqrt{2a^2 + 2b^2 - c^2}$$

Вариант 2. Вычислить биссектрисы треугольника со сторонами *a, b, c* по формулам:

$$\beta_a = \frac{2\sqrt{b * c * p(p-a)}}{b+c}$$
$$\beta_b = \frac{2\sqrt{a * c * p(p-b)}}{a+c}$$
$$\beta_c = \frac{2\sqrt{a * b * p(p-c)}}{b+a}$$
$$p = \frac{a+b+c}{2}$$

Вариант 3. Вычислить координаты центра тяжести трех материальных точек с массами *m*1, *m*2, *m*3 и координатами (*x*1, *y*1), (*x*2, *y*2), (*x*3, *y*3) по формулам:

$$x_c = \frac{m1 * x1 + m2 * x2 + m3 * x3}{m1 + m2 + m3}$$
$$y_c = \frac{m1 * y1 + m2 * y2 + m3 * y3}{m1 + m2 + m3}$$

Вариант 4. Вычислить координаты точки, делящей отрезок *a*1*a*2 в отношении *n*1:*n*2 по формулам:

$$x = \frac{x1 + \lambda * x2}{1+\lambda}$$
$$y = \frac{y1 + \lambda * y2}{1+\lambda}$$
$$\lambda = \frac{n1}{n2}$$

Вариант 5. Вычислить площадь поверхности $S = \pi*(R + r)*l + \pi*R^2 + \pi*r^2$ и объем усечённого конуса $V = \frac{\pi*h*(R^2+r^2+R*r)}{3}$, где *R* и *r* – радиусы верхнего и нижнего оснований; *l* - образующая конуса; *h* – высота конуса.

Вариант 6. Составить программу для вычисления расстояний между двумя точками с координатами (*x*1, *y*1, *z*1) и (*x*2, *y*2, *z*2) в трёхмерном пространстве по формуле:

$$d = \sqrt{(x2-x1)^2 + (y2-y1)^2 + (z2-z1)^2}$$

Вариант 7. Составить программу для вычисления значений функций:

$$y = \frac{e^{-x1} + e^{-x2}}{2}$$

$$z = \frac{a\sqrt{x1} - b\sqrt{x2}}{c}$$

$$x1 = \frac{b + \sqrt{|b^2 - 4ac|}}{2a}$$

$$x2 = \frac{b - \sqrt{|b^2 - 4ac|}}{2a}$$

Вариант 8. Написать программу вычисления стоимости поездки на автомобиле на дачу (туда и обратно). Исходными данными являются: расстояние до дачи (в километрах); количество бензина, которое потребляет автомобиль на 100 км пробега; цена одного литра бензина.

Вариант 9. Три сопротивления $R1$, $R2$ и $R3$ соединены параллельно. Найти общее сопротивление соединения $R0$.

Вариант 10. Даны два числа. Найти среднее арифметическое кубов этих чисел и среднее геометрическое модулей этих чисел.

Вариант 11. Вычислить расстояние между двумя точками с данными координатами $(x1, y1)$ и $(x2, y2)$.

Вариант 12. Дана сторона равностороннего треугольника. Найти площадь этого треугольника, его высоту, радиусы вписанной и описанной окружностей.

Вариант 13. Заданы координаты трёх вершин треугольника $(x1, y1)$, (x2, $y2$), $(x3, y3)$. Найти его периметр и площадь.

Вариант 14. Вычислить значения функций:

$$z_1 = 2sin^2(3\pi - 2\alpha)cos^2(5\pi + 2\alpha)$$

$$z_2 = \frac{1}{4} - \frac{1}{4}\sin\left(\frac{5}{2}\pi - 8\alpha\right)$$

Вариант 15. Вычислить значения функций:

$$z_1 = \cos\alpha + \sin\alpha + \cos 3\alpha + \sin 3\alpha$$

$$z_2 = 2\sqrt{2}\cos\alpha * \sin\left(\frac{\pi}{4} + 2\alpha\right)$$

Вариант 16. Вычислить значения функций:

$$z_1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos\alpha + 1 - 2sin^2 2\alpha}$$

$$z_2 = 2\sin\alpha$$

Вариант 17. Вычислить значения функций:

$$z_1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos \alpha - \cos 3\alpha + \cos 5\alpha}$$

$$z_2 = tg\ 3\alpha$$

Вариант 18. Вычислить значения функций:

$$z_1 = 1 - \frac{1}{4} sin^2 2\alpha + \cos 2\alpha$$

$$z_2 = cos^2\alpha + cos^4\alpha$$

Вариант 19. Вычислить значения функций:

$$z_1 = \cos\alpha + \cos 2\alpha + \cos 6\alpha + \cos 7\alpha$$

$$z_2 = 4\cos\frac{\alpha}{2} \cdot \cos\frac{5}{2}\alpha \cdot \cos 4\alpha$$

Вариант 20. Вычислить значения функций:

$$z_1 = \cos^2\left(\frac{3}{8}\pi - \frac{\alpha}{4}\right) - \cos^2\left(\frac{11}{8}\pi + \frac{\alpha}{4}\right)$$

$$z_2 = \frac{\sqrt{2}}{2}\sin\frac{\alpha}{2}$$

Вариант 21. Вычислить значения функций:

$$z_1 = \cos^4 x + \sin^2 y + \frac{1}{4}\sin^2 2x - 1$$

$$z_2 = \sin(y + x) \cdot \sin(y - x)$$

Вариант 22. Вычислить значения функций:

$$z_1 = (\cos\alpha - \cos\beta)^2 - (\sin\alpha - \sin\beta)^2$$

$$z_2 = -4\sin^2\frac{\alpha - \beta}{2} \cdot \cos(\alpha + \beta)$$

Вариант 23. Вычислить значения функций:

$$z_1 = \frac{\sin\left(\frac{\pi}{2} + 3\alpha\right)}{1 - \sin(3\alpha - \pi)}$$

$$z_2 = \operatorname{ctg}\left(\frac{5}{4}\pi + \frac{3}{2}\alpha\right)$$

Вариант 24. Вычислить значения функций:

$$z_1 = \frac{1-2\sin^2\alpha}{1+\sin 2\alpha}$$

$$z_2 = \frac{1-\operatorname{tg}\alpha}{1+\operatorname{tg}\alpha}$$

Вариант 25. Вычислить значения функций:

$$z_1 = \frac{\sin 4\alpha}{1+\cos 4\alpha} \cdot \frac{\cos 2\alpha}{1+\cos 2\alpha}$$

$$z_2 = \operatorname{ctg}\left(\frac{3}{2}\pi - \alpha\right)$$

Вариант 26. Вычислить значения функций:

$$z_1 = \frac{\cos\alpha + \sin\alpha}{\cos\alpha - \sin\alpha}$$

$$z_2 = \operatorname{tg} 2\alpha + \sec 2\alpha$$

Вариант 27. Вычислить значения функций:

$$z_1 = \frac{\sqrt{2b+2\sqrt{b^2-4}}}{\sqrt{b^2-4}+b+2}$$

$$z_2 = \frac{1}{\sqrt{b+2}}$$

Вариант 28. Вычислить значения функций:

$$z_1 = \frac{x^2+2x-3+(x+1)\sqrt{x^2-9}}{x^2-2x-3+(x-1)\sqrt{x^2-9}}$$

$$z_2 = \sqrt{\frac{x+3}{x-3}}$$

Вариант 29. Вычислить значения функций:

$$z_1 = \frac{\sqrt{(3m+2)^2-24m}}{3\sqrt{m}-\frac{2}{\sqrt{m}}}$$

$$z_2 = -\sqrt{m}$$

4 Лабораторная работа №4 – Операторы выбора

4.1 Цель работы

Цель работы:

– закрепление навыков работы в интегрированной среде *Lazarus*;

– знакомство с операторами управления в программах в интегрированной среде *Lazarus*;

– освоение приёмов программирования и отладки разветвлённых алгоритмов в интегрированной среде *Lazarus*.

4.2 Порядок выполнения работы

Перед выполнением этой работы следует:

1. Ознакомиться с теоретическим разделом 4.5. В качестве дополнительного источника знаний можно использовать [7, 8].
2. Разработать программу в соответствии с индивидуальным заданием (раздел 4.7).
3. Войти в свой личный каталог, загрузить и настроить систему программирования *Lazarus*.
4. Ввести шаблон программы, сделанный на лабораторной работе №1. Сделать его копию, записав под новым именем (разветвлённой программы).
5. Ввести разработанную Вами программу, корректируя и дополняя свой шаблон.
6. Освоить методику поиска причин и исправления синтаксических ошибок, а также отладки программы «по шагам». Добиться, чтобы программа дала правильные результаты.
7. Ответить на контрольные вопросы.
8. Оформить отчёт и защитить его у преподавателя.

4.3 Отчетность

Отчёт должен быть выполнен в соответствии с [9] и состоять из следующих разделов:

1. Тема и цель работы.
2. Индивидуальное задание.
3. Блок-схема алгоритма.
4. Откомпилированный текст программы (в электронном виде).
5. Ответы на контрольные вопросы.
6. Результаты выполнения программы.
7. Выводы.

При защите отчёта по работе для получения зачёта студент должен:

– уметь отвечать на контрольные вопросы;

– обосновать структуру выбранного алгоритма и доказать, что разработанное приложение работает правильно;

– уметь пояснить назначение элементов разработанного приложения;
– продемонстрировать навыки работы в интегрированной среде *Lazarus*.

4.4 Контрольные вопросы

1. Что такое составной оператор? Каковы причины его использования? В чем состоит особенность расстановки точек с запятой при записи составного оператора?

2. Что представляет собой оператор безусловного перехода? Укажите его назначение и особенности применения. Почему рекомендуется избегать оператора ***goto***?

3. Какая алгоритмическая конструкция называется ветвлением? Для чего необходимо ветвление в алгоритмах?

4. Какие формы ветвления различают? Сравните формы ветвления между собой. Приведите примеры её использования.

5. Что такое условие? Приведите примеры логических выражений. Из чего они состоят?

6. К какому типу данных всегда принадлежит результат проверки логического выражения? Приведите примеры его использования в инструкциях присваивания и вывода.

7. Перечислите основные операции отношения, используемые в *Lazarus*.

8. Перечислите основные логические операции, используемые в *Lazarus*.

9. Как определяется порядок выполнения операций в составе логических выражений?

10. Поясните, почему каждое простое условие в логическом выражении необходимо заключать в скобки. Приведите примеры.

11. Чем отличаются друг от друга ветвление и обход? Поясните с использованием блок-схем алгоритмов. Приведите примеры листингов, содержащих варианты использования инструкции ***if***.

12. Найдите значение *Y* при *X* = 20 (рис. 4.1).

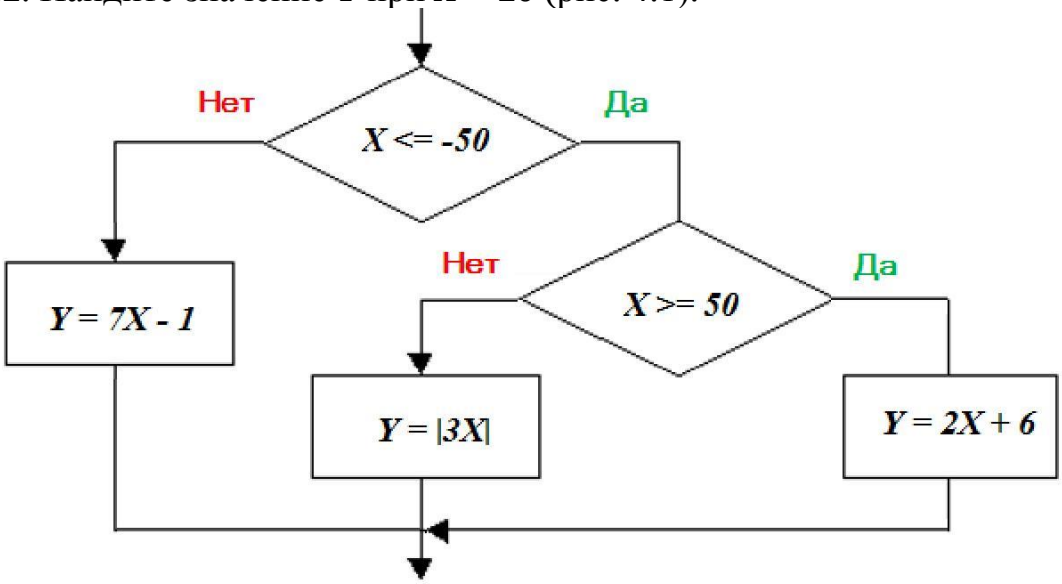


Рис. 4.1 – Анализ работы разветвлённого алгоритма

13. При каких начальных значениях переменных алгоритм на блок-схеме рис. 4.2 закончит работу (*a* mod 2 – это остаток от деления *a* на 2)?

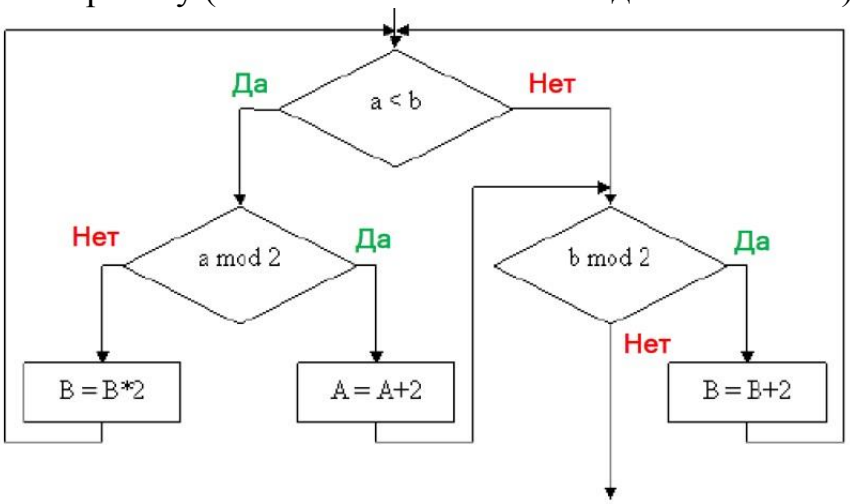


Рис. 4.2 – Анализ условий выхода разветвлённого алгоритма 13

14. При каких начальных значениях переменных алгоритм на блок-схеме рис. 4.3 закончит работу?

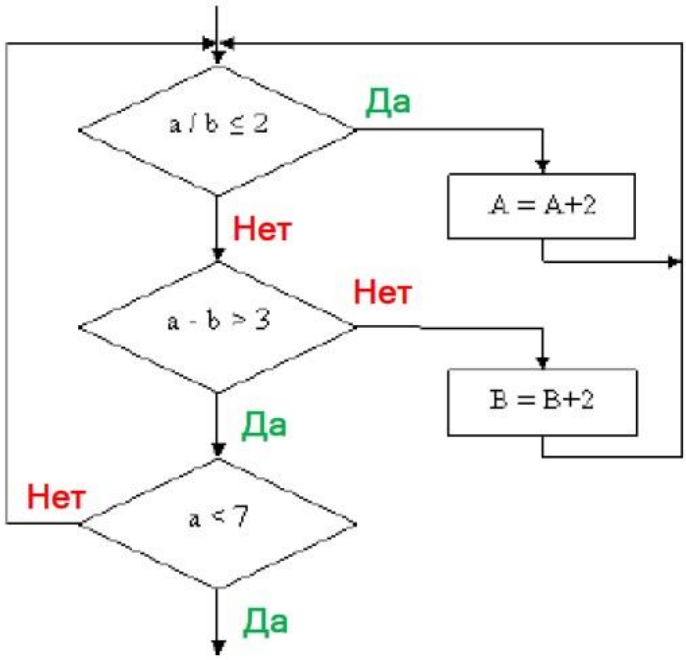


Рис. 4.3 – Анализ условий выхода разветвлённого алгоритма 14

15. В каких случаях возникает необходимость использования оператора выбора ***case***? Перечислите основные особенности использования оператора ***case***.

16. Можно ли вообще обойтись без оператора ***case***?

17. Что произойдёт, если проверяемое условие – ложно, а часть оператора, стоящая после служебного слова ***else***, отсутствует?

18. В чем сущность процесса отладки?

4.5 Операторы управления программой

Нередко в зависимости от ситуации, возникшей в ходе решения задачи, нужно вы брать один из двух или более вариантов решения. Выбор той или иной ветви вычислительного процесса в алгоритмах с разветвляющейся структурой осуществляется в зависимости от выполнения поставленного условия.

Для организации изменения естественного порядка выполнения операторов (так выполняются операторы в линейной программе – один за другим) в языке ***Free Pascal*** предусмотрены следующие управляющие операторы:

– составной оператор ***begin .. end***;
– оператор безусловной передачи управления ***goto***;
– оператор ветвления (условный) ***if .. then .. else***;
– оператор выбора (варианта) ***case .. of*** .

Как видим, в языке ***Free Pascal*** имеется два типа условных оператора ветвления: ***if*** и ***case***. Причём условным оператором чаще всего называют оператор ***if***, а оператор ***case*** именуют оператором выбора или оператором варианта.

4.6 Пример разработки программы

Задание. Необходимо отобразить рис 4.4 на форме программы и дать возможность пользователю вводить координаты точек. Программа проверяет и сообщает: точка с данными координатами принадлежит области или нет.

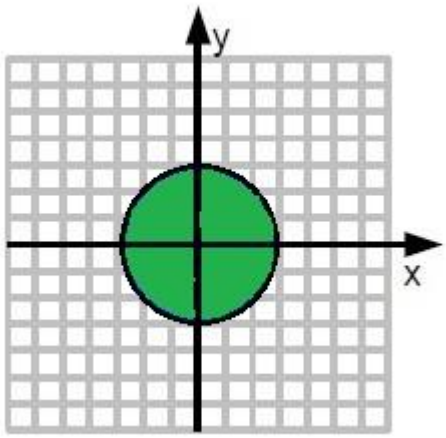


Рис. 4.4 – Задание

Создадим интерфейс. В среде IDE *Lazarus* выбираем ***Файл***/***Создать***/ ***Проект*** /***Приложение***. Расставим компоненты как на рис. 4.5. Слева на форме компонент ***TImage*** (прямоугольник из пунктирных линии), он находится на закладке компонент ***Additional***. Справа: два ***TEdit***, два ***TLabel*** н один ***TButton***.

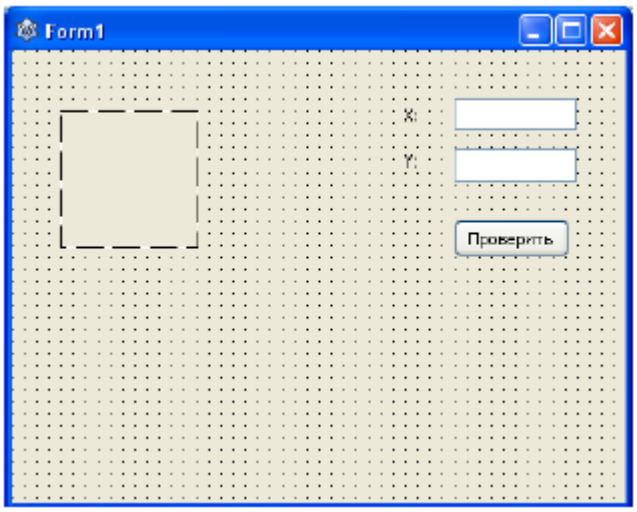


Рис. 4.5 – Расположение компонент на форме

Изменим заголовки у компонентов ***TLabel*** и ***TButton*** (свойство ***Caption***) и значения поля ***Text*** у компонентов ***TEdit***. В результате получим надписи как на рис. 4.5. Настроим компонент ***TImage1*** – на нем мы будем рисовать области. В инспекторе объектов найдем свойства ***Width*** и ***Height*** и устанавим значения равными **200**. Мы выбрали значение такими, что бы легче было рисовать н рассчитывать координаты.

Система координат (рис. 4.6) компонента ***TImage***, как и у всех других компонент, перевернута относительно обычной, которой мы привыкли пользоваться. Центр координат находится в левом верхнем углу компонента и имеет значение (0,0). Ось X – направлена горизонтально вправо, ось Y направлена вертикально вниз. Значения координат отсчитываются в пикселях (одни пиксель – одна единица).

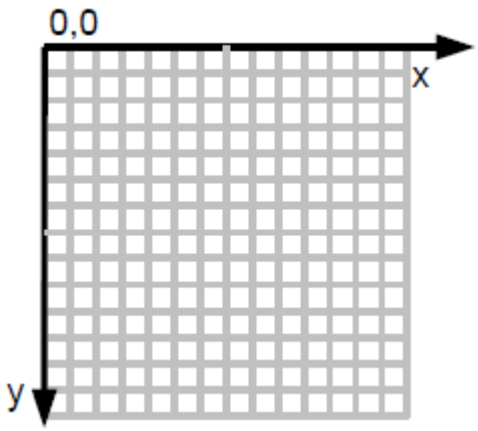


Рис. 4.6 – Система координат, используемая у компонента ***TImage***

В задании центр координат находится по центру рисунка и ось Y направлена в другую сторону (вверх). Следует учесть, что значения координат в задании довольно маленькие (от –5 до 5), поэтому введем масштабный коэффициент н перенесем центр координат в центр картинки. Масштаб выберем равным 20:1, а центр в точке с координатами $x = 100$; $y = 100$.

Компоненты ***TLabel***, ***TEdit*** и ***TButton*** видны на форме, а компонент ***TImage*** на форме не виден, т.к. на нем ничего не нарисовано. Наиболее

подходящее место по отображения области – это процедура создания формы. Когда форма создается (ее границы, метки, поля ввода, кнопки и другие компоненты) нарисуем область из задания.

Сделаем «двойной клик» в свободной области формы (где нет других компонентов) для обработчика создания формы (рис. 4.7).

```
  { TForm1 }
35
procedure TForm1.FormCreate(Sender: TObject);
begin
38
end;
```

Рис. 4.7 – Обработчик создания формы

Между элементами ***begin*** и ***end*** добавим код, который будет рисовать область. Для рисования обратимся к тому компоненту, на котором мы будем рисовать, в данном случае к компоненту ***TImage1***. Свойство, отвечающее за рисование примитивов, называется ***Canvas***.

Существует множество процедур у свойства ***Canvas***, которые рисуют примитивные объекты: линии, эллипсы и т.д.

Canvas имеет два свойства отвечающие за цвета: ***Кисть*** (***Brush***) и ***Карандаш*** (***Pen***). Все линии объектов рисуются свойствами карандаша: цвет линии, толщина, стиль и т.д., а все объемные части объектов закрашиваются свойствами кисти: цвет заливки, штриховка и др. Например внутренняя часть эллипса будет закрашена свойствами кисти, а внешняя граница нарисована свойствами карандаша.

Нарисуем окружность. Перевод координат из физических в экранные:

$$X_Э = X_Ф \cdot 20 + 100;\ Y_Э = -Y_Ф \cdot 20 + 100.$$

Напишем код между словами ***begin*** и ***end***:

Image1.Canvas.Ellipse(80, 80, 120, 120);

Обычно на белом фоне рисуют черные линии, поэтому нужно сменить фон. По умолчанию «карандаш» черный, а «кисть» – белая. Для закраски фона воспользуемся белым прямоугольником размером равным размеру компонента ***Image1***. Очистку фона надо вызвать до прорисовки окружности, чтобы не стереть окружность.

Image1.Canvas.Rectangle(0, 0, 200, 200);
Image1.Canvas.Ellipse(80, 80, 120, 120);

Закрасим внутреннюю часть окружности в зеленый цвет – для информативности области. До рисования окружности назначим зеленый цвет кисти.

Image1.Canvas.Rectangle(0, 0, 200, 200);
Image1.Canvas.Brush.Color := clGreen;
Image1.Canvas.Ellipse(80, 80, 120, 120);

Осталось добавить оси координат. Воспользуемся функцией ***Line***.

Image1.Canvas.Rectangle(0, 0, 200, 200);
Image1.Canvas.Brush.Color := clGreen;

Image1.Canvas.Ellipse(80, 80, 120, 120);
Image1.Canvas.Line(1, 100, 200, 100);
Image1.Canvas.Line(100, 0, 100, 200);

Сохраним и запустим проект. В результате форма программы выглядит следующим образом (рис. 4.8).

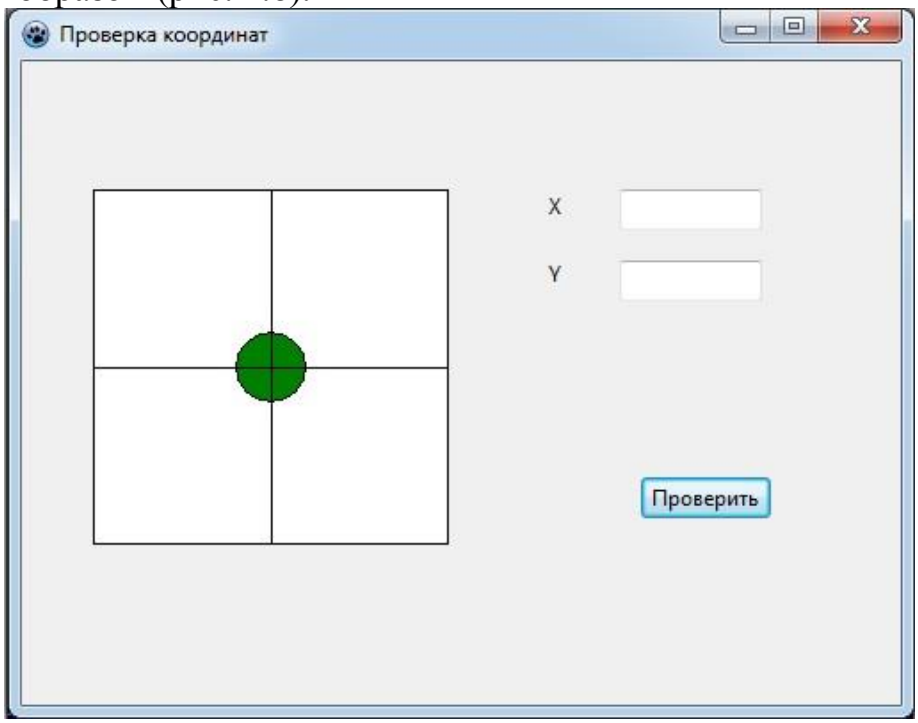


Рис. 4.8 – Готовая форма

Программа рисует область, а проверять координаты пока не умеет. Добавим эту возможность в программу. Необходимо написать обработчик на кнопку **Проверить**. Закроем программу и в редакторе формы сделаем двойной «клик» на кнопке **Проверить**. В результате отобразится следующий код (рис. 4.9).

```
  procedure TForm1.Button1Click(Sender: TObject);
  begin
55
  end;
```

Рис. 4.9 – Обработчик на кнопку **Проверить**

Для проверки понадобятся две вспомогательные переменные, в которых будем хранить координаты, введенные пользователем в поля ***X*** и ***Y***. Координаты вещественные (могут иметь дробную часть), объявим их как ***real***. Запишем в переменные ***X*** и ***Y*** значения из соответствующих полей:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  x, y : real;
begin
  x := StrToFloat(Edit1.Text);
  y := StrToFloat(Edit2.Text);
```

end;

Необходимо проанализировать значения координат с помощью оператора выбора ***if***. Воспользуемся операцией сравнения < (меньше). Если точка принадлежит фигуре окружности, то проверяемое событие – истинно. Здесь уже используются координаты физические, а не экранные (те, что заданы на рисунке в задании). Для проверки на принадлежность окружности необходимо задать уравнение окружности:

$$(X - X_0)^2 + (Y - Y_0)^2 = R^2,$$

где X, Y – координаты проверяемой точки; X_0, Y_0 – координаты центра окружности; R – радиус окружности.

Если в уравнении стоит знак **=**, то точка с координатами (***X***, ***Y***) находится на окружности, если поставить знак <, то – внутри окружности, а если поставить знак >, то – вне окружности. Воспользуемся знаком < (строго меньше, т.к. граница не входит в область проверки) н определим значения X_0 и Y_0. Центр окружности находится в точке с координатами (0, 0) тогда $X_0 = 0$ и $Y_0 = 0$.

Если условие выполняется, то будем выдавать при помощи функции ***ShowMessage*** текст **Точка принадлежит области**, а если условие не выполнится, то **Точка НЕ принадлежит области**. Добавим код:

```
x := StrToFloat(Edit1.Text);
y := StrToFloat(Edit2.Text);
if (x*x + y*y < 4) then
   ShowMessage('Точка принадлежит области')
else
   ShowMessage('Точка НЕ принадлежит области');
```

Сохраним проект и запустим на выполнение. Опробуем проверку координат. В соответствующие поля введем координаты (например, ***1***, ***1***) и нажмем кнопку **Проверить** (рис. 4.10).

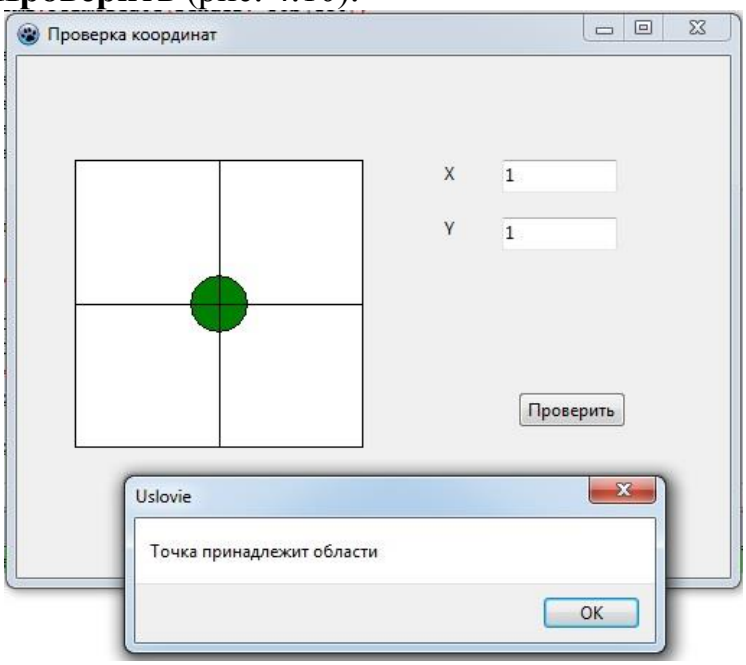


Рис. 4.10 – Проверка координат

Проверим координаты точек (2; 2), (–2; –2), (–0,5; –0,5) на принадлежность области. Первые две точки должны не принадлежать области (т.к. граница области не входит в проверку), а третья принадлежит области.

Программа почти закончена, но необходимо добавить в нее еще одну небольшую функциональность, которая значительно облегчит проверку правильности составления условия (оператора ***if***) на принадлежность точек области. Пользователю удобней не вводить координаты в соответствующие поля ввода данных, а «кликать» на рисунке. Место «клика» программа проанализирует и занесет в соответствующие поля значения координат.

Получения координат происходит в момент «клика» по компоненту ***Image1***. Событие ***OnClick*** (связанное с ***Image1***) не подойдет. Хотя оно происходит в момент «клика», но оно не содержит значений координат «мыши» в момент клика, которые необходимы для анализа. Более подходящие события: ***OnMouseDown*** и ***OnMouseUp***. Они содержат координаты курсора мыши в момент «клика». Первое происходит в момент нажатия на кнопку «мыши» – ***Down***, а второе в момент отпускания кнопки мыши – ***Up***. Эти события происходят в разные моменты времени и могут содержать разные координаты. Можно отпустить кнопку мыши не в том месте, где нажали ее.

Воспользуемся событием ***OnMouseDown*** для анализа координат мыши. Найдем его в **Инспекторе объектов**, предварительно выбрав компонент ***Image1***, на вкладке С**обытия** (рис. 4.11).

Рис. 4.11 – Привязка события ***OnMouseDown*** к ***Image1***

Список возможных привязываемых событий пуст, но справа есть кнопка «**...**». Нажмем на нее и получим заготовку события ***OnMouseDown*** (рис. 4.12).

```
55  procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
    .   Shift: TShiftState; X, Y: Integer);
    .  begin
    .
    .  end;
```

Рис. 4.12 – Обработчик события ***OnMouseDown*** для ***Image1***

У этого обработчика есть два параметра – координаты мыши в момент «клика». Они соответственно хранятся в переменных ***X*** и ***Y***. Напишем код между ***begin*** и ***end***.

```
Edit1.Text := FloatToStr((X - 100)/20);
Edit2.Text := FloatToStr((Y - 100)/20);
Image1.Canvas.Ellipse(x - 2, y - 2, x + 2, y + 2);
```

Первые две строки преобразуют координаты из экранных в физические и заносят в соответствующие поля ввода, а третья строка в точке «клика» рисует маленькую окружность.

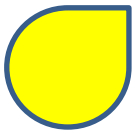
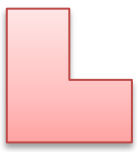
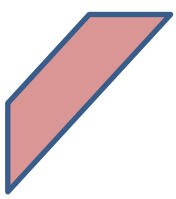
Сохраним проект и запустим на выполнение.

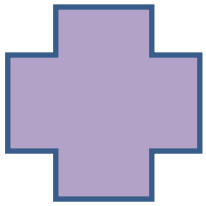
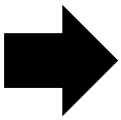
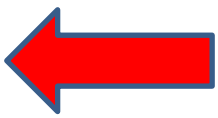
4.7 Варианты заданий для составления условных программ

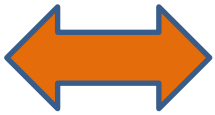
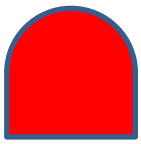
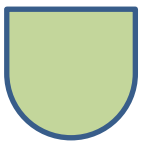
Необходимо отобразить фигуру по варианту заданию (таблица 4.1) на форме программы и дать возможность пользователю вводить координаты точек. Программа проверяет и сообщает: точка с данными координатами принадлежит области внутри фигуры или нет. Размер области компонента ***TImage*** равен 200×200 пикселов. Размер фигуры – произвольный, но такой чтобы находился внутри области 200×200 пикселов и симметрично относительно начала координат.

Таблица 4.1 – Варианты заданий

№ варианта	Фигура
1	
2	
3	

4	
5	
6	
7	
8	
9	
10	
11	
12	
13	

14	
15	
16	
17	
18	
19	
20	
21	
22	

23	
24	
25	
26	
27	
28	
29	

Список литературы

1. Алексеев Е.Р., Чеснокова О.В., Кучер Т.В. Free Pascal и Lazarus: Учебник по программированию / Е.Р.Алексеев, О.В.Чеснокова, Т.В.Кучер. - М.: Издательский дом «ДМК-пресс», 2010. - 440 с.
2. Алексеев Е.Р., Чеснокова О.В., Кучер Т.В.. Самоучитель по программированию на Free Pascal и Lazarus.. - Донецк: ДонНТУ, Технопарк ДонНТУ УНИТЕХ, 2011. - 503 с.
3. Кетков Ю.Л. Свободное программное обеспечение. FREE PASCAL для студентов и школьников / Ю.Л. Кетков, А.Ю. Кетков. - СПб.: БХВ-Петербург, 2011. - 384 с.
4. Мансуров К.Т. Основы программирования в среде Lazarus. - М.: Нобель пресс, 2013. – 772 с.
5. Фаронов В.В. Turbo Pascal. Наиболее полное руководство (в подлиннике). - СПб.: БХВ-Петербург, 2004. - 1056 с.
6. Фленов М.Е. Библия Delphi. - 2-е изд., перераб. и доп. - СПб.: БХВ-Петербург, 2008. - 800 с.
7. Lazarus Tutorial/ru [Электронный ресурс] // База знаний о Free Pascal, Lazarus и родственных проектах: [сайт]. URL: http://wiki.freepascal.org/Lazarus_Tutorial/ru
8. Программирование на Lazarus [Электронный ресурс] // «ИНТУИТ» Национальный открытый университет: [сайт]. URL: http://www.intuit.ru/studies/courses/13745/1221/ lecture/23276?page=1
9. ОС ТУСУР 01-2013 (СТО 02069326.1.01-2013). Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления. - Томск: ТУСУР, 2013. – 57 с.
10. Кобрин Ю.П. Приложение к лабораторной работе "Работа в интегрированной среде Lazarus (free Pascal)". Томск: ТУСУР, 2016. - 21 с.