Министерство науки и высшего образования Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

Д.В.Озеркин

ОТКРЫТАЯ СРЕДА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ LAZARUS

Методические указания для лабораторных работ
по дисциплинам «Информатика»,
«Информатика и программирование»,
«Информационные технологии в электронике»
для направления подготовки
110303 Конструирование и технология электронных средств

Томск
2022

УДК 004.91
ББК 32.85
О-46

Озеркин, Денис Витальевич
Открытая среда разработки программного обеспечения Lazarus: Методические указания для лабораторных работ по дисциплинам «Информатика», «Информатика и программирование», «Информационные технологии в электронике» для направления подготовки 110303 Конструирование и технология электронных средств / Д.В.Озеркин. – Томск: Томск. гос. ун-т систем упр. и радиоэлектроники, 2022. – 111 с.

Lazarus - открытая среда разработки программного обеспечения на языке Object Pascal для компилятора Free Pascal (часто используется сокращение FPC - Free Pascal Compiler, бесплатно распространяемый компилятор языка программирования Pascal). Интегрированная среда разработки предоставляет возможность кроссплатформенной разработки приложений в Delphi-подобном окружении. Позволяет достаточно несложно переносить Delphi-программы с графическим интерфейсом в различные операционные системы: Linux, FreeBSD, Mac OS X, Microsoft Windows, Android. Начиная с Delphi XE2 в самом Delphi имеется возможность компиляции программ для Mac OS X, с версии XE4 - для iOS, с версии XE5 - для Android.

Методические указания предназначены для бакалавров направления подготовки 110303 Конструирование и технология электронных средств

Одобрено на заседании кафедры РЭТЭМ протокол № 78 от 16.02.2022.

УДК 004.91
ББК 32.85

СОДЕРЖАНИЕ

1 Лабораторная работа №1 – Создание шаблона для разрабатываемых программ 5
1.1 Цель работы 5
1.2 Порядок выполнения лабораторной работы 5
1.3 Отчетность 5
1.4 Практическое задание 6
1.5 Контрольные вопросы 20
2 Лабораторная работа №2 – Консольные программы на Free Pascal 21
2.1 Цель работы 21
2.2 Порядок выполнения работы 21
2.3 Отчетность 21
2.4 Контрольные вопросы 21
2.5 Краткие теоретические сведения 22
2.6 Работа в редакторе исходного текста Free Pascal 24
2.7 Пример разработки программы 25
2.8 Рекомендации по составлению программы 26
2.9 Индивидуальные задания 26
3 Лабораторная работа №3 – Операторы присваивания 29
3.1 Цель работы 29
3.2 Порядок выполнения работы 29
3.3 Отчетность 29
3.4 Контрольные вопросы 30
3.5 Линейные программы 30
3.6 Пример разработки программы 34
3.7 Варианты заданий для составления линейных программ 37
4 Лабораторная работа №4 – Операторы выбора 42
4.1 Цель работы 42
4.2 Порядок выполнения работы 42
4.3 Отчетность 42
4.4 Контрольные вопросы 43
4.5 Операторы управления программой 44
4.6 Пример разработки программы 45
4.7 Варианты заданий для составления условных программ 51
5 Лабораторная работа №5 – Оператор цикла с заданным числом повторов 55
5.1 Цель и задачи работы 55
5.2 Порядок выполнения работы 55
5.3 Отчетность 55
5.4 Контрольные вопросы 55

5.5 Методика разработки циклических программ 57
5.6 Практические задания 60
6 Лабораторная работа №6 - Оператор цикла с предусловием 69
6.1 Цель и задачи работы 69
6.2 Порядок выполнения работы 69
6.3 Отчетность 69
6.4 Контрольные вопросы 69
6.5 Теоретические сведения о циклах с предусловием 70
6.6 Практические задания 72
7 Лабораторная работа №7 – Массивы 77
7.1 Цель и задачи работы 77
7.2 Порядок выполнения работы 77
7.3 Отчетность 77
7.4 Контрольные вопросы 77
7.5 Теоретические сведения о массивах 78
7.6 Практические задания 80
8 Лабораторная работа №8 – Программирование с использованием записей 89
8.1 Цель и задача работы 89
8.2 Порядок выполнения работы 89
8.3 Отчетность 89
8.4 Контрольные вопросы 89
8.5 Применение записей при разработке баз данных 90
8.6 Практические задания 92
9 Лабораторная работа №9 – Двумерные массивы 104
9.1 Цель и задачи работы 104
9.2 Порядок выполнения работы 104
9.3 Отчетность 104
9.4 Контрольные вопросы 104
9.5 Теоретические сведения о двумерных массивах 105
9.6 Практические задания 106
Список литературы 111

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 [1 - 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* (рисунок 1.1).

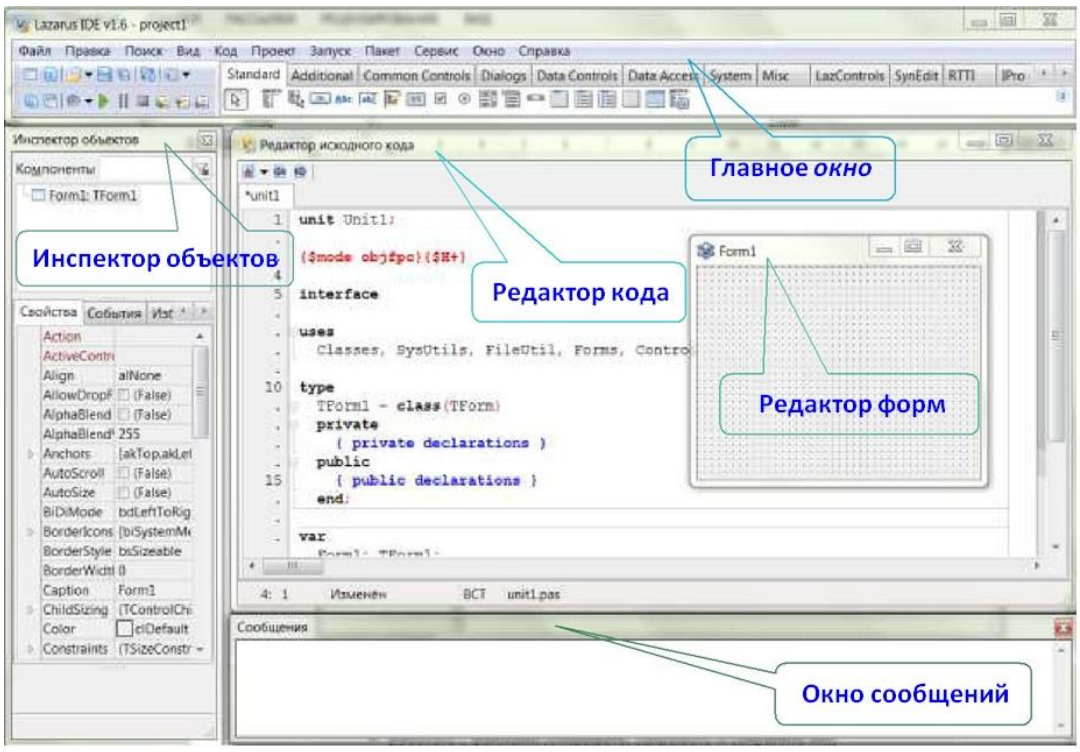


Рисунок 1.1 - Окна интегрированной среды разработки программ ***Lazarus***

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 иерархическом 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 (рисунок 1.2) следует выбрать опцию **Приложение** и нажать кнопку **ОК**.

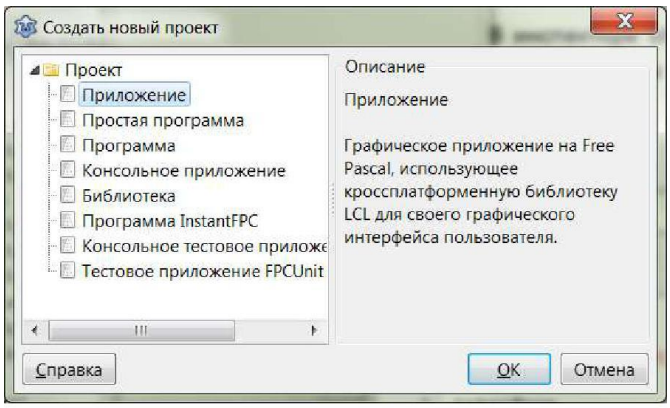


Рисунок 1.2 - Диалоговое окно **1.4.4 Сохранение созданного проекта**

Сохранить созданный проект можно с помощью последовательности команд **Проект / Сохранить проект как…** В открывшемся окне **Сохранить проект** (рисунок 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.

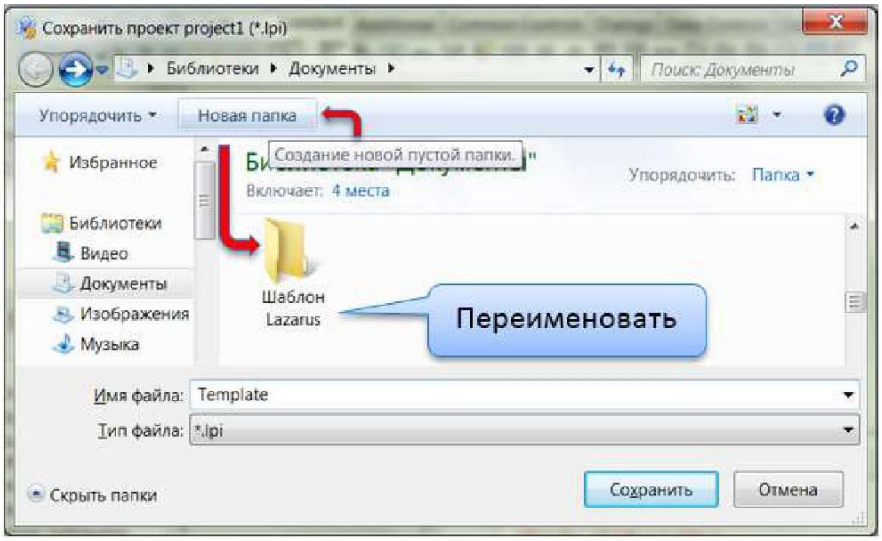


Рисунок 1.3 - Диалоговое окно **Сохранить проект**

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 (рисунок 1.4).

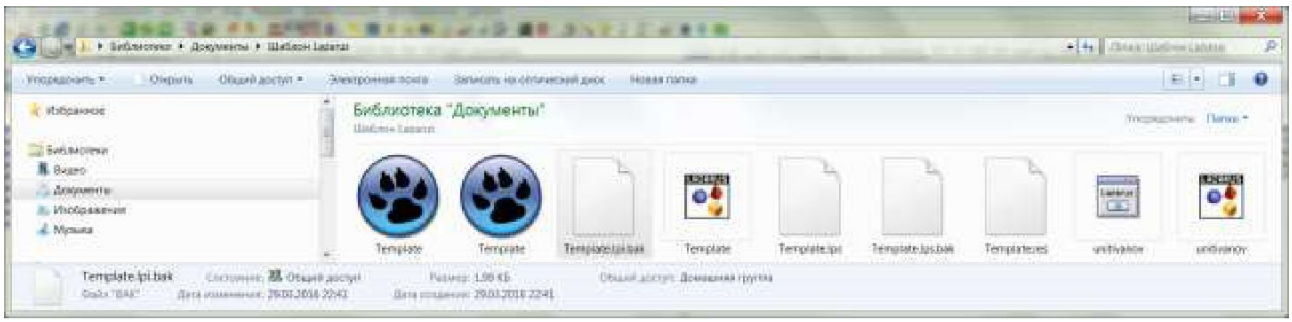


Рисунок 1.4 - Содержание папки **Шаблон Lazarus** после сохранения проекта

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.

Окончание таблицы 1.1

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

Событие

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 (рисунок1.5).

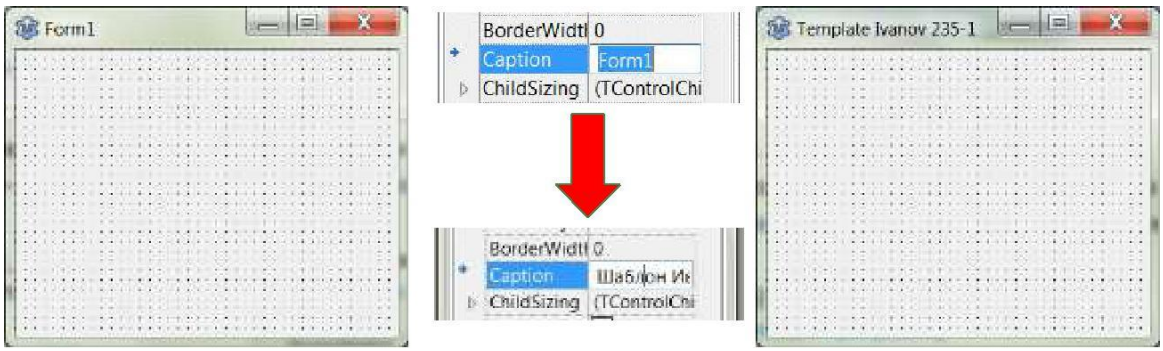


Рисунок 1.5 - Уточнение заголовка формы

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) (рисунок 1.6).

Рисунок 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 (рисунок 1.7).

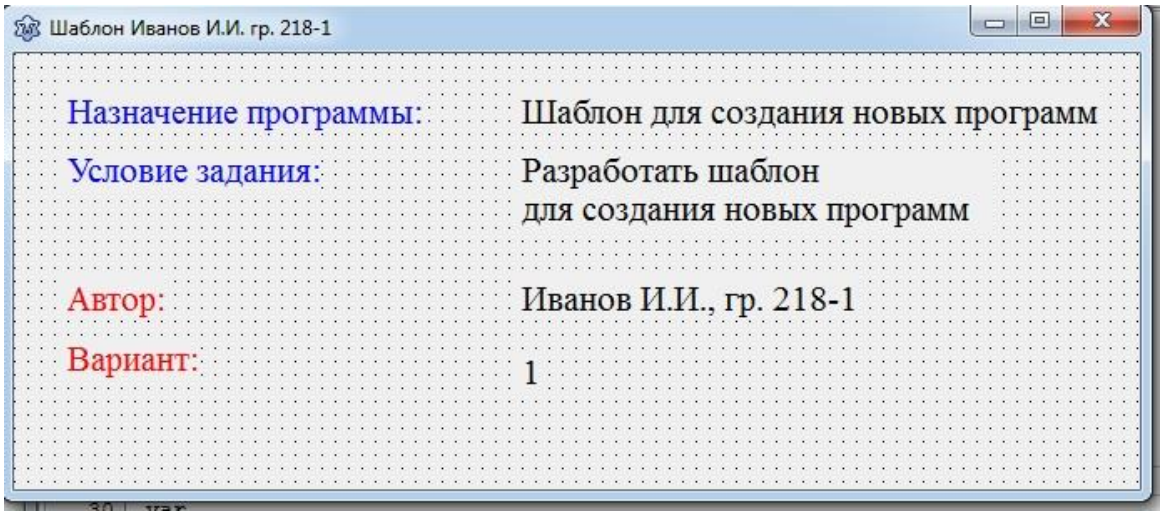


Рисунок 1.7 - Создание информационных строк в Форме

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 (рисунок 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.

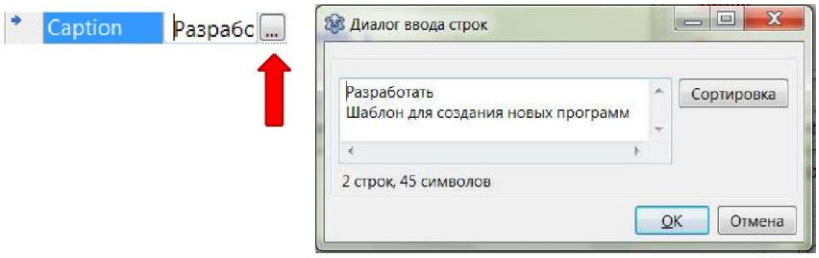


Рисунок 1.8 - Редактирование многострочной надписи

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 (рисунок 1.9).

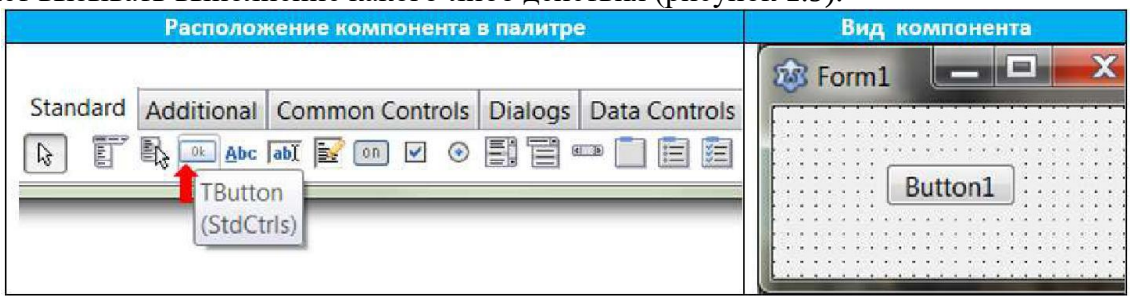


Рисунок 1.9 - Положение компонента ***TButton*** (Кнопка) во вкладке «Standard» палитры и его вид

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 (рисунок 1.10).

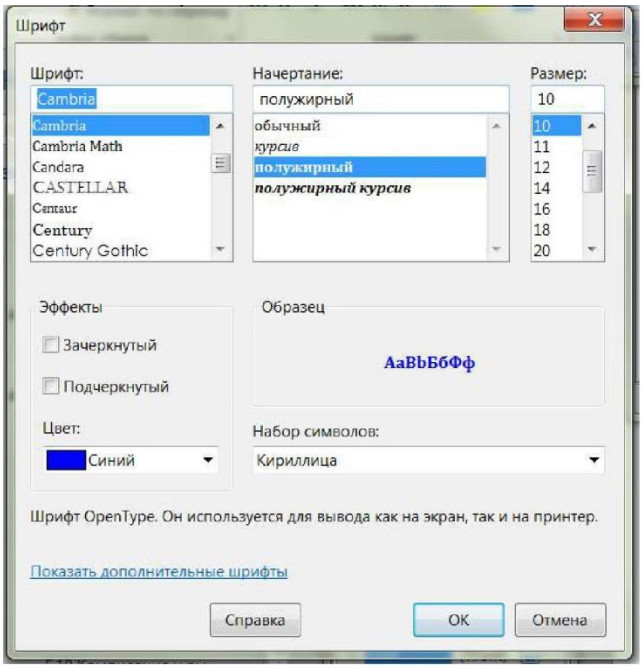


Рисунок1.10 - Диалоговое окно Шрифт. Установка параметров выводимого текста

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 (рисунок 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.

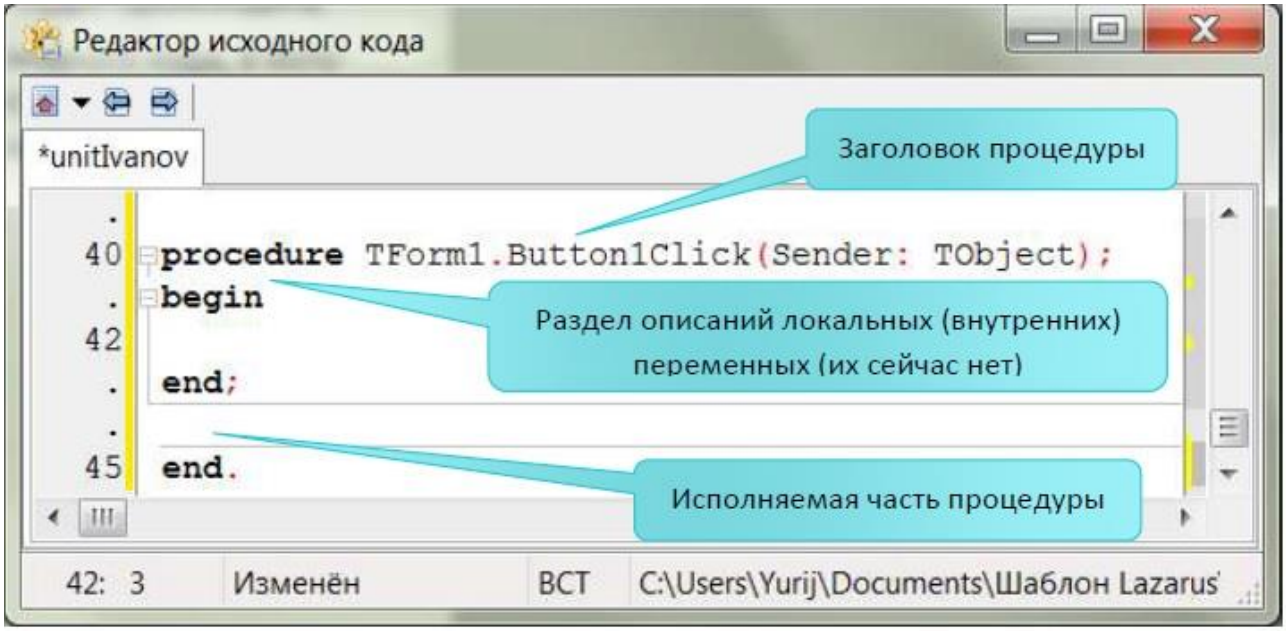


Рисунок 1.11 – Пустой шаблон процедуры обработчика события **Щелчок на кнопке**

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*** (рисунок 1.12). Подсказка представляет собой всплывающее меню, в котором перечислены допустимые свойства и методы компонента ***Label9***.

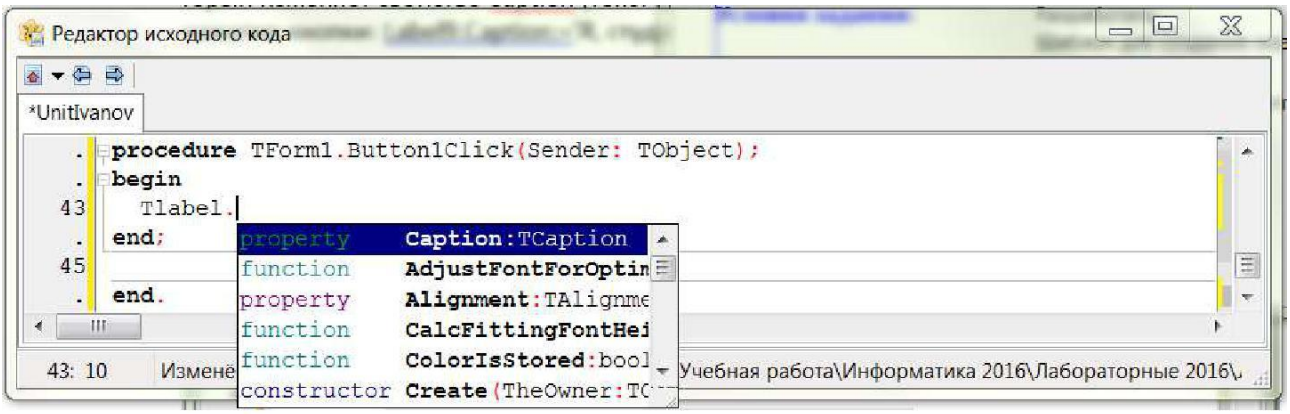


Рисунок 1.12 - Всплывающее меню, в котором перечислены допустимые свойства и методы компонента ***Label9***

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 рисунок 1.13.

Рисунок 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 рисунок 1.14 представлен вид процедуры обработчика события кнопки **Завершить работу**.

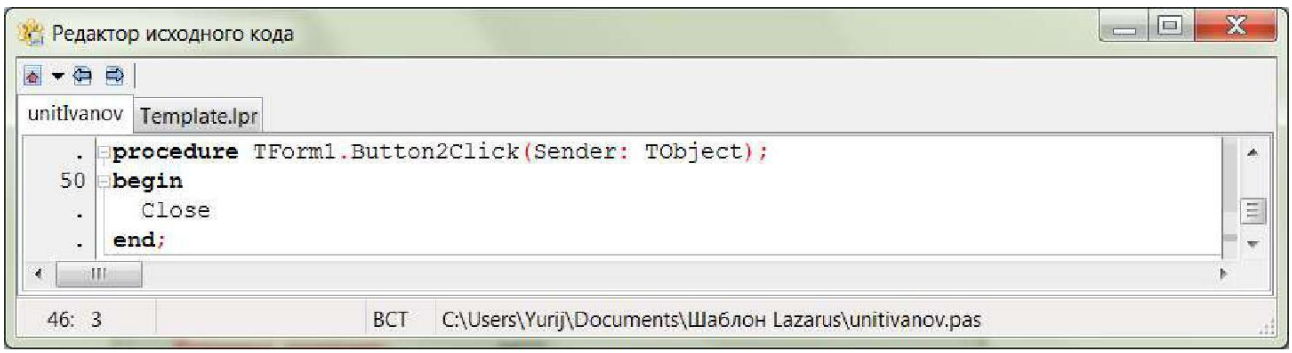
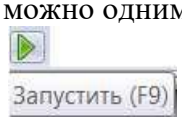


Рисунок 1.14 - Вид процедуры обработчика события кнопки «Завершить работу»

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 (рисунок 1.15).

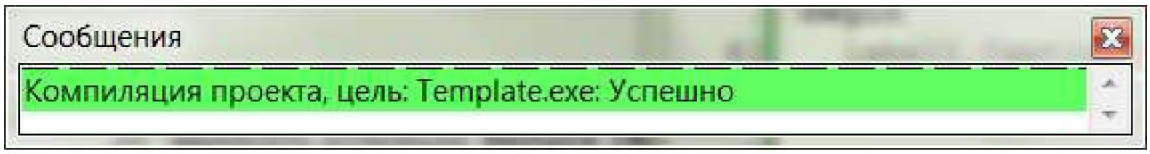


Рисунок 1.15 - Сообщение об успешной компиляции

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!**» (рисунок 1.16).

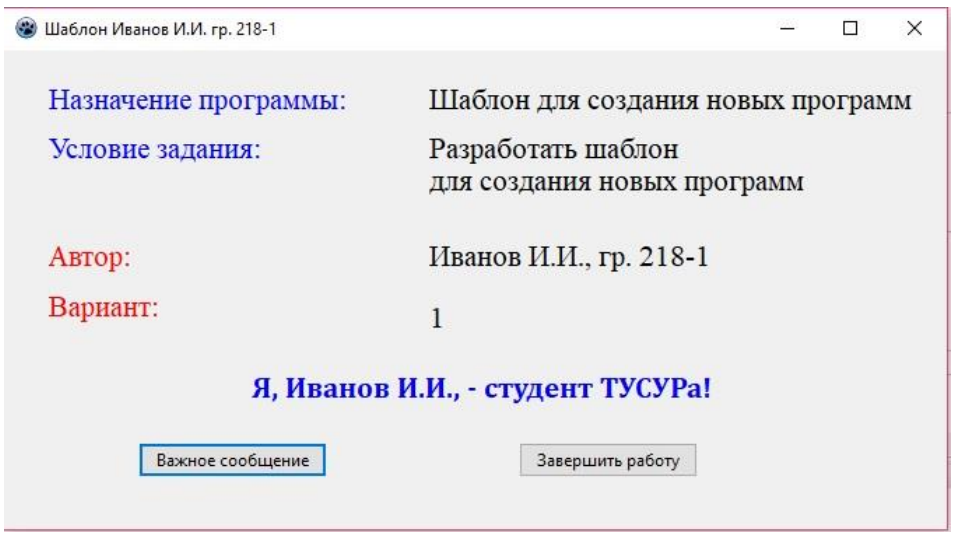


Рисунок 1.16 - Результаты работы программы

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 [1 - 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 (рисунок 2.1) выбрать *Проект / Программа* и нажать кнопку *ОК*.

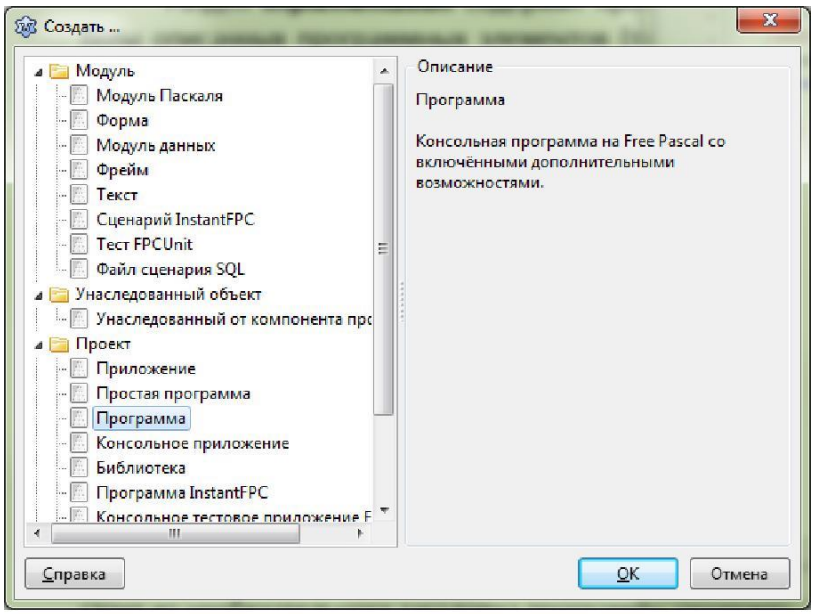


Рисунок 2.1 - Диалоговое окно "Создать…"

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

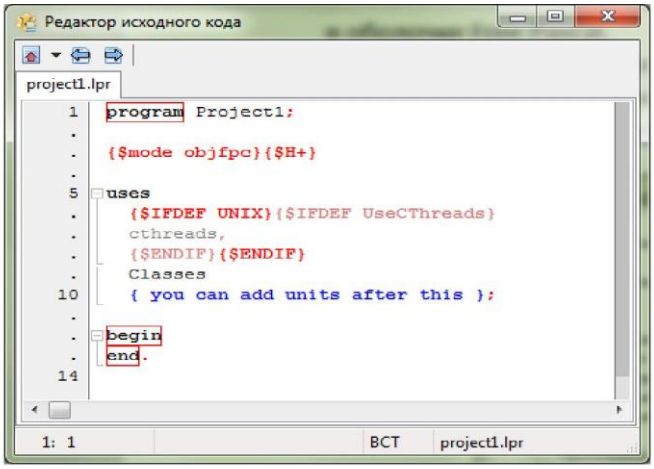


Рисунок 2.2 – Заготовка программы в **Редакторе исходного кода**

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* (рисунок 2.3). Там же приведены и «Горячие клавиши», соответствующие клавишам меню.

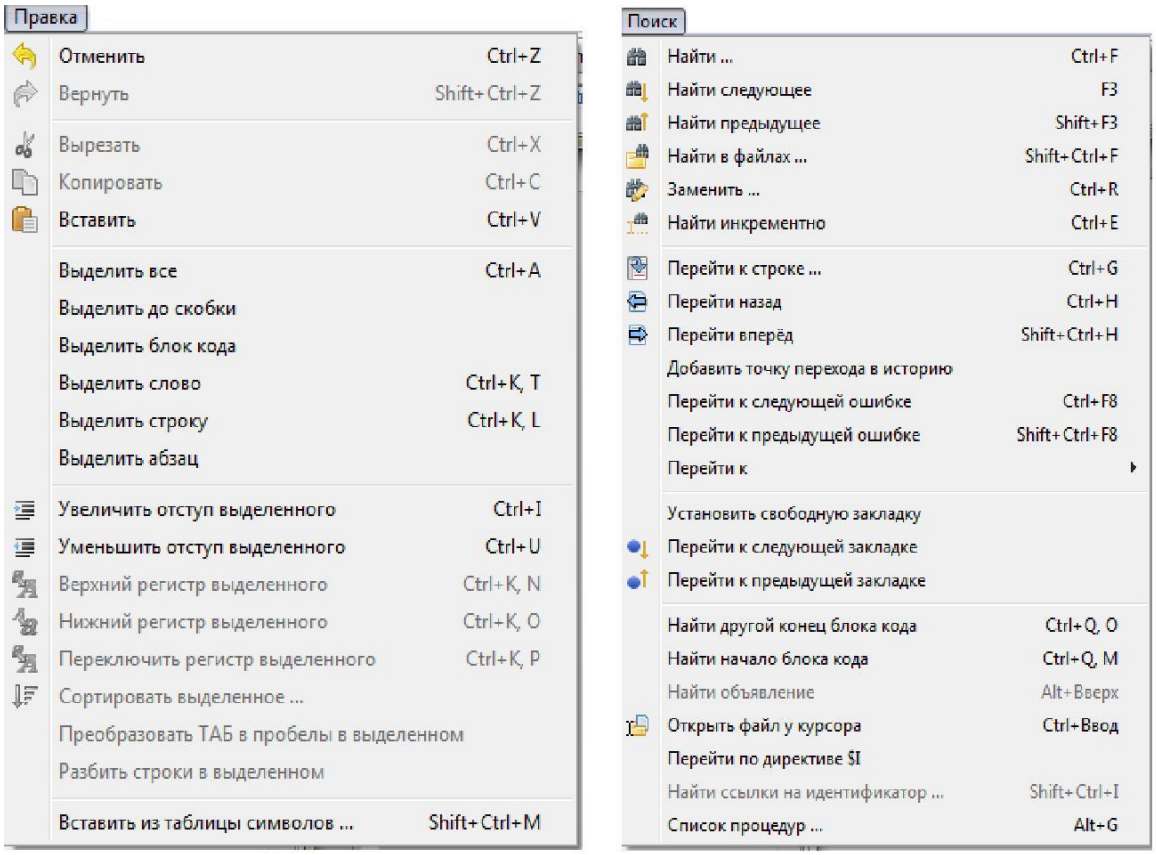


Рисунок 2.3 - Меню **Правка** и **Поиск** Редактора исходного текста

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 (рисунок 2.4).

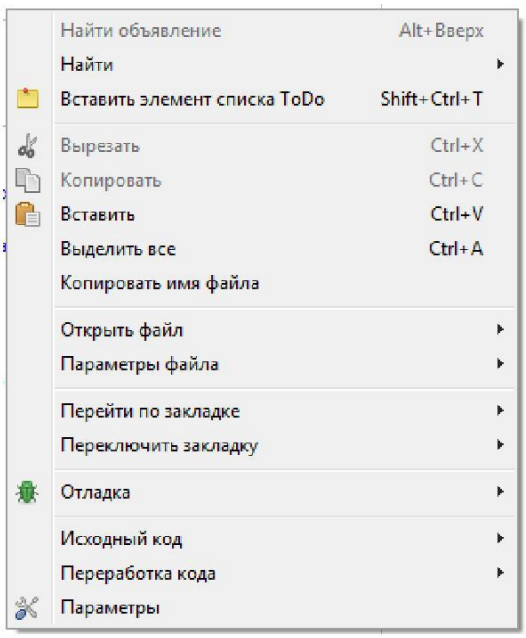


Рисунок 2.4 - Контекстное меню Редактора

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.	$Y := \sqrt[8]{\frac{lg(a+x)}{b^{a+x}+b}} + e^{5 \cdot a}$	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.	$Y := \frac{\left(a + \sqrt[3]{x+3}\right) \cdot (x+b)}{\ln b}$	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.	$Y := \frac{\sqrt[3]{x} + \frac{1}{a}}{lg\left(\left\lvert \sqrt[3]{x} + \frac{1}{a} \right\rvert\right)}$	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 [1 - 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 (рисунок 3.1). Такой порядок выполнения действий называется *естественным*.

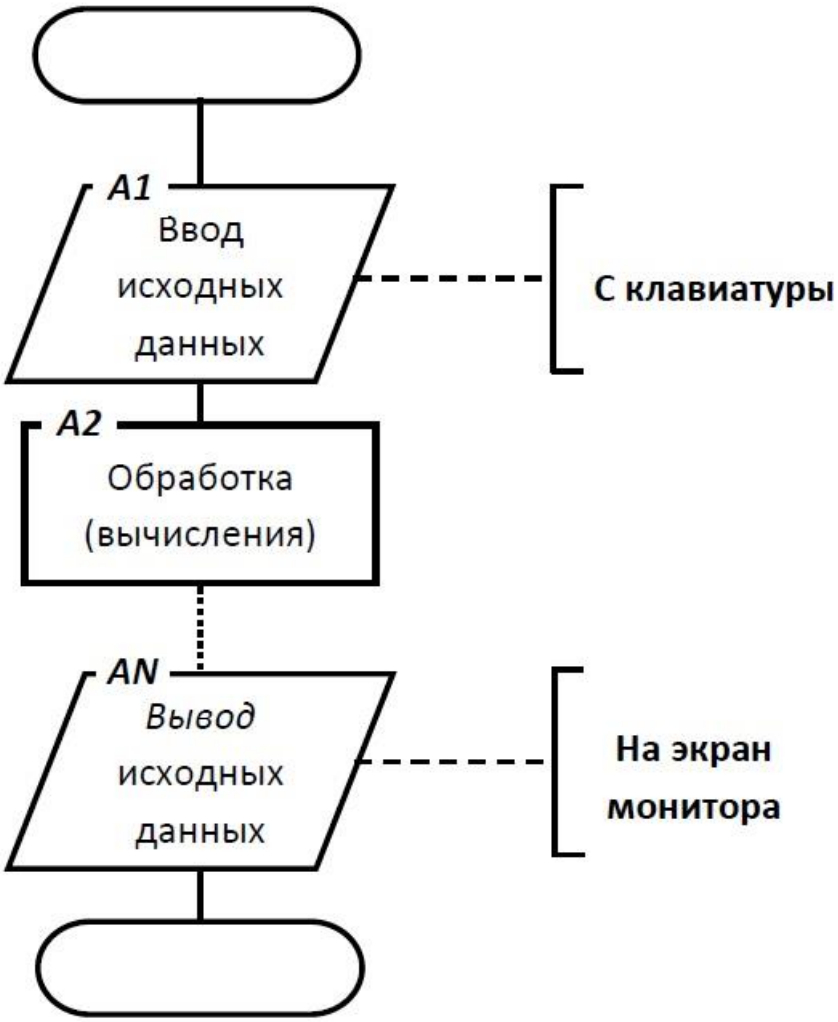


Рисунок 3.1 – Линейный алгоритм

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 (рисунок 3.2).

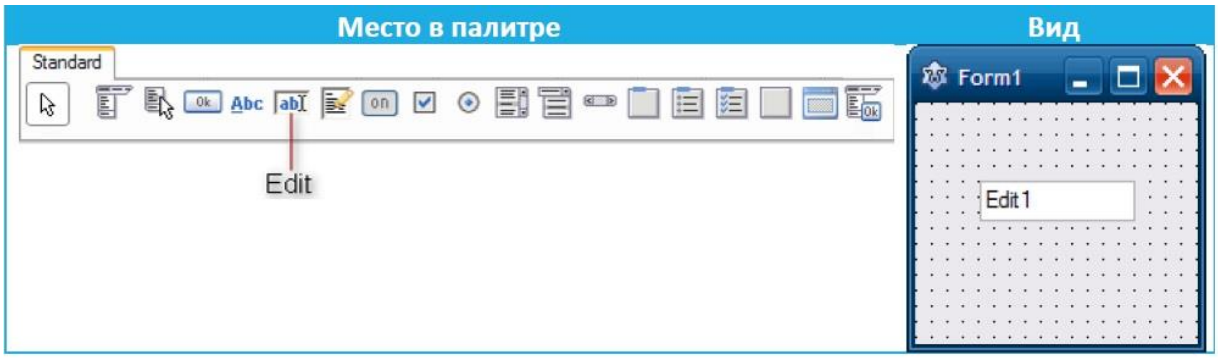


Рисунок3.2 – Компонент ***TEdit*** и его место в Палитре инструсентов

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

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 3.1 - 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*).

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 (рисунок 3.3).

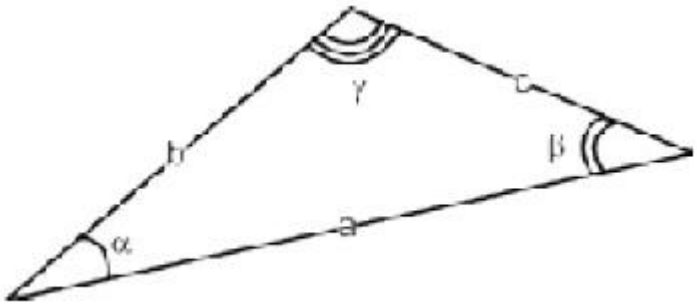


Рисунок 3.3 – Треугольник

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 (рисунок 3.4).

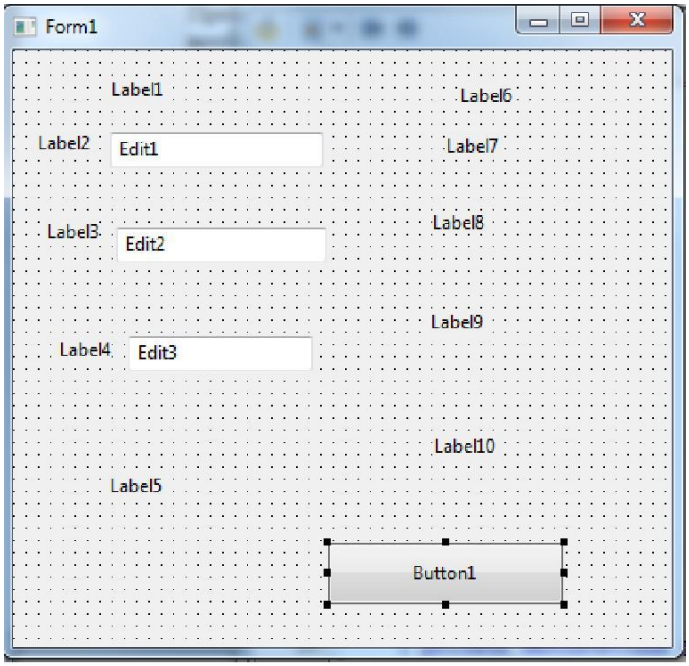


Рисунок 3.4 - Вид формы с компонентами

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

Рисунок 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 $a1a2$ в отношении $n1{:}n2$ 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 ($x1$, $y1$, $z1$) и ($x2$, $y2$, $z2$) 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 [1 - 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 (рисунок 4.1).

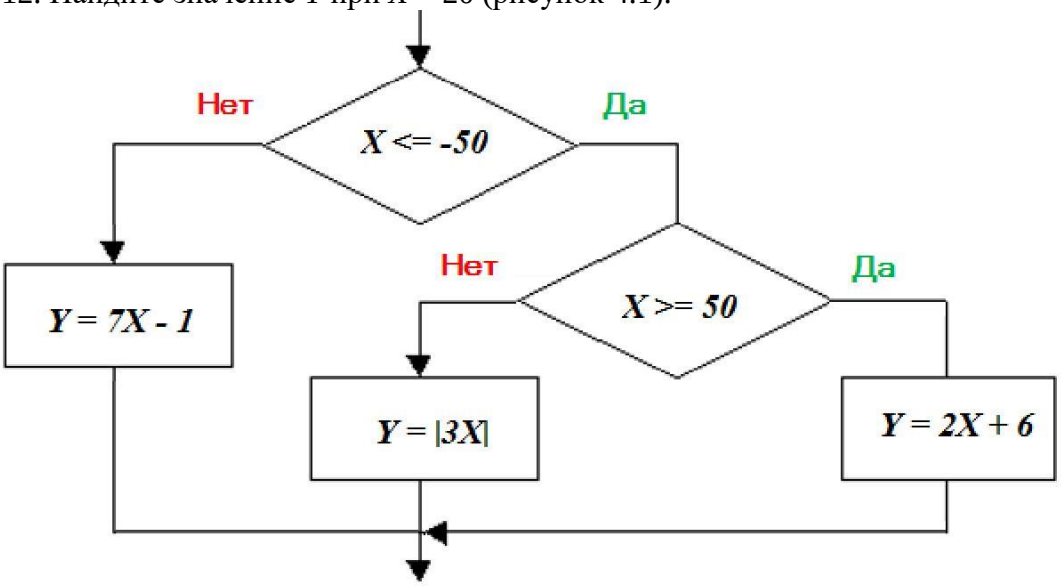


Рисунок 4.1 – Анализ работы разветвлённого алгоритма

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

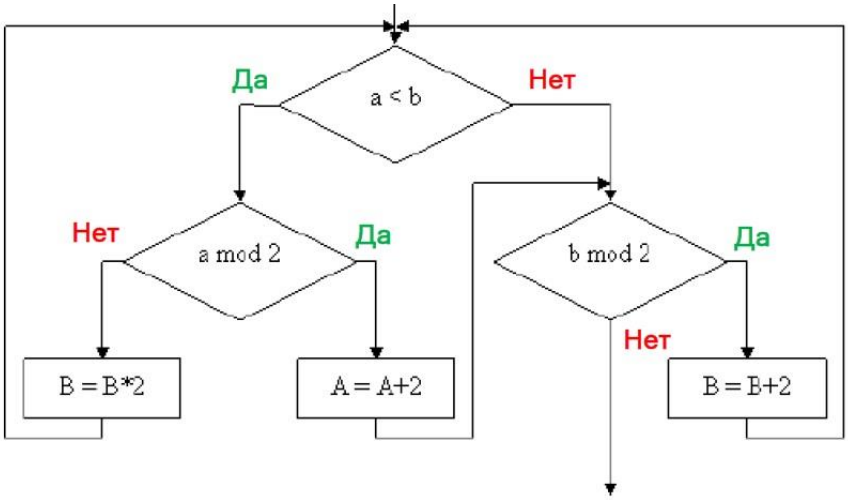


Рисунок 4.2 – Анализ условий выхода разветвлённого алгоритма 13

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

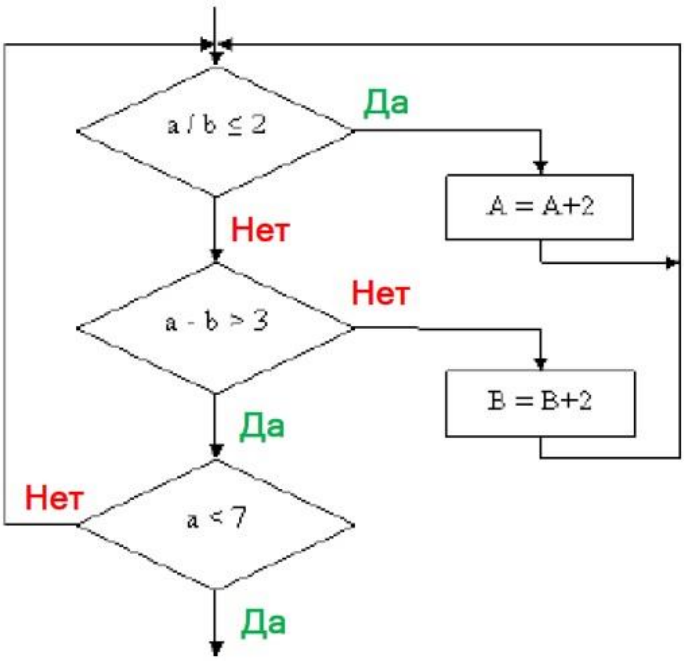


Рисунок 4.3 – Анализ условий выхода разветвлённого алгоритма 14

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.

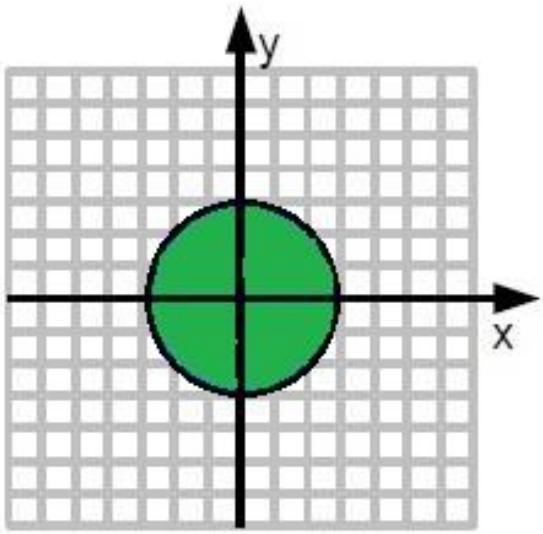


Рисунок 4.4 – Задание

Создадим интерфейс. В среде IDE *Lazarus* выбираем ***Файл***/***Создать***/ ***Проект*** /***Приложение***. Расставим компоненты как на рисунок 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***.

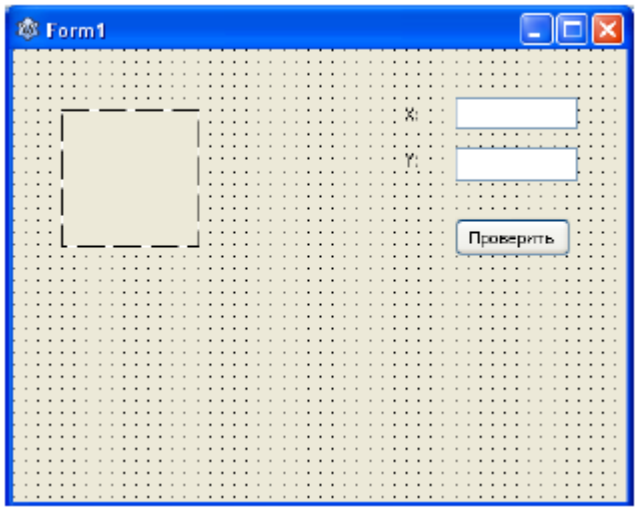


Рисунок 4.5 – Расположение компонент на форме

Изменим заголовки у компонентов ***TLabel*** и ***TButton*** (свойство ***Caption***) и значения поля ***Text*** у компонентов ***TEdit***. В результате получим надписи как на рисунок 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 (рисунок 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).

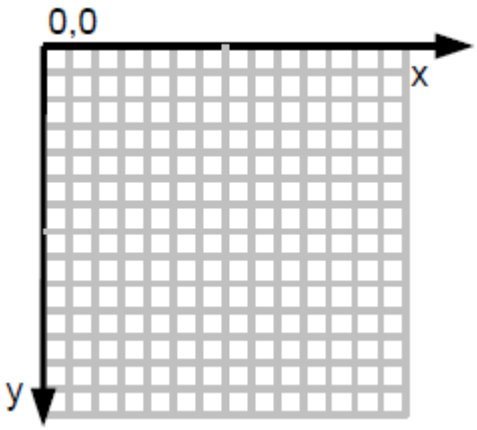


Рисунок 4.6 – Система координат, используемая у компонента ***TImage***

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 (рисунок 4.7).

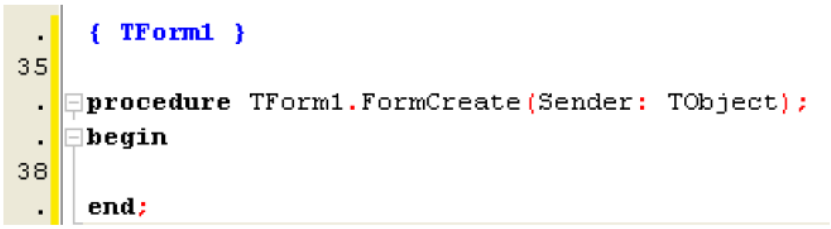


Рисунок 4.7 – Обработчик создания формы

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 (рисунок 4.8).

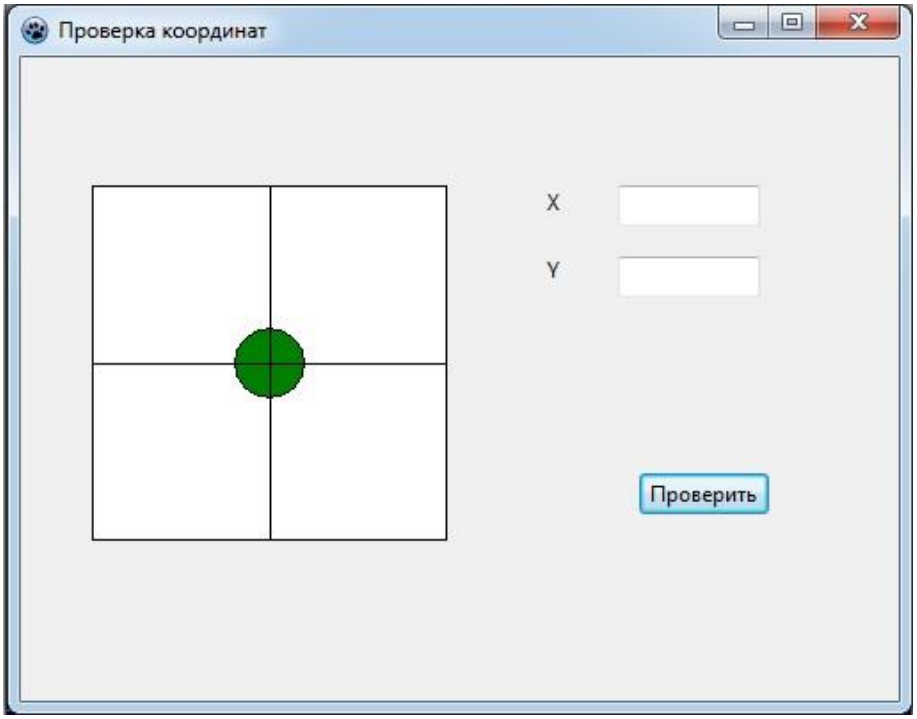


Рисунок 4.8 – Готовая форма

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 (рисунок 4.9).

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

Рисунок 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** (рисунок 4.10).

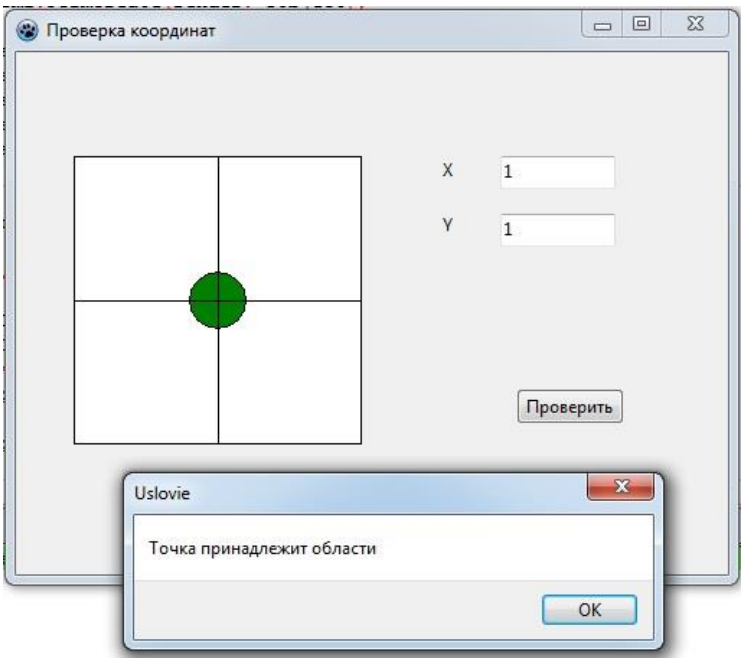


Рисунок 4.10 – Проверка координат

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** (рисунок 4.11).

Рисунок 4.11 – Привязка события ***OnMouseDown*** к ***Image1***

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

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

Рисунок 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.

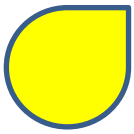
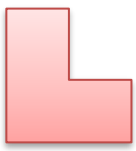
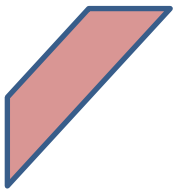
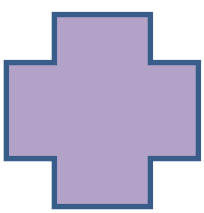
4.7 Варианты заданий для составления условных программ

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.

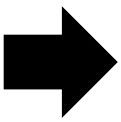
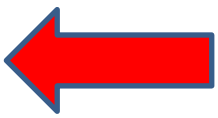
Таблица 4.1 – Варианты заданий

№ варианта	Фигура
1	
2	
3	
4	
5	
6	

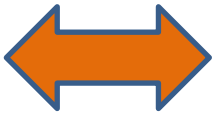
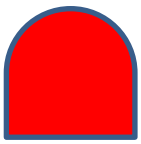
Продолжение таблицы 4.1

7	
8	
9	
10	
11	
12	
13	
14	

Продолжение таблицы 4.1

15	
16	
17	
18	
19	
20	
21	
22	

Окончание таблицы 4.1

23	
24	
25	
26	
27	
28	
29	

5 Лабораторная работа №5 – Оператор цикла с заданным числом повторов

5.1 Цель и задачи работы

Цель работы: закрепление навыков работы в интегрированной среде *Lazarus*.

Задачи работы:

- знакомство с операторами цикла *for* в программах в интегрированной среде *Lazarus*;
- освоение приёмов программирования и отладки циклических алгоритмов в интегрированной среде *Lazarus*.

5.2 Порядок выполнения работы

Перед выполнением этой работы следует:

1. Ознакомиться с разделом 5.5 «Методика разработки циклических программ». В качестве дополнительной литературы можно использовать [1- 8].
2. Войти в свой личный каталог, загрузить и настроить систему программирования *Lazarus*.
3. Повторить пример разработки циклической программы, рассмотренный в п. 5.6.
4. Разработать программу по своему варианту задания (п. 5.6.2), корректируя и дополняя свой шаблон.
5. Добиться с помощью отладки, чтобы программа давала правильные результаты.
6. Ответить на контрольные вопросы.
7. Оформить отчёт и защитить его у преподавателя.

5.3 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. Откомпилированные тексты программ (в электронном виде).
5. Ответы на контрольные вопросы.
6. Результаты выполнения программ.
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 разработанных приложений;
- продемонстрировать навыки работы в интегрированной среде *Lazarus*.

5.4 Контрольные вопросы

1. Что такое цикл?
2. В каких случаях необходимо применять цикл?
3. Что такое «зацикливание» программы?
4. Как изменяются данные, если программа «зациклилась»?
5. Какие изменения нужно сделать в алгоритме (рисунок 5.1), чтобы избежать зацикливания?

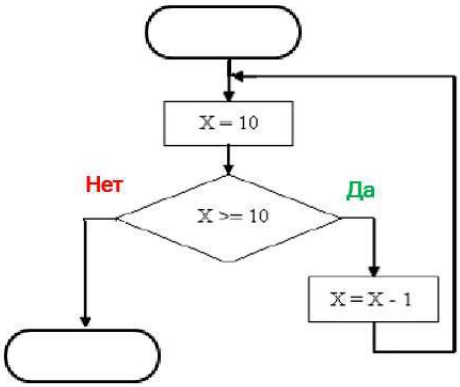


Рисунок 5.1 – Зацикленность алгоритма

6. Какие значения примут переменные в результате работы алгоритмов, представленных на блок-схемах (рисунок 5.2, *а, б, в*).

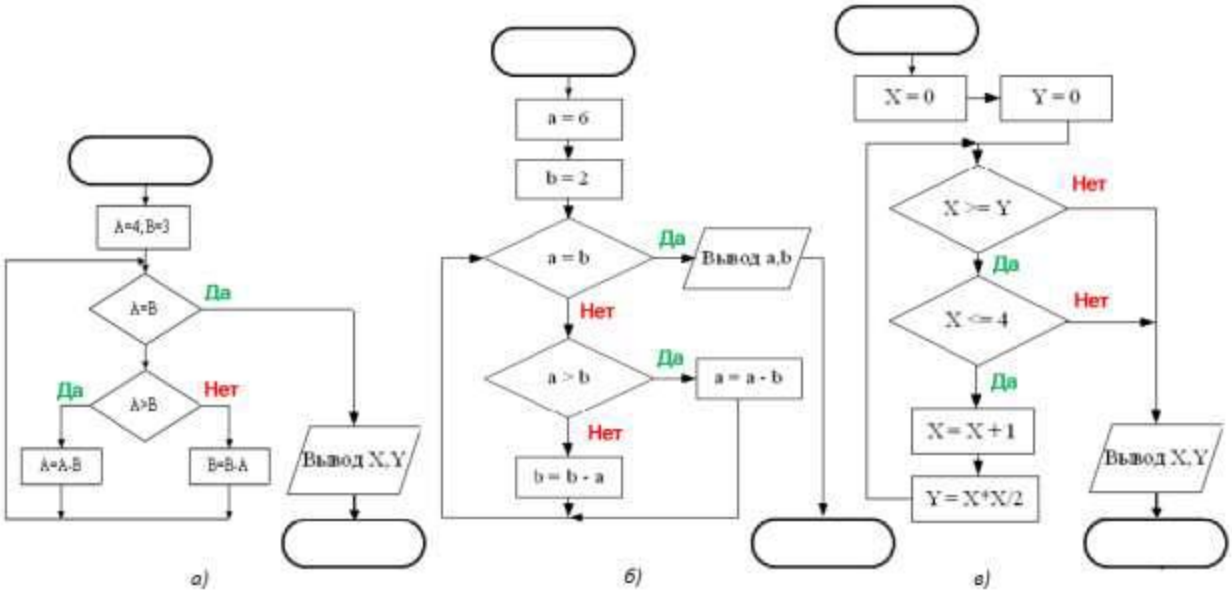


Рисунок 5.2 – Три вида алгоритмов

7. В каких случаях оператор *for* предпочтительнее использовать для организации циклов?

8. Как записывается и как работает оператор *for*?

9. Какие ограничения по типу накладываются на переменную в счетчике оператора *for*?

10. С каким шагом может изменяться счетчик оператора *for*?

11. Может ли тело оператора цикла со счётчиком не выполниться ни разу?

12. Как можно рассчитать число итераций в операторе *for*?

13. Сколько итераций совершится при использовании следующего цикла?

```
for i := 5 to 15 do
writeln(i);
```

14. Чему равно количество повторений тела оператора цикла со счётчиком, если счетчик цикла принимает все целые значения от 1 до 10?

15. Почему в программировании цикла со счётчиком существует правило: нельзя изменять счетчик цикла в теле цикла?

5.5 Методика разработки циклических программ

Одними из важнейших алгоритмических структур при решении многих задач являются циклические структуры, в которых для достижения результата определённая последовательность действий повторяется несколько раз. Возможность многократного повторения некоторой группы операторов позволяет существенно упростить схему алгоритма и сократить длину соответствующей ему программы. Программирование циклов – одна из основных и в тоже время важных задач, которые часто приходится решать программисту.

Цикл – это многократно повторяющаяся последовательность одних и тех же действий. Параметр цикла (управляющая переменная цикла) определяет момент выхода из цикла на продолжение программы. Тело цикла – последовательность инструкций, предназначенная для многократного исполнения. В теле цикла может быть всего одно действие, может быть несколько действий, может не быть ни одного действия. Последнюю разновидность цикла называют пустым циклом. Пустой цикл иногда применяют, например, для задержки времени.

Итерация – однократное выполнение тела цикла. Шаг цикла – это значение, на которое будет увеличиваться или уменьшаться параметр цикла при каждой итерации. Условие выхода или условие окончания цикла – выражение определяющее – будет ли в очередной раз выполняться итерация или цикл завершится. Если условие выхода из цикла задано неправильно – цикл может никогда не кончиться, произойдет зацикливание (бесконечный цикл).

Счётчик цикла (счётчик итераций) – переменная, хранящая текущий номер итерации. Цикл не обязательно содержит счётчик циклов. Циклические вычисления можно распределить по трём группам:

1. Счётные циклы (циклы с заданным количеством повторений) – циклические процессы, для которых количество повторений известно. Например, заполнение массива или вычисления (табулирования) функции $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$, у которой $\boldsymbol{x}$ меняется на конечном интервале от начального $\boldsymbol{X}_{\text{Н}}$ до конечного значения $\boldsymbol{X}_{\text{К}}$ с заданным шагом $\Delta\boldsymbol{x}$ (рисунок 5.3). Здесь много раз используется одна и та же формула с разными значениями аргумента $\boldsymbol{x}$.

2. Итерационные циклы – циклические процессы, завершающиеся по достижении или нарушении некоторых условий. Например, нахождение суммы бесконечного сходящегося ряда с заданной точностью.

3. Поисковые циклы – циклические процессы, из которых возможны два варианта выхода: выход по завершению процесса и досрочный выход по какому-либо дополнительному условию, например, нахождению первого отрицательного элемента массива.

Составляя циклические программы необходимо помнить, что результаты вычислений, запоминаемые переменной в левой части формулы оператора присвоения на каждом цикле, записываются вместо предыдущих значений. Чтобы не потерять эти значения на следующей итерации, нужно их как-то использовать: своевременно выводить на экран или принтер, запоминать (например, в качестве элемента массива) или употреблять как-то иначе. Так, например, в алгоритме на рисунок 5.3 выполняется циклическое вычисление функции $y = f(x)$, а результаты вычислений выводятся в документ в каждом цикле, например, в виде строки таблицы.

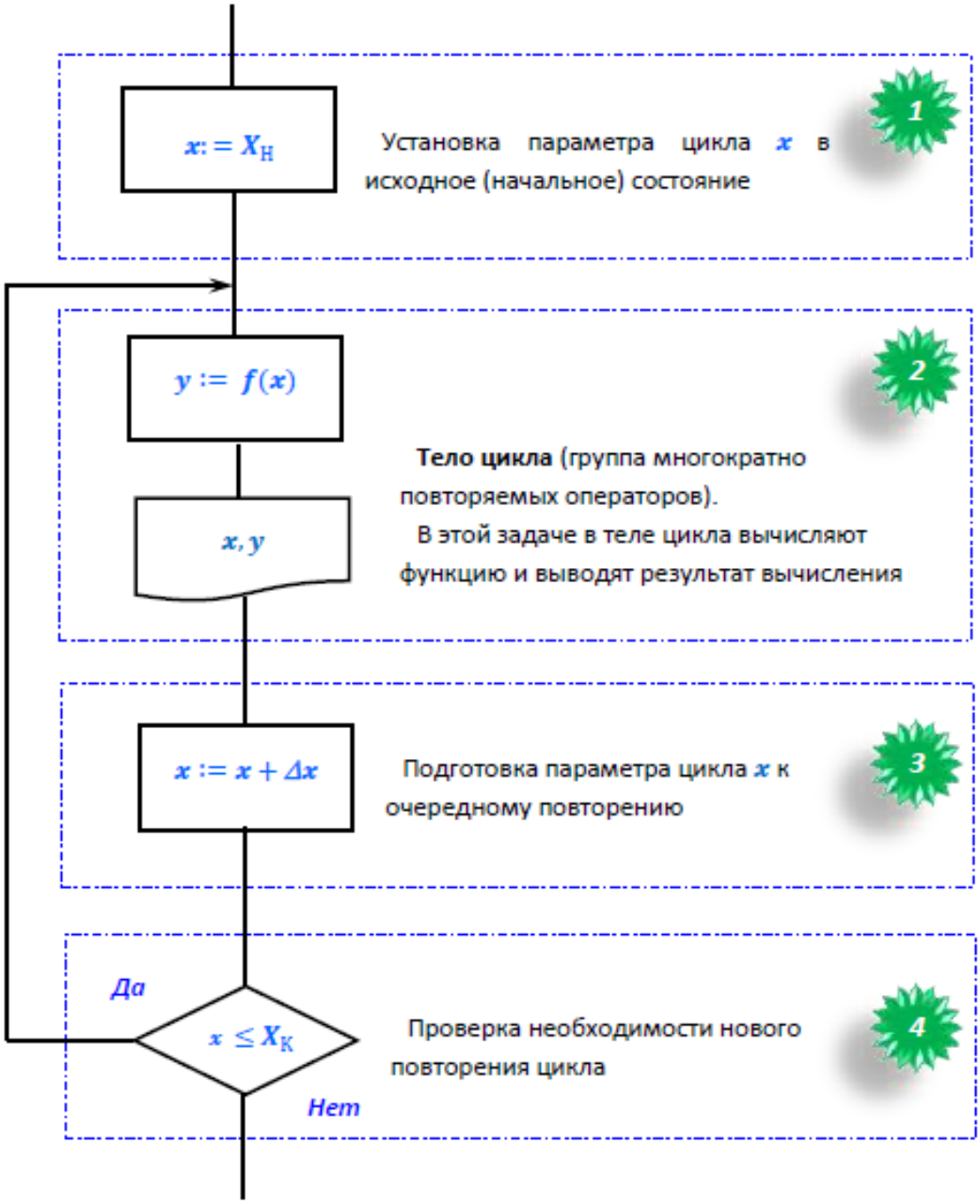


Рисунок 5.3 - Блок-схема циклической программы вычисления значений функции $y = f(x)$ в диапазоне $\boldsymbol{X}_{\text{Н}}$ .. $\boldsymbol{X}_{\text{К}}$ с шагом $\Delta\boldsymbol{x}$

В общем случае в соответствии с ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения» циклы изображают с помощью символа, состоящего из двух частей, отображающих начало и конец цикла (рисунок 5.4). Обе части символа имеют один и тот же идентификатор. Условия для инициализации, приращения, завершения и т.д. помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условие.

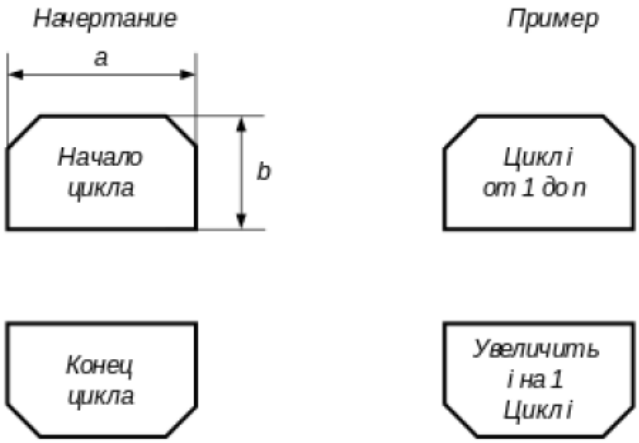


Рисунок 5.4 - Изображение циклов по ГОСТ 19.701-90

Цикл со счётчиком (рисунок 5.5) – цикл, в котором некоторая переменная изменяет своё значение от заданного начального значения до конечного значения с заданным шагом, и для каждого значения этой переменной тело цикла выполняется один раз.

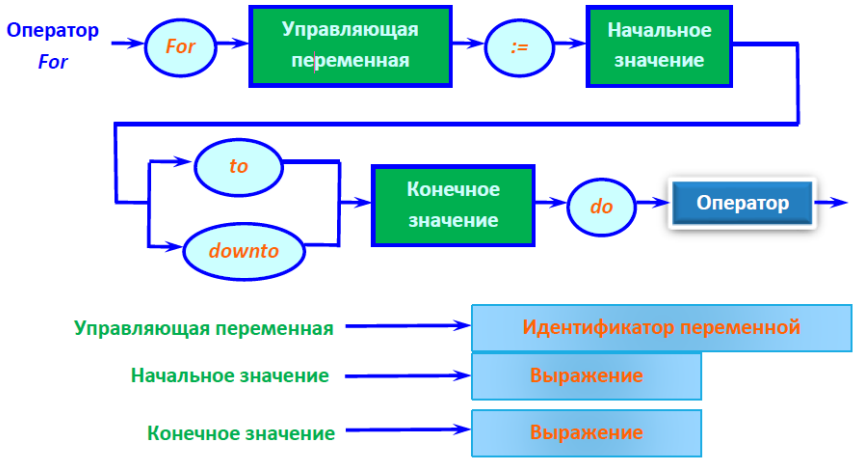


Рисунок 5.5 - Синтаксическая диаграмма оператора цикла *for*

Цикл со счётчиком *for* подходит для программирования таких циклических программ, в которых до выполнения цикла известны начальное и конечное значения счётчика повторений цикла (например, индекс первого и последнего элемента массива). Поэтому цикл со счётчиком *for* называют циклом с заданным числом повторений. Возможны два варианта исполнения цикла.

1. Если счётчик при выполнении цикла наращивает своё значение на +1:

for <Управляющая переменная> := <Нач. значение> to <Кон. значение> do
<Оператор>;

Пусть вычисленные значения выражений <**Исходное значение**> и <**Конечное значение**> равны, соответственно, $\boldsymbol{i}_{нач}$ и $\boldsymbol{i}_{кон}$ и переменой цикла (счётчику циклов) ***i*** присвоено начальное значение $\boldsymbol{i}_{нач}$. Совершается проверка – не превосходит ли переменная цикла конечное значение $\boldsymbol{i}_{кон}$? Если $\boldsymbol{i}_{нач} \leq \boldsymbol{i}_{кон}$, то выполняется тело цикла (оператор или группа операторов). Далее значение управляющей переменной увеличивается на единицу ***i*** **:=** ***i*** **+ 1** (шаг приращения переменной цикла задан неявно – служебным словом *to*)**.** Процесс, включающий проверку, выполнение тела цикла и изменение управляющей переменной, повторяется до тех пор, пока проверка не даст положительный результат, т.е. $\boldsymbol{i} > \boldsymbol{i}_{кон}$. В этом

случае оператор *for* завершает работу, переменная цикла объявляется неопределённой и осуществляется переход на оператора, следующий за оператором ***for*.**

2. Если счётчик при выполнении цикла уменьшает своё значение на -1:

for <Управляющая переменная> := <Нач. значение> downto <Кон. значение> do
<Оператор>;

Если <**Конечное значение**> больше <**Начального значения**>, тело цикла не выполняется ни разу. При каждом повторении в этом варианте цикла *for* переменная цикла ***i*** уменьшается на единицу $i := i - 1$ (отрицательный шаг приращения переменной цикла задан неявно – служебным словом *downto*). Проверяется: если значение переменной цикла больше или равно конечного значения $i \geq i_{кон}$, то тело цикла выполняется. Если же нет, и $i < i_{кон}$, то оператор *for* завершает работу, переменная цикла объявляется неопределённой и осуществляется переход на оператор, следующий за оператором *for*.

К сожалению, никакой другой шаг, кроме ±1, в цикле *for* не предусмотрен!

В блок-схемах алгоритмов программ цикл со счётчиком ***for*** может также изображаться так, как показано на рисунок 5.6 – с помощью блочного символа «Модификация», задающего закон изменения управляющей переменной.

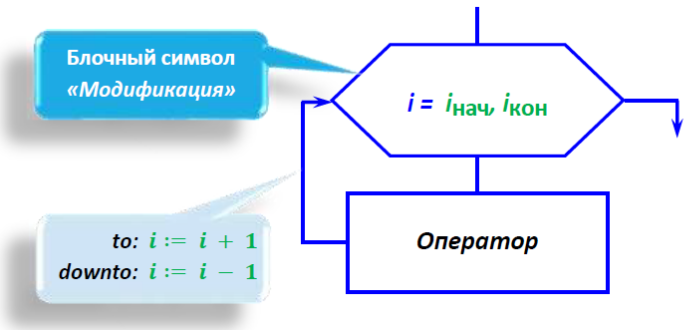


Рисунок 5.6 – Изображение цикла *for* с помощью символа «Модификация»

o

В обоих вариантах записи оператора *for*:

< **Управляющая переменная**> – переменная порядкового типа, к которым относятся данные типа *shortint, integer, word, byte* и *char*; она принимает последовательные значения от заданного < **Нач. значения** > до заданного < **Кон. значения** >;

< **Нач. значение** > и < **Кон. значение** > – выражения порядкового типа, определяющие, соответственно, начальное и конечное значение управляющей переменной цикла. Они рассчитываются только один раз перед началом выполнения оператора *for*, и не должны меняться на протяжении всего процесса его выполнения.

< **Оператор** > – единственный повторяемый оператор. Если необходимо выполнить несколько действий, то они должны быть объединены в один составной оператор путём заключения в составной оператор (группа операторов, заключённая между операторными скобками *begin* и *end*).

Число повторений цикла легко найти по формуле:

$$M = < Кон.\ значение > - < Нач.\ значение > + 1.$$

5.6 Практические задания

5.6.1 Пример для повторения

Задание. Ввести количество чисел, сами числа (вещественного типа) и вывести те, которые больше либо равны среднему значению введенных чисел. Программа

заранее не знает сколько, чисел будет введено, и их количество запрашивает у пользователя.

Создадим интерфейс программы, расставив компоненты (рисунок 5.7).

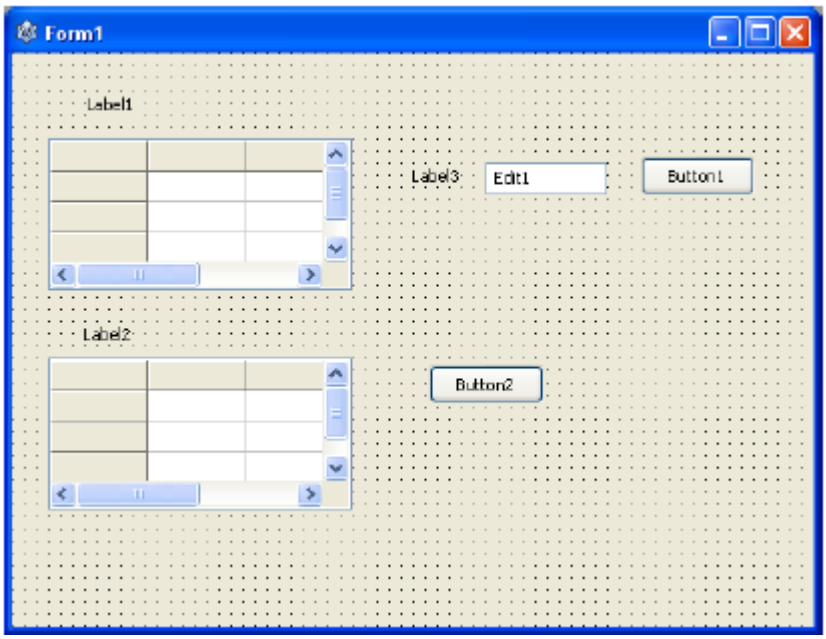


Рисунок 5.7 – Расположение компонентов на форме

Настройка компонентов *TLabel*, *TEdit* и *TButton* рассмотрена в лабораторных работах №1, 3. На форму установим два компонента *TStringGrid* – таблица строк (один под другим, палитра компонент *Additional*). Установим для обеих таблиц в **Инспекторе объектов** свойства *FixedRows* = **0** и *FixedCols* = **0**, чтобы избавиться от «шапки» у таблиц. Настроим «метки», «кнопки» и «поле ввода данных» – внешний вид элементов показан на рис 5.8.

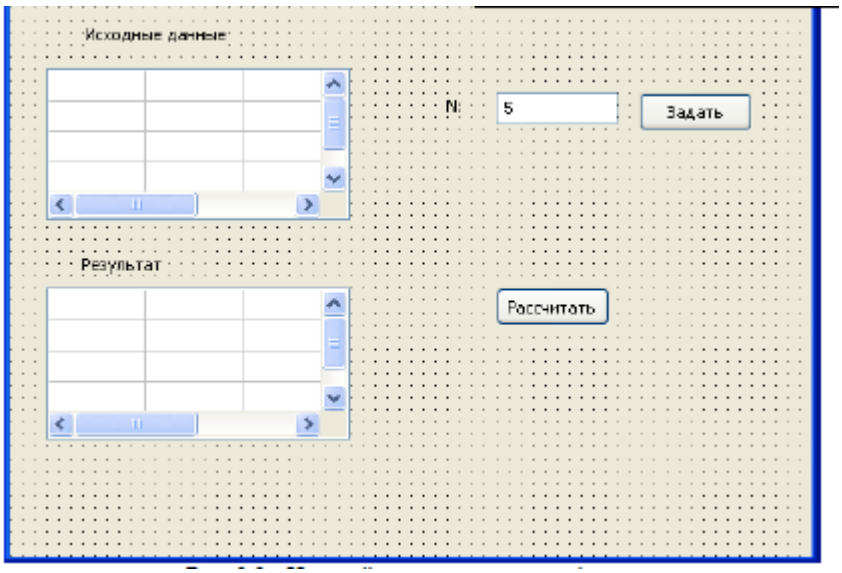


Рисунок 5.8 – Настройки компонентов на форме

По умолчанию таблица создается размером 5×5 элементов. В программе понадобится 2 строки. В верхней строке будет храниться порядковый номер элемента – индекс, а в нижней его значение. Уменьшим количество строк в таблицах до двух,

установив у таблиц свойство *RowCount* = **2** и «растянем» их в длину при помощи маркеров у компонентов *StringGridl* и *StringGrid2* (рисунок 5.9).

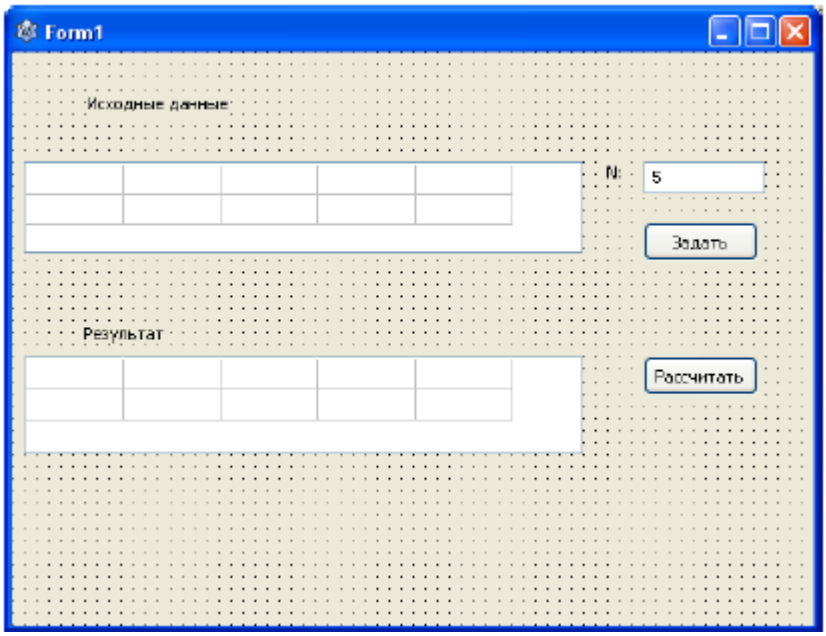


Рисунок 5.9 – Настройка компонентов *StringGrid*

Интерфейс программы практически готов, но вводить данные в таблицу пока нет возможности (перемещаться по ячейкам можно). Для разрешения редактирования ячеек таблицы нужно изменить свойство *Options* у таблиц. Данное свойство не просто текст или число, а сложное явление – множество. Каждый элемент множества может быть внесен или исключен из него.

Раскроем список всех возможных элементов множества (для доступа к элементам множества), кликнув по знаку **+**. Интересующий элемент множества *goEditing* – значения ячеек таблицы во время выполнения программы – представлен на рисунок 5.10. Установим значение *true* для разрешения редактирования ячеек таблицы.

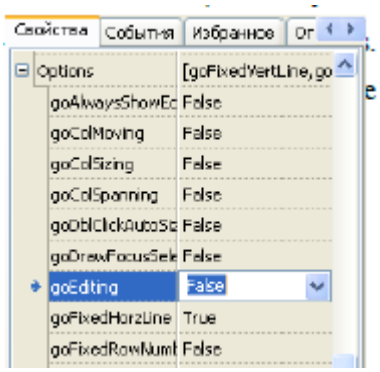


Рисунок 5.10 – Изменение свойства *Options / goEditing*

Настроим *Options* у обеих таблиц. Появилась возможность внесения текста в таблицы (рисунок 5.11).

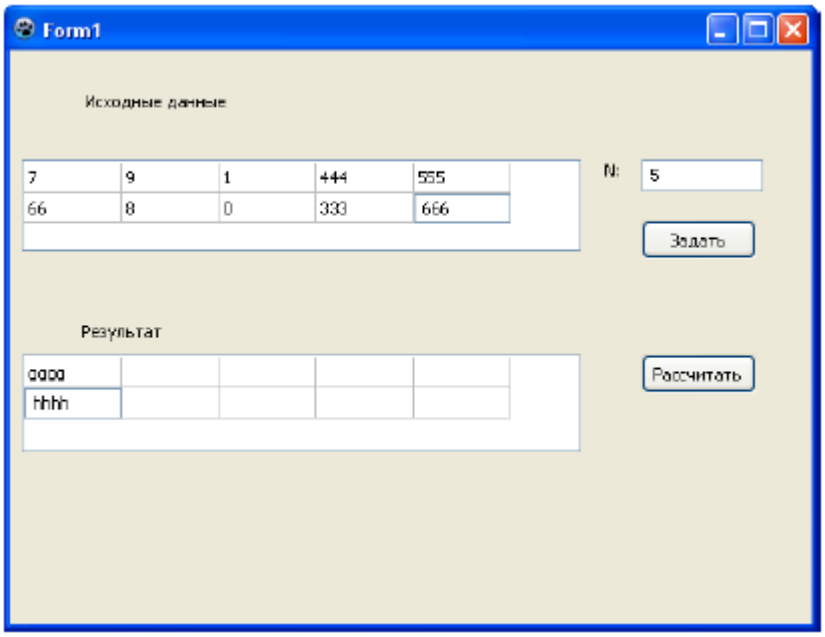


Рисунок 5.11 – Внесение данных в таблицы

Добавим «функциональную начинку» для программы – обработчики событий. Напишем обработчик для кнопки **Задать**. Он запоминает число, которое ввел пользователь в поле над этой кнопкой, создает указанное число столбцов в первой таблицы и нумерует (создает индексы) в нулевой (верхней) строке. Сделаем «двойной клик» в редакторе форм на кнопке **Задать** н добавим следующий код (рисунок 5.12).

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i,n : integer;
begin
   n := StrToInt(Edit1.Text);
   StringGrid1.ColCount:=n;
   for i:=0 to n-1 do
     StringGrid1.Cells[i,0] := IntToStr(i+1);
end;
```

Рисунок 5.12 – Обработчик кнопки **Задать**

В разделе описания переменных объявим две переменные *i* и *n*. Переменная *i* понадобится как параметр цикла, а переменная *n* – для хранения количества элементов. *n := StrToInt(Editl.Text)* – запоминает в переменной *n* количество элементов массива. Устанавливаем количество столбцов у верхней таблицы равной *N*. Третья и четвертая строки – это цикл, который выполняется ровно *N* раз и последовательно заносит в нулевую строку таблицы порядковые номера элементов. Цикл начинается с **0,** т.к. левая верхняя ячейка имеет индекс (**0,0**).

По заданию необходимо найти среднее значение и вывести во вторую таблицу только те элементы, которые больше среднего. Из курса математики известно: среднее значение – это сумма всех элементов, разделенная на их количество. Процедура (обработчик) для кнопки **Рассчитать** представлена на рисунок 5.13.

```
procedure TForm1.Button2Click(Sender: TObject);
var
  i,n : integer;
  sr : Real;
begin
  sr := 0;
  n := StrToInt(Edit1.Text);
  for i:=0 to n-1 do
    sr := sr + StrToFloat(StringGrid1.Cells[i,1]);
  sr := sr / n;
  StringGrid2.ColCount:=1;
  for i:=0 to n-1 do
    if StrToFloat(StringGrid1.Cells[i,1]) >= sr then
    begin
      StringGrid2.Cells[StringGrid2.ColCount-1,0] := IntToStr(StringGrid2.ColCount);
      StringGrid2.Cells[StringGrid2.ColCount-1,1] := StringGrid1.Cells[i,1];
      StringGrid2.ColCount:= StringGrid2.ColCount +1;
    end;
  StringGrid2.ColCount:= StringGrid2.ColCount -1;
end;
```

Рисунок 5.13 – Код обработчика для кнопки **Рассчитать**

Объявили три переменные: две целых *i* – индекс (номер столбца таблицы) и *n* – количество элементов в таблице; переменная *sr* вещественного типа данных, в ней будем подсчитывать сумму элементов. Разделив подсчитанную сумму на их количество, получим среднее значение. Выбран вещественный тип данных *real* для третьей переменной, т.к. при делении может получиться дробное значение. Последовательность действий выполняемых в процедуре:

sr := 0 – обнуление счетчика суммирования;

n := StrToInt(Editl.Text) – получение из поля *Editl* количества элементов таблице на обработку;

for i := 0 to n – 1 do

sr := sr + StrToFloat(StrmgGridl.Cells[i, 1]) – подсчет суммы элементов в таблице.

Здесь первая строка – цикл, повторяющийся ровно *n* раз, вторая строка – наращивание значения переменной *sr* за счет добавления к ней значений, хранящихся в ячейках таблицы. Перед суммированием преобразовываем значения ячеек из строк в числа с помощью функции *StrToFloat*.

Для вычисления среднего значения разделим найденную сумму на число элементов:

sr := sr / n.

В таблице как минимум один элемент больший или равный среднему значению (если таблица содержит хотя бы одно значение).

StringGrid2.ColCount := 1 – устанавливаем количество столбцов в выходной таблице равной **1**.

for i:=0 to n – 1 do – повторный перебор таблицы.

if StrToFloat(StringGrid1.Cells[i, 1]) >= sr then – проверка каждой ячейки на условие отбора: больше или равно среднему значению.

Если условие истинно, то программа выполнит действия указанные после слова *then* в операторе выбора. В синтаксисе языка *Object Pascal* после ключевого слова *then* может находиться только один оператор, а необходимо выполнить больше одного действия. Для этого объединим эти действия в операторные скобки *begin - end*.

begin

StringGrid2.Cells[StringGrid2.ColCount–1,0] := IntToStr(StnngGrid2.ColCount);

```
    StringGrid2.Cells[StringGrid2.ColCount–1,1] := StringGrid1.Cells[i,1];
    StringGrid2.ColCount := StringGrid2.ColCount+1;
  end;
```

Первая строка присваивает порядковый номер, вторая – само значение, а третья подготавливает следующий столбец для внесения туда новых элементов – расширяет таблицу на один столбец. После того, как найдены все элементы большие, либо равные среднему значению, удаляем последний столбец (он заготовлен, но не использован).

StringGrid2.ColCount := StringGrid2.ColCount – 1 – удаление пустого последнего столбца.

Сохраним проект и запустим на выполнение.

Установим количество элементов в первой таблице, внесем в нее данные н рассчитаем вторую таблицу (рисунок 5.14).

Данные для расчета необходимо вносить полностью и корректно. Количество столбцов должно быть целым н положительным. Перед расчетом необходимо заполнить все ячейки таблицы №1, т.к. программа не умеет анализировать пустые ячейки и не знает, как действовать в таких ситуациях. При недостаточном количестве информации программа аварийно остановится, выдав предупредительное сообщение. В более крупных проектах все такие ситуации проверяются, и пользователь не видит подобных сообщений, а программа подсказывает, что необходимо исправить или производит действия с учетом значений по умолчанию. Такой стиль программирования считается более дружелюбным к пользователю.

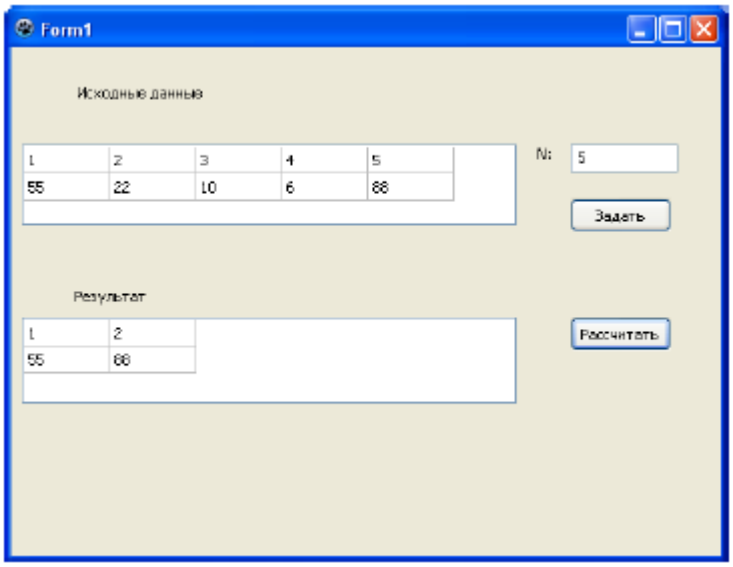


Рисунок 5.14 – Пример работы программы

5.6.2 Варианты индивидуальных заданий

Вариант №1. Ввести количество чисел, сами числа (вещественного типа) и вывести те, которые меньше либо равны среднему значению введенных чисел. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №2. Ввести количество чисел, сами числа (вещественного типа, положительные и отрицательные) и вывести те, которые больше либо равны нулю. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №3. Ввести количество чисел, сами числа (вещественного типа, положительные и отрицательные) и вывести те, которые меньше либо равны нулю. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №4. Ввести количество чисел, сами числа (вещественного типа, положительные, отрицательные, нулевые) и вывести те, которые равны нулю. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №5. Ввести количество чисел, сами числа (целого типа, положительные) и вывести те, которые являются простыми. Простое число – это число, которое делится без остатка только само на себя или на единицу. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №6. Ввести количество чисел, сами числа (целого типа, положительные) и вывести те, которые являются четными. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №7. Ввести количество чисел, сами числа (целого типа, положительные) и вывести те, которые являются нечетными. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №8. Ввести количество чисел, сами числа (целого типа, положительные) и вывести те, которые делятся без остатка на 3. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №9. Ввести количество чисел, сами числа (целого типа, положительные) и вывести те, которые делятся без остатка на 5. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №10. Ввести количество чисел, сами числа (целого типа, положительные) и вывести те, которые делятся без остатка на 7. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №11. Ввести количество чисел, сами числа (целого типа, положительные) и вывести те из них, квадратный корень из которых является целым числом. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №12. Ввести количество чисел, сами числа (вещественного типа, положительные и отрицательные) и вывести те из них, синус которых является положительным числом. Предполагается, что введенные числа – это радианная мера углов. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №13. Ввести количество чисел, сами числа (вещественного типа, положительные и отрицательные) и вывести те из них, синус которых является отрицательным числом. Предполагается, что введенные числа – это радианная мера углов. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №14. Ввести количество чисел, сами числа (вещественного типа, положительные и отрицательные) и вывести те из них, косинус которых является положительным числом. Предполагается, что введенные числа – это радианная мера углов. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №15. Ввести количество чисел, сами числа (вещественного типа, положительные и отрицательные) и вывести те из них, косинус которых является отрицательным числом. Предполагается, что введенные числа – это радианная мера углов. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №16. Ввести количество чисел, сами числа (вещественного типа, положительные и отрицательные) и вывести те из них, тангенс которых является нулевым числом. Предполагается, что введенные числа – это радианная мера углов. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №17. Ввести количество чисел, сами числа (вещественного типа, положительные и отрицательные) и вывести те из них, тангенс которых не существует (равен бесконечности). Предполагается, что введенные числа – это радианная мера углов. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №18. Ввести количество чисел, сами числа (вещественного типа, положительные и отрицательные) и вывести те из них, тангенс которых является положительным числом. Предполагается, что введенные числа – это радианная мера углов. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №19. Ввести количество чисел, сами числа (вещественного типа, положительные и отрицательные) и вывести те из них, тангенс которых является отрицательным числом. Предполагается, что введенные числа – это радианная мера углов. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №20. Ввести количество чисел, сами числа (вещественного типа, положительные и отрицательные) и вывести те из них, котангенс которых является нулевым числом. Предполагается, что введенные числа – это радианная мера углов. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №21. Ввести количество чисел, сами числа (вещественного типа, положительные и отрицательные) и вывести те из них, котангенс которых не существует (равен бесконечности). Предполагается, что введенные числа – это радианная мера углов. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №22. Ввести количество чисел, сами числа (вещественного типа, положительные и отрицательные) и вывести те из них, котангенс которых является положительным числом. Предполагается, что введенные числа – это радианная мера углов. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №23. Ввести количество чисел, сами числа (вещественного типа, положительные и отрицательные) и вывести те из них, котангенс которых является отрицательным числом. Предполагается, что введенные числа – это радианная мера углов. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №24. Ввести количество чисел, сами числа (вещественного типа, только положительные) и вывести те из них, натуральный логарифм которых является отрицательным числом. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №25. Ввести количество чисел, сами числа (вещественного типа, только положительные) и вывести те из них, натуральный логарифм которых является положительным числом. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №26. Ввести количество чисел, сами числа (вещественного типа, только положительные) и вывести те из них, десятичный логарифм которых является

отрицательным числом. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №27. Ввести количество чисел, сами числа (вещественного типа, положительные и отрицательные) и вывести те из них, экспонента которых больше единицы. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №28. Ввести количество чисел, сами числа (вещественного типа, положительные и отрицательные) и вывести те из них, экспонента которых меньше единицы. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

Вариант №29. Ввести количество чисел, сами числа (целого типа, только положительные) и вывести те из них, факториал которых больше миллиона. Программа заранее не знает, сколько чисел будет введено, и их количество запрашивает у пользователя.

6 Лабораторная работа №6 - Оператор цикла с предусловием

6.1 Цель и задачи работы

Цель работы: закрепление навыков программирования циклических алгоритмов в интегрированной среде *Lazarus*.

Задачи работы:

- знакомство с операторами цикла *while* в программах в интегрированной среде *Lazarus*;
- освоение приёмов программирования и отладки циклических алгоритмов с предусловием в интегрированной среде *Lazarus*.

6.2 Порядок выполнения работы

Перед выполнением этой работы следует:

1. Ознакомиться с разделом 6.5 «Теоретические сведения о циклах с предусловием». В качестве дополнительной литературы можно использовать [1- 8].
2. Войти в свой личный каталог, загрузить и настроить систему программирования *Lazarus*.
3. Повторить пример разработки циклической программы, рассмотренный в п. 6.6.
4. Разработать программу по своему варианту задания (п. 6.6.2), корректируя и дополняя свой шаблон.
5. Добиться с помощью отладки, чтобы программа давала правильные результаты.
6. Ответить на контрольные вопросы.
7. Оформить отчёт и защитить его у преподавателя.

6.3 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. Откомпилированные тексты программ (в электронном виде).
5. Ответы на контрольные вопросы.
6. Результаты выполнения программ.
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 разработанных приложений;
- продемонстрировать навыки работы в интегрированной среде *Lazarus*.

6.4 Контрольные вопросы

1. Как работает оператор цикла *while*?

2. Может ли тело цикла с предусловием не выполниться ни разу?
3. Сколько раз выполнится цикл *while* в программах на рисунок 6.1, *a*, *b*?

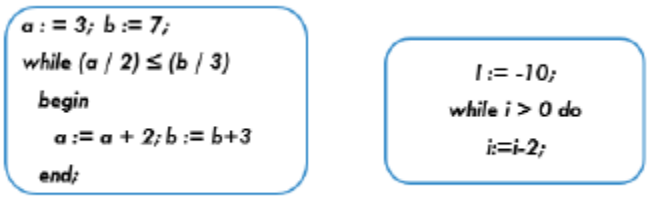


Рисунок 6.1 – Анализ работы цикла *while*

4. Когда удобнее использовать оператор *while*, а когда оператор *for*?
5. При помощи какого оператора можно досрочно выйти из цикла?
6. При помощи какого оператора можно перейти к следующему шагу выполнения цикла?
7. Чему равно значение *a* после выполнения фрагмента программы на рисунок 6.2?

```
a := 0; i := 1;
while i <= 10 do
begin
  a := a + 1; i := i + 1
end
```

Рисунок 6.2 – Фрагмент программы с циклическим оператором *while*

6.5 Теоретические сведения о циклах с предусловием

Оператор цикла с предусловием *while* (пока) позволяет многократно выполнять одни и те же действия, пока выполняется некоторое логическое условие (рисунок 6.3).

Рисунок 6.3 - Синтаксическая диаграмма оператора цикла *while*

В цикле *while* переменная цикла и шаг приращения (в отличие от цикла *for*) могут быть и вещественными, т.е. иметь дробную часть. Чтобы цикл *while* был выполнен хотя бы один раз, необходимо, чтобы перед выполнением оператора *while* значение логического выражения **Условие** было истинно (*true*). Типичными примерами использования цикла *while* являются итерационные вычисления до достижения заданной точности, поиск в массиве или в файле и т.п. Оператор *while* выполняется следующим образом:

- сначала вычисляется значение логического выражения **Условие**;
- если значение выражения **Условие** равно *true* (условие истинно), то выполняется **Оператор** (тело цикла);
- если значение выражения **Условие** равно *false* (условие ложно), то выполнение цикла *while* завершается, и управление передаётся оператору, следующему за телом цикла;
- условие вычисляется перед каждой итерацией цикла и каждый раз вновь проверяется до тех пор, пока **Условие** не станет ложным (*false*);
- если при первой же проверке **Условие** ложно (*false*), цикл не выполнится ни разу.

В блок-схемах алгоритмов программ цикл с неизвестным числом повторений *while* изображается так, как показано на рисунок 6.4 или рисунок 6.5.

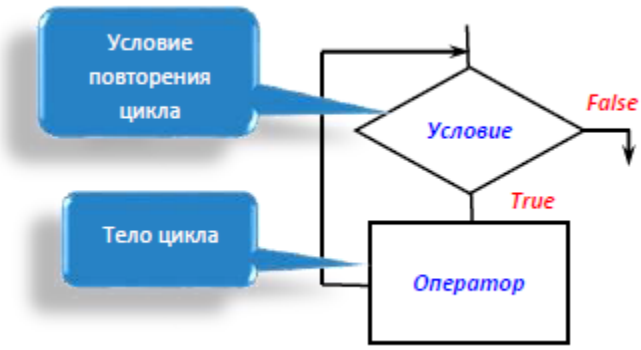


Рисунок 6.4 – Цикл с предусловием *while*

Рисунок 6.5 – Изображение цикла *while* по ГОСТ 19.701-90

Как видно из синтаксической диаграммы (рисунок 6.3), оператор *while* повторяет единственный оператор, пока **Логическое условие** принимает значение *true*. Чтобы обойти это ограничение, как и в случае оператора цикла *for* применяют составной оператор.

Чтобы не «зациклиться», нужно, чтобы в теле цикла непременно менялись значения переменных, входящих в выражение **Условие** так, чтобы когда-либо значения выражения **Условие** неизбежно стало бы ложным (*false*).

Таблица 6.1 показывает варианты оптимальной записи для разных случаев применения оператора *while*.

Таблица 6.1 - Наиболее популярные варианты записи оператора *while*

Если тело цикла состоит из одного оператора	Если тело цикла включает более одного оператора
while <**Условие**> *do* *Оператор;*	*while* <**Условие**> *do* *begin* {составной оператор} *Оператор 1;* *Оператор 2;* ... *Оператор N* *end;*

6.6 Практические задания

6.6.1 Пример для повторения

Задание. Написать программу, которая выдаст ряд чисел Фибоначчи пока их сумма (ряда) не превысит 100.

Числа Фибоначчи – это ряд, в котором каждое последующее чисто равно сумме двух предыдущих, первые два числа равны 1. Пример: 1, 1, 2, 3, 5, 8, ...

Создадим интерфейс программы (рисунок 6.6).

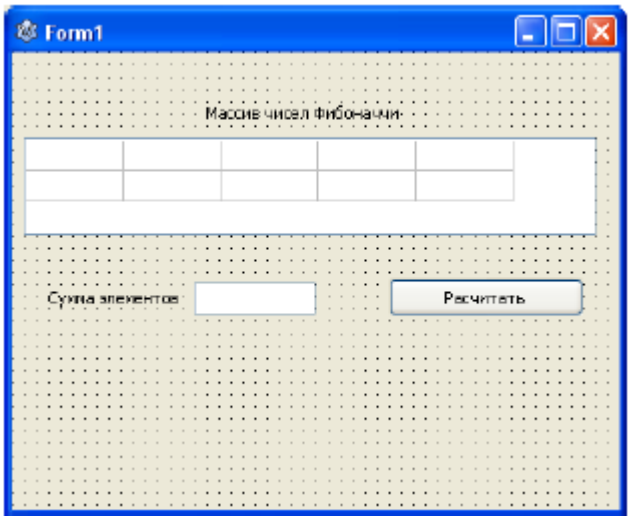


Рисунок 6.6 – Интерфейс программы

Пользователь не вводит никаких данных, а только инициирует расчет ряда чисел и получает результат в табличном виде.

Вверху расположена таблица *StringGrid1*, внизу поле для вывода суммы элементов ряда и кнопка **Рассчитать**. Таблицу настроим так же, как в лабораторной работе №5, кнопку переименуем.

Программа содержит один обработчик, связанный с кнопкой **Рассчитать**. Напишем процедуру обработчика для кнопки **Рассчитать** (рисунок 6.7).

В коде программы написано следующее:

a, b, c : integer – объявлены три переменные для хранения двух последних чисел ряда Фибоначчи и текущего числа (рассчитываемого).

s : integer – сумма данного ряда.

Выполняемые операторы:

StringGrid1.ColCoint := 2;

StringGrid1.Cells[0, 0] := ‘1’;

StringGrid1.Cells[0, 1] := ‘1’;

StringGrid1.Cells[l, 0] := ‘2’;

StringGrid1.Cells[l, l] := ‘1’;

a := 1;

b := 1;

s := a + b – инициализация (установление первоначальных) значений переменных для первых двух элементов ряда чисел Фибоначчи. Первоначальная установка количества столбцов и инициализация суммы.

while s < 100 do – цикл с предусловием, выполняется до тех пор, пока сумма (переменная *s*) меньше 100.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  a,b,c : integer;
  s : integer;
begin
  StringGrid1.ColCount:=2;
  StringGrid1.Cells[0,0]:='1';
  StringGrid1.Cells[0,1]:='1';
  StringGrid1.Cells[1,0]:='2';
  StringGrid1.Cells[1,1]:='1';
  a:=1;
  b:=1;
  s := a+b;
  while s < 100 do
  begin
    StringGrid1.ColCount:= StringGrid1.ColCount+1;
    StringGrid1.Cells[StringGrid1.ColCount -1,0] := IntToStr(StringGrid1.ColCount);
    c := a+b;
    StringGrid1.Cells[StringGrid1.ColCount -1,1] := IntToStr(c);
    a := b;
    b := c;
    s := s + c;
  end;
  Edit1.Text:=IntToStr(S);
end;
```

Рисунок 6.7 – Обработчик кнопки **Рассчитать**

Тело цикла:

StringGrid1.ColCount := StringGrid1.ColCount + 1 – увеличение количества столбцов таблице на 1;

StringGrid1.Cells[StringGrid1.ColCount – 1, 0] := IntToStr(StringGrid1.ColCount) – присвоение порядкового номера элементу ряда (нулевая строка таблицы).

c := a + b – расчет текущего элемента ряда Фибоначчи;

StringGrid1.Cells[StringGrid1.ColCount – 1, 1] := IntToStr(c) – занесение этого элемента в таблицу;

a := b;

b := c – сдвиг на один элемент по ряду;

s := s + c – подсчет суммы элементов.

Тело цикла заканчивается. За циклом:

Edit1.Text := IntToStr(s) – вывод суммы в поле *Editl*.

Сохраним проект и запустим на выполнение (рисунок 6.8).

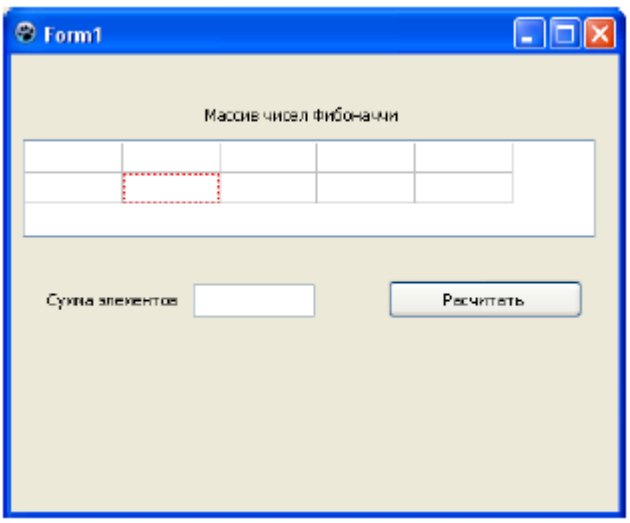


Рисунок 6.8 – Форма для расчета ряда Фибоначчи

Нажав на кнопку **Рассчитать** получим следующие данные (рисунок 6.9).

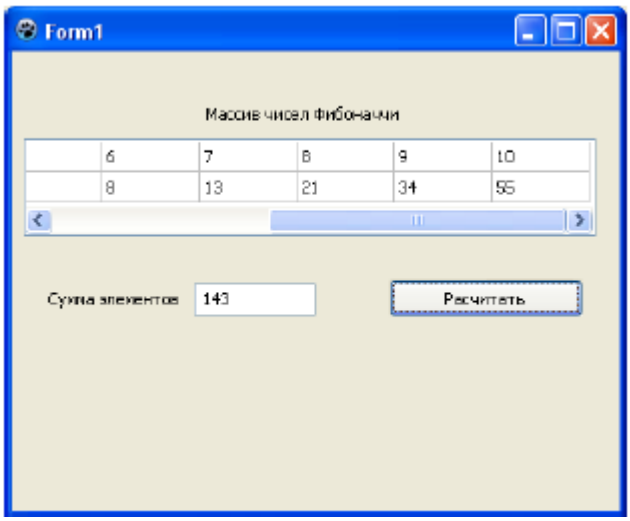


Рисунок 6.9 - Рассчитанный ряд Фибоначчи

6.6.2 Варианты индивидуальных заданий

Вариант №1. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{6}{9n^2+12n-5}$. Первый член ряда вычисляется при $n = 0$. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.001. Вычислить сумму ряда.

Вариант №2. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{24}{9n^2-12n-5}$. Первый член ряда вычисляется при $n = 0$. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.1. Вычислить сумму ряда.

Вариант №3. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{9}{9n^2+6n-8}$. Первый член ряда вычисляется при $n = 0$. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.01. Вычислить сумму ряда.

Вариант №4. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{9}{9n^2+21n-8}$. Первый член ряда вычисляется при $n = 0$. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.003. Вычислить сумму ряда.

Вариант №5. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{4-5n}{n(n-1)(n-2)}$. Первый член ряда вычисляется при $n = 3$. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.01. Вычислить сумму ряда.

Вариант №6. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{n+6}{n(n+2)(n+3)}$. Первый член ряда вычисляется при $n = 1$. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.001. Вычислить сумму ряда.

Вариант №7. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{5n+3}{n(n+1)(n+3)}$. Первый член ряда вычисляется при $n = 1$. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.005. Вычислить сумму ряда.

Вариант №8. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{4n-2}{(n^2-1)(n-2)}$. Первый член ряда вычисляется при $n = 3$. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.005. Вычислить сумму ряда.

Вариант №9. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{1}{n(n+1)(n+3)}$. Первый член ряда вычисляется при n = 1. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.001. Вычислить сумму ряда.

Вариант №10. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{2}{4n^2+8n+3}$. Первый член ряда вычисляется при n = 0. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.001. Вычислить сумму ряда.

Вариант №11. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{\left(\sin(n\sqrt{n})\right)^2}{n\sqrt{n}}$. Первый член ряда вычисляется при n = 1. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.005. Вычислить сумму ряда.

Вариант №12. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{2+(-1)^n}{n-\ln(n)}$. Первый член ряда вычисляется при n = 1. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.1. Вычислить сумму ряда.

Вариант №13. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{2}{5^{n-1}+n-1}$. Первый член ряда вычисляется при n = 0. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.001. Вычислить сумму ряда.

Вариант №14. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{1}{n}\tan\frac{1}{\sqrt{n}}$. Первый член ряда вычисляется при n = 1. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.005. Вычислить сумму ряда.

Вариант №15. Написать программу, которая выдаст ряд чисел по формуле $\sum \ln\frac{n^2+5}{n^2+4}$. Первый член ряда вычисляется при n = 0. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.001. Вычислить сумму ряда.

Вариант №16. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{1}{\sqrt{n}}\sin\frac{1}{n}$. Первый член ряда вычисляется при n = 1. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.005. Вычислить сумму ряда.

Вариант №17. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{\left(n^2+3\right)^2}{n^5+\left(\ln(n)\right)^4}$. Первый член ряда вычисляется при n = 1. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.1. Вычислить сумму ряда.

Вариант №18. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{n^3+2}{n^5+\sin(2^n)}$. Первый член ряда вычисляется при n = 0. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.005. Вычислить сумму ряда.

Вариант №19. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{n+1}{2^n\,(n-1)!}$. Первый член ряда вычисляется при n = 1. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.001. Вычислить сумму ряда.

Вариант №20. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{(n!)^2}{2^{n^2}}$. Первый член ряда вычисляется при n = 0. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.001. Вычислить сумму ряда.

Вариант №21. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{2^{n+1}(n^3+1)}{(n+1)!}$. Первый член ряда вычисляется при n = 0. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.001. Вычислить сумму ряда.

Вариант №22. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{10^n \cdot 2n!}{(2n)!}$. Первый член ряда вычисляется при $n = 0$. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.001. Вычислить сумму ряда.

Вариант №23. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{(2n+2)!}{3n+5} \cdot \frac{1}{2^n}$. Первый член ряда вычисляется при $n = 0$. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.001. Вычислить сумму ряда.

Вариант №24. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{1}{3^n} \cdot \left(\frac{n}{n+1}\right)^{-n^2}$. Первый член ряда вычисляется при $n = 0$. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.005. Вычислить сумму ряда.

Вариант №25. Написать программу, которая выдаст ряд чисел по формуле $\sum \left(1 + \frac{1}{n}\right)^{n^2} \cdot \frac{1}{4^n}$. Первый член ряда вычисляется при $n = 1$. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.001. Вычислить сумму ряда.

Вариант №26. Написать программу, которая выдаст ряд чисел по формуле $\sum \left(\frac{2n^2+1}{n^2+1}\right)^{n^2}$. Первый член ряда вычисляется при $n = 0$. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет больше миллиона. Вычислить сумму ряда.

Вариант №27. Написать программу, которая выдаст ряд чисел по формуле $\sum n^4 \left(\frac{2n}{3n+5}\right)^n$. Первый член ряда вычисляется при $n = 0$. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет больше 30. Вычислить сумму ряда.

Вариант №28. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{1}{n(\ln(3n+1))^2}$. Первый член ряда вычисляется при $n = 1$. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.001. Вычислить сумму ряда.

Вариант №29. Написать программу, которая выдаст ряд чисел по формуле $\sum \frac{1}{(2n+3)(\ln(3n+1))^2}$. Первый член ряда вычисляется при $n = 1$. Вычисление ряда прекратить, когда очередное слагаемое по абсолютной величине будет меньше 0.001. Вычислить сумму ряда.

7 Лабораторная работа №7 – Массивы

7.1 Цель и задачи работы

Цель работы: изучение структуры одномерного массива.

Задачи работы:

- знакомство с конструкцией *array of* в программах в интегрированной среде *Lazarus*;
- освоение приёмов программирования и отладки алгоритмов с одномерными массивами в интегрированной среде *Lazarus*.

7.2 Порядок выполнения работы

Перед выполнением этой работы следует:

1. Ознакомиться с разделом 7.5 «Теоретические сведения об одномерных массивах». В качестве дополнительной литературы можно использовать [1- 8].
2. Войти в свой личный каталог, загрузить и настроить систему программирования *Lazarus*.
3. Повторить пример разработки программы с одномерным массивом, рассмотренный в п. 7.6.
4. Разработать программу по своему варианту задания (п. 7.6.2), корректируя и дополняя свой шаблон.
5. Добиться с помощью отладки, чтобы программа давала правильные результаты.
6. Ответить на контрольные вопросы.
7. Оформить отчёт и защитить его у преподавателя.

7.3 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. Откомпилированные тексты программ (в электронном виде).
5. Ответы на контрольные вопросы.
6. Результаты выполнения программ.
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 разработанных приложений;
- продемонстрировать навыки работы в интегрированной среде *Lazarus*.

7.4 Контрольные вопросы

1. Что такое массивы?
2. Зачем нужны массивы?

3. Что такое элемент массива?
4. Что такое индекс массива?
5. Что такое размерность массива?
6. Как можно обратиться к ячейке массива?
7. Какого типа могут быть элементы массива?
8. Какого типа может быть индекс массива?
9. Как можно описать массив из десяти элементов целого типа?
10. Как можно осуществить ввод и вывод элементов массива с клавиатуры?
11. Как можно осуществить ввод массива через датчик случайных чисел?

7.5 Теоретические сведения о массивах

Во многих программах результатом вычислений является значение простой переменной, для записи которого в памяти компьютера выделяется одна ячейка. Если в процессе вычислений значение переменной изменяется, то после окончания работы остается лишь последнее значение результата. Чтобы записать все вычисленные значения, нужно выделить для их хранения необходимое количество ячеек памяти (массив), а текущий результат обозначить переменной с индексом (рисунок 7.1).

Массив – это ограниченная упорядоченная последовательность однотипных данных, имеющая общее имя. Для доступа к нужному компоненту массива используется его порядковый номер – индекс.

Рисунок 7.1 – Представление одномерных массивов (векторов) в компьютере

К необходимости применения массивов мы приходим всякий раз, когда требуется связать и использовать целый ряд родственных величин. Например, результаты многократных замеров напряжения на электрорадиоэлементе при изменении тока через него при снятии вольтамперной характеристики удобно рассматривать как совокупность вещественных чисел, объединенных в один сложный объект – массив измеренных напряжений.

При описании массива необходимо указать общее число входящих в массив компонентов и тип этих компонентов. Например:

var {объявления массивов}
A : array [1..10] of real; {массив *A* состоит из 10 компонентов типа *real*}
B : array [0..50] of char; {массив *B* состоит из 51 компонентов типа *char*}
C : array [-3..4] of boolean; {массив *C* состоит из 8 элементов типа *boolean*}

Как видим, при описании массива используются зарезервированные слова *array* и *of* (массив, из). За словом *array* в квадратных скобках указывается тип-диапазон, с помощью которого компилятор определяет общее число компонентов массива. Тип-диапазон задается левой и правой границами изменения индекса массива. За словом *of* указывается тип компонентов, образующих массив. Компонентами массивов могут быть как простые

переменные любых типов, так и переменные составных типов (массивов, строк, записей, объектов и т.д.).

Доступ к каждому компоненту массива в программе осуществляется с по- мощью индекса – целого числа (точнее, выражения порядкового типа), служащего своеобразным именем компонента в массиве (если левая граница типа-диапазона равна 1, индекс компонента совпадает с его порядковым номером). При упоминании в программе любого компонента массива сразу за именем массива должен следовать индекс компонента в квадратных скобках, например:

```
var
  A : array [1..10] of integer; {массив A состоит из 10 элементов типа integer}
  B : array [0..40] of char; {массив B состоит из 41 элемента типа char}
  C : array [-2..2] of boolean; {массив C состоит из 5 элементов типа boolean}
  K : integer; {индекс элементов массивов}
  begin
  ...
    B[27] := 'Ж'; {27 элементу массива B присваивается буква Ж}
    C[-2] := A[1] > A[2];
  {-2 элементу массива C присваивается результат проверки условия A[1]>A[2]}
    for k : = 1 to 10 do {обнуление массива A с помощью цикла for}
      A[k] := 0;
  end.
```

В правильно составленной программе индекс не должен выходить за пределы, определенные типом-диапазоном. Например, можно использовать элементы *A[1], B[38], C[0],* но нельзя *A[0]* или *C[38]* (определение массивов см. выше). Среда *Lazarus* может контролировать использование индексов в программе на этапе компиляции и на этапе счета программы, если установлена соответствующая опция.

Основной способ присвоения значений элементам массивов – ввод с помощью оператора *read*. Этот способ утомителен, требует повышенной концентрации внимания (особенно при вводе длинных последовательностей данных).

Нередко приходится в массив многократно вводить одну и ту же последовательность данных. Например, для тестового примера или для неизменно используемых в программе таблично заданных функциональных зависимостей. Облегчить решение подобных проблем можно путем инициализации массивов (присвоения начальных значений всем компонентам массивов) с помощью типизированных констант. Напомним, что типизированные константы приобретают указанные в их объявлениях значения лишь один раз: к моменту начала работы программы (на этапе компиляции). В ходе выполнения программы им можно присваивать другие значения, т.е. фактически они представляют собой переменные с начальными значениями.

Пример.

Проинициализировать значения одномерного массива *A[10]*.

	1	2	3	4	5	6	7	8	9	10
A	0	2.1	4.0	5.65	6.1	6.7	7.2	8.0	8.7	9.3

Рисунок 7.2 – Инициализация одномерного массива

Решение.

```
type
  tVector10 = array[1..10] of real;
const
```

A: tVector10 = (0, 2.1, 4, 5.65, 6.1, 6.7, 7.2, 8, 8.7, 9.3);

Нередко при проведении тестирования необходимо ввести несколько вариантов случайных последовательностей данных, причем важен лишь диапазон значений случайных чисел. В среде ***Lazarus*** имеются функции генерации как вещественных, так и целых псевдослучайных чисел (таблица 7.1). Причем, чтобы последовательности данных не повторялись, целесообразно перед использованием этих функций проинициализировать встроенный генератор псевдослучайных чисел случайным значением (текущим системным временем). Для этого нужно воспользоваться процедурой *Randomize*.

Таблица 7.1 - Функции генерации псевдослучайных чисел

Заголовок функции	Назначение	Тип результата	Примеры использования
Random: real	Генерирует значение случайного числа из диапазона *0 .. 1*	Вещественный	*Random⇒ 0.4876* (случайное число)
Random(Range: word): word	Генерирует значение целого случайного числа из диапазона *0 .. Range*	Целый	*Random(1000) ⇒ 745* (случайное число)

Пример.

Заполнить одномерный массива *A[1000]* целыми псевдослучайными числами в диапазоне ±250.

Решение.

Диапазон случайных чисел для этой задачи *Diapason = 250 – (-250) = 500***.** Функция *Random(500)* будет генерировать псевдослучайные числа от 0 до 500. Для смещения диапазона в отрицательную область нужно от каждого случайного числа вычесть 250.

```
const
  N = 1000; {Количество компонентов в массиве}
  Diapason = 500; {Диапазон значений случайных чисел}
  Diap2 = Diapason div 2; {Смещение диапазона значений случайных чисел}
var
  A: array [1..N] of integer; {Массив чисел}
  i : integer; {Индекс элемента массива}
  ...
  {Наполняем в цикле массив случайными числами и выводим их на экран:}
  for i := 1 to N do
  begin {Начало тела цикла}
    A[i] := random(Diapason) – Diap2;
    write('A[', i, '] = ', A[i]:5);
  end;
  writeln; {Переход на начало строки}
  ...
```

7.6 Практические задания

7.6.1 Пример для повторения

Задание. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – целочисленные.

Начальные условия:

$$S_i = 2S_{i-2} - 3S_{i-1} + 5; \quad S_1 = 2; \quad S_2 = 4; \; n = 16.$$

Разработаем интерфейс программы.

Пользователь не вводит никаких данных, а только инициирует расчет ряда чисел и получает результат в табличном виде. Интерфейс программы на рисунок 7.3.

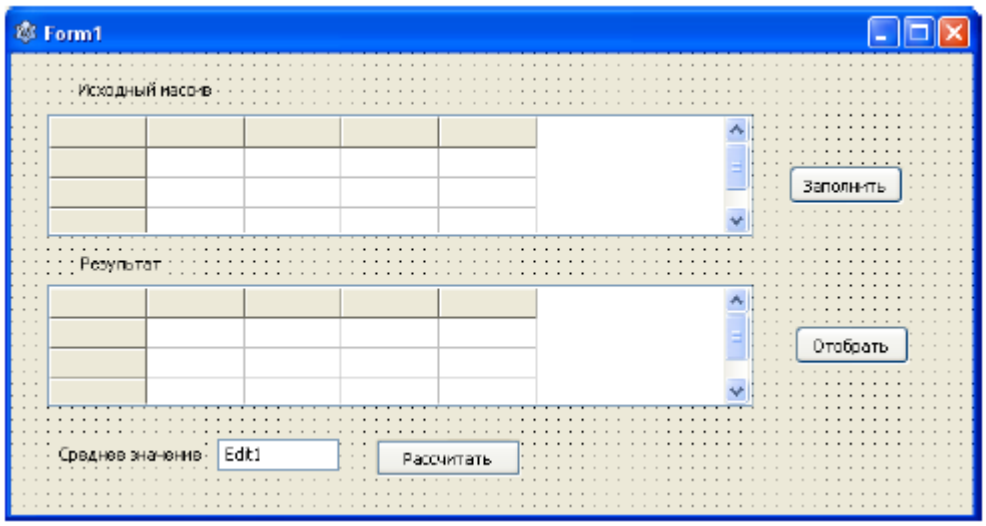


Рисунок 7.3 - Интерфейс программы

На форме разместим две таблицы (*TStringGrid*), три метки (*TLabel*) и три кнопки (*TButton*). Переименуем их как показано на рисунке 7.3 (изменяем свойство *Caption* у элементов формы).

Настроим обе таблицы: уберем зафиксированные строки и столбцы; уменьшим количество строк до двух. Назначим свойствам таблиц (рисунок 7.4):

FixedRows = 0;

FixedCols = 0;

RowCout = 2.

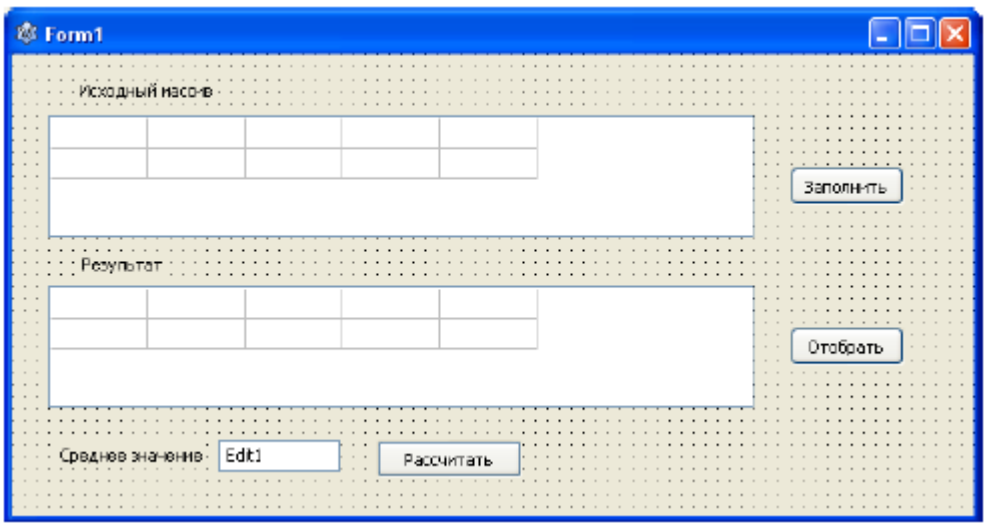


Рисунок 7.4 – Настройка таблиц, меток и кнопок у формы

Интерфейс готов, напишем обработчики событий.

Обращаться к ряду чисел будем неоднократно – вычислить его, подсчитывать среднее значение и выдавать часть элементов удовлетворяющих условию задания. Чтобы не вычислять ряд чисел при каждом обращении, воспользуемся массивом для хранения элементов ряда.

Опишем массив как глобальный массив модуля в разделе *var* за формой программы (рисунок 7.5). По условию элементов 16, они целые и нумерация начинается с 1. Этой информации достаточно для описания массива в среде *Lazarus*. Создадим массив с именем *mas*.

```
  var
    Form1: TForm1;
    mas : array[1..16] of integer;
35
36  implementation
```

Рисунок 7.5 - Создание массива из 16 целых элементов

Напишем процедуру вычисляющую все элементы ряда в первую таблицу и привяжем ее к кнопке **Заполнить**. Понадобится несколько вспомогательных переменных:

i – индекс массива;

A, *A1*, *A2* – элементы массива, соответственно текущий, перед текущим и перед-перед текущим.

Процедура вычисления элементов, занесения в массив и таблицу представлена на рисунок 7.6.

```
  { TForm1 }

40 procedure TForm1.Button1Click(Sender: TObject);
  var
    i : integer;
    A,A1,A2 : integer;
  begin
45    A1 := 4;
    A2 := 2;
    mas[1] := A2;
    mas[2] := A1;
    for i:= 3 to 16 do
50  begin
      A := 2*A2 -3 *A1 +5;
      mas[i] := A;
      A2:=A1;
      A1:=A;
55  end;
    StringGrid1.ColCount:=16;
    for i := 1 to 16 do
  begin
      StringGrid1.Cells[i-1,0] := IntToStr(i);
60    StringGrid1.Cells[i-1,1] := intToStr(mas[i]);
  end;
  end;
```

Рисунок 7.6 – Расчет ряда, занесение в массив и в таблицу

Запоминаем в массиве вспомогательные переменные *A1* и *A2* – первые два числа. Вычисляем и заносим в массив элементы ряда с 3-го по 16-й. Устанавливаем количество столбцов в таблице 16. Выводим в верхнюю строку порядковый номер элемента, а в нижнюю строку значение элемента. Процедура обработчик события готова. Сохраним проект, запустим на выполнение.

Проверим работу алгоритма по кнопке **Заполнить**. Программа выдает ряд чисел (рисунок 7.7).

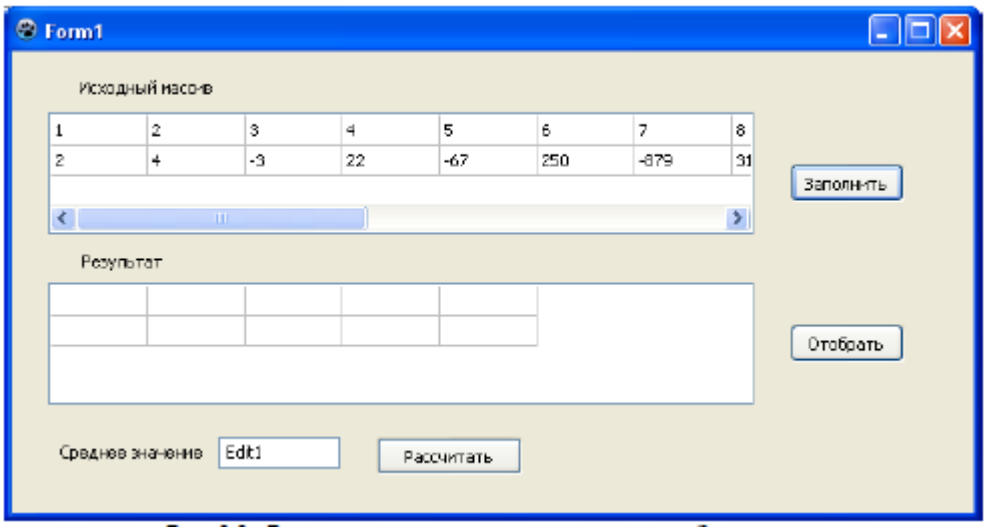


Рисунок 7.7 – Результат работы программы по кнопке **Заполнить**

Вычислим среднее значение ряда – напишем обработчик кнопки **Рассчитать**, а результат запишем в поле *Editl*. Для вычисления понадобится переменная – индекс массива и переменная под среднее значение. Код процедуры расчета подробно рассматривался в примере лабораторной работы №5, принцип расчета не изменился (рисунок 7.8).

```
65  procedure TForm1.Button3Click(Sender: TObject);
    var
      i : integer;
      S : Real;
    begin
70    S := 0;
      for i:= 1 to 16 do
        S := S + mas[i];
      S := S / 16;
      Edit1.Text:= FloatToStr(S);
75  end;
```

Рисунок 7.8 – Код обработчика – расчет среднего значения

Запустим программ (рисунок 7.9) и вычислим среднее значение ряда.

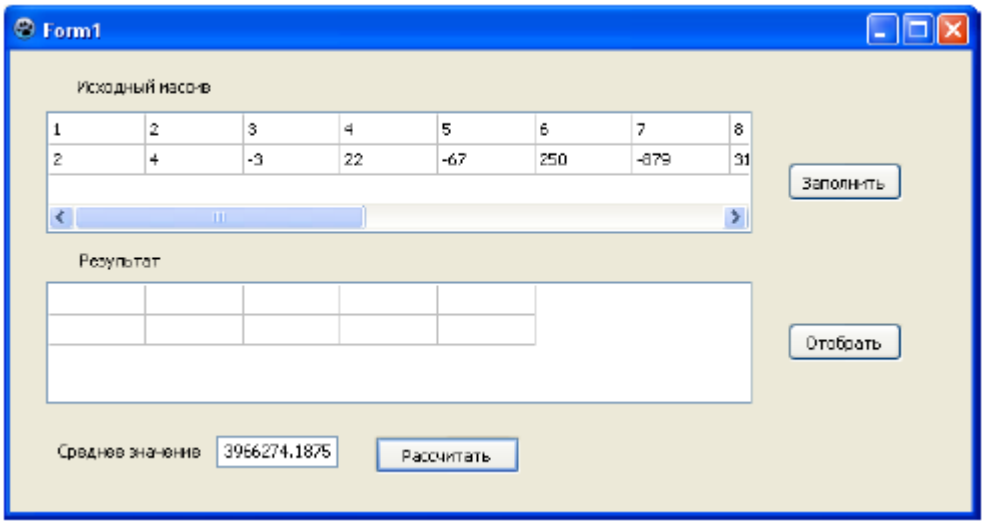


Рисунок 7.9 – Результат расчета среднего значения ряда

В нижнюю таблицу выведем только те элементы массива, которые больше среднего значения. Напишем обработчик – выбор элементов больше среднего и привяжем его к кнопке **Отобрать** (рисунок 7.10).

```
65
.   procedure TForm1.Button2Click(Sender: TObject);
.   var
.     i : integer;
.     s : Real;
70  begin
.     S := 0;
.     for i:= 1 to 16 do
.       S := S + mas[i];
.     S := S / 16;
75    StringGrid2.ColCount:=1;
.     for i:= 1 to 16 do
.       if mas[i] > S then
.       begin
.         StringGrid2.Cells[StringGrid2.ColCount-1,0] :=IntToStr(i);
80        StringGrid2.Cells[StringGrid2.ColCount-1,1] := IntToStr(mas[i]);
.         StringGrid2.ColCount:=StringGrid2.ColCount+1;
.       end;
.     StringGrid2.ColCount:=StringGrid2.ColCount-1;
.   end;
```

Рисунок7.10 – Процедура отбора значений больше среднего

Алгоритм отбора элементов массива такой же, как в примерах предыдущих лабораторных работ. Первоначально (как в предыдущей процедуре) рассчитываем среднее значение ряда, а затем проходим по всему массиву и отбираем только те элементы, которые больше среднего значения.

Запустим программу на выполнение и получим результирующий массив (рисунок 7.11).

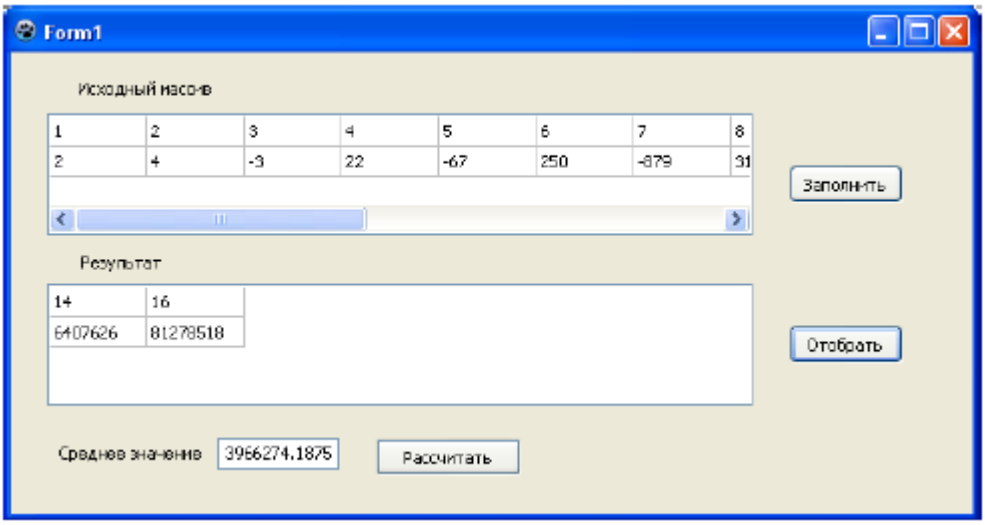


Рисунок 7.11 – Отбор элементов больше среднего

7.6.2 Варианты индивидуальных заданий

Вариант №1. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{6}{9n^2+12n-5}$. Первый элемент массива вычисляется при $n = 0$.

Вариант №2. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{24}{9n^2-12n-5}$. Первый элемент массива вычисляется при $n = 0$.

Вариант №3. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{9}{9n^2+6n-8}$. Первый элемент массива вычисляется при $n = 0$.

Вариант №4. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{9}{9n^2+21n-8}$. Первый элемент массива вычисляется при $n = 0$.

Вариант №5. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{4-5n}{n(n-1)(n-2)}$. Первый элемент массива вычисляется при $n = 3$.

Вариант №6. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{n+6}{n(n+2)(n+3)}$. Первый элемент массива вычисляется при $n = 1$.

Вариант №7. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10.

Каждый элемент массива вычисляется по формуле $\frac{5n+3}{n(n+1)(n+3)}$. Первый элемент массива вычисляется при $n = 1$.

Вариант №8. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{4n-2}{(n^2-1)(n-2)}$. Первый элемент массива вычисляется при $n = 3$.

Вариант №9. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{1}{n(n+1)(n+3)}$. Первый элемент массива вычисляется при $n = 1$.

Вариант №10. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{2}{4n^2+8n+3}$. Первый элемент массива вычисляется при $n = 0$.

Вариант №11. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{\left(\sin(n\sqrt{n})\right)^2}{n\sqrt{n}}$. Первый элемент массива вычисляется при $n = 1$.

Вариант №12. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{2+(-1)^n}{n-\ln(n)}$. Первый элемент массива вычисляется при $n = 1$.

Вариант №13. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{2}{5^{n-1}+n-1}$. Первый элемент массива вычисляется при $n = 0$.

Вариант №14. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{1}{n}\tan\frac{1}{\sqrt{n}}$. Первый элемент массива вычисляется при $n = 1$.

Вариант №15. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\ln\frac{n^2+5}{n^2+4}$. Первый элемент массива вычисляется при $n = 0$.

Вариант №16. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{1}{\sqrt{n}}\sin\frac{1}{n}$. Первый элемент массива вычисляется при $n = 1$.

Вариант №17. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{(n^2+3)^2}{n^5+(\ln(n))^4}$. Первый элемент массива вычисляется при $n = 1$.

Вариант №18. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{n^3+2}{n^5+\sin(2^n)}$. Первый элемент массива вычисляется при $n = 0$.

Вариант №19. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{n+1}{2^n\,(n-1)!}$. Первый элемент массива вычисляется при $n = 1$.

Вариант №20. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 5. Каждый элемент массива вычисляется по формуле $\frac{(n!)^2}{2n^2}$. Первый элемент массива вычисляется при $n = 0$.

Вариант №21. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 10. Каждый элемент массива вычисляется по формуле $\frac{2^{n+1}(n^3+1)}{(n+1)!}$. Первый элемент массива вычисляется при $n = 0$.

Вариант №22. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 5. Каждый элемент массива вычисляется по формуле $\frac{10^n \cdot 2n!}{(2n)!}$. Первый элемент массива вычисляется при $n = 0$.

Вариант №23. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 5. Каждый элемент массива вычисляется по формуле $\frac{(2n+2)!}{3n+5}\frac{1}{2^n}$. Первый элемент массива вычисляется при $n = 0$.

Вариант №24. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 9. Каждый элемент массива вычисляется по формуле $\frac{1}{3^n} \cdot \left(\frac{n}{n+1}\right)^{-n^2}$. Первый элемент массива вычисляется при $n = 0$.

Вариант №25. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 9. Каждый элемент массива вычисляется по формуле $\left(1+\frac{1}{n}\right)^{n^2} \cdot \frac{1}{4^n}$. Первый элемент массива вычисляется при $n = 1$.

Вариант №26. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 9. Каждый элемент массива вычисляется по формуле $\left(\frac{2n^2+1}{n^2+1}\right)^{n^2}$. Первый элемент массива вычисляется при $n = 0$.

Вариант №27. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 9. Каждый элемент массива вычисляется по формуле $n^4\left(\frac{2n}{3n+5}\right)^n$. Первый элемент массива вычисляется при $n = 0$.

Вариант №28. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 9. Каждый

элемент массива вычисляется по формуле $\frac{1}{n(\ln(3n+1))^2}$. Первый элемент массива вычисляется при $n = 1$.

Вариант №29. Создать и отобразить исходный массив. Выбрать элементы больше среднего значения. Элементы массива – вещественные. Количество элементов – 9. Каждый элемент массива вычисляется по формуле $\frac{1}{(2n+3)(\ln(3n+1))^2}$. Первый элемент массива вычисляется при $n = 1$.

8 Лабораторная работа №8 – Программирование с использованием записей

8.1 Цель и задача работы

Цель работы – знакомство с данными типа запись в среде *Lazarus.*

Задача работы – получение практических навыков в разработке программ, использующих записи.

8.2 Порядок выполнения работы

В ходе выполнения этой работы следует:

1. Изучить описание лабораторной работы, обратив особое внимание на правила описания и использования записей в среде *Lazarus*. В качестве дополнительной литературы можно использовать [1 - 8].
2. Ответить письменно на контрольные вопросы.
3. Войти в свой личный каталог и настроить интегрированную среду *Lazarus* для последующей работы. Записать файл конфигурации в личный каталог.
4. Повторить пример из п. 8.6.1.
5. Выполнить индивидуальное задание из п. 8.6.2, в том числе провести разработку алгоритма и программы.
6. Продемонстрировать работоспособность программы.
7. Оформить отчет по лабораторной работе и защитить его у преподавателя.

8.3 Отчетность

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

1. Тема и цель работы.
2. Индивидуальное задание.
3. Схема алгоритма решения задачи.
4. Текст программы и вводимые тестовые исходные данные.
5. Откомпилированный текст программы (в электронном виде).
6. Результаты выполнения программы.
7. Ответы на контрольные вопросы.
8. Выводы.

При защите отчета по работе студент должен:

- уметь отвечать на контрольные вопросы;
- обосновать структуру выбранного алгоритма и показать его работоспособность;
- уметь пояснять работу программы;
- продемонстрировать навыки работы в среде *Lazarus.*

8.4 Контрольные вопросы

1. Что такое запись?
2. В каких случаях целесообразно использовать данные типа запись?
3. Как объявляются тип и переменные типа запись?
4. Поясните правила объявления типизированных констант записи.
5. Для чего используются составные имена и из каких компонентов они состоят?

6. Каково назначение и форма оператора присоединения *with*?
7. Какова форма обращения к полям записей без использования оператора *with*?
8. Какие преимущества дает оператор присоединения?
9. Какие операции допустимы над полями записей?

8.5 Применение записей при разработке баз данных

Большинство современных информационных технологий, задач экономики и управления базируется на концепции баз данных (БД). По определению, БД – это совокупность взаимосвязанных файлов, в которых хранятся определенным образом организованные данные, адекватно отражающие соответствующую предметную область (например, информацию о моделях электрорадиоэлементов, материалах, проектных документах и т.д.). Невозможно представить себе САПР, которая не опиралась бы на БД, хранящую используемые при проектировании данные.

Как правило, файлы БД содержат большое число однотипных записей. Каждая запись объединяет под одним именем логически связанные данные разного типа, относящиеся к одному объекту. Запись состоит из фиксированного числа объектов, называемых ее полями. Поле – это переменная определенного типа. Объем памяти, необходимый для хранения записи, складывается из длин полей.

Записи – наиболее общий и гибкий тип данных языка Паскаль. Следовательно, задача разработки программ, позволяющих работать с записями, представляется весьма жизненной.

Определить запись в языке Паскаль можно следующим образом:

```
type {Раздел объявлений нестандартных пользовательских типов данных}
  <Имя типа записи> = record {Заголовок объявления типа записи}
    <идентификатор поля>: <тип поля>;
    ……
    <идентификатор поля>: <тип поля>
      {Список определений элементов записи - полей}
  end; {Конец определения записи}
```

Элементами (полями) записи могут быть любые стандартные типы данных, и, кроме того, определенные пользователем множества, файлы, другие записи и массивы.

Экземпляр записи можно создать в разделе описания переменных:

```
var
  <Идентификатор переменной, …> : <Имя типа записи>;
```

Например, в БД Госавтоинспекции возможны записи такого вида:

```
type {раздел объявления пользовательских типов данных}
  tData = record {заголовок типа данных: запись даты}
    Year : integer; {поле № 1, год}
    Month : 1..12; {поле № 2, номер месяца}
    Day : 1..31; {поле № 3, день}
  end;

tAuto = record {заголовок типа данных: запись об автомобиле}
  GosNomer : string[11];{поле № 1, номер государственной регистрации}
  Marka : string[20];{поле № 2, марка автомобиля}
  God, GodReg : tData; {одинаковые поля №3 и 4, записи о датах выпуска и}
  {регистрации автомобиля, состоящие из полей Year, Month, Day записи tData}
  FIO : string[40]; {поле № 5, фамилия, имя и отчество владельца}
  Adres : string[50]; {поле № 6, адрес владельца}
```

```
end; {конец определения записи об автомобиле}

var {раздел объявления переменных}
  Auto : tAuto; {экземпляр записи об автомобиле}
  TablAuto : array[1..50] of tAuto; {таблица (массив) записей об автомобилях}
```

Обращение к полю записи выполняется с помощью составного имени. Первая часть составного имени – это имя записи, вторая часть уточняет имя поля. В качестве разделителя между ними применяется точка. Изменять значения полей можно либо присвоением, либо вводом. Например:

Auto.Marka := 'Kia Ceed';

{присвоение полю *Marka* записи *Auto* нового значения}

Read(TablAuto[5].*GosNomer);*

{ввод нового значения поля *GosNomer* для 5 элемента массива записей *TablAuto*}

Для вложенных полей (подструктур) уточнений имени может быть несколько, например:

Auto.God.Year := 2017; {присвоение полю *Year* записи *Auto* нового значения}

Допускается применение оператора присваивания и к записям в целом, если они имеют один и тот же тип. Например, чтобы значения всех полей текущей записи *Auto* запомнить в *i*-ом элементе массива записей *TablAuto,* следует записать оператор:

TablAuto[*I*] *:= Auto*

Для краткого обращения к полям записи удобно использовать оператор присоединения *with*. В операторе *with* к полям одной или более конкретных переменных типа запись можно обращаться, используя только лишь простые идентификаторы полей:

with <Имя записи> do
  Оператор;
 {В этом единственном операторе указывают только имена полей записи}
 {<Имя записи> добавляется к каждому имени поля автоматически}

Для распространения действия оператора присоединения *with* на некоторую группу операторов используют составной оператор (заключают эту группу операторов в *begin .. end*).

С учетом введенных ранее описаний записи, приведем пример использования оператора *with*:

```
with Auto.God do
{Теперь к полям записи Auto достаточно обращаться только }
{ по их именам}
if Month = 12
then
begin
  Month := 1;
  Year := Year + 1
end
else
  Month := Month + 1;
```

Это эквивалентно следующему фрагменту программы:

```
{Для доступа к полям записи Auto используются длинные составные }
{ имена}
if Auto.God.Month = 12
then
begin
  Auto.God.Month := 1;
```

```
  Auto.God.Year := Auto.God.Year + 1
end
else
  Auto.God.month := Auto.God.Month + 1;
```

При необходимости, можно использовать более сложный тип записи, содержащий несколько вариантов структур полей. Любой из вариантов структуры записи может быть активизирован в зависимости от значения поля-переключателя.

В заключение приведем пример еще одной возможности задания начальных значений полей записей – с помощью типизированных констант (переменных с начальными значениями).

```
type {Объявления нестандартных типов данных}
  tPoint = record {запись о координатах точки}
    X, Y : Real;
  end;

  tVector = array[0..1] of tPoint; {Массив из двух точек}

 {Перечислимый тип данных - месяцы}
  tMonth = (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec);

  tDate = record {Запись о некоторой дате}
    Day: 1..31;
    Month: tMonth;
    Year: 1900..2999;
  end;

const {Примеры определения типизированных констант}
  Origin : tPoint = (X : 0.0; Y : 0.0); {Начальные координаты точки Origin}
  {Координаты начала и конца отрезка прямой Line}
  Line : tVector = ((X : -3.1; Y : 1.5), (X : 5.8; Y : 3.0));
  {Запоминается дата дня рождения SomeDay}
  SomeDay : tDate = (Day : 5; Month : Sep; Year : 1974);
```

8.6 Практические задания

8.6.1 Пример для повторения

Задание. Создать структуру типа запись, состоящую из следующих элементов:

1 – Номер по порядку.
2 – Фамилия.
3 – Имя.
4 – Отчество.
5 – Год рождения.
6 – Балл за физику.
7 – Балл за математику.
8 – Балл за русский.

Заполнить таблицу минимум 10 разными записями. Данные рекомендуется записать в процедуре *FormCreate* единожды (иначе их придется вводить постоянно при запуске программы). Отобразить исходную таблицу и таблицу перечислений фамилий

абитуриентов, у которых при поступлении по физике было 100 баллов.

Разработаем интерфейс программы. Пользователь не вводит никаких данных, но может редактировать персональные данные абитуриентов и получает результат в табличном виде. Расставим основные элементы на форме (рисунок 8.1).

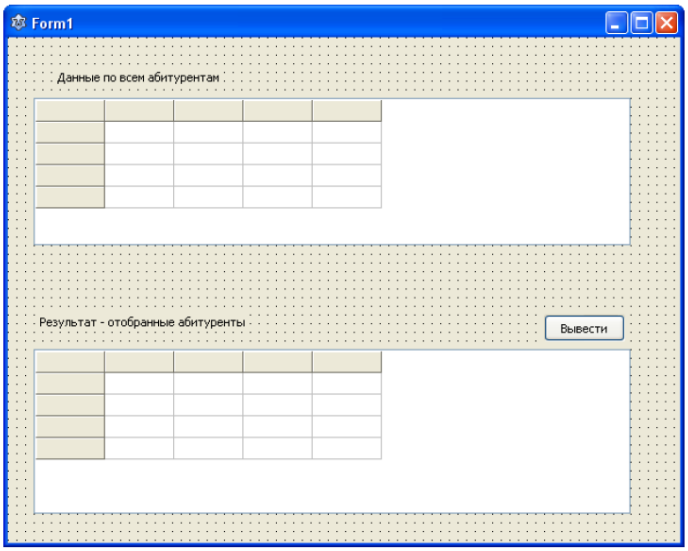


Рис. 8.1 – Интерфейс программы

Переименуем метку и кнопку так, как показано на рисунке 8.1. Настроим таблицы, установив свойства *StringGridl*:

RowCount = 8;

ColCount = 11;

Option.goEditing = true; (разрешаем редактировать данные таблицы из окна программы).

В редакторе формы через контекстное меню у таблицы *StringGridl* **Редактировать *StringGrid1*** вносим данные об абитуриентах и баллы от 0 до 100 (рисунок 8.2).

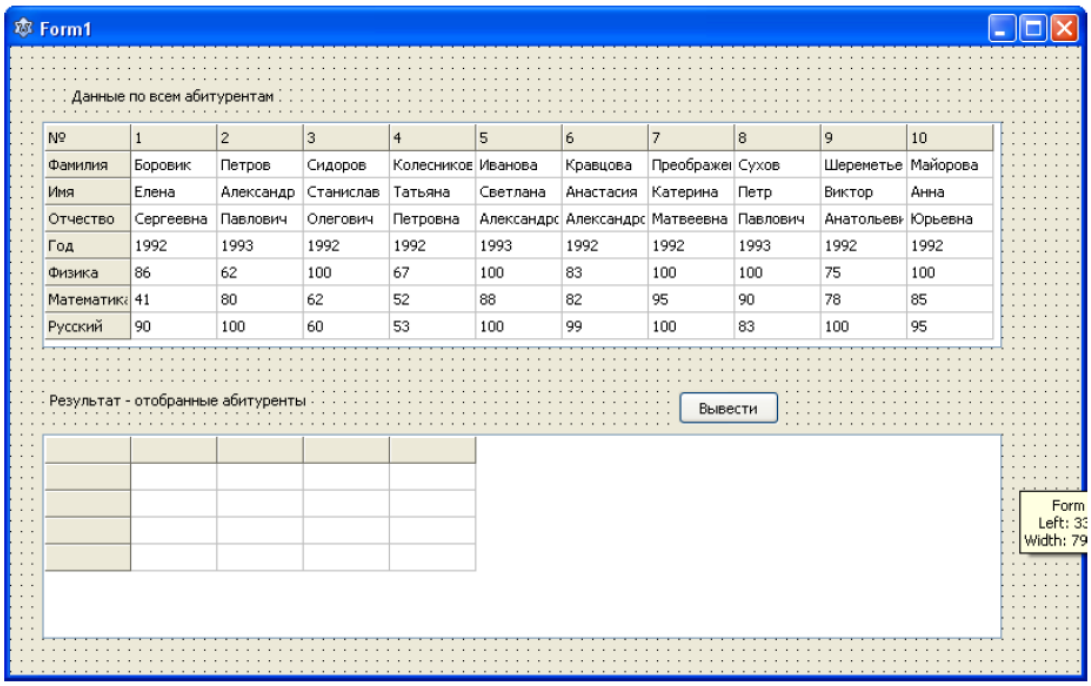


Рисунок 8.2 – Заполненная таблица исходных данных

Настроим таблицу результатов (нижняя), установив значения свойств в инспекторе объектов:

RowCount = 4;

ColCount = 2.

В редакторе формы через контекстное меню у таблицы *StringGrid2* **Редактировать *StringGrid2*** вносим первоначальные данные в «шапку» таблицы. В результате форма имеет вид (рисунок 8.3).

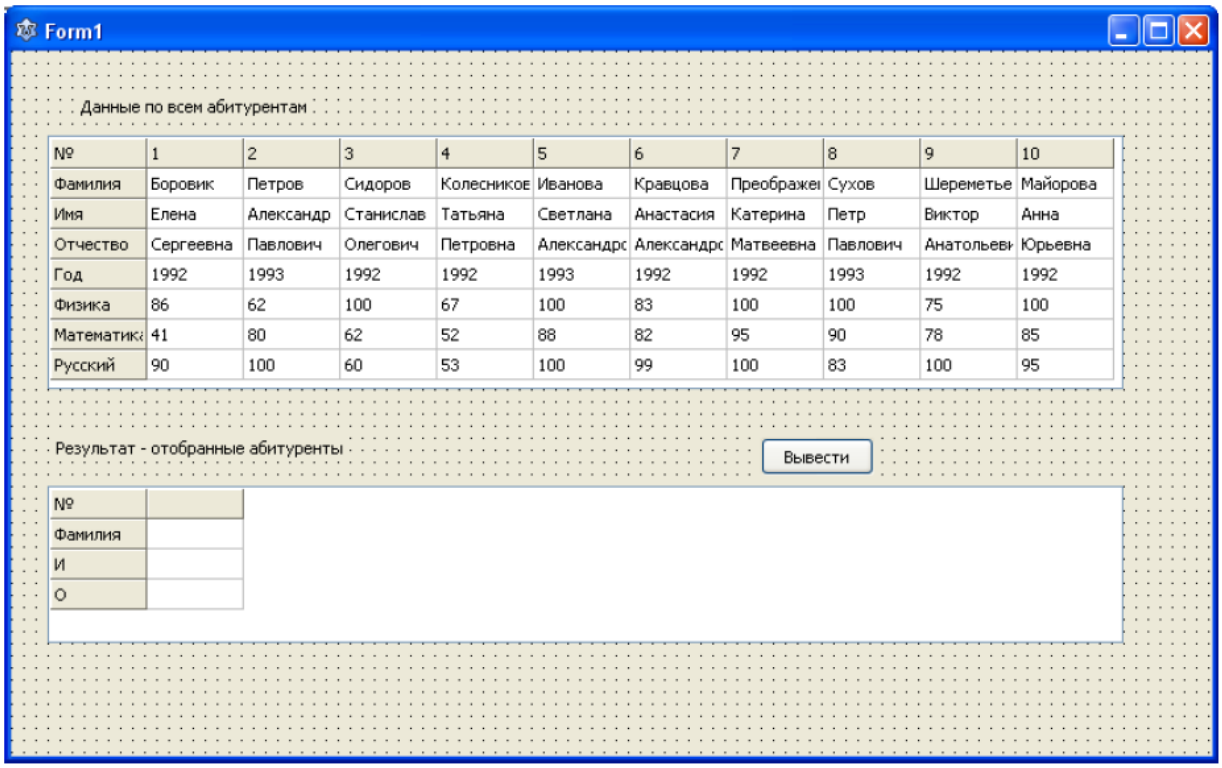
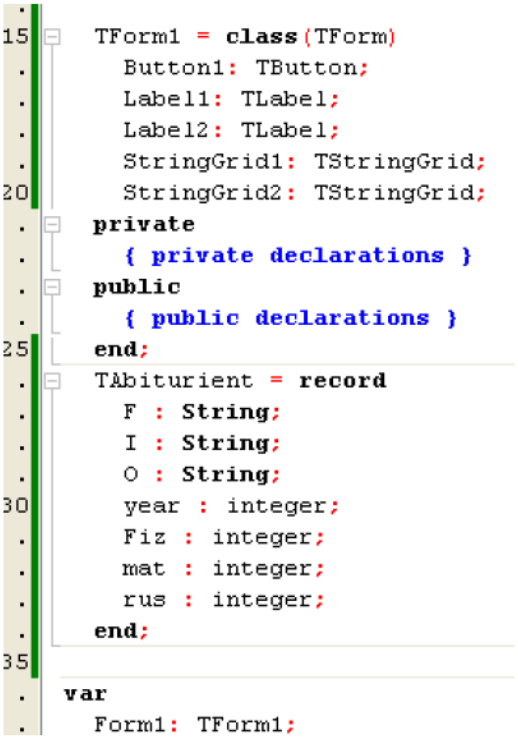


Рисунок 8.3 – Окончательный дизайн формы — настроена таблица *StringGrid2*

Каждый абитуриент обладает наборам персональных данных, поэтому сгруппируем их в структуру – запись и назовем ее *tAbiturient*. Опишем ее в разделе *type*, т.к. здесь описываются новые типы данных (рисунок 8.4). По принятым стилевым правилам оформления кода все типы имеют префикс *t*.

Структура типа запись объявляется с помощью ключевого слова *record*, за которым идет список всех полей. Поля перечислены через «**;**», а после последнего поля стоит ключевое слово *end* – говорящее, что список полей структуры закончен. Каждый элемент списка структуры имеет такой же способ описания, как и переменных в разделе *var*. Сначала идет имя поля и через «**:**» указывается его тип данных. Под **Фамилию**, **Имя**, **Отчество** выбрали строковые поля, а под **Год рождения** и **Оценки** – целые. Описывать новый тип данных необходимо после описания формы и до описания глобальных переменных в разделе *var*.

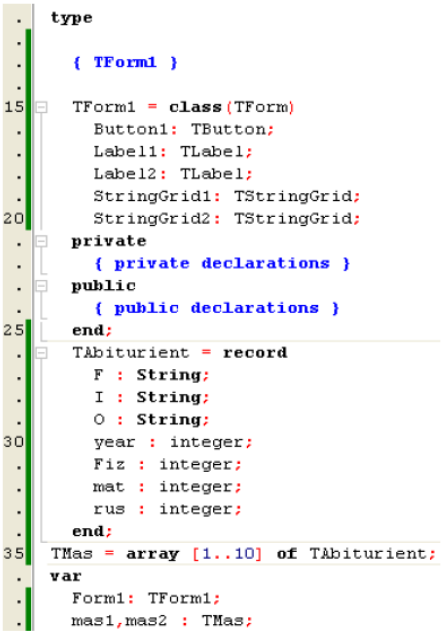


```
  TForm1 = class(TForm)
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    StringGrid1: TStringGrid;
    StringGrid2: TStringGrid;
  private
    { private declarations }
  public
    { public declarations }
  end;
  TAbiturient = record
    F : String;
    I : String;
    O : String;
    year : integer;
    Fiz : integer;
    mat : integer;
    rus : integer;
  end;

var
  Form1: TForm1;
```

Рисунок 8.4 – Объявление структуры *tAbiturient* в разделе *type*

Такая структура описывает данные только для одного абитуриента, а по заданию их 10. Чтобы не создавать 10 переменных, создадим массив, т.к. каждый абитуриент имеет однотипные персональные данные (рисунок 8.5). Объявлено два новых типа данных, затем определены переменные соответствующих типов данных. Необходимо два массива абитуриентов (рисунок 8.5). В первом мы будем хранить всех абитуриентов, а во втором только тех, которые удовлетворяют критериям (отобранные абитуриенты). Код представлен ниже.



```
type

  { TForm1 }

  TForm1 = class(TForm)
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    StringGrid1: TStringGrid;
    StringGrid2: TStringGrid;
  private
    { private declarations }
  public
    { public declarations }
  end;
  TAbiturient = record
    F : String;
    I : String;
    O : String;
    year : integer;
    Fiz : integer;
    mat : integer;
    rus : integer;
  end;
TMas = array [1..10] of TAbiturient;
var
  Form1: TForm1;
  mas1,mas2 : TMas;
```

Рисунок 8.5 – Добавление к уже существующему типу нового типа – массива *tMas*; определение переменных массивов *mas1* и *mas2*

Опишем обработчик кнопки **Вывести**. Он выполняет:

1. Перенос данных *StringGridl* в массив *mas1*.
2. Из массива *masl* выбирает подходящих абитуриентов и копирует их в массив *mas2*.
3. Запоминает количество скопированных элементов в массиве *mas2*.
4. Выдает искомые данные из массива *mas2* в таблицу *StringGrid2*.

Код обработчика представлен ниже (рисунок 8.6).

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : integer;
  n : integer;
begin
  for i := 1 to 10 do
  begin
    masl[i].F := StringGrid1.Cells[i, 1];
    masl[i].I := StringGrid1.Cells[i, 2];
    masl[i].O := StringGrid1.Cells[i, 3];
    masl[i].Year := StrToInt(StringGrid1.Cells[i, 4]);
    masl[i].Fiz := StrToInt(StringGrid1.Cells[i, 5]);
    masl[i].Mat := StrToInt(StringGrid1.Cells[i, 6]);
    masl[i].Rus := StrToInt(StringGrid1.Cells[i, 7]);
  end;
  n := 0;
  for i := 1 to 10 do
  if masl[i].Fiz = 100 then
  begin
    n := n + 1;
    mas2[n] := masl[i];
  end;
  StringGrid2.ColCount := n + 1;
  for i := 1 to n do
  begin
    StringGrid2.Cells[i, 0] := IntToStr(i);
    StringGrid2.Cells[i, 1] := mas2[i].F;
    StringGrid2.Cells[i, 2] := mas2[i].I;
    StringGrid2.Cells[i, 3] := mas2[i].O;
  end;
end;
```

Рисунок 8.6 – Код обработчика кнопки **Вывести**

Запускаем программу и получаем искомый результат (рисунок 6.8).

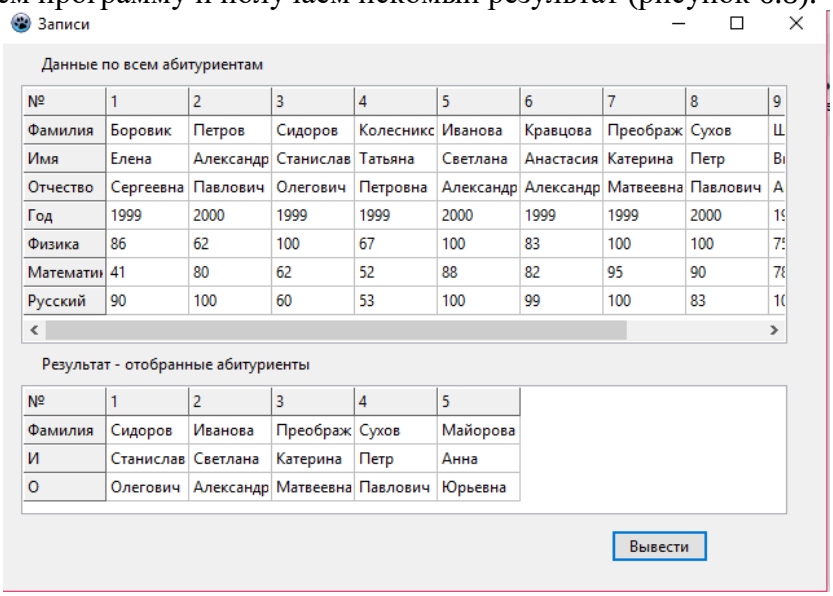


Рисунок 8.7 – Результат выполнения программы

8.6.2 Варианты индивидуальных заданий

Вариант 1. Создать запись «Самолеты», состоящую из следующих полей:
1 – Номер по порядку.
2 – Наименование типа самолета.
3 – Фамилия конструктора.
4 – Год выпуска.
5 – Количество мест.
6 – Взлетная масса.
Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений типов самолетов, у которых год выпуска моложе 2014 года.

Вариант 2. Создать запись «Расчет движения», состоящую из следующих полей:
1 – Номер по порядку.
2 – Наименование авиарейса (маршрут).
3 – Тип самолета.
4 – Количество рейсов в неделю.
5 – Протяженность маршрута в км.
6 – Среднее количество авиапассажиров на одном рейсе.
Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений авиарейсов, протяженность которых больше 5000 км.

Вариант 3. Создать запись «Перевозки», состоящую из следующих полей:
1 – Номер по порядку.
2 – Тип самолета.
3 – Номер борта.
4 – Количество рейсов в неделю.
5 – Налет самолета в тысячах км.
6 – Налет самолета в часах.
Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений бортов самолетов, у которых налет 10 000 часов.

Вариант 4. Создать запись «Сооружения аэропорта», состоящую из следующих полей:
1 – Номер по порядку.
2 – Наименование сооружения.
3 – Площадь сооружения в м2.
4 – Этажность сооружения.
5 – Год сооружения.
6 – Балансовая стоимость в млн. руб.
Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений сооружений, у которых этажность больше 3 этажей.

Вариант 5. Создать запись «Ремонт аэродромных сооружений», состоящую из следующих полей:
1 – Номер по порядку.

2 – Наименование ремонтных работ.
3 – Шифр ремонтных работ.
4 – Вид ремонтных работ.
5 – Стоимость ремонта в млн. руб..
6 – Наименование подрядчика.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений ремонтных работ, стоимость которых превышает 10 млн. руб.

Вариант 6. Создать запись «Кассы авиабилетов», состоящую из следующих полей:
1 – Номер по порядку.
2 – ФИО кассира.
3 – Количество проданных билетов за день.
4 – Суммарная выручка за день.
5 – Дата продаж.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений ФИО кассиров, с выручкой за день больше 100 000 руб.

Вариант 7. Создать запись «Характеристики персональных компьютеров», состоящую из следующих полей:
1 – Номер по порядку.
2 – Тип процессора.
3 – Тактовая частота в ГГц.
4 – Емкость оперативной памяти в ГБ.
5 – Емкость винчестера в ТБ.
6 – Тип монитора.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений компьютеров с емкостью винчестера больше 1 ТБ.

Вариант 8. Создать запись «Города», состоящую из следующих полей:
1 – Номер по порядку.
2 – Название города.
3 – Количество жителей.
4 – Площадь города в га.
5 – Год основания.
6 – Количество школ.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений городов с количеством школ больше 100.

Вариант 9. Создать запись «Московские мосты», состоящую из следующих полей:
1 – Номер по порядку.
2 – Наименование моста.
3 – Высота в м.
4 – Ширина в м.
5 – Количество опор.
6 – Протяженность в м.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений мостов, длина которых больше 100 м.

Вариант 10. Создать запись «Линии московского метро», состоящую из следующих полей:

1 – Номер по порядку.
2 – Наименование линии.
3 – Количество станций.
4 – Год пуска.
5 – Протяженность в км.
6 – Количество поездов.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений линий с количеством станций меньше 10.

Вариант 11. Создать запись «Легковые автомобили», состоящую из следующих полей:

1 – Номер по порядку.
2 – Марка.
3 – Цвет.
4 – Стоимость в млн. руб.
5 – Изготовитель.
6 – Максимальная скорость в км/ч.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений автомобилей, максимальная скорость которых больше 150 км/ч.

Вариант 12. Создать запись «Продажа программных продуктов», состоящую из следующих полей:

1 – Номер по порядку.
2 – Наименование программного продукта.
3 – Фирма-изготовитель.
4 – Стоимость в тыс. руб.
5 – Объем занимаемого дискового пространства в МБ.
6 – Количество комплектов на складе.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений программных продуктов, изготовленных Microsoft.

Вариант 13. Создать запись «Абонентская плата за телефон», состоящую из следующих полей:

1 – Номер по порядку.
2 – ФИО абонента.
3 – Номер телефона.
4 – Год установки.
5 – Наименование тарифа.
6 – Средняя оплата телефонных разговоров в месяц.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений телефонов, установленных раньше 1991 года.

Вариант 14. Создать запись «Города», состоящую из следующих полей:

1 – Номер по порядку.

2 – Наименование детского сада.

3 – Номер детского сада.

4 – Количество детей.

5 – Район города.

6 – Средняя плата за месяц.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений детских садов, расположенных в Кировском районе.

Вариант 15. Создать запись «Сотрудники», состоящую из следующих полей:

1 – Номер по порядку.

2 – ФИО.

3 – Табельный номер.

4 – Дата рождения.

5 – Оклад в тыс. руб.

6 – Трудовой стаж.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений сотрудников с трудовым стажем больше 25 лет

Вариант 16. Создать запись «Ведомость зарплаты за текущий месяц», состоящую из следующих полей:

1 – Номер по порядку.

2 – ФИО.

3 – Наименование подразделения.

4 – Табельный номер.

5 – Количество рабочих часов в неделю.

6 – Размер зарплаты.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений сотрудников с кафедры РЭТЭМ.

Вариант 17. Создать запись «Музеи», состоящую из следующих полей:

1 – Номер по порядку.

2 – Наименование.

3 – Тематика.

4 – Адрес.

5 – Время работы.

6 – Стоимость билета.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений музеев с военной тематикой.

Вариант 18. Создать запись «Экскурсии», состоящую из следующих полей:

1 – Номер по порядку.

2 – Наименование.

3 – Страна.

4 – Стоимость в тыс. руб.

5 – Продолжительность в днях.
6 – Вид транспорта.
Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений экскурсий, на которые нужно добираться самолетом.

Вариант 19. Создать запись «Кинофильмы», состоящую из следующих полей:
1 – Номер по порядку.
2 – Наименование кинотеатра.
3 – Стоимость билета.
4 – Время начала сеансов.
5 – Адрес кинотеатра.
6 – Количество мест в кинотеатре.
Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений кинотеатров, в которых количество мест в кинозале больше 100.

Вариант 20. Создать запись «Книги – почтой», состоящую из следующих полей:
1 – Номер по порядку.
2 – Наименование книги.
3 – ФИО писателя.
4 – Номер по каталогу.
5 – Издательство.
6 – Стоимость книги.
Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений книг, написанных Б.Акуниным.

Вариант 21. Создать запись «Квартиры», состоящую из следующих полей:
1 – Номер по порядку.
2 – Адрес.
3 – Площадь в м2.
4 – Сторона света.
5 – Стоимость 1 м2.
6 – Этаж.
7 – Количество комнат.
Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений квартир, окна которых выходят на юг.

Вариант 22. Создать запись «Склад товаров», состоящую из следующих полей:
1 – Номер по порядку.
2 – Название магазина.
3 – Наименование товара.
4 – Артикул товара.
5 – Цена единицы товара.
6 – Количество товара.
Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений магазинов, в которых продают электрокамины.

Вариант 23. Создать запись «Телевизоры на складе магазина», состоящую из следующих полей:

1 – Номер по порядку.

2 – Наименование телевизора.

3 – Фирма-изготовитель.

4 – Стоимость.

5 – Диагональ экрана.

6 – Количество на складе.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений телевизоров, диагональ которых превышает 42 дюйма.

Вариант 24. Создать запись «Холодильники на складе магазина», состоящую из следующих полей:

1 – Номер по порядку.

2 – Наименование холодильника.

3 – Фирма-изготовитель.

4 – Стоимость.

5 – Емкость морозильной камеры.

6 – Количество на складе.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений холодильников, которые имеются на складе в количестве больше 10 штук.

Вариант 25. Создать запись «Студенты ТУСУРа», состоящую из следующих полей:

1 – Номер по порядку.

2 – Факультет.

3 – Курс.

4 – Группа.

5 – ФИО.

6 – Средний балл за весь период обучения.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений студентов, у которых средний балл 4,75 и выше.

Вариант 26. Создать запись «Бани Томска», состоящую из следующих полей:

1 – Номер по порядку.

2 – Наименование бани.

3 – Адрес.

4 – Выходной день.

5 – Часы работы.

6 – Количество залов.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений бань, которые не работают в субботу

Вариант 27. Создать запись «Рок-группы России», состоящую из следующих полей:

1 – Номер по порядку.

2 – Название коллектива.

3 – Количество музыкантов.

4 – Направление рок-музыки.

5 – Год основания группы.

6 – Среднее количество гастролей в год.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений рок-групп, состоящих из более, чем 4 человека.

Вариант 28. Создать запись «Российские цари», состоящую из следующих полей:

1 – Номер по порядку.

2 – Имя царя.

3 – Год рождения.

4 – Год вхождения на престол.

5 – Продолжительность правления.

6 – Количество одержанных побед в войнах.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений царей, которые правили больше 10 лет.

Вариант 29. Создать запись «Смартфоны», состоящую из следующих полей:

1 – Номер по порядку.

2 – Наименование смартфона.

3 – Фирма-изготовитель.

4 – Стоимость.

5 – Емкость памяти.

6 – Средняя продолжительность работы от одной зарядки аккумулятора.

Заполнить таблицу 10 разными записями. Данные записать в процедуре *FormCreate* единожды. Отобразить исходную таблицу и таблицу перечислений смартфонов Apple IPhone X с любой емкостью памяти.

9 Лабораторная работа №9 – Двумерные массивы

9.1 Цель и задачи работы

Цель работы – программирование алгоритмов обработки двумерных массивов.

Задачи работы:

1. Изучение двумерных массивов (матриц).
2. Изучение способов обработки матричных данных.
3. Освоение форматного вывода двумерных массивов в виде таблиц.

9.2 Порядок выполнения работы

В ходе выполнения этой работы следует:

1. Изучить описание лабораторной работы. В качестве дополнительной литературы можно использовать [1 - 8].
2. Ответить письменно на контрольные вопросы.
3. Войти в свой личный каталог и настроить интегрированную среду *Lazarus* для последующей работы. Записать файл конфигурации в личный каталог.
4. Повторить пример из п. 9.6.1.
5. Выполнить индивидуальное задание из п. 9.6.2.
6. Продемонстрировать работоспособность программы.
7. Оформить отчет по лабораторной работе и защитить его у преподавателя.

9.3 Отчетность

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

1. Тема и цель работы.
2. Индивидуальное задание.
3. Схема алгоритма решения задачи.
4. Текст программы и вводимые тестовые исходные данные.
5. Откомпилированный текст программы (в электронном виде).
6. Результаты выполнения программы.
7. Ответы на контрольные вопросы.
8. Выводы.

При защите отчета по работе студент должен:

- уметь отвечать на контрольные вопросы;
- обосновать структуру выбранного алгоритма и показать его работоспособность;
- уметь пояснять работу программы;
- продемонстрировать навыки работы в среде *Lazarus***.**

9.4 Контрольные вопросы

1. Поясните понятия двумерного массива, матрицы.
2. Что обозначают индексы матрицы?
3. Сколько элементов в матрице из 7 строк и 9 столбцов?
4. Дайте понятие квадратной матрицы, диагоналей квадратной матрицы.
5. Приведите пример описания двумерных массивов на языке Паскаль.

6. Поясните порядок использования вложенных циклов при вводе элементов двумерного массива

9.5 Теоретические сведения о двумерных массивах

Двумерный массив – это индексированная с использованием двух индексов совокупность однотипных элементов, имеющая общее имя. Каждый элемент двумерного массива однозначно определяется именем массива и двумя индексами (номерами этого элемента массива). Для обращения к отдельному элементу двумерного массива указывается имя этого массива и индексы элемента, заключенные в квадратные скобки, например:

arrm[5,1], mm[i,k].

В математике аналогом двумерного массива является матрица, состоящая из элементов, расположенных по строкам и столбцам. Первый индекс элемента обозначает номер строки, в которой находится элемент массива, второй индекс – номер столбца, в котором находится элемент массива.

Индексы чаще всего имеют целочисленный тип и могут быть выражениями целого типа. Двумерные массивы могут быть статические и динамические. Для статического массива границы индексов и, соответственно, размеры массива задаются при объявлении (описании) массива, т.е. они известны еще до компиляции программы.

Формат описания типа статического массива:

array [тип индексов] of <тип элементов>

Примеры объявлений (описаний) двумерных массивов:

```
var
  ab : array [10..100, 1..100] of integer;
  v : array [1..20, 1..10] of real;
```

Здесь *ab* и *v* являются двумерными массивами. Каждый из элементов массива *ab* – целое число типа *integer*, массива *v* – вещественное число типа *real*. Массив *ab* состоит из 9100 элементов (из 91 строки по 100 элементов в каждой, при этом нижняя граница индекса строки – 10, верхняя граница индекса строки – 100, нижняя граница индекса столбца – 1 верхняя граница индекса столбца – 100). Массив *v* состоит из 200 элементов (из 20 строк по 10 элементов в каждой).

Пример вложенного цикла для суммирования элементов каждой строки двумерного массива *v* с размещением полученных сумм в тех элементах одномерного массива *z*, индексы которых совпадают с номерами строк двумерного массива *v*:

```
for i :=1 to 20 do
begin
  z[i] := 0;
  for k := 1 to 10 do
    z[i] := z[i]+ v[i, k]
end;
```

При вводе и выводе значений двумерного массива можно указывать только элемент массива с индексами, например:

```
readln(z[1, 10], z[i, i + k], v[k, i +1]);
writeln(z[10, 1], z[i, i +1], v[i, k]);
```

9.6 Практические задания

9.6.1 Пример для повторения

Задание. Создать двухмерный массив (матрицу) размером 4×4 элемента, заполнить его и вычислить разницу между суммами главной и побочной диагоналей.

Разработаем интерфейс программы. Для отображения двумерных массивов (матриц) можно также использовать компонент *tStringGrid* (Строковая таблица). Внешний вид формы представлен на рисунок 9.1.

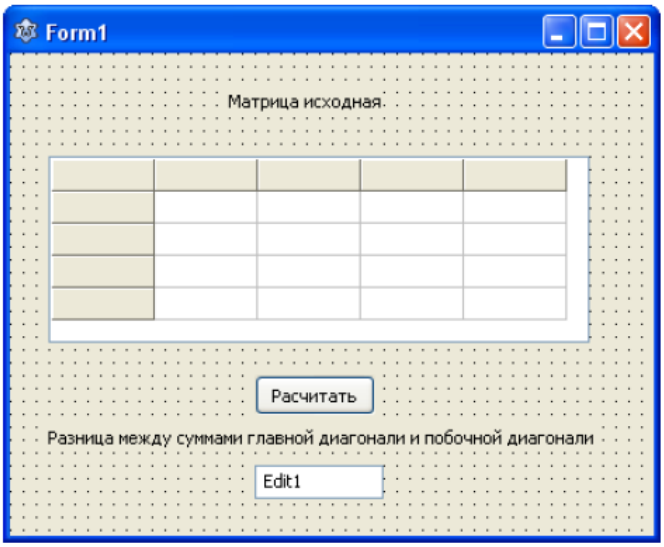


Рисунок 9.1 - Интерфейс программы

На форме разместим две «метки» (соответствующим образом их переименуем), «таблицу», «кнопку» и «поле ввода данных», в которое будем выводить результат. Настроим таблицу:

RowCount = 4;

ColCount = 4;

FixedRows = 0;

FixedCols = 0;

Options.goEditing = true.

Переименуем кнопку на рассчитать (*Caption*) и удалим надпись *Editl* компонента *Editl* (свойство *Text*). В результате получим форму (рисунок 9.2).

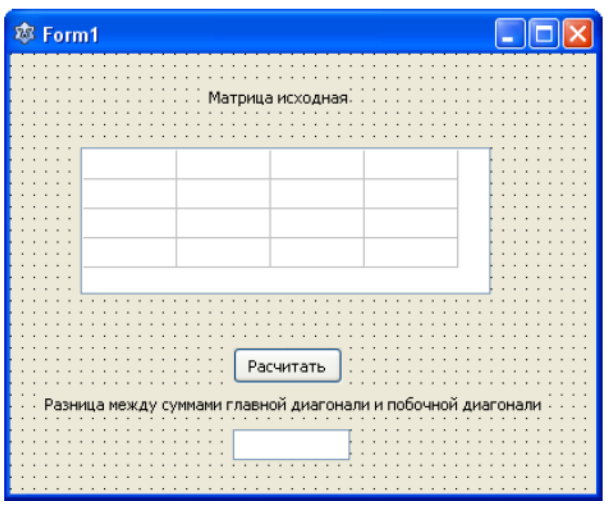


Рисунок 9.2 – Настройка компонентов интерфейса

Разработаем структуру данных для хранения двумерного массива – переменную опишем за формой (*Form1*) в глобальном разделе описания переменных формы *var* (рисунок 9.3).

```
var
  Form1: TForm1;
  mas : array [1..4,1..4] of real;
```

Рисунок 9.3 – Описание массива

Для создания двумерного массива необходимо указать диапазоны его индексов в квадрантных скобках через запятую. Переменная для хранения данных (двумерного массива вещественных чисел) определена. Напишем процедуру, считающую разницу между суммами главной и побочной диагоналей. Создадим обработчик на кнопку **Рассчитать** (рисунок 9.4).

```
{ TForm1 }

procedure TForm1.Button1Click(Sender: TObject);
var
  i,j : integer;
  S1,S2,R : real;
begin
  for i:=1 to 4 do
    for j:=1 to 4 do
      mas[i,j] := StrToFloat(StringGrid1.Cells[i-1,j-1]);
  S1 := 0;
  for i:=1 to 4 do
    S1 := S1 + mas[i,i];
  S2 := 0;
  for i:=1 to 4 do
    S2 := S2 + mas[i,5-i];
  R := S1-S2;
  Edit1.Text:= FloatToStr(R);

end;
```

Рисунок 9.4 - Обработчик кнопки **Рассчитать**

Для расчета результата введем вспомогательные переменные:

i, j – индексы массивов (целого типа);

S1 – сумма главной диагонали;

S2 – сумма побочной диагонали;

R – разность сумм главной и побочной диагонали.

Первый двойной вложенный цикл переносит данные из таблицы в массив *mas*.Далее идут два цикла: подсчет сумм главной *S1* и побочной *S2* диагоналей. Переменной *R* присваиваем разность сумм диагоналей и выводим ее в поле данных *Editl*.

Сохраняем проект и запускаем его на выполнение. Пример работы программы представлен на рисунок 9.5.

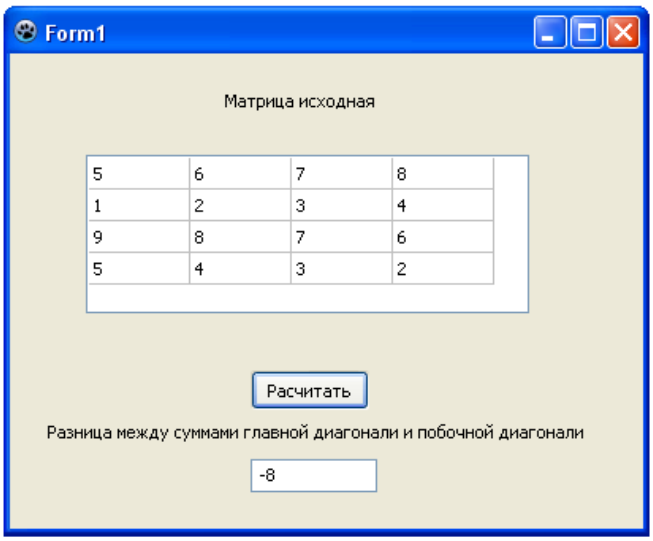


Рисунок 9.5 – Результат выполнения программы

9.6.2 Варианты индивидуальных заданий

Вариант №1. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и вычислить сумму элементов в каждой строке.

Вариант №2. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти наименьший элемент.

Вариант №3. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти наименьший элемент, а также номер строки, в которой он находится.

Вариант №4. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти сумму всех элементов.

Вариант №5. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и подсчитать количество положительных элементов в каждой строке.

Вариант №6. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти номер столбца массива, в котором находится наибольшее количество отрицательных элементов.

Вариант №7. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и упорядочить каждый столбец матрицы по возрастанию.

Вариант №8. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти наибольшее нечетное число в матрице.

Вариант №9. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и вычислить количество положительных элементов в каждом столбце матрице.

Вариант №10. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и вычислить количество отрицательных элементов в каждой строке матрицы.

Вариант №11. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и вычислить количество четных элементов в каждом столбце матрицы.

Вариант №12. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и вычислить количество четных отрицательных элементов в матрице.

Вариант №13. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти номер строки, столбца для наибольшего элемента массива.

Вариант №14. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти номер строки, столбца для наименьшего элемента массива.

Вариант №15. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и сумму элементов в каждом столбце.

Вариант №16. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и произведение элементов в каждом столбце.

Вариант №17. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и произведение элементов в каждой строке.

Вариант №18. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти произведение элементов в главной диагонали.

Вариант №19. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и произведение в побочной диагонали.

Вариант №20. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти номер столбца, в котором находится наименьшее количество положительных элементов.

Вариант №21. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти номер строки, в которой находится наименьшее количество положительных элементов.

Вариант №22. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти номер строки, в которой находится наибольшее количество четных элементов.

Вариант №23. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти номер строки, столбца с наименьшим четным числом.

Вариант №24. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти наибольшее число, кратное 3.

Вариант №25. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти наименьшее число, кратное 3.

Вариант №26. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти количестов нулевых элементов.

Вариант №27. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти наименьшее число, кратное 2.

Вариант №28. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и найти наибольшее число, кратное 2.

Вариант №29. Создать двухмерный массив (матрицу) размером 5×5 элементов, заполнить его произвольными значениями и подсчитать количество простых чисел.

Список литературы

1. Алексеев, Е. Р. Free Pascal и Lazarus: Учебник по программированию / Е. Р. Алексеев, О. В. Чеснокова, Т. В. Кучер. – Москва : Издательский дом «ДМК-пресс», 2010. – 440 с.

2. Алексеев, Е. Р. Самоучитель по программированию на Free Pascal и Lazarus / Е. Р. Алексеев, О. В. Чеснокова, Т. В. Кучер. – 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. – Санкт-Петербург : БХВ-Петербург, 2011. – 384 с.

4. Мансуров, К. Т. Основы программирования в среде Lazarus. – Москва : Нобель пресс, 2013. – 772 с.

5. Lazarus Tutorial/ru : сайт / База знаний о Free Pascal, Lazarus и родственных проектах – URL: http://wiki.freepascal.org/Lazarus_Tutorial/ru (дата обращения 04.03.2022). – Режим доступа: свободный.