МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ЮЖНО-РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
(НОВОЧЕРКАССКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ)»

методические указания

по написанию программ на языке Object Pascal в среде программирования Lazarus для строительных специальностей.

Новочеркасск 2011

Составитель А. П. Савин

УДК

методические указания по написанию программ на языке Object Pascal в среде программирования Lazarus для строительных специальностей/ Южно — Российский государственный технический университет (Новочеркасский политехнический университет), Новочеркасск 2011, 80 стр.

Настоящие методические указания имеют целью ознакомить студентов с принципами написания программ на языке Object Pascal в среде программирования Lazarus применительно к строительным специальностям. В методических указания детально рассмотрены этапы программирования от азов написания элементарных программ, до применяемых в расчетах металлических конструкций зданий и сооружений. Дается общее представление методов реализации математических и физических моделей оснований зданий и сооружений в программных продуктах. Данный материал необходим при изучении курса «Информатика» и при выполнении курсового проекта по данной дисциплине студентами строительных специальностей.

Знакомство со средой программирования Lazarus.

Лабораторная работа№ 1

Тема: Операторы присваивания.

Задание: Написать программу позволяющую ввести два целых числа и вычислить их сумму.

Выполнение:

Лабораторная работа состоит из двух этапов:

1. Разработка интерфейса;
2. Создание обработчиков.

Создадим интерфейс программы.

Для запуска среды программирования необходимо выбрать «Пуск→Все программы→Lazarus→Lazarus». Среду программирования Lazarus необходимо установлена на ПК. Дистрибутив можно взять с официального сайта, он распространяется по лицензии GPL/LGPL - бесплатный для некоммерческого использования.

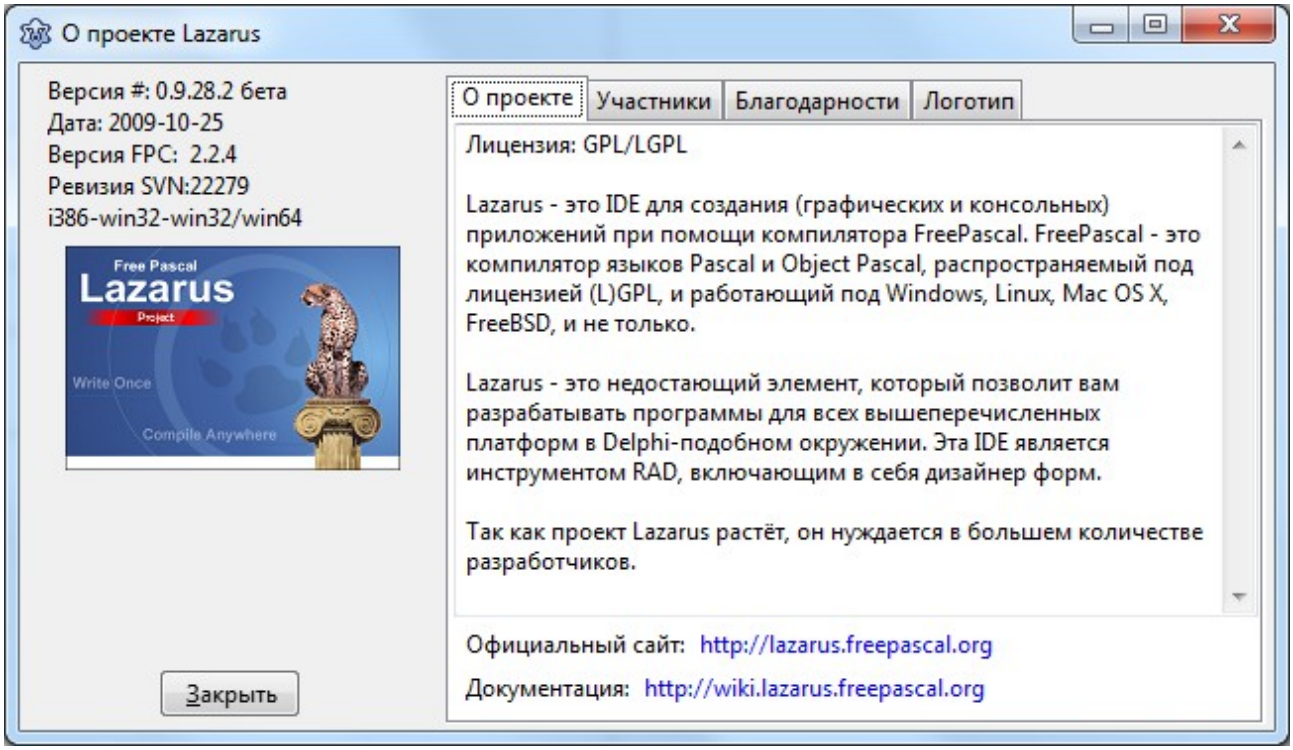


Рис. 1.1 — Официальный сайт и документация об среде программирования Lazarus.

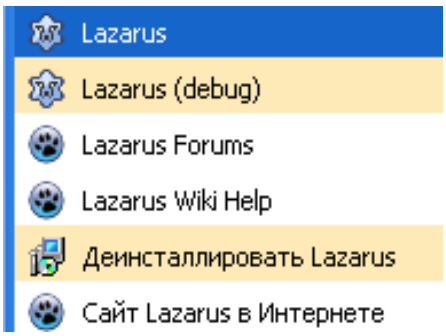


Рис. 1.2 — Запуск среды программирования Lazarus.

После запуска программы появятся несколько окон:

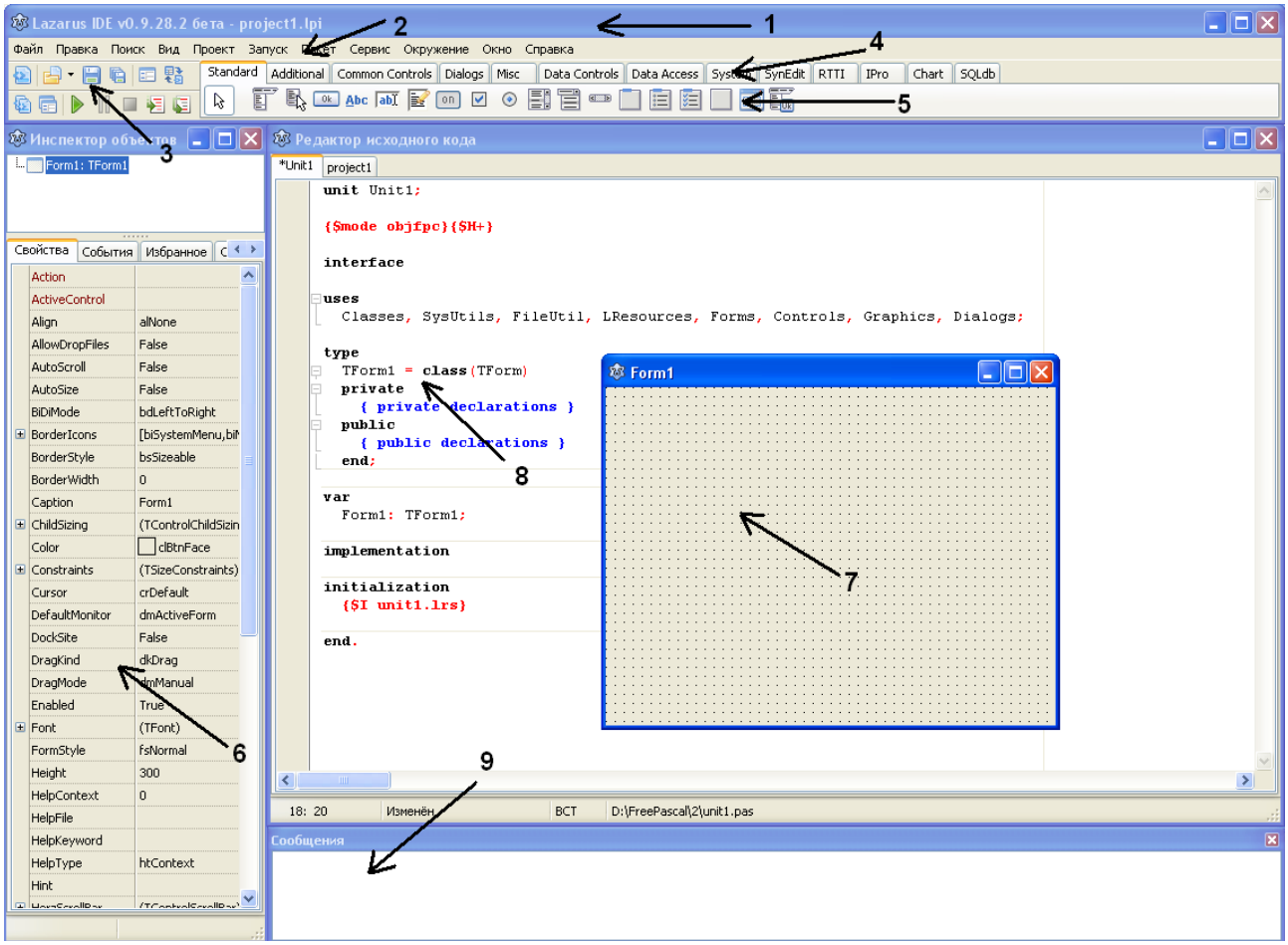


Рис. 1.3 — Основные окна среды программирования Lazarus.

1. Основная форма программы;
2. Главное меню программы;
3. Основные кнопки управления проектом;
4. Закладки компонентов;
5. Компоненты;
6. Инспектор объектов;
7. Форма программы;

8. Модуль программы (листинг);

9. Окно ошибок и подсказок;

Рассмотрим каждый элемент интерфейса программы подробно:

Основная форма программы — содержит доступ ко всем остальным формам программы. Если закрыть подчиненное окно, то проект останется открытым. Если закрыть главную форму программы — закроется все приложение.

Главное меню программы — имеет древовидную структуры и содержит доступ ко всем элементам программы от «Файл» до «Справка».

Кнопки управления проектом:

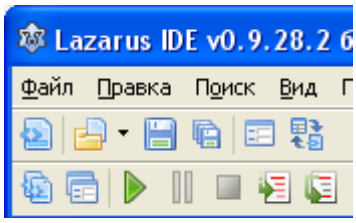


Рис. 1.4 — Основные кнопки управления проектом.

Рассмотри назначение каждой кнопки отдельно:

Верхний ряд: слева — направо:

1. Создать модуль — создает новый модуль форму.

2. Открыть — открывает проект, модуль или любой другой файл (понимаемый средой Lazarus).

3. Сохранить — сохраняет текущий модуль на диск, перед первым сохранением — запросит имя и папку, куда сохранить файл. Если текущий файл был сохранен и не редактировался, то кнопка серого цвета. Среда программирования Lazarus не понимает русские буквы в именах модулей (это связано с синтаксисом языка Object Pascal).

4. Сохранить все — Сохраняет все модули проекта, если модуль ни разу не сохранялся — предложит указать имя и папку для файла.

5. Создать форму — создает новую форму программы и прикрепляет к ней «pas» файл, по умолчанию при создании пустого проекта в нем присутствует форма. Проект может содержать несколько форм, в т.ч. и ни одной (консольное приложение, сервис, библиотека и т.д.). Первая форма создаваемая

в проекте называется главной, остальные подчиненные. При закрытии главной формы закрывается весь проект.

6. Переключить форму/модуль — осуществляет переход между окном формы (рис 1.3.7) и окном модуля (рис 1.3.8). Можно переключаться с помощью клавиши <F12>.

Нижний ряд слева — направо:

1. Показать модули — отображает список всех модулей проекта и позволяет переключится на любой из них. (Модуль может содержать, а может и не содержать форму проекта);

2. Показать формы — отображает список всех форм проекта и позволяет переключится на любую из форм.

3. Запуск проекта — осуществляет сборку проекта и его запуск в виде исполняемого файла под отладчиком, т. е. жизнью проекта управляет среда программирования Lazarus. (Если кнопка серая — проект запущен и находится в списке запущенных программ Windows)

4. Пауза — во время работы проекта его можно приостановить — поставив на «паузу».

5. Останов — принудительная остановка проекта, можно остановить проект сочетанием клавиш <Ctrl + F2>.

6. Шаг со входом — отладка проекта пошаговая с заходом во все процедуры и последовательная отладка этих процедур (можно зайти как в процедуры своего проекта, так и библиотечные функции Lazarus, но в системные процедуры Windows (API Windows) попасть не получится);

7. Шаг в обход — отладка проекта пошаговая и каждая вызываемая процедура выполняется за один шаг отладки (если внутри вызова не встретятся дополнительные точки останова);

Закладки компонентов — содержат наборы различных компонентов сгруппированные по различным признакам. В лабораторных работах использоваться вкладки «Standard» и «Additional».

Компоненты — готовые блоки программы, которые можно

устанавливать на форму, подключать в модули. Чтобы установить компонент на форму, необходимо один раз «кликнуть» на нем в панели компонентов, затем второй раз на форме — в том месте где необходимо его разместить. Компоненты можно выбирать «кликнув» по нему или с помощью рамки выделения. Выбранные компоненты имеют по краям черные прямоугольные маркеры, за которые можно перемещать объект по форме и изменять его размеры. Некоторые компоненты можно только перемещать.

Инспектор объектов — форма позволяющая настроить свойства каждого компонента индивидуально. Каждый компонент имеет набор свойств, большинство из них перечислено в инспекторе объектов (хотя и не все). Свойство объекта это имя и значение. Левый столбец — имя свойства, правый — значение. Свойства, как и типы данных, могут быть: целыми, вещественными, строковыми, логическими, множествами или сложными. Инспектор объектов имеет несколько закладок. Закладка «События» — позволяет посмотреть список большинства событий, на которые может реагировать компонент, а также процедуры привязанные к каждому из событий. (По умолчанию большинство событий пустые — с ними не связаны никакие обработчики событий).

Форма программы — форма на которой разрабатывается дизайн программы с помощью компонентов.

Модуль программы — окно в которой содержится исходный код на языке Object Pascal. Среда программирования Lazarus автоматически вносит изменения в код программы при добавлении компонент на форму и создании обработчиков событий, а программист при создании тела программы вносит свои изменения в код.

Окно ошибок — окно в котором отображается все ошибки и подсказки при сборке проекта.

Ошибки могут быть:

1. Синтаксические — кода исходный текст программы не понимает среда программирования Lazarus;

2. Логические — когда код с точки зрения среды программирования написан верно, но программа выполняет не те действия, которые ожидает пользователь от программы.

Первый тип ошибок может отследить среда программирования и программист, второй тип — только программист.

Под каждый проект рекомендуется отводить отдельную папку, т. к. при создании проекта — создается не отдельный файл, а группа связанных файлов.

Для каждой лабораторной рекомендуется создать следующую структуру:

Чтобы иметь доступ к папке по сети в папке общие документы (Мой компьютер → Общие документы), создать папку с номером вашей группы (например «СФ 1-8а»), далее ваша фамилия (например «Иванов»), далее «1» для первой лабораторной, для второй «2» и т. д.

В результате получим примерно такой путь «Мой компьютер\Общие документы\СФ-1-8\Иванов\1\».

После создания папок, создадим пустой проект и сохраним его в папку «Мой компьютер\Общие документы\СФ-1-8\Ваша фамилия\1\».

Для создания пустого проекта в среде программирования Lazarus выберем «Файл → Создать...»

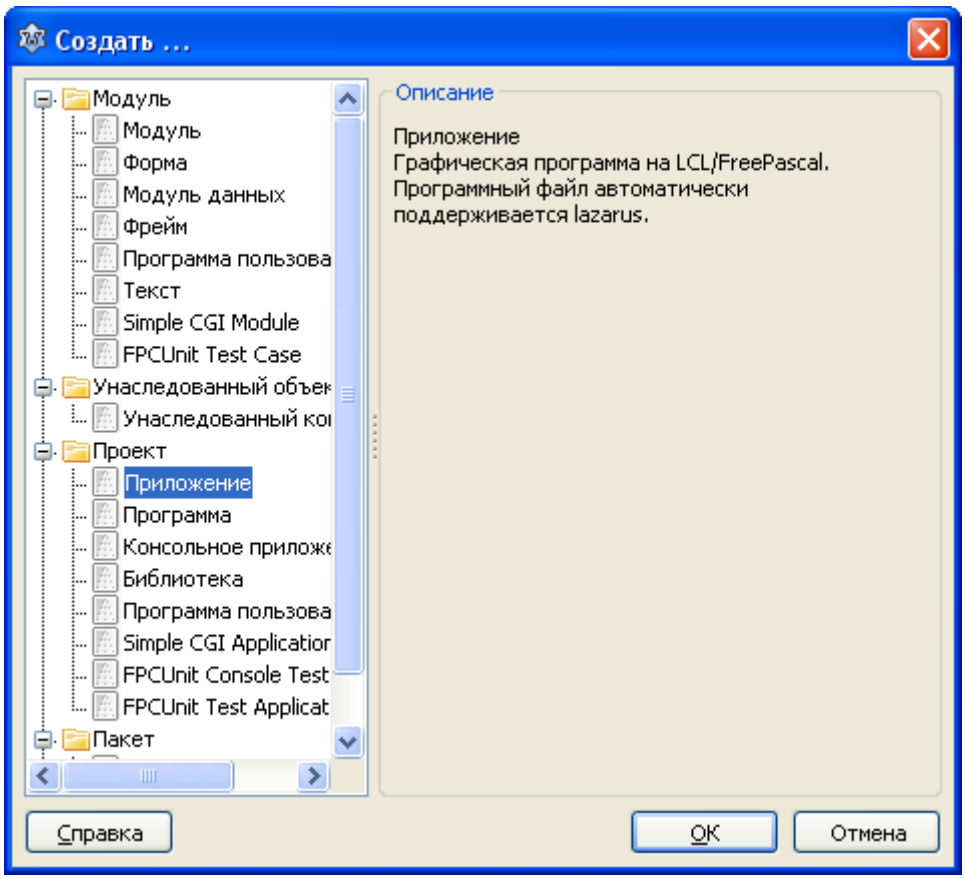


Рис. 1.5 — Создание нового проекта.

В окне выберем раздел «Проект» («Project») и пункт «Приложение» («Application»).

Нажмем кнопку «Ok».

На вопрос о сохранении проекта выбираем «нет» (если предложит).

Сохраним пустой проект — «Файл → Сохранить все».

Программа предложит указать папку и имя файла. Папку выбираем «Мой компьютер\Общие документы\СФ-1-8\Ваша фамилия\1\», а имя файла не изменяем — оставляем по умолчанию «Unit1». Среда программирования Lazarus не понимает русские буквы в имени фалов, необходимо использовать набор латинских букв и цифр. Данное ограничение связано с синтаксисом языка Object Pascal.

Нажимаем на кнопку «Сохранить».

Программа предупредит о преобразовать имя файла в нижний регистр — соглашаемся.

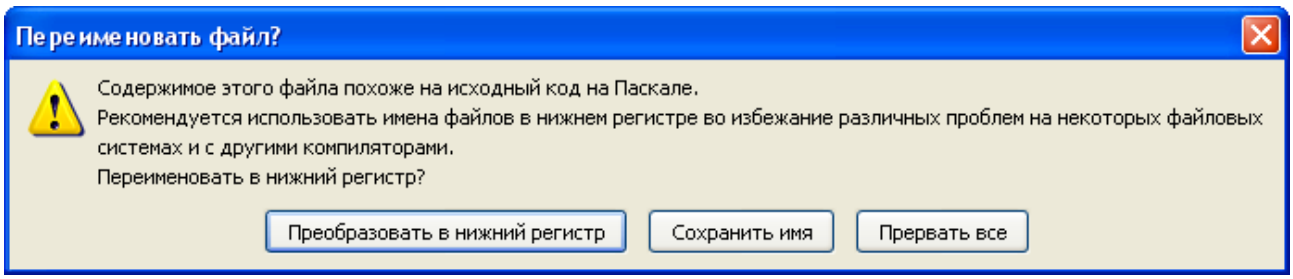


Рис. 1.6 — Предупреждение о преобразовании имени файла в нижний регистр.

Далее аналогичным образом сохраняем проект в туже папку.

Если проект сохранен то на главной форме программы Lazarus иконка «сохранить» будет иметь «серый» вид.

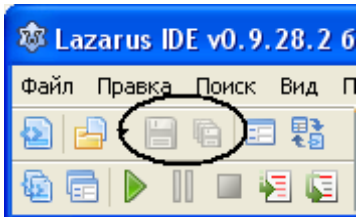


Рис. 1.7 — Проект сохранен на диске.

Запустим проект — для этого нажмем на кнопу с зеленым треугольником. Среда программирования на некоторое время «задумается» и после сборки проекта — запустит его.

На экране появится та же форма, но без точек. Также можно заметить, что появилась новая программа в панели программ Windows с именем Projoct1.

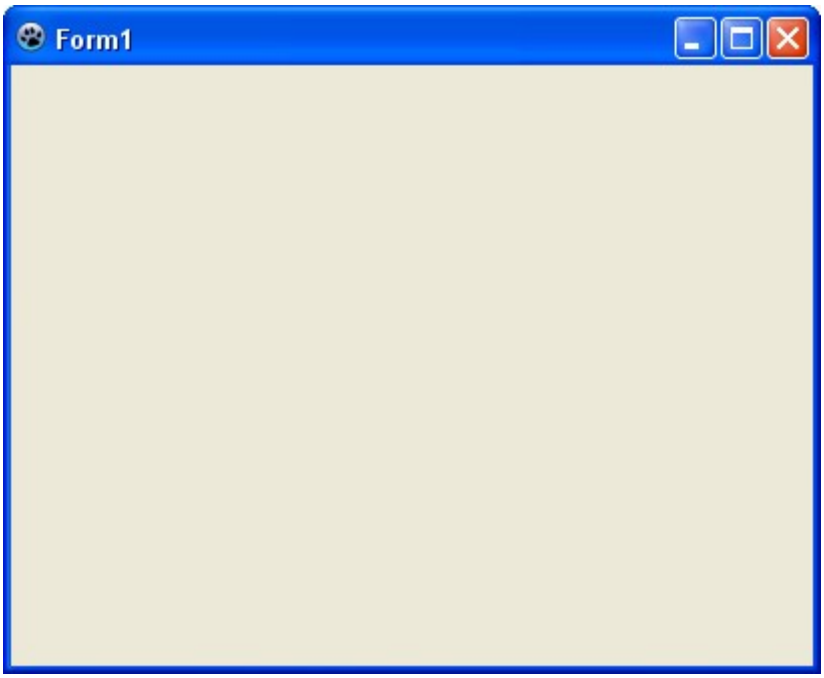


Рис. 1.8 — Пустая форма программы.

Закрываем программу (крестик в правом верхнем углу) и возвращаемся

в среду программирования Lazarus.

После закрытия проекта появится окно с информацией, что проект остановлен.

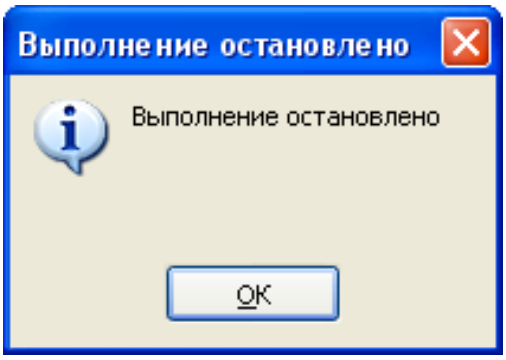


Рис. 1.9 — Предупреждение — программа остановлена.

Проект рекомендуется изменять только когда остановлено выполнение программы.

Отобразим форму программы — для этого нажмем «F12».

Поместим на форму компонент «TMainMenu». Данный компонент находится на закладке «Standard» — самый первый.

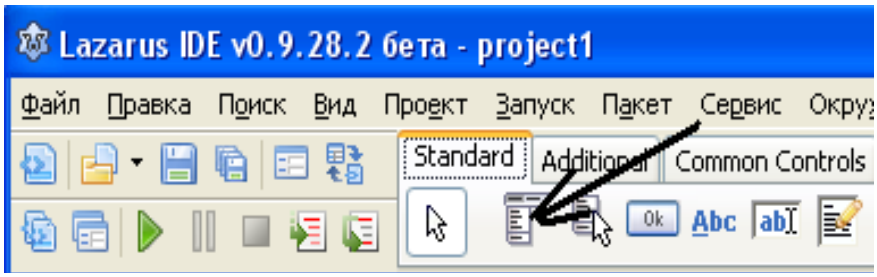


Рис. 1.10 — Выбор компонента из палитры компонентов.

В результате на форме отобразится иконка этого компонента, как показано на рисунке 1.11:

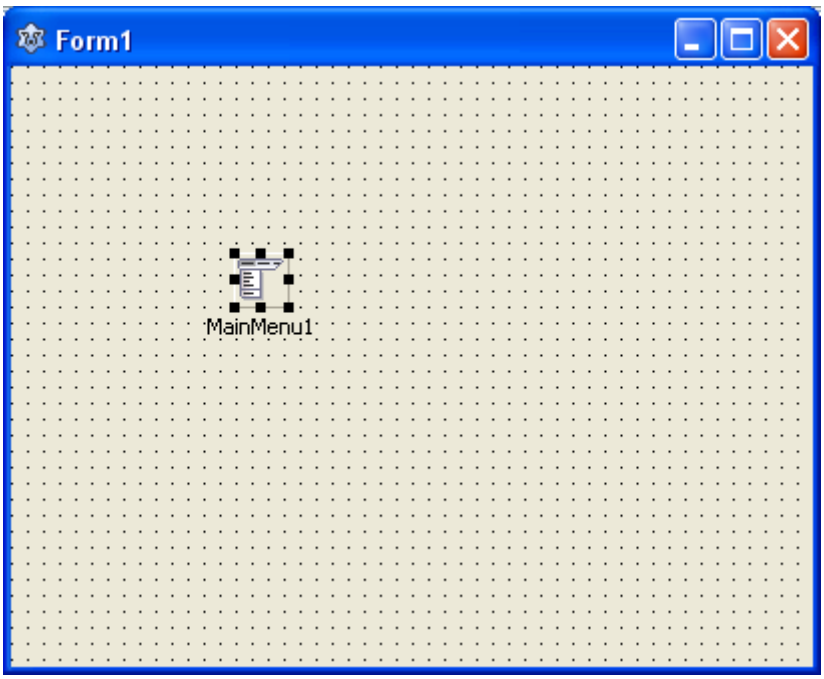


Рис. 1.11 — Установка компонента на форму.

Данный компонент предназначен для создания главного меню программы.

Откроем редактор главного меню: сделаем «двойной клик» на иконке главного меню, которая находится на форме программы (Ее только что добавили на форму).

Откроется «редактор меню»:

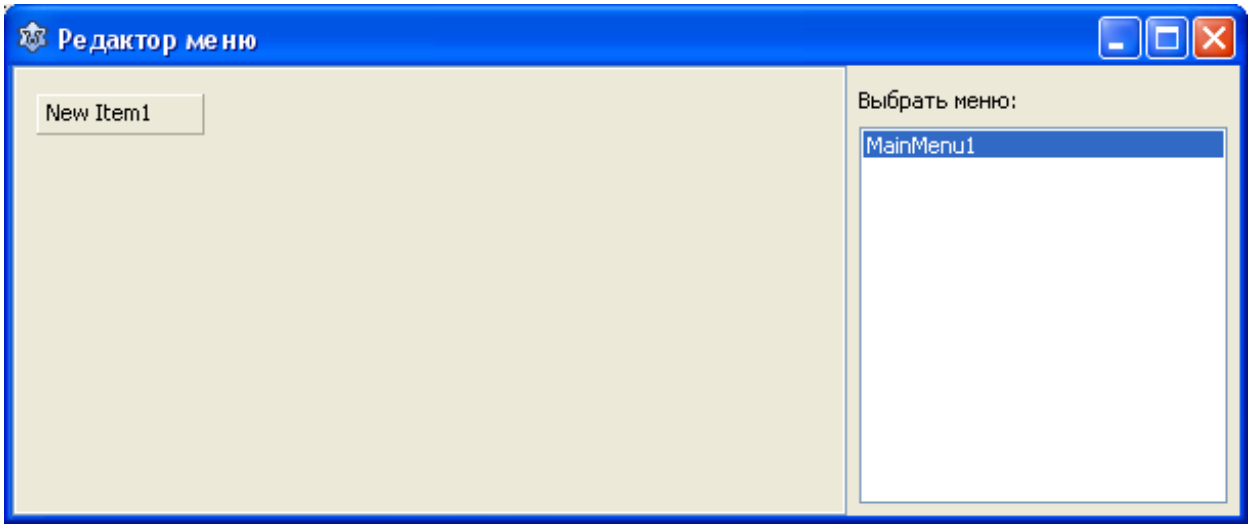


Рис. 1.12 — Внешний вид первого элемента меню в «Редакторе меню».

Во время открытия «Редактора меню» будет автоматически создан первый элемент меню с именем «New Item1». Переименуем его в «Файл». Для этого один раз «кликнем» левой клавишей «мыши» на элементе «New Item1» в

«Редакторе меню» (чтобы его выбрать), затем, в инспекторе объектов, найдем свойство «Caption» и изменим его с «New Item1» на «Файл».

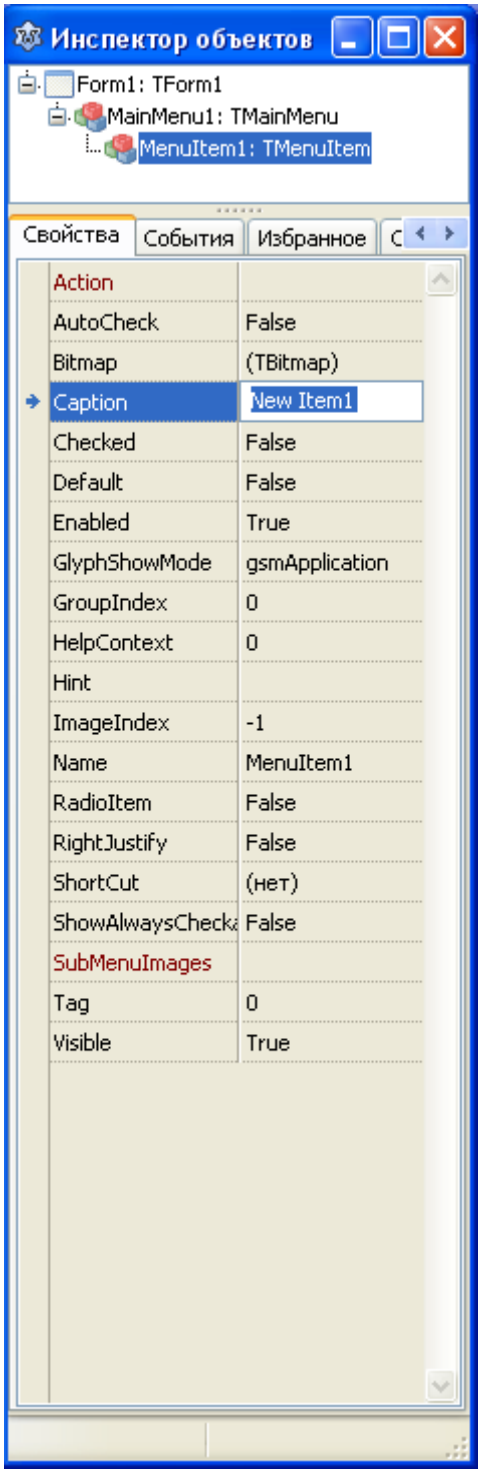


Рис. 1.13 — Переименование элемента меню из «New Item1» в «Файл».

Автоматически изменится название элемента меню в «Редакторе меню» и на форме программы.

В «Редакторе меню» на слове «Файл» «кликнем» правой клавишей «мыши» и в контекстном меню (выпадающем списке) выберем «Создать

подменю».

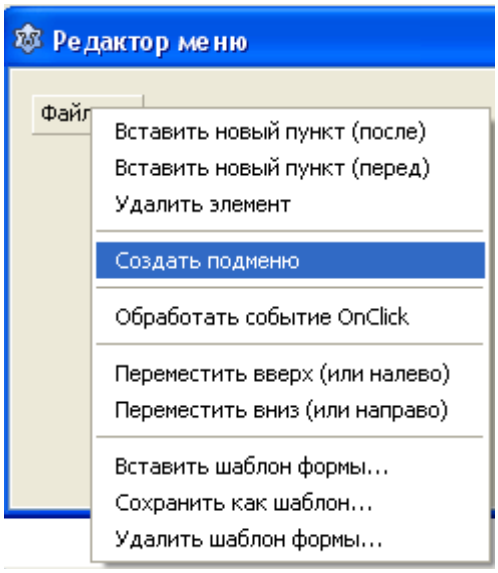


Рис. 1.14 — Создание подменю «Выход» для меню «Файл».

Появится элемент подменю в пункте «Файл» с именем «New Item2» — переименуем его в «Выход» с помощью инспектора объектов.

В «Редакторе меню» кликнем правой клавишей «мыши» на слове «файл» и в выпадающем списке выберем «Вставить новый пункт (после)» - создадим новый элемент меню правее элемента «Файл».

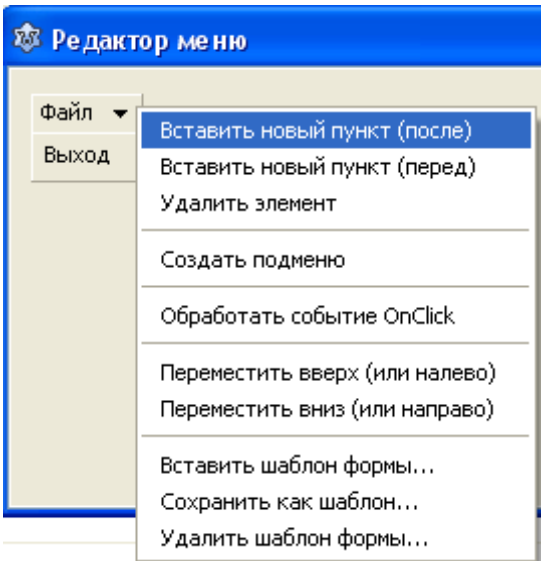


Рис. 1.15 — Создание меню «Вычислить».

Переименуем его в «Вычислить». Аналогично создаем пункты меню «Помощь» и «О программе» как показано на рисунке 1.16:

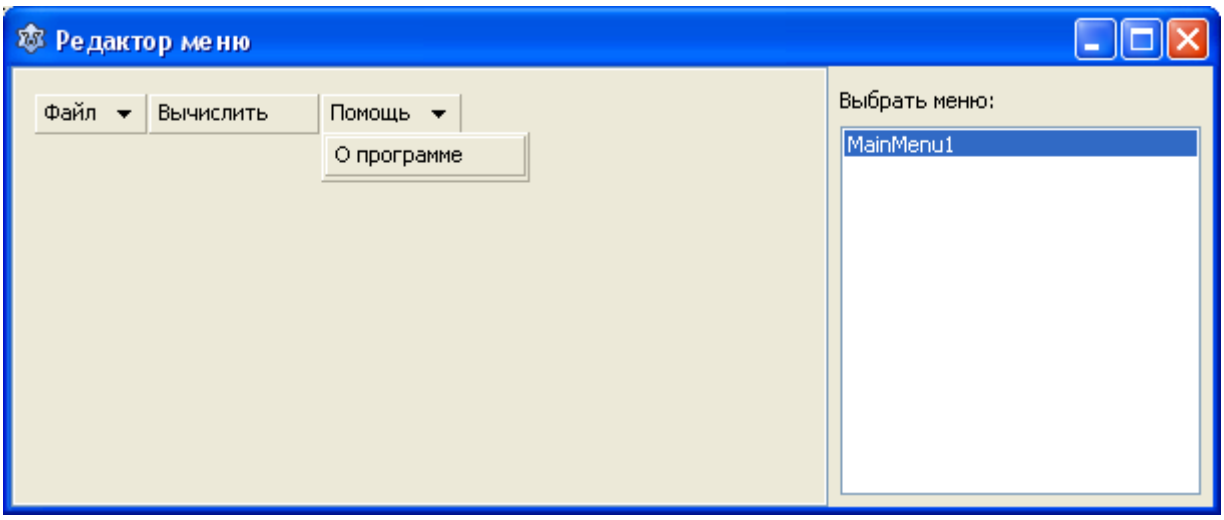


Рис. 1.16 — Создание остальных пунктов меню.

Закрываем редактор меню (крестик у окна «Редактор меню» в правом верхнем углу), сохраняем проект и запускаем его на выполнение. Проконтролируем результат — наличие главного меню на форме запущенной программы. Рекомендуется сохранять и запускать на выполнение программу при каждом существенном изменении программы.

После запуска программы у пустой формы появилось «главное меню», по которому можно перемещаться. При выборе любого пункта меню не происходит никаких действий, т. к. мы не объяснили Lazarus, что должно происходить при выборе соответствующих пунктов меню — т. е. не написали обработчики событий соответствующих пунктов меню.

Закрываем программу, возвращаемся в Lazarus. Продолжим разработку интерфейса программы.

Расположим на форме элементы как показано на рис 1.17:

3 компонента «TEdit»;

5 компонентов «TLabel»;

1 компонент «TButton»;

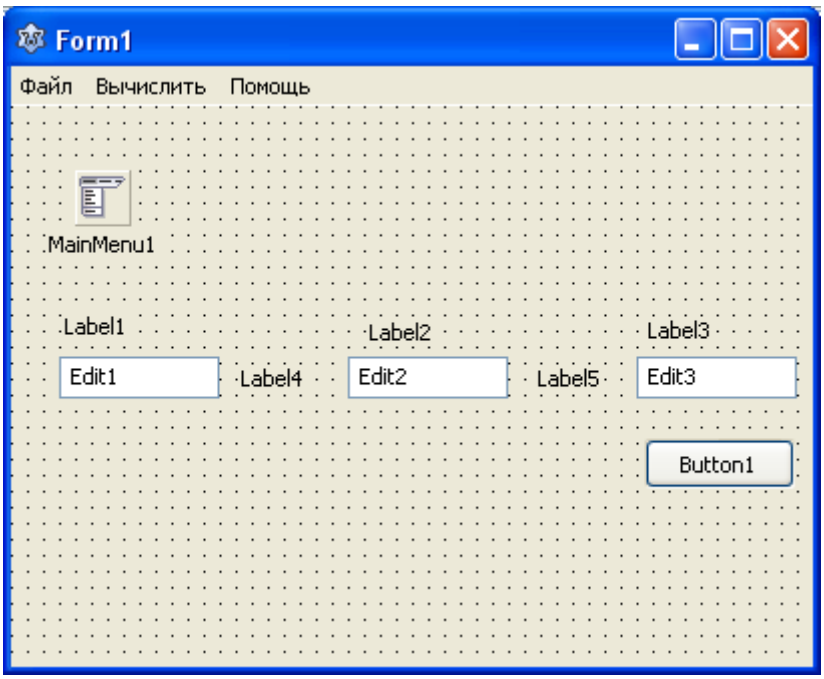


Рис. 1.17 — Расположение компонентов на форме.

Последовательно будем выбирать каждый выставленный компонент «TLabel» и изменять у него свойство «Caption» в инспекторе объектов. Изменим это свойство у кнопки «Botton1». В результате получим форму:

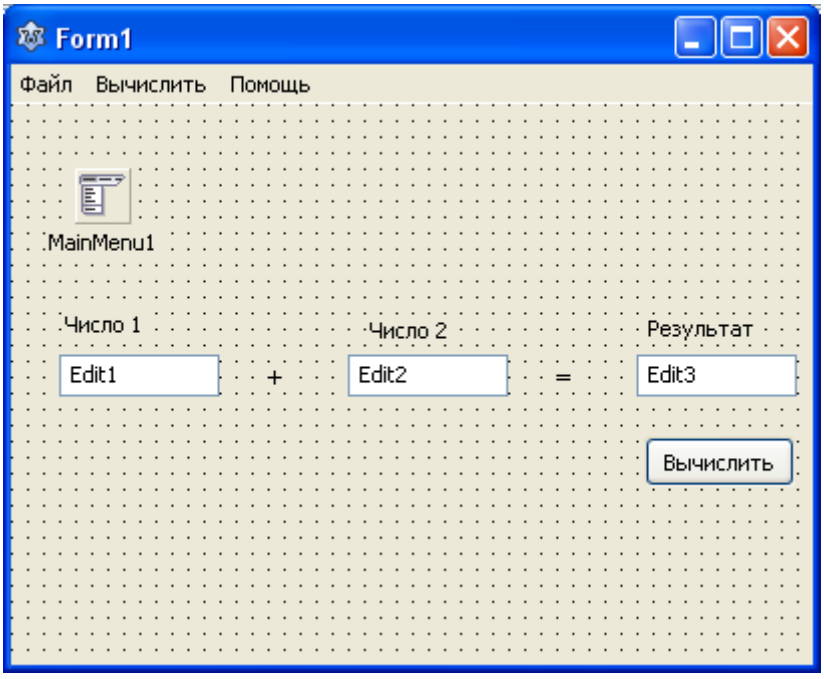


Рис. 1.18 — Переименование «Меток» и «Кнопки».

Выберем последовательно каждый компонент «Edit» и в «Инспекторе объектов» изменим свойство «Text» на «5», «7» и «». В результате получим форму:

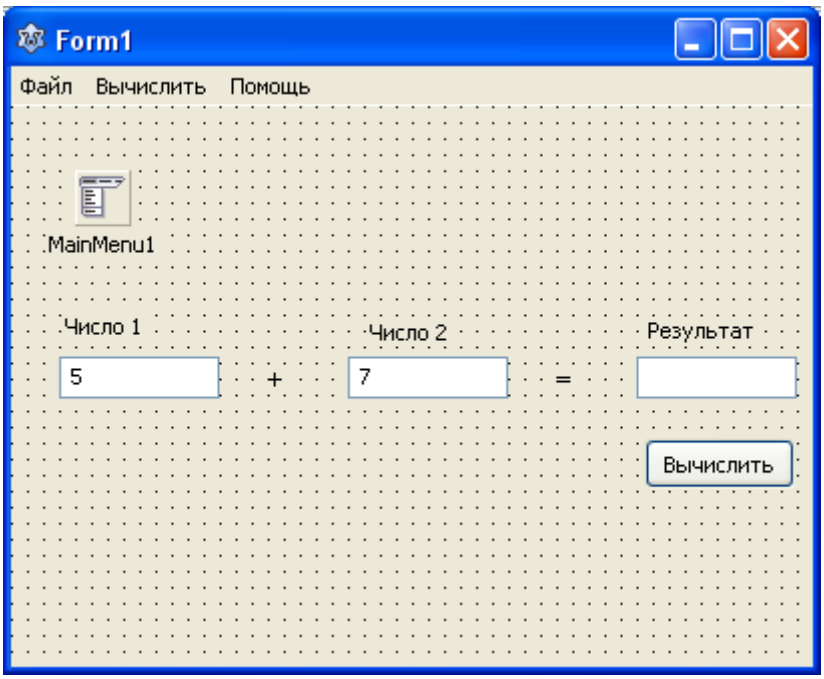


Рис. 1.19 — Изменение свойства у элемента «TEdit».

Сохраним проект и запустим его на выполнение, появится следующая форма:

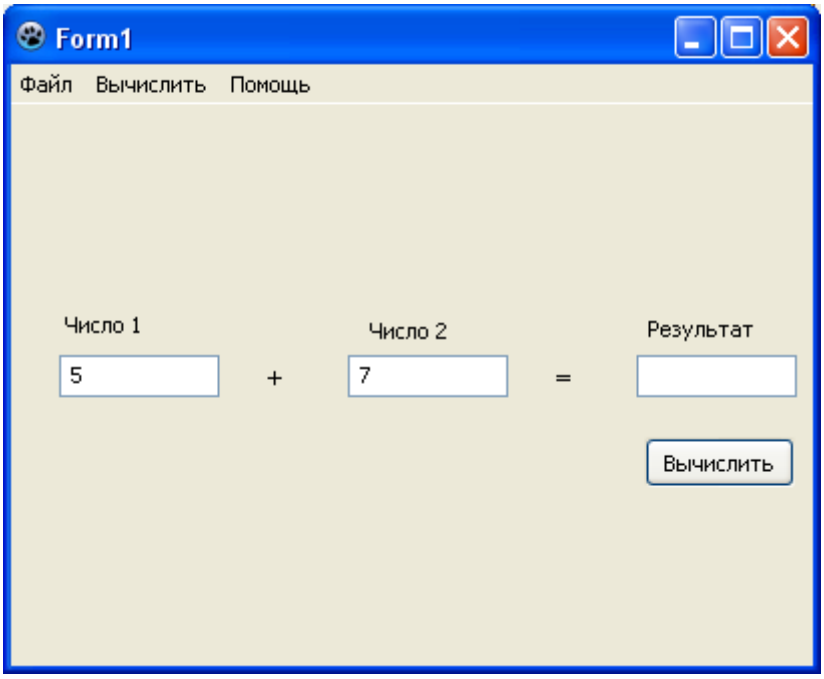


Рис. 1.20 — Внешний вид формы после сборки проекта.

На данный момент разработан полностью интерфейс программы, но пока нет функционального наполнения.

Вторая часть работы — создание обработчиков (наполнение интерфейса функциональностью).

Возвращаемся в среду программирования Lazarus, отображаем форму

программы и выбираем пункт меню «Файл» → «Выход»:

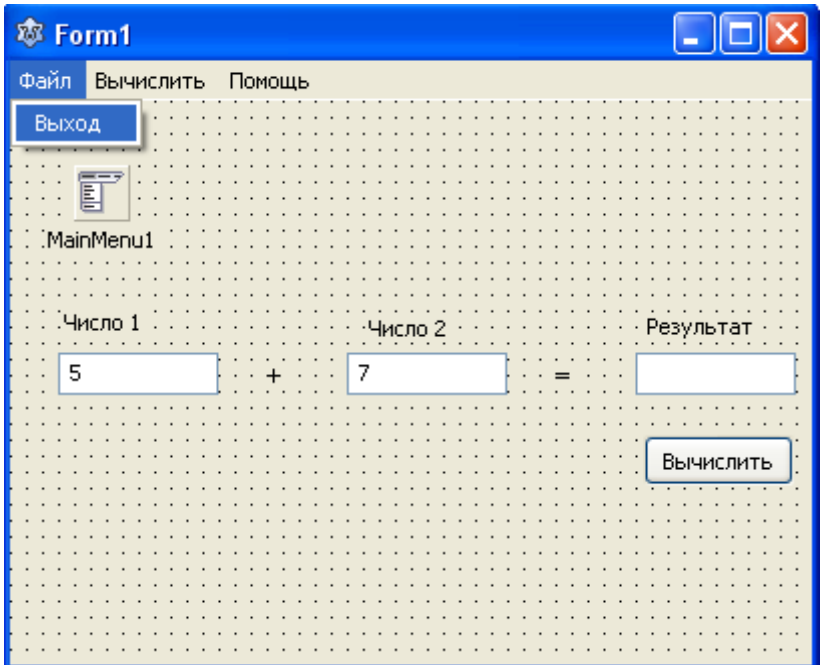


Рис. 1.21 — Привязка обработчика к меню «Выход».

В результате форма дизайна программы уйдет на задний план, а на передний план покажется «Редактор исходного кода» с заготовкой события.

```
{ TForm1 }

procedure TForm1.MenuItem2Click(Sender: TObject);
begin

end;
```

Рис. 1.22 — Заготовка процедуры для события «Выход».

Пустая процедура сообщает, что при выборе данного меню не выполняется никаких действий.

По заданию при выборе меню «Файл» → «Выход» программа должна закрываться. Процедура которая закрывает форму, а заодно и программу называется close. Между «begin» и «end», там где установлен курсор, наберем следующий код, как показано на рисунке в конце поставив «;».

```
procedure TForm1.MenuItem2Click(Sender: TObject);
begin
  close;
end;
```

Рис. 1.23 — Добавление функции закрытия программы.

Отобразим форму программы «F12» и выберем пункт меню

«Помощь»→«О программе». Напишем обработчик данного события — выбор соответствующего пункта меню. Программа должна сообщить кто автор. Сообщать будем с помощью процедуры ShowMessage. Она имеет один параметр (строковый): текст который будет отображаться в виде сообщения. Вид обработчика примерно следующий:

```
procedure TForm1.MenuItem2Click(Sender: TObject);
begin
  close;
end;

procedure TForm1.MenuItem5Click(Sender: TObject);
begin
  ShowMessage('Выполнил студент СФ-1-8 Иванов И. И.');
end;
```

Рис. 1.24 — Создание обработчика «О программе».

Текст сообщения необходимо писать в скобочках и одиночных кавычках. Сохраним проект и запустим его на выполнение.

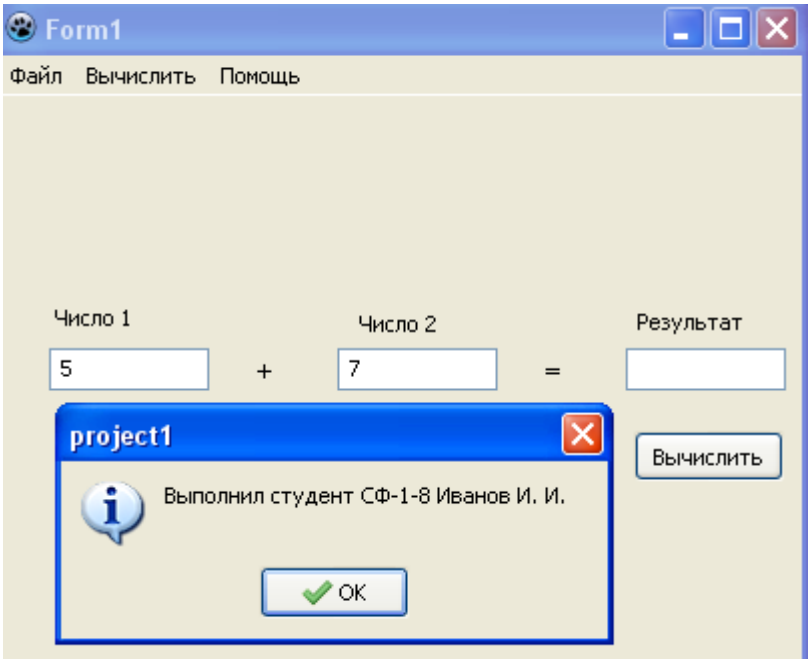


Рис. 1.25 — Проверка работоспособности обработчиков.

При выборе пункта меню «Помощь»→«О программе» должно отображаться следующее окно рис. 1.25. При выборе «Файл»→«Выход» программа должна зарыться так же, как при нажатии на крестик в правом верхнем углу программы.

Создана часть функциональности, теперь добавим процедуру

вычисления суммы двух чисел.

Для этого закроем программу и на форме программы сделаем двойной клик на кнопке «Вычислить», в результате будет создана заготовка (пустая процедура) обработчика нажатия на кнопку:

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

end;
```

Рис. 1.26 — Создание обработчика на кнопку «Вычислить».

Между «procedure» и «begin» добавим раздел описания переменных «var» и опишем три целых переменные. Между «begin» и «end» добавим четыре действия:

1. Получение данных с первого поля ввода данных, преобразование из текста в число и запись в переменную «A»;
2. Получение данных со второго поля ввода данных, преобразование из текста в число и запись в переменную «B»;
3. Суммирование этих двух переменных и запись результата в переменную «C»;
4. Значение суммы из переменной «C» преобразуем из числа в строку и запишем в третье поле данных — результат.

Преобразование из строк в числа (1 и 2 пункт) нам необходима, т. к. в поля Edit можно вводить не только числа, но и текст. Что бы программа могла суммировать надо преобразовать текст в числа. Если будем суммировать текстовые переменные то получим текстовую переменную вдвое большей длинны. (Например: '123' + '456' = '123456', что не соответствует правилам математики). После того как получим сумму в переменной «C» необходимо ее обратно преобразовать в текст и записать в поле «Edit3». Процедура (обработчик события нажатия кнопки «Button1» на форме) имеет следующий вид:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  a,b,c : integer;
begin
  a := StrToInt(Edit1.Text);
  b := StrToInt(Edit2.Text);
  c := a+b;
  Edit3.Text:= IntToStr(c);
end;
```

Рис. 1.27 — Код процедуры: расчет суммы двух чисел.

Сохраним проект и запустим на выполнение. В результате при нажатии на кнопку «Вычислить» программа должна вычислить сумму двух чисел.

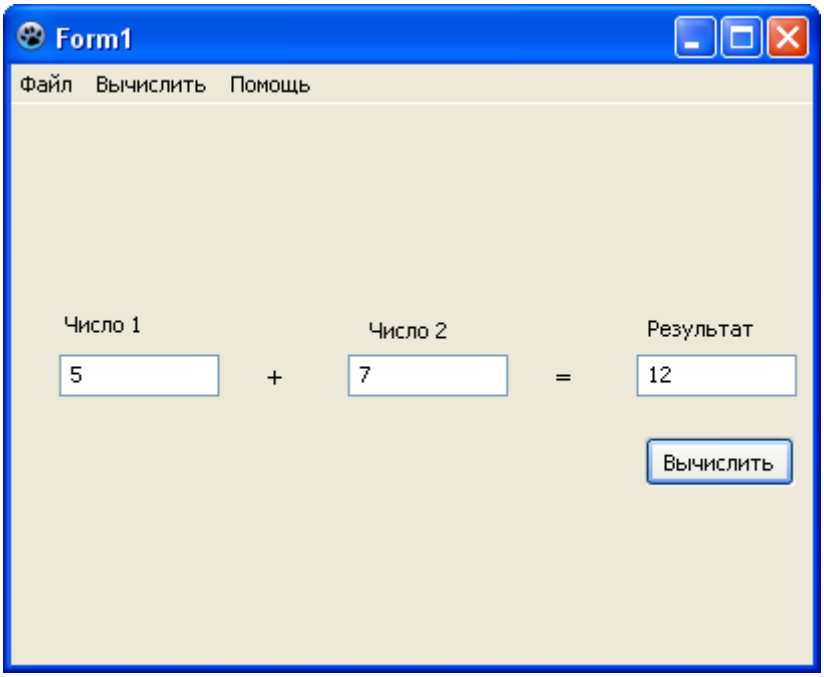


Рис. 1.28 — Пример расчета с помощью кнопки «Вычислить».

На данный момент при выборе пункта меню «Вычислить» — расчет не производится. Вернемся в «Редактор меню» Lazarus:

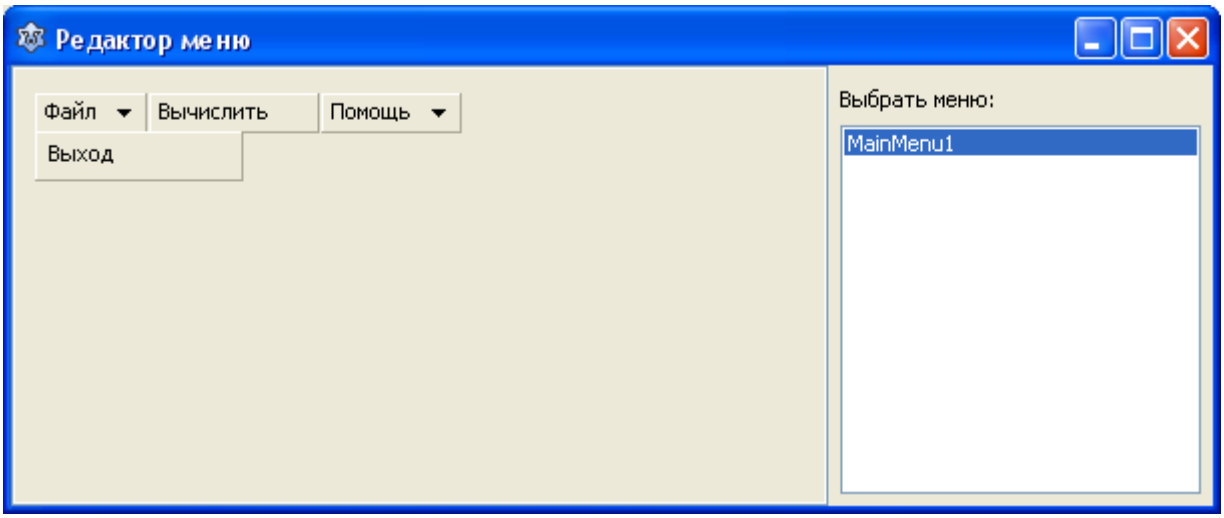


Рис. 1.29 — Выбор пункта меню «Вычислить» в «Редакторе меню».

Один раз «кликнем» левой кнопкой «мыши» на пункте «Вычислить» и в «Инспекторе объектов» выбираем закладку «События», найдем событие «OnClick».

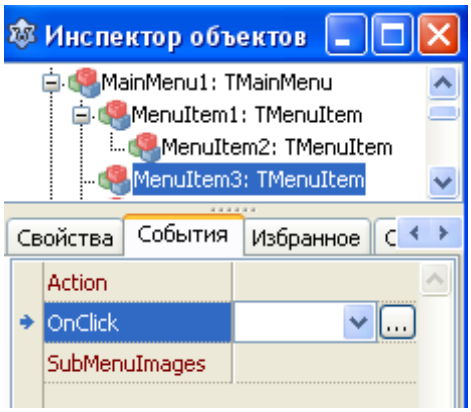


Рис. 1.30 — Список событий у меню «Вычислить».

В выпадающим списке выбираем обработчик «Button1Click» — тот же, что и у кнопки «Рассчитать».

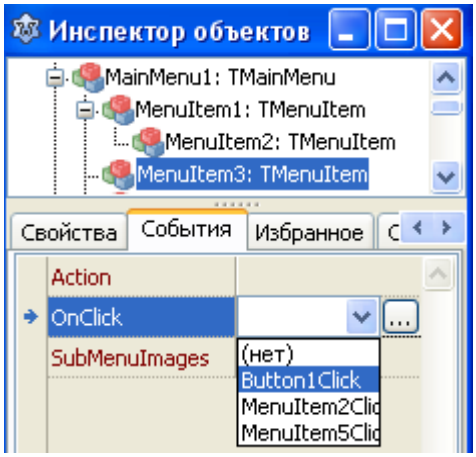


Рис. 1.31 — Привязка обработчика кнопки «Рассчитать» к меню «Вычислить».

Сохраняем проект, запускаем, проверяем. Теперь расчет происходит не только по кнопке, но и по пункту меню «Вычислить».

Демонстрационный проект готов.

Вам необходимо на основе данного проекта выполнить свое индивидуальное задание. Большую часть можно взять из данного примера.

Лабораторная работа №2

Тема: Операторы выбора.

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

Пример к лабораторной работе данная область.

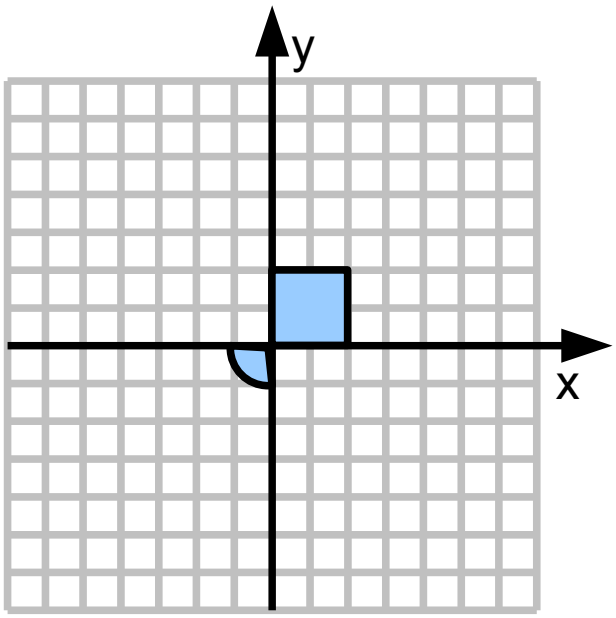


Рис. 2.1 — Задание.

Рассмотрим условный первый вариант (по заданию у нечетных вариантов граница не принадлежит области).

Работу необходимо сохранить в папке «2». Место расположения проекта: «Мой компьютер\Общие документы\СФ-1-8а\Иванов\2».

Разработаем интерфейс программы.

В редакторе Lazarus выбираем «Файл»→«Создать...»→«Проект/Project» →«Приложение/Application».

Интерфейс выглядит следующим образом:

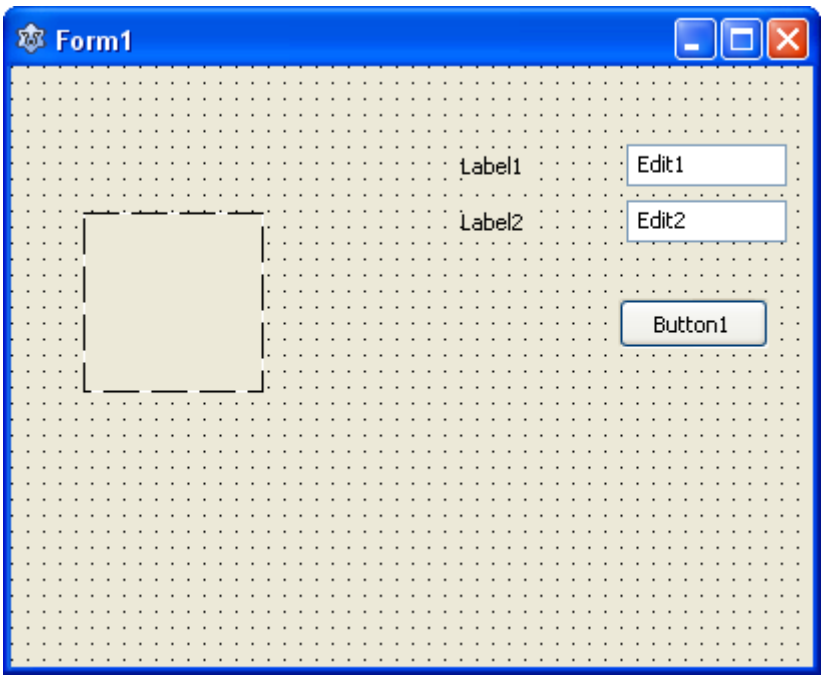


Рис. 2.2 — Расположение компонентов на форме.

Слева на форме компонент TImage (прямоугольник из пунктирных линий), он находится на закладке компонент Additional. Перенесем его с палитры компонентов на форму так же, как и другие компоненты. Справа: два TEdit, два TLabel и один TBatton. Расставим их так, как на рисунке 2.2.

Необходимо переименовать элементы TLabel и TBatton (свойство Caption) и стереть значения полей Text у компонентов TEdit. В результате получим форму рис 2.3:

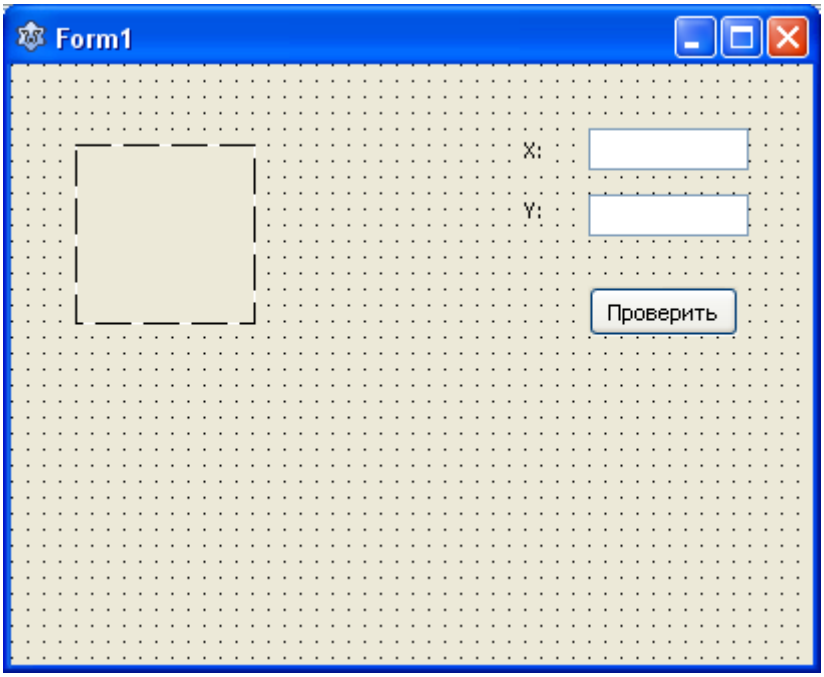


Рис. 2.3 — Переименование и настройка основных компонент.

Настроим компонент «Image1» – на нем мы будем рисовать области.

В инспекторе объектов найдем у него свойства «Width» и «Height» и устанавливаем их значения равными 200. Мы выбрали значение такими, что бы легче было рисовать и рассчитывать координаты.

Немного о системе координат.

У компонента «TImage», как и у всех других компонент, система координат перевернута от обычной (которой вы привыкли пользоваться). Центр координат находиться в левом верхнем углу картинки и имеет значение 0,0. Ось X — направлена горизонтально вправо, ось Y направлена вертикально вниз. Значения координат отсчитываются в пикселях (один пиксель — одна единица).

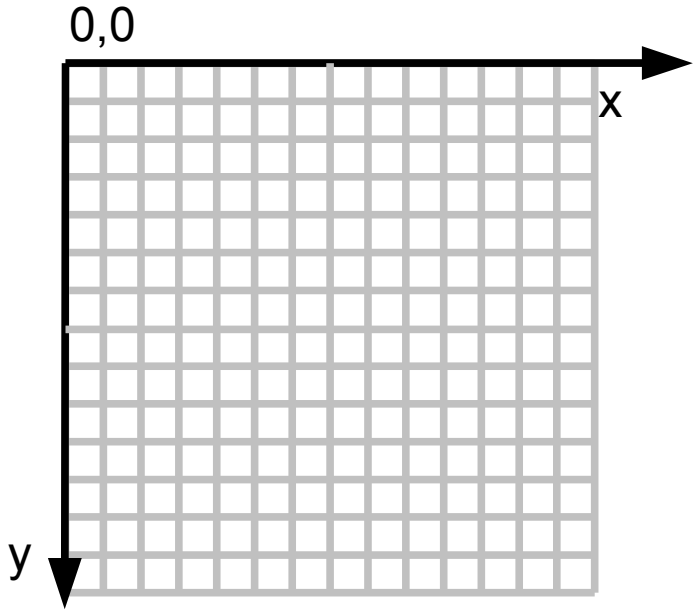


Рис. 2.4 — Система координат используемая у компонента TImage.

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 мы введем масштабный коэффициент и перенесем центр координат в центр картинки. Масштаб выберем равным 20:1, а центр в точке с координатами x=100; y=100. Что идеально подходит для нашего рисунка.

Интерфейс разработан — сохраняем проект и запускаем на выполнение.

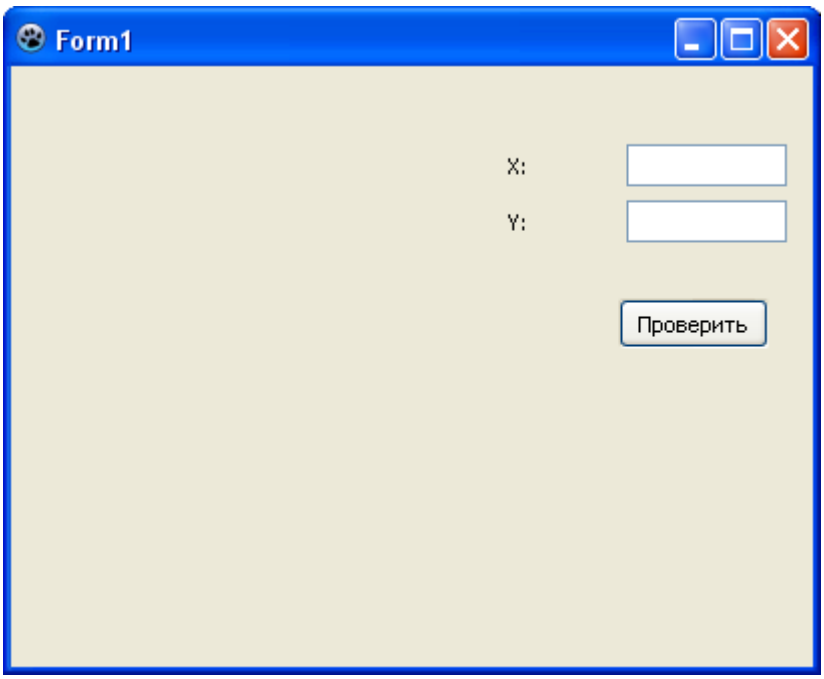


Рис. 2.5 — Внешний вид формы после сборки и запуска проекта.

Компоненты «TLabel», «TEdit» и «TButton» видны на форме, а вот компонент «TImage» на форме не виден т. к. на нем ничего еще не нарисовано.

С помощью компонента «TImage» можно отобразить на форме любой рисунок. Для этого можно воспользоваться двумя способами:

1. Создать рисунок в любом графическом редакторе (Paint, Photoshop и др.) и загрузить получившийся рисунок (файл рисунка) в компонент Image1 (формат BMP подойдет, хотя можно использовать и другие).

2. Воспользоваться примитивными функциями рисования и последовательно создать рисунок из таких элементов как: линии, точки, дуги, окружности, эллипсы и т. д.

Первый способ лучше подходит для таких рисунков как фотографии, а второй для схем.

Мы воспользуемся вторым способом.

Теперь решим когда необходимо отображать рисунок нашей области? Конечно же в самом начале выполнения программы, т. к. пользователь должен сразу видеть область, а не выполнять какие то действия по ее отображению.

Наиболее подходящее место по отображения области это процедура создания формы. Когда форма создается (ее границы, метки, поля ввода, кнопки

и др. компоненты) мы рисуем область.

Что бы выбрать обработчик создания формы сделаем двойной клик в свободной области формы (где нет других компонентов). В результате получим следующий обработчик.

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

Рис. 2.6 — Обработчик создания формы.

Между элементам Begin и End добавим код, который будет рисовать область.

Для рисования необходимо обратиться к тому компоненту, на котором мы будем рисовать, в данном случае компонент Image1. У компонента имеется много свойств, с некоторыми уже познакомились. Свойство отвечающее за рисование примитивов называется Canvas.

Рассмотрим как происходит прорисовка.

Существует ряд процедур у свойства Canvas которые рисуют примитивные объекты. Например для отображения линии необходимо выбрать процедуру Line и передать ей 4 координаты (координаты начала и конца линии). На экране после вызова процедуры отобразится линия. Для построения эллипса необходимо вызвать процедуру Ellipse с 4-мя координатами. Это координаты левого верхнего и правого нижнего угла прямоугольника, в который будет вписан эллипс (направления осей эллипса совпадают с направлением осей координат).

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.

В среде Lazarus нет одной процедуры, которая сразу нарисовала бы такой сложный рисунок как в задании. Есть несколько примитивных функций с помощью которых можно последовательно (по правилу художника) нарисовать практически любой рисунок.

Разделим рисунок на 3 области, которые будем рисовать:

1 — четверть окружности;

2 — прямоугольник;

3 — оси координат.

Прямоугольник и линии можно нарисовать средствами Canvas вызвав по одной процедуре, а четверть окружности — нет.

Для отображения четверти окружности мы нарисуем всю окружность, а лишнее сотрем.

Как рассчитывать координаты? Воспользуемся следующими формулами для перевода координат из физических в экранные.

X = X*20 + 100

Y =100 – Y*20

Отобразим в центре рисунка окружность. Добавим следующий код.

```
35
 .  procedure TForm1.FormCreate(Sender: TObject);
 .  begin
 .    Image1.Canvas.Ellipse(80,80,120,120);
 .  end;
40
```

Рис. 2.7 — Прорисовка окружности в центре картинке.

Сохраним проект и запустим на выполнение. В результате программа изменит вид.

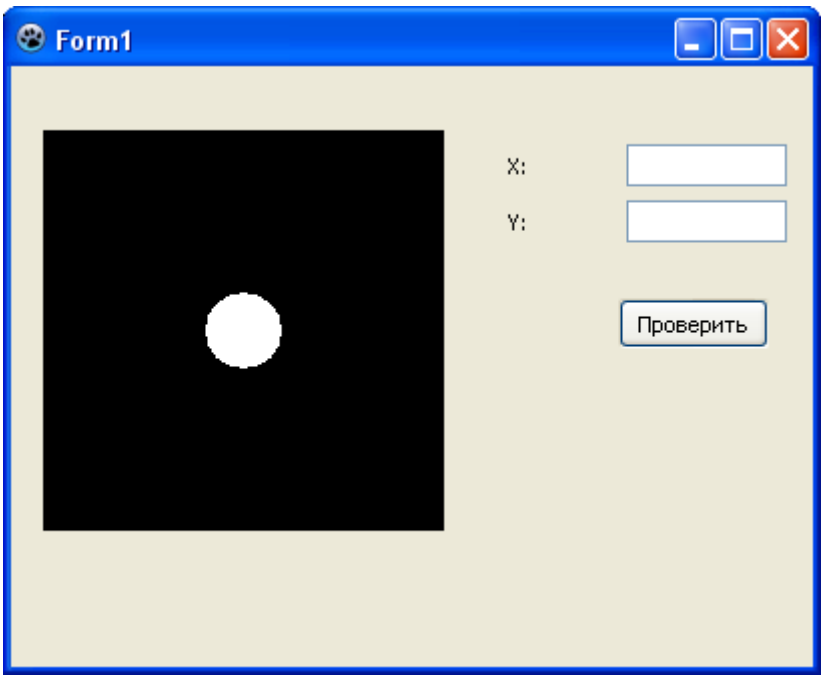


Рис. 2.8 — Окружность в центре Image.

Не совсем то, что ожидали, но рисунок.

Рисовать на черном фоне белым цветом можно, разве что на доске мелом. Обычно на белом фоне рисуют черные линии — нужно закрасить фон в белый цвет.

По умолчанию карандаш имеет цвет черный, а кисть белый. Поэтому окружность белого цвета — она закрашена белой кистью.

Для закраски фона воспользуемся белым прямоугольником размером равным размеру компонента Image1. Очистку фона надо вызвать до прорисовки окружности, что бы не стереть окружность. Добавим следующую строчку выше отображения окружности.

```
35
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Image1.Canvas.Rectangle(0,0,200,200);
    Image1.Canvas.Ellipse(80,80,120,120);
40 end;
```

Рис. 2.9 — Очистка картинки и прорисовка окружности.

Сохраним проект и запустим на выполнение, в результате получим следующий вид:

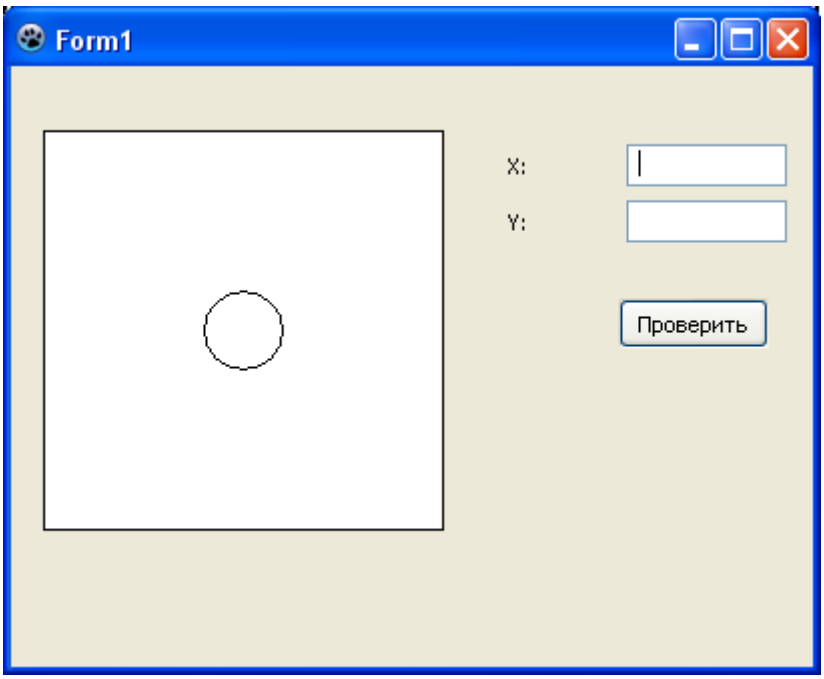


Рис 2.10 — Отображение окружности на белом фоне.

Что бы четко различать границы области закрасим внутреннюю часть окружности в зеленый цвет.

Немного о цветах в ОС Windows и среде Lazarus: Все чистые цвета и многие оттенки цветов могут быть представлены в виде трех составляющих: красного, зеленого и синего. В ОС Windows эта комбинация трех цветов называется RGB по первым буквам соответствующих цветов на английском (Red, Green, Blue). Каждая составляющая берется в диапазоне от 0 до 255 и чем больше число, тем сильнее выражен соответствующий цвет. Пример: для красного цвета надо красную составляющую повысить на максимум, а остальные убрать на минимум (255,0,0). В среде Lazarus присутствует константы цветов — те которыми мы привыкли пользоваться в повседневной жизни но на английском языке. Например для красного цвета это константа clRed. Константа состоит из префикса «cl» (сокращение от английского Color) и самого названия цвета Red (красный на английском языке).

Закрасим нашу окружность зеленым цветом. Для этого необходимо до рисования окружности назначить зеленый цвет кисти.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.Rectangle(0,0,200,200);
  Image1.Canvas.Brush.Color:=clGreen;
  Image1.Canvas.Ellipse(80,80,120,120);
end;
```

Рис 2.11 — Выбор зеленого цвета кисти и отображение зеленой окружности.

Сохраним проект и запустим на выполнение. В результате получим следующую форму:

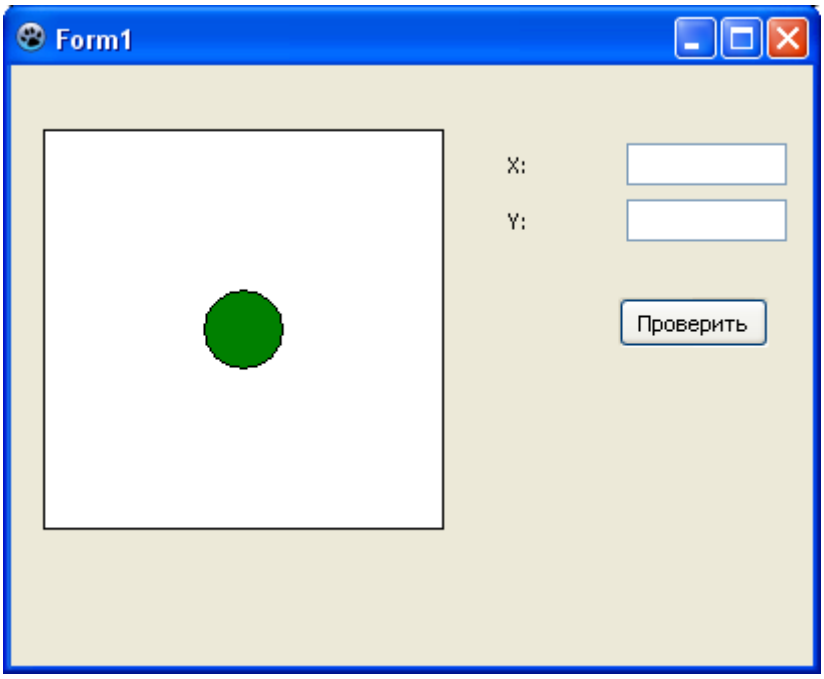


Рис. 2.12 — Зеленая окружность в центре области.

По заданию необходима не вся окружность, а только ее часть (четверть). Что бы убрать лишние части окружности — изменим кисть не белую и карандаш то же выберем белый. Прорисуем поверх зеленого круга белые прямоугольники. Допишем следующий код:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.Rectangle(0,0,200,200);
  Image1.Canvas.Brush.Color:=clGreen;
  Image1.Canvas.Ellipse(80,80,120,120);
  Image1.Canvas.Brush.Color:= clWhite;
  Image1.Canvas.Pen.Color:=clWhite;
  Image1.Canvas.Rectangle(80,80,120,100);
  Image1.Canvas.Rectangle(100,100,120,120);
end;
```

Рис. 2.13 — Выбор белой кисти и карандаша со стиранием 3/4 окружности.

Сохраним проект и запустим на выполнение.

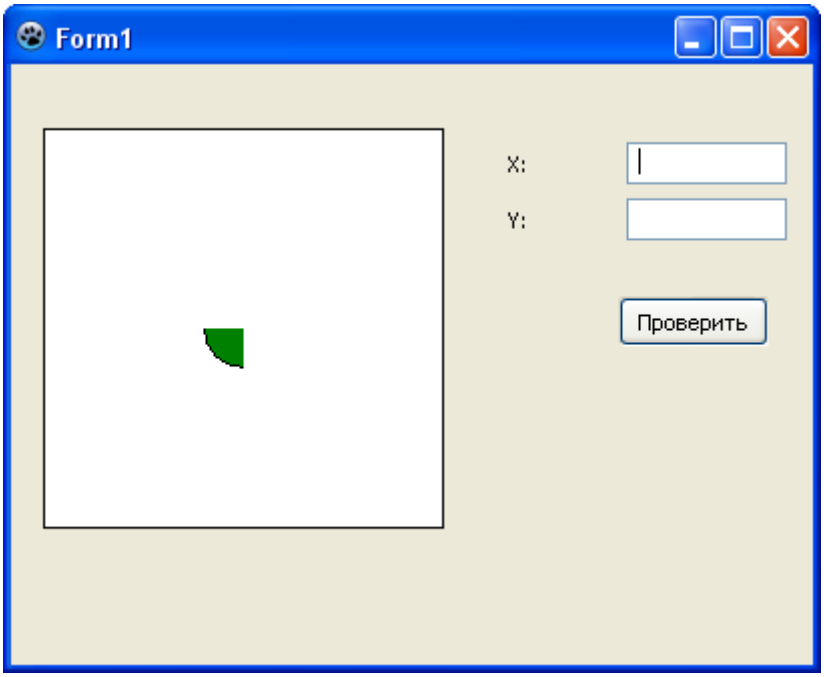


Рис 2.14 — Четверть окружности.

В результате от окружности осталась одна четверть.

Приступим к рисованию второй части области — прямоугольнику. Для этого сменим кисть опять на зеленую, а карандаш на черный и нарисуем прямоугольник. Добавим следующий код ниже.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   Image1.Canvas.Rectangle(0,0,200,200);
   Image1.Canvas.Brush.Color:=clGreen;
   Image1.Canvas.Ellipse(80,80,120,120);
   Image1.Canvas.Brush.Color:= clWhite;
   Image1.Canvas.Pen.Color:=clWhite;
   Image1.Canvas.Rectangle(80,80,120,100);
   Image1.Canvas.Rectangle(100,100,120,120);
   Image1.Canvas.Pen.Color:=clBlack;
   Image1.Canvas.Brush.Color:=clGreen;
   Image1.Canvas.Rectangle(100,60,140,100);
end;
```

Рис. 2.15 — Прорисовка прямоугольника черным карандашом и зеленой кистью.

Сохраняем и запускаем проект. Получим следующую форму.

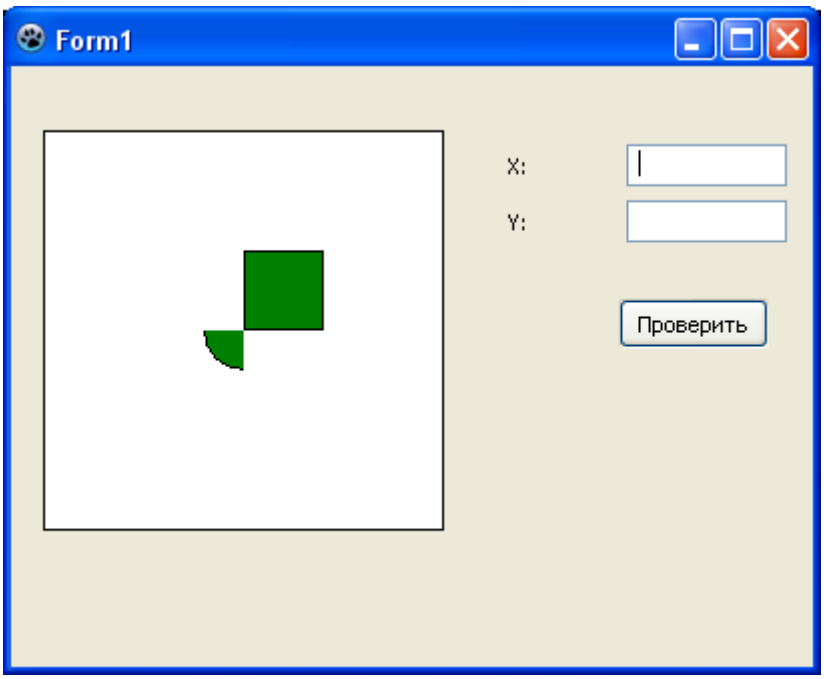


Рис 2.16 — Четверть окружности и прямоугольник.

Осталось добавить оси координат, воспользуемся функцией Line.

Добавим следующий код.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.Rectangle(0,0,200,200);
  Image1.Canvas.Brush.Color:=clGreen;
  Image1.Canvas.Ellipse(80,80,120,120);
  Image1.Canvas.Brush.Color:= clWhite;
  Image1.Canvas.Pen.Color:=clWhite;
  Image1.Canvas.Rectangle(80,80,120,100);
  Image1.Canvas.Rectangle(100,100,120,120);
  Image1.Canvas.Pen.Color:=clBlack;
  Image1.Canvas.Brush.Color:=clGreen;
  Image1.Canvas.Rectangle(100,60,140,100);
  Image1.Canvas.Line(0,100,200,100);
  Image1.Canvas.Line(100,0,100,200);
end;
```

Рис 2.17 — Прорисовка осей координат.

Сохраним и запустим проект. В результате форма программы выглядит следующим образом.

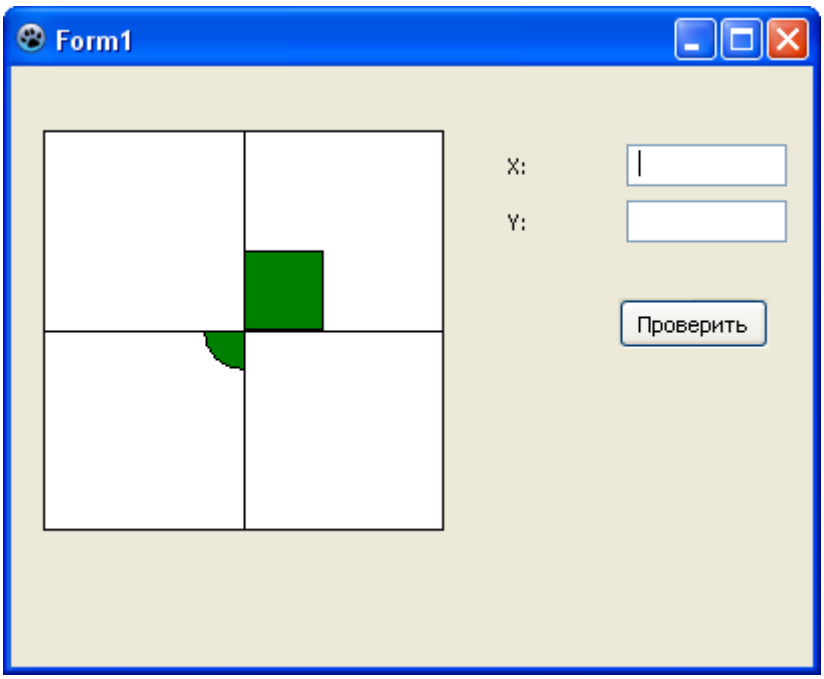


Рис 2.18 — Полностью готовый рисунок с четвертью окружности, прямоугольником и системой координат.

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

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

Рис. 2.19 — Обработчик на кнопку «Проверить».

Для проверки понадобятся две вспомогательные переменные в которых будем хранить координаты, которые ввел пользователь в поля «X:» и «Y:». Координаты вещественные (могут иметь дробную часть), объявим их как Real. Запишем в эти переменные значения из соответствующих полей.

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

Рис. 2.20 — Объявление двух переменных «x» и «y» и их инициализация.

Запустив проект мы не заметим никаких изменений в работе программы.

Необходимо проанализировать значения координат с помощью оператора выбора «IF». Воспользуемся логическими операциями AND, OR, NOT и операциями сравнения «>» «<» «>=» «<=» «<>» «=». Анализ координат разделим на анализ в каждой отдельной фигуре (четверти окружности и прямоугольнике). Если точка принадлежит хотя бы одной из фигур, то она принадлежит нашей области.

Точка принадлежит прямоугольнику, когда координата «X» в пределах от левой до правой грани, а координата «Y» от верхней до нижней. Здесь уже используются координаты физические, а не экранные (те что заданы на рисунке в задании). Для проверке четверти окружности необходимо знать уравнение всей 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) находятся на окружности, если поставить знак «<» то внутри окружности, а если «>» то вне окружности.

Воспользуемся знаком «<» (строго меньше, т. к. граница не входит в область — вариант первый — нечетный) и определим значения X_0 и Y_0 . Центр окружности находится в точке с координатами 0,0 тогда $X_0=0$ и $Y_0=0$. По заданию не вся окружность, а только четверть, значит дополнительно ограничим нашу окружность одной четвертью. Фигуры (четверть окружности и прямоугольник) между собой объединяются с помощью оператора OR (или), а условия принадлежности фигуре с помощью AND (и).

Оператор выбора (IF) может содержать один или два (как в данном случае) вложенных в него оператора. Первый оператор находящийся после слова «Then» выполнится если условие выполнится, а второй ели условие не выполнится. Обо одновременно не могут выполнится, и ни одни тоже не может выполнится. Хотя бы один из операторов (после слова «Then» или «Elase»)

всегда выполняется, но какой зависит от условия. Если условие выполняется будем выдавать (при помощи функции ShowMessage) текст «Точка принадлежит области», а если условие не выполнится, то «Точка НЕ принадлежит области».

Добавим следующий код.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  x,y : Real;
begin
  x := StrToFloat(Edit1.Text);
  y := StrToFloat(Edit2.Text);
  if ((x>0)and(x<2)and(y>0)and (y<2))or
     ((x*x+y*y<4)and(x<0)and(y<0)) then
     ShowMessage('Точка принадлежит  области') else
     ShowMessage('Точка НЕ принадлежит области');
end;
```

Рис. 2.21 — Проверка на принадлежность координат точки прямоугольнику и четверти окружности.

Сохраним проект и запустим на выполнение. Проверим работает ли наша проверка или нет? В соответствующие поля введем координаты (например 1, 1) и нажмем кнопку проверить.

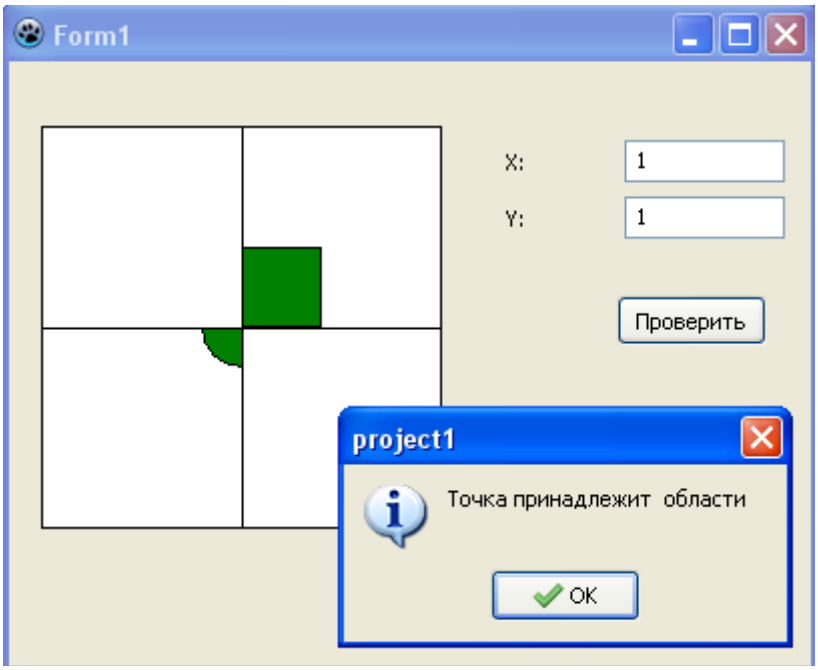


Рис. 2.22 — Проверка координат.

Проверим координаты точек (2;2), (0;0) (-0.5;-0.5) на принадлежность области. Первые две точки должны не принадлежать области (т. к. граница области не входит), а третья принадлежит области, почему?

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» «мышью» 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 мент 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.

Более подходящее событие для анализа координат «мыши» это OnMuseDown. Найдем его в «Инспекторе объектов», предварительно выбрав компонент Image1, на вкладке «события».

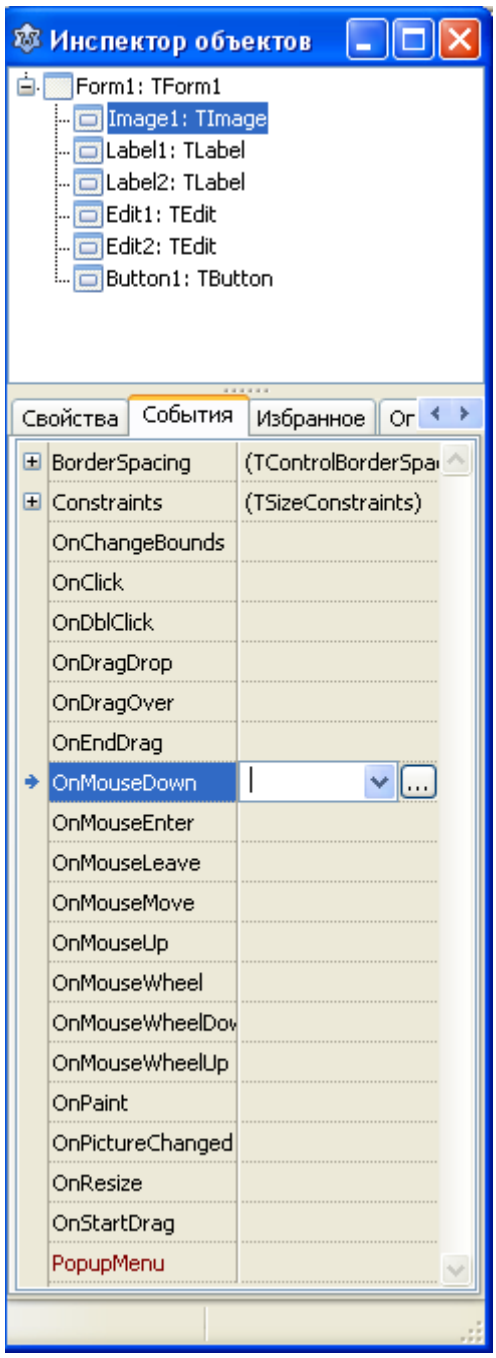


Рис. 2.23 — Привязка события «OnMouseDown» к «Image1».

Список возможных привязываемых событий пуст, но справа есть кнопка «...» нажмем на нее и получим заготовку события OnMouseDown.

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

Рис. 2.24 — Обработчик события «OnMouseDown» для «Image1».

У этого обработчика есть два интересующих нас параметра — координаты «мыши» в момент «клика». Они соответственно хранятся в

переменных «X» и «Y». Напишем код между begin и end. Первые две строки преобразуют координаты из экранных в физические и заносят в соответствующие поля ввода, а третья строка в точке «клика» рисует маленькую окружность.

```
55  procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
    .     Shift: TShiftState; X, Y: Integer);
    .  begin
    .     Edit1.Text:=FloatToStr((X-100)/20);
    .     Edit2.Text:=FloatToStr((Y-100)/(-20));
60  .     Image1.Canvas.Ellipse(x-2,y-2,x+2,y+2);
    .  end;
```

Рис. 2.25 — Преобразование координат с экранных в физические и занесение в Edit-ы, отображение в месте клика небольшой окружности.

Сохраним проект и запустим на выполнение.

Сделав несколько «кликов» на компоненте Image1 получим примерно следующий вид программы.

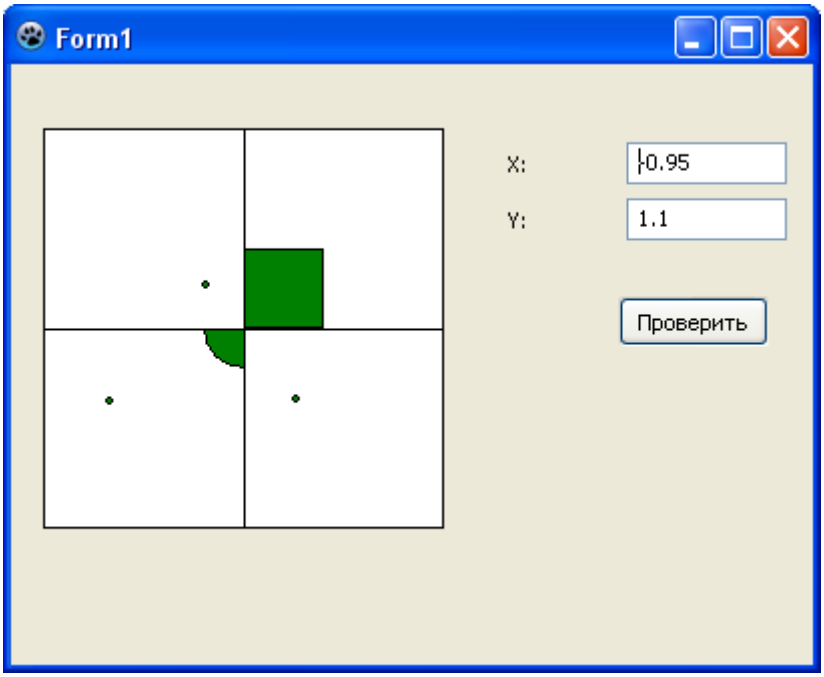


Рис. 2.26 — Внешний вид программы с проверкой координат.

Работа над демонстрационном примером закончена. Воспользуйтесь этой программой и нарисуйте свою область не забыв изменить процедуру проверки координат. Интерфейс программы менять нет необходимости, а все изменения производятся в процедурах «Button1Click» и «FormCreate».

Лабораторная работа №3

Тема: Оператор цикла заданного числа раз.

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. Расставим на форме компоненты:

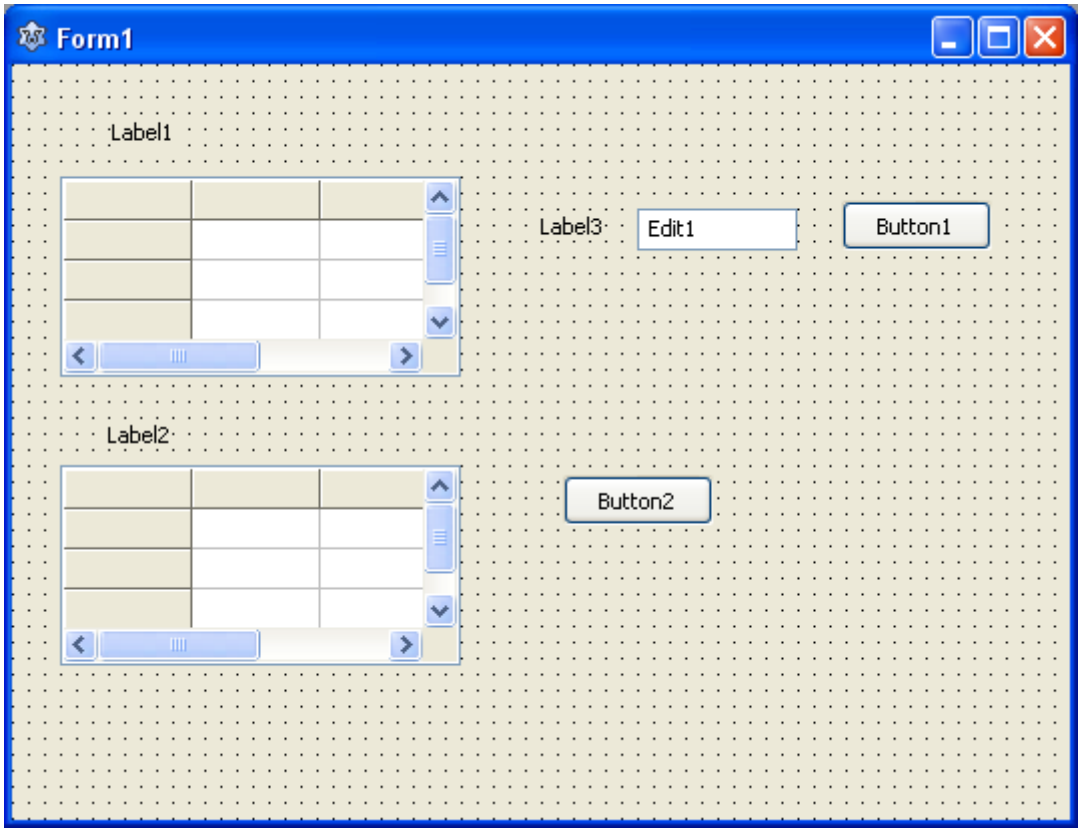


Рис. 3.1 — Расположение компонентов на форме.

Настройка компонентов TLabel, TEdit и TButton и их функционирование рассматривалось в предыдущих примерах. В этой программе познакомимся с новым компонентом TStringGrid – таблица строк. На форму установим два экземпляра этого компонента (один под другим). При помощи таблицы строк можно отображать на форме и редактировать большие массивы строковых данных. Каждая строка в таблице хранится в отдельной ячейке, доступ к которой осуществляется по номеру столбца и строки. Столбцы нумеруются слева направо, начиная с 0, а строки нумеруются с верху вниз начиная с 0. В

результате левая верхняя ячейка имеет номер (0,0). Таблицы могут иметь несколько зафиксированных столбцов и/или строк — шапка таблицы. Она всегда видна независимо от того с какой ячейкой работает пользователь. Шапку редактировать пользователь не может, а содержимое таблицы можно.

В данной программе шапки у таблиц не понадобятся — от них избавляемся установив для обоих таблиц в «инспекторе объектов» свойства FixedRows=0 и FixedCols = 0. Настроим «метки», «кнопки» и «поле ввода данных» — количество элементов как показано на рис 3.2.

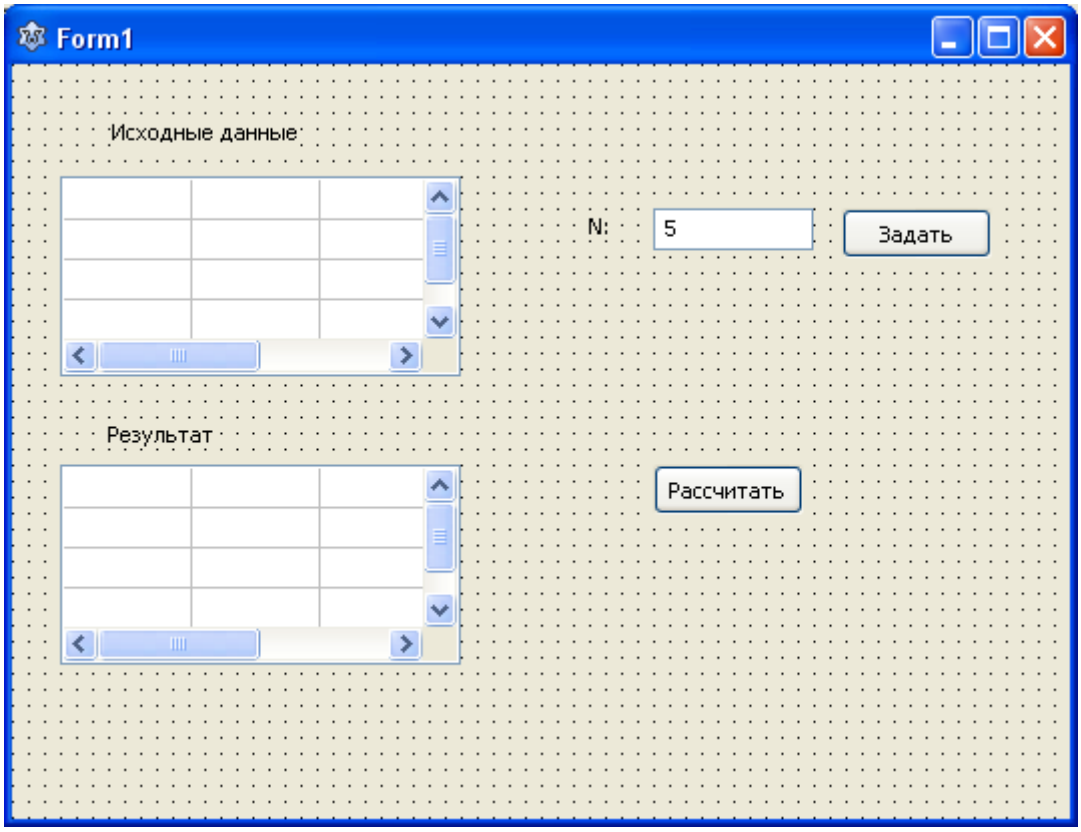


Рис. 3.2 — Настройка компонентов на форме.

По умолчанию таблица создается размером 5x5 элементов. В программе понадобится 2 строки. В верхней хранится 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 StringGrid1 и StringGrid2.

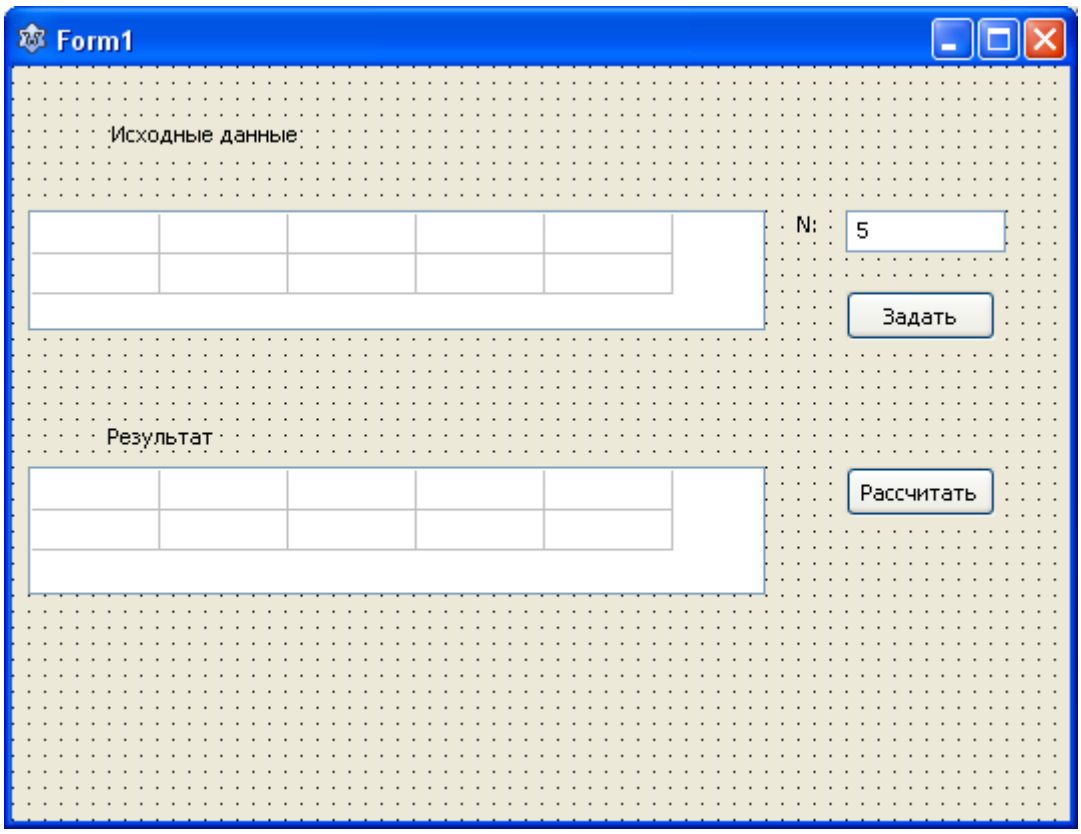


Рис. 3.3 — Настройка компонентов StringGrid.

Сохраним проект и запустим на выполнение.

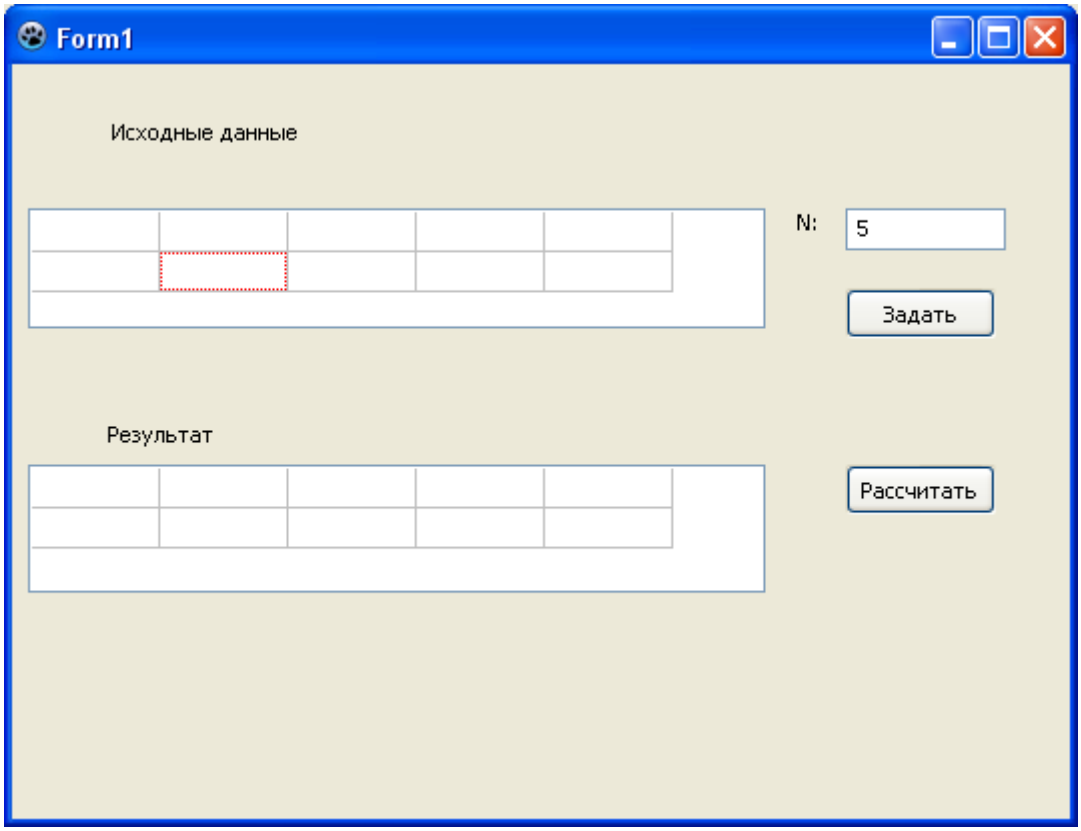


Рис. 3.4 — Внешний вид программы после запуска.

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 таблицы. Данное свойство не просто текст или число, а сложное — множество. Каждый элемент множества может быть внесен или исключен из него. Для работы с элементами множества необходимо «раскрыть» список всех возможных элементов множества, для этого надо «кликнуть» по знаку «+» слева от свойства Options. Интересующий элемент множества goEditing — отвечает за редактирование значений ячеек таблицы во время выполнения программы. Для разрешения редактирования ячеек таблицы необходимо его внести во множество — установив значение true. Поменяем свойство Options у обеих таблиц, сохраним проект и запустим на выполнение.

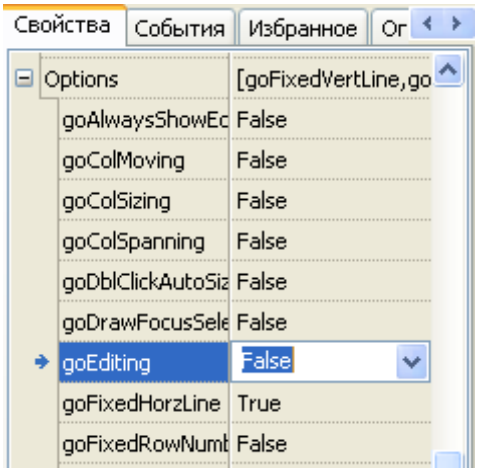


Рис. 3.5 — Изменение свойства Options → goEditing у обоих таблиц.

После сохранения и запуска программы появилась возможность внесения данных в таблицы.

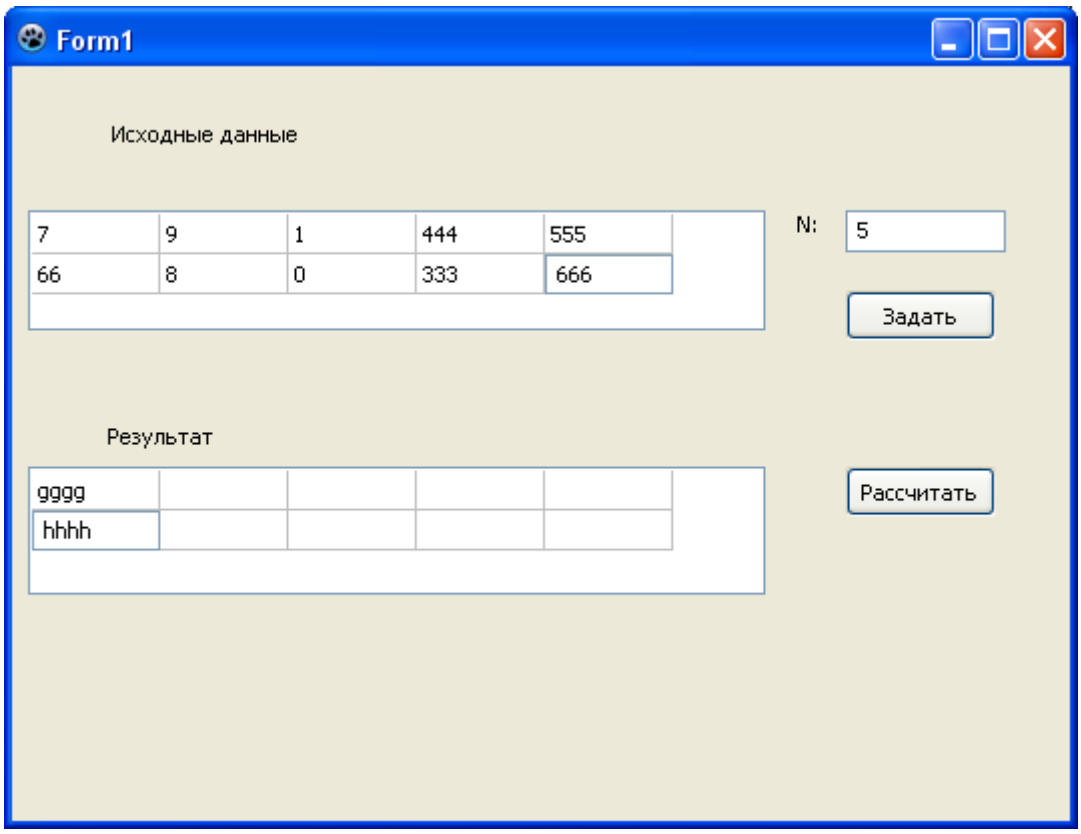


Рис. 3.6 — Внесение данные в таблицы.

В таблицы можно вносить не только цифры но и текст.

Добавим «функциональную начинку» для нашей программы — обработчики событий.

Напишем обработчик для кнопки «Задать». Он запоминает число которое ввел пользователь в поле над этой кнопкой, создает указанное число столбцов у таблиц 1 и нумерует (создает индексы) в верхней таблице в нулевой (верхней) строке.

Сделаем «двойной клик» в редакторе форм на кнопке «Задать» и добавим следующий код.

```
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;
```

Рис. 3.7 — 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(Edit1.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).

Сохраним проект и запустим на выполнение. Внесем в поле «N:» число 4 (или любое другое положительное целое, но не очень большое) и нажмем на кнопку «Задать». Отобразиться форма.

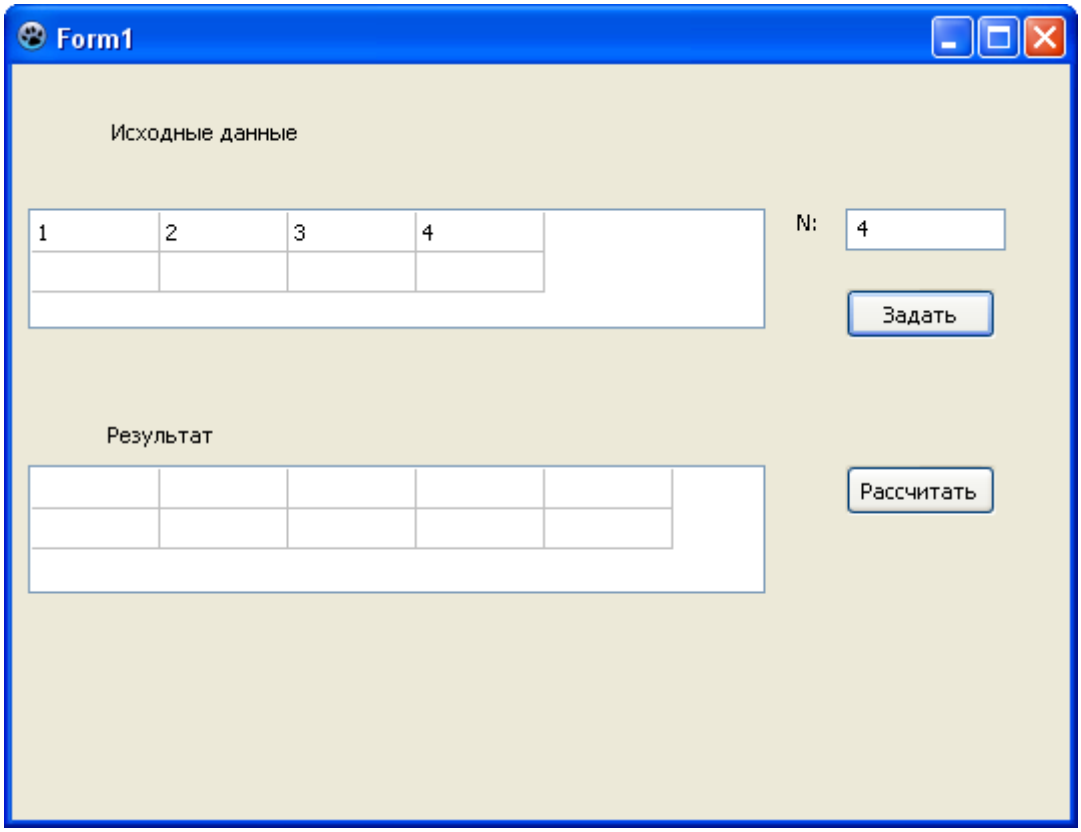


Рис. 3.8 — Задание количества столбцов и нумерации у первой таблицы.

По заданию необходимо найти среднее значение и вывести во вторую таблицу только те элементы, которые больше среднего. Из курса математике известно, чтобы найти среднее значение необходимо подсчитать сумму всех элементов и разделить на их количество.

Процедура (обработчик) для кнопки «Рассчитать» следующая:

```
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;
```

Рис. 3.9 — Код обработчика для кнопки «Рассчитать».

Рассмотрим подробно, что в ней происходит.

Объявили три переменные: две целых «i» - индекс (номер столбца таблицы) и «n» - количество элементов в таблице, переменная «sr» вещественного типа данных — в которой будем подсчитывать сумму элементов. Разделив подсчитанную сумму на их количество получим среднее значение. Выбран вещественный тип данных (Real) для третей переменной т. к. при делении может получиться дробное значение.

Рассмотрим последовательность действий выполняемые в процедуре.

« sr := 0;» - обнуление счетчика суммирования.

«n := StrToInt(Edit1.Text);» - получение из поля Edit1 количество элементов в таблице на обработку.

«for i:=0 to n-1 do

sr := sr + StrToFloat(StringGrid1.Cells[i,1]);» - подсчет суммы элементов в таблице. Первая строчка — цикл повторяющийся ровно «n» раз, вторая — наращивание значение переменной «sr» за счет добавления к ней значений хранящихся в ячейках таблицы. Обращение к ячейкам осуществляется по двум индексам. 1-й индекс это номер столбца (он меняется), второй — номер строки — он постоянен = 1. Перед суммированием преобразовываем значения ячеек из строк в числа — функция StrToFloat().

Подсчет суммы готов. Для вычисления среднего значения — разделим найденную сумму на число элементов: «sr := sr / n;».

Как минимум в таблице будет 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» - осуществляем повторный перебор таблицы.

«if StrToFloat(StringGrid1.Cells[i,1]) >= sr then» - осуществляем проверку каждой ячейке на условие отбора — значение ячейки должно быть больше или равно среднему значению.

Если условие выполнится то программа выполнит действия указанные после слова «then» в операторе выбора. По синтаксису языка Object Pascal там может находится 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(StringGrid2.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.

Код обработчика для кнопки «Рассчитать» готов. Сохраним проект и запустим на выполнение.

Установим количество элементов в первой таблице, внесем в нее данные и рассчитаем вторую таблицу. В результате у нас должна появиться примерно такая форма.

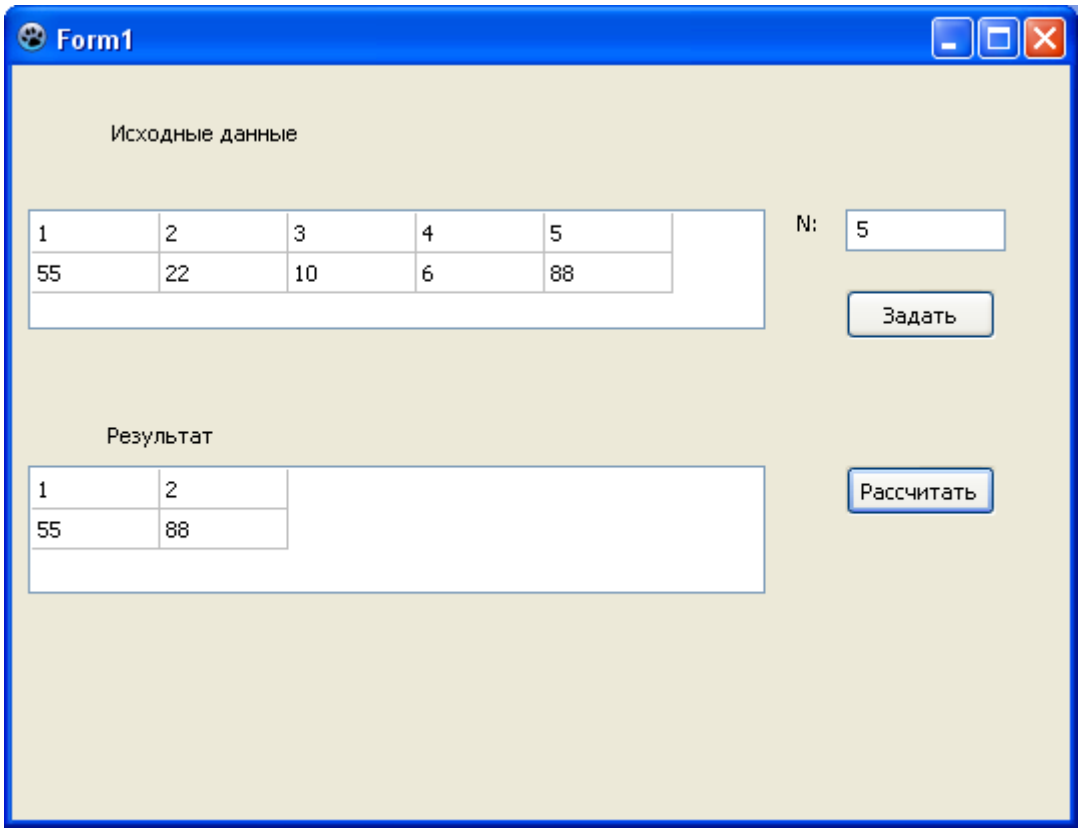


Рис. 3.10 — Пример работы программы.

Программа готова, в индивидуальной работе необходимо изменить обработчики на кнопки «Задать» и «Рассчитать», интерфейс изменяется минимально.

Лабораторная работа №4

Тема: Цикл с предусловием и с пост условием.

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

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

Создадим интерфейс программы.

Пользователь не вводит никаких данных, а только инициирует расчет ряда чисел и получает результат в табличном виде. Интерфейс на рисунке 4.1.

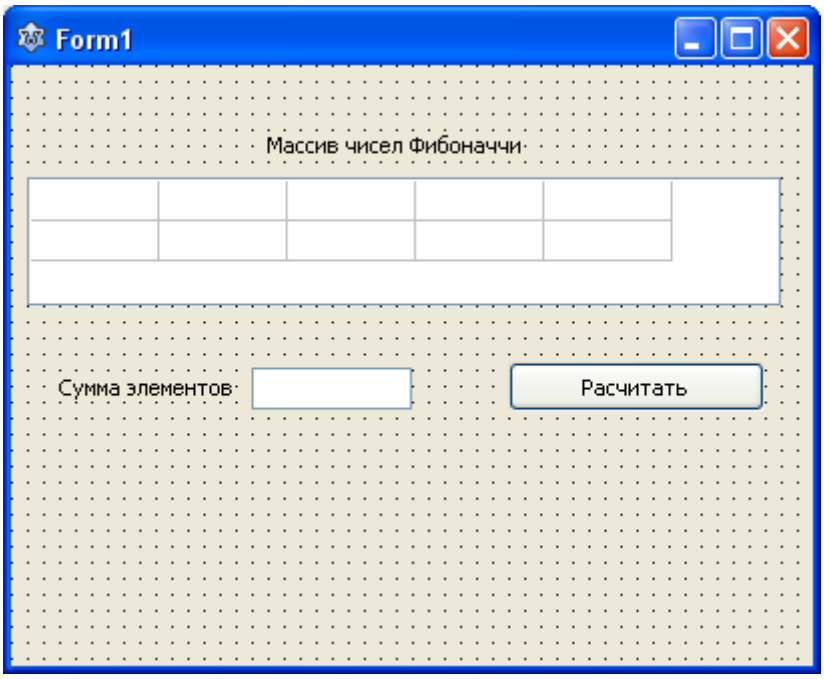


Рис 4.1 — Интерфейс программы.

Вверху расположена таблица «StringGrid», внизу поле для вывода суммы элементов ряда и кнопка «Рассчитать». Таблицу настроим так же, как в 3-й работе, а кнопку переименуем. Надписи меток оформим согласно представленному рисунку 4.1.

В программе всего один обработчик связанный с кнопкой «Рассчитать».

```
35  procedure TForm1.Button1Click(Sender: TObject);
    var
       a,b,c : integer;
       s : integer;
    begin
40     StringGrid1.ColCount:=2;
       StringGrid1.Cells[0,0]:='1';
       StringGrid1.Cells[0,1]:='1';
       StringGrid1.Cells[1,0]:='2';
       StringGrid1.Cells[1,1]:='1';
45     a:=1;
       b:=1;
       s := a+b;
       while s < 100 do
       begin
50       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;
55       b := c;
         s := s + c;
       end;
       Edit1.Text:=IntToStr(S);
    end;
```

Рис. 4.2 — Обработчик кнопки «Рассчитать».

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.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;» - инициализация (установление первоначальных) значений переменных для первых двух элементов ряда чисел Фибоначчи, подбор количество столбцов и расчет суммы.

«while s < 100 do» - цикл с предусловием, выполняться до тех пор пока сумма (переменная «s») меньше 100.

Далее тело цикла.

«StringGrid1.ColCount := StringGrid1.ColCount + 1;» — увеличение количества столбцов в 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.

На этом тело цикла заканчивается.

За циклом:

«Edit1.Text:=IntToStr(S);» - вывод суммы в поле «Edit1».

Сохраним проект и запустим на выполнение, отобразится форма.

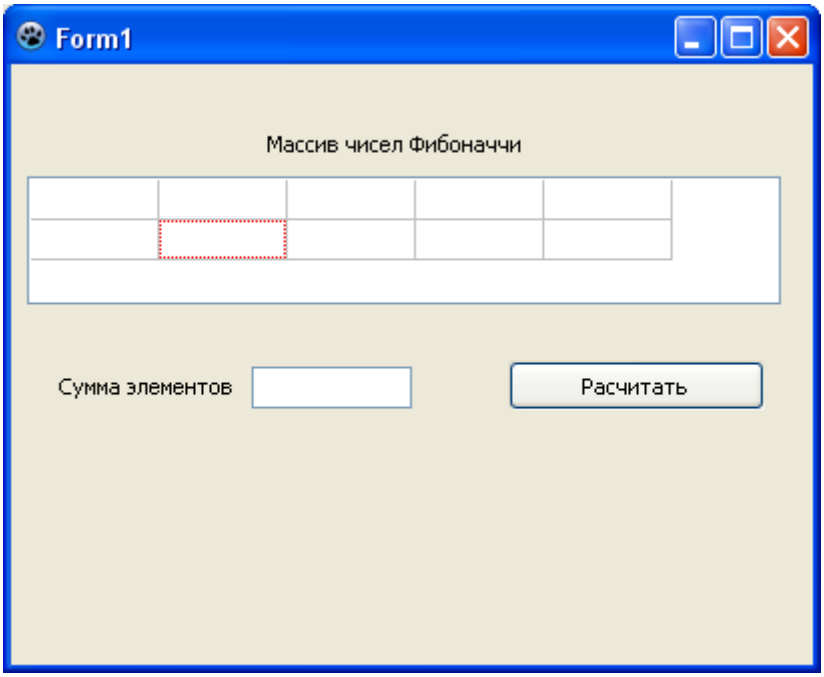


Рис. 4.3 — Форма для расчета ряда Фибоначчи.

При нажатии на кнопку «Рассчитать» получим следующие данные.

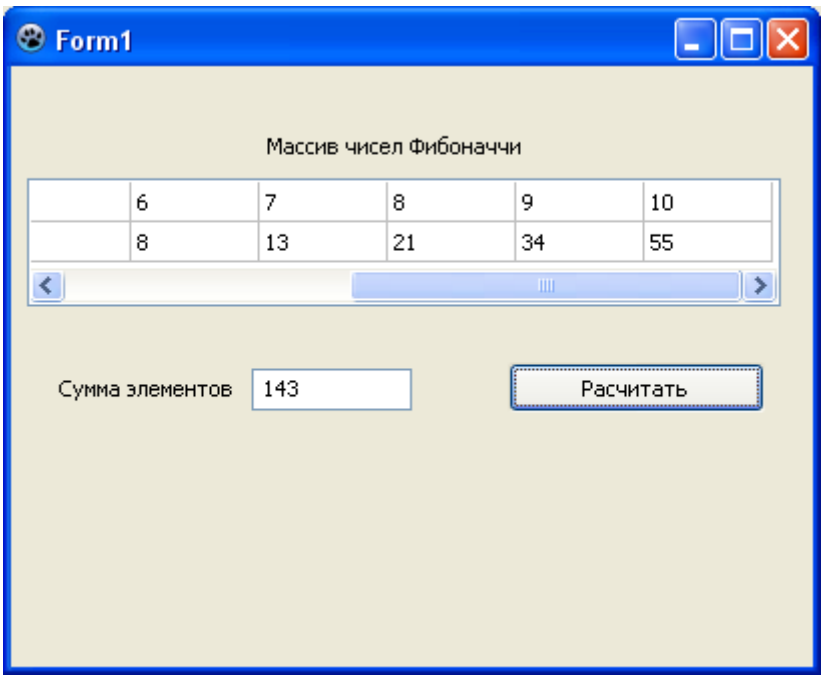


Рис. 4.4 — Рассчитанный ряд Фибоначчи.

Программа готова, на основе данного примера выполните свое индивидуальное задание. В место ряда чисел Фибоначчи используйте ряд заданный формулой (формула схожа с формулой Фибоначчи — она оперирует предыдущими двумя значениями ряда).

Лабораторная работа №5

Тема: Массивы.

Задание: Создать и отобразить исходный массив. Выбрать элементы больше среднего значения.

Начальные условия:

$S_i = 2S_{i-2} - 3S_{i-1} + 5$

$S_1 = 2$

$S_2 = 4$

$n = 16$

type = int

Разработаем интерфейс программы.

Пользователь не вводит никаких данных, а только инициирует расчет ряда чисел и получает результат в табличном виде. Интерфейс представлен на рисунке 5.1.

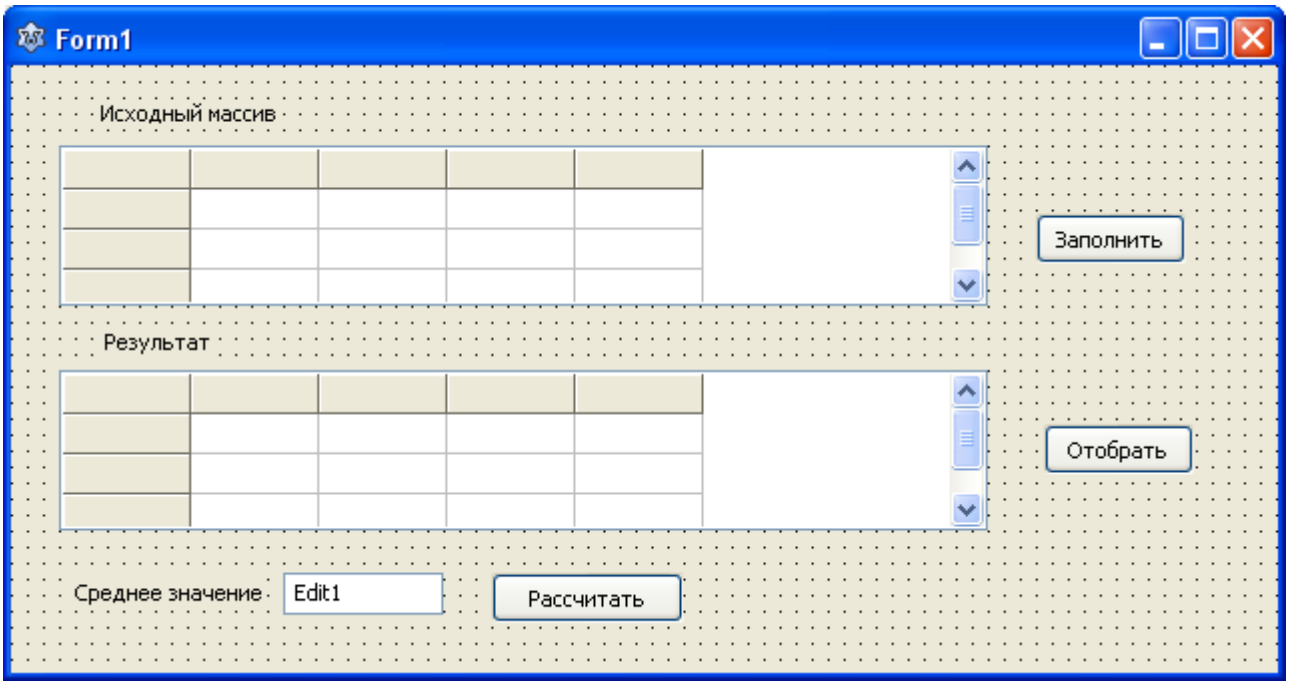


Рис 5.1 — Интерфейс программы.

На форме разместим две таблицы (TstringGrid), три метки (TLabel) и три кнопки (TButton). Переименуем их как показано на рисунке 5.1 (меняем свойство Caption у элементов формы).

Настроим обе таблицы: уберем зафиксированные строки и столбцы и

уменьшим количество строк до двух. Назначим свойствам таблиц:

FixedRows = 0

FixedCols = 0

RowCout =2

Форма примет вид:

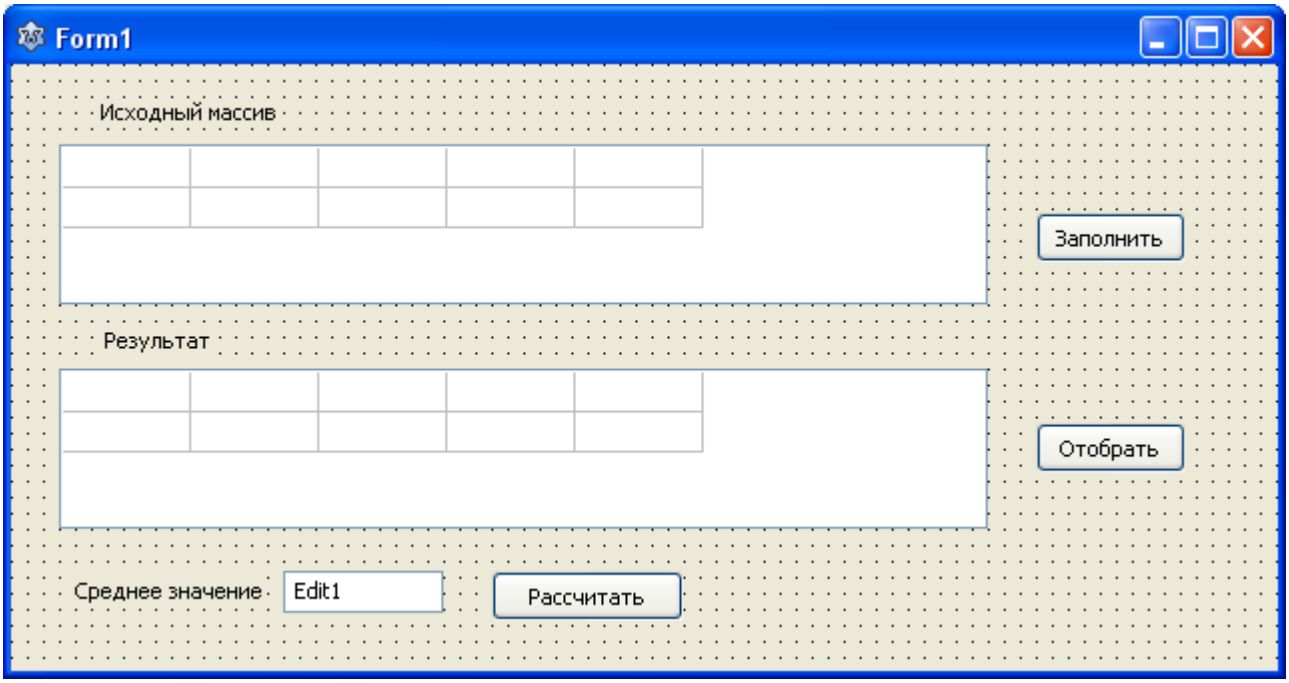


Рис 5.2 — Настройка таблиц, меток и кнопок у формы.

Интерфейс готов, напишем обработчики событий.

Обращаться к массиву будем неоднократно — заполнять его, подсчитывать среднее значение и выдавать часть массива удовлетворяющего условию задания. Что бы не вычислять ряд каждый раз будем использовать массив для хранения элементов.

Опишем массив — как глобальный массив модуля в разделе var.

По условию элементов 16, они целые и нумерация начинается с 1. Этой информации достаточно для описания массива в Object Pascal. Создадим массив с именем mas.

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

Рис 5.3 — Создание массива из 16 целых элементов.

Напишем процедуру вычисляющую все элементы ряда в первую таблицу

и привяжем ее к кнопке «Запомнить».

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 рисунке 5.4.

```
  { TForm1 }

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

Рис 5.4 — Расчет ряда, занесение в массив и таблицу.

Запоминаем в массиве вспомогательные A1 и A2 переменные — первые два числа.

Вычисляем и заносим в массив элементы ряда с 3-го по 16-й.

Устанавливаем количество столбцов у таблице 16.

Выводим в верхнюю строку порядковый номер элемента и в нижнюю строку значение элемента.

Процедура (обработчик события) готова. Сохраним проект, запустим на выполнение, проверим работу алгоритма у кнопки «Заполнить».

Программа выдает ряд чисел:

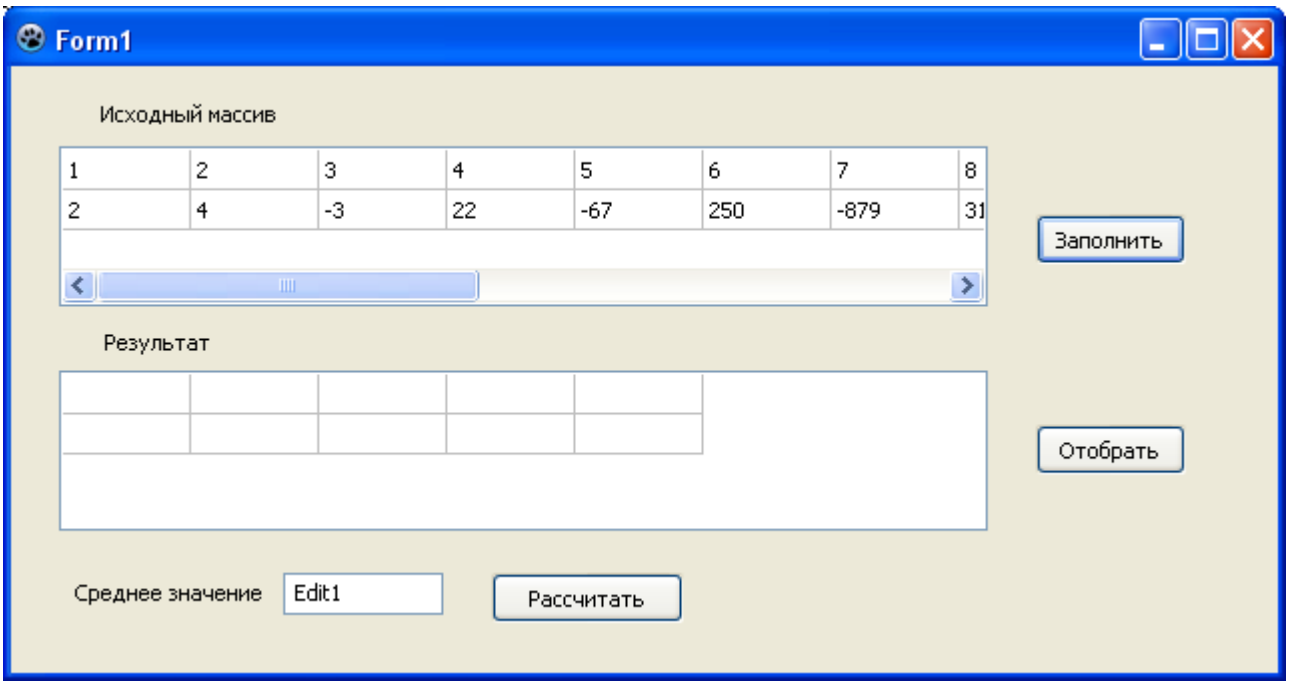


Рис 5.5 — Результат работы программы по кнопке «Запомнить».

Вычислим среднее значение ряда. Напишем обработчик кнопки «Рассчитать», а результат запишем в поле Edit1.

Для вычисления понадобится переменная — индекс массива и переменная под среднее значение.

Код процедуры расчета подробно рассматривался в предыдущем примере — принцип расчета не изменился:

```
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;
```

Рис 5.6 — Код обработчика — расчет среднего значения.

Запустим программу на выполнение и вычислим среднее значение ряда.

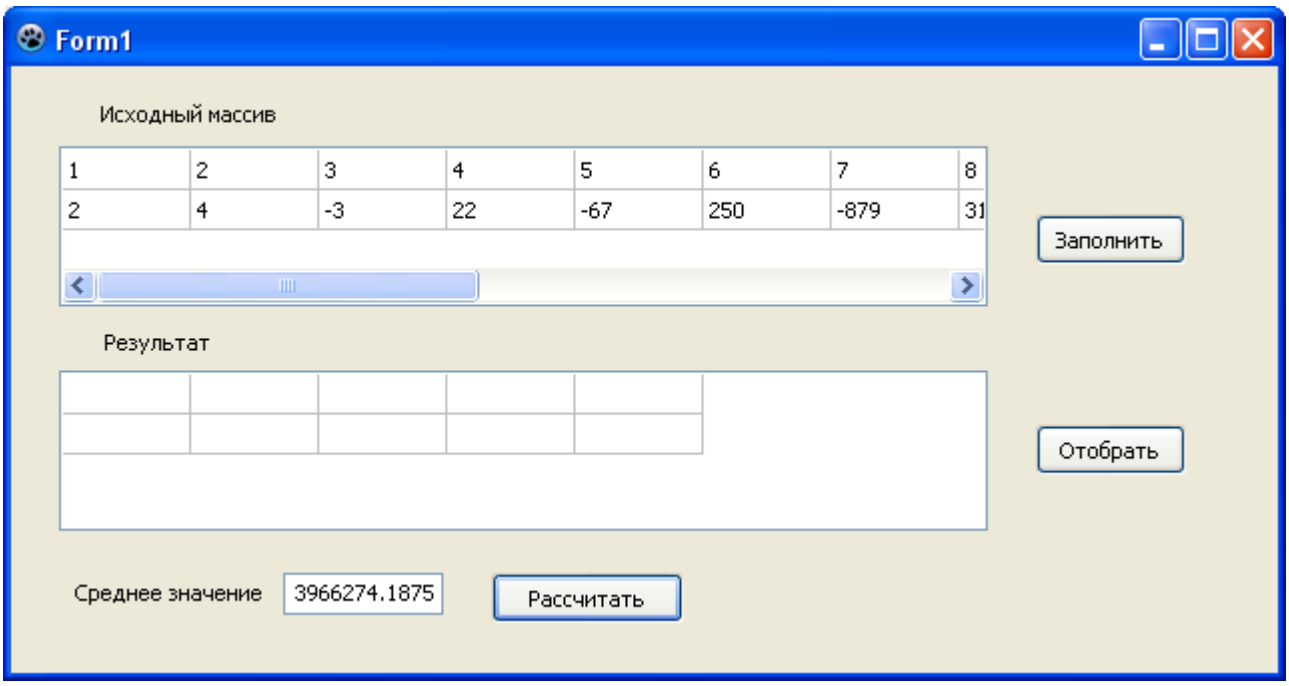


Рис 5.7 — Результат расчета среднего значения ряда.

В нижнюю таблицу выведем только те элементы массива, которые больше среднего значения.

Напишем обработчик — выбор элементов больше среднего и привяжем его к кнопки «Отобрать».

```
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;
```

Рис 5.8 — Процедура отбора значений больше среднего.

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.

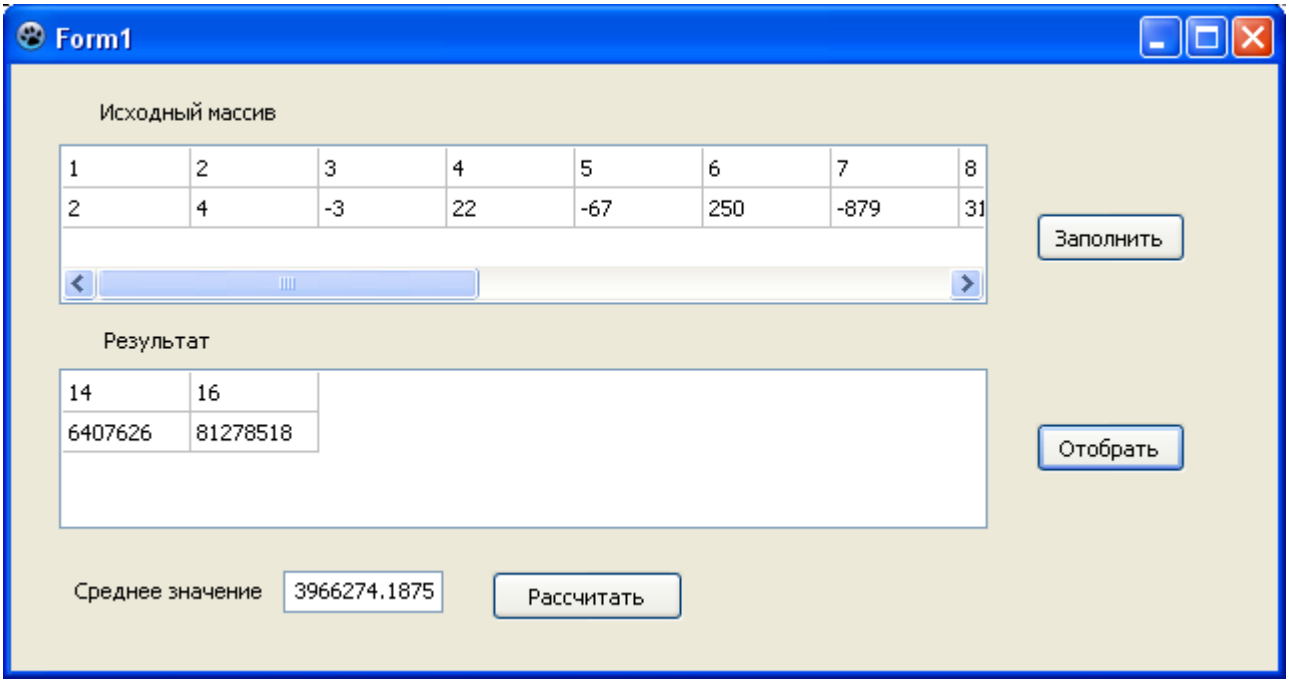


Рис 5.9 — Отбор элементов больше среднего.

Программа готова. На основе данного примера необходимо выполнить свое индивидуальное задание. Расчет ряда чисел осуществляется по другой формуле и отбираются элементы во вторую таблицу по другим критериям.

Лабораторная работа №6

Тема: Записи.

Задание: Создать структуру — типа запись состоящую из следующих элементов:

1 — Номер по порядку

2 — Фамилия

3 — Имя

4 — Отчество

5 — Год рождения

6 — Бал за физику

7 — Бал за математику

8 — Бал 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 в которой перечислены фамилия и инициалы абитуриентов у которых при поступлении по физике 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.

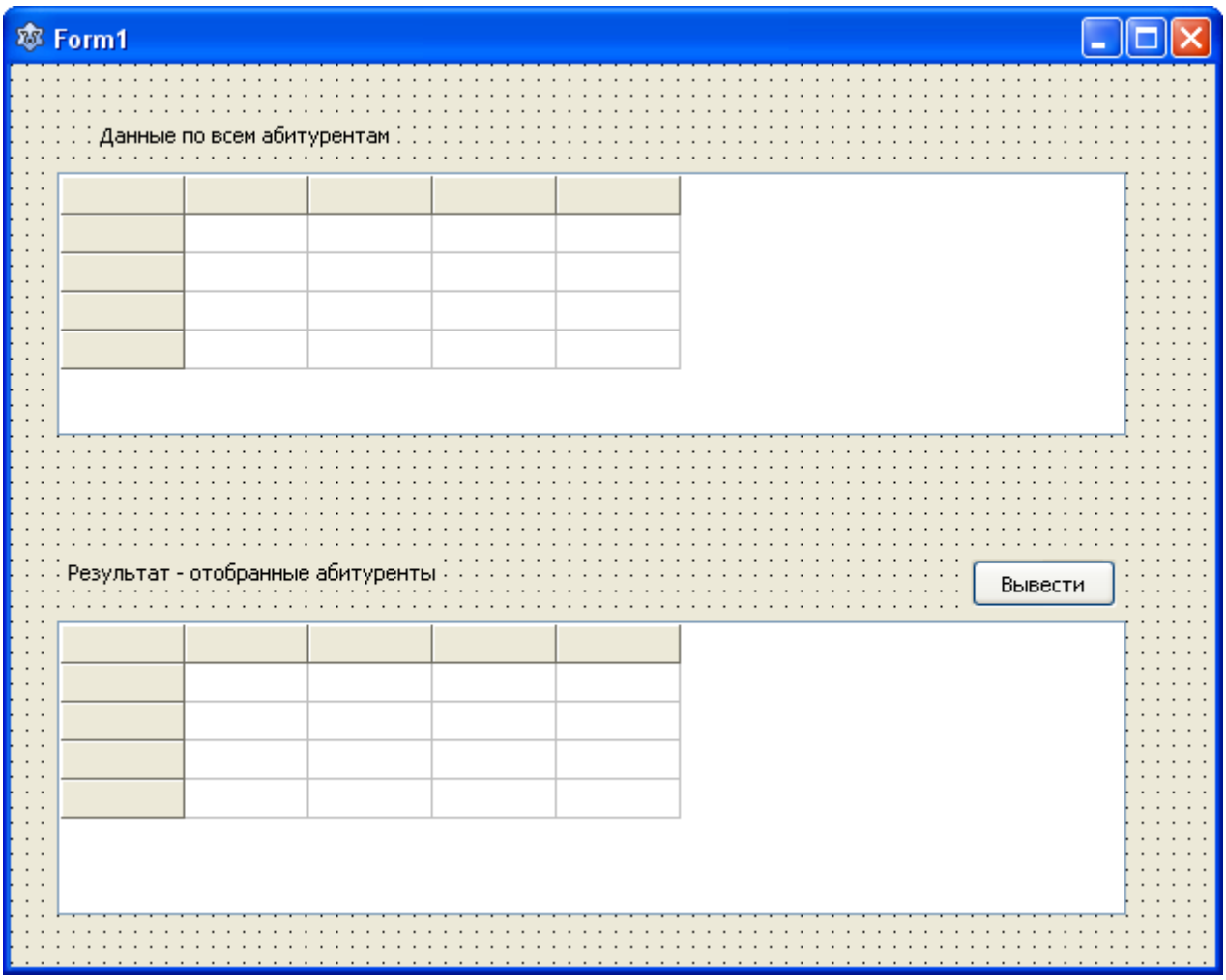


Рис 6.1 — Интерфейс программы.

Переименуем метку и кнопку так, как показано на рисунке 6.1.

Настроим таблицы установив свойства StringGrid1:

RowCount = 8;

ColCount = 11;

Options.goEditing = true; - разрешаем редактировать данные таблицы из окна программы.

В редакторе формы через контекстное меню у StringGrid1 «Редактировать StringGrid» вносим данные об абитуриентах и балы от 0 до 100.

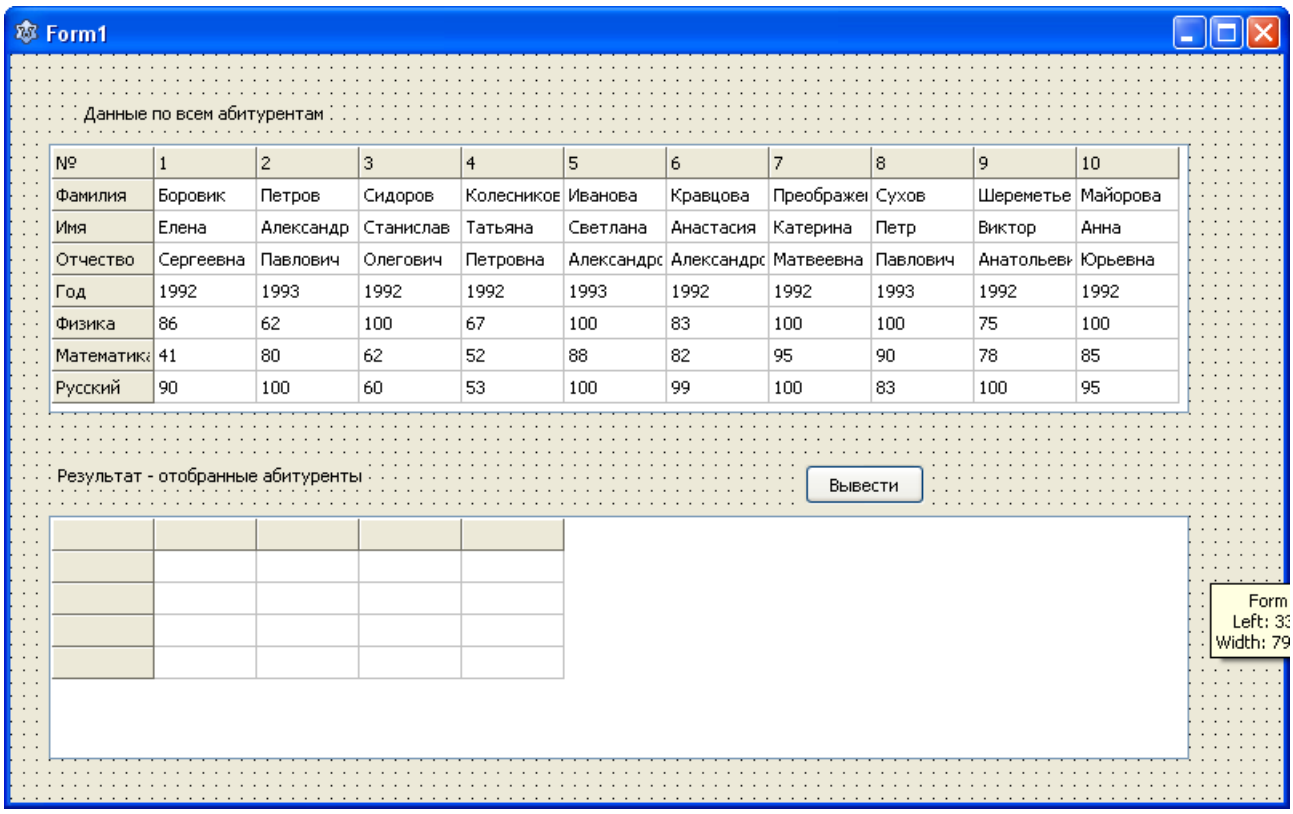


Рис 6.2 — Заполненная таблица исходных данных.

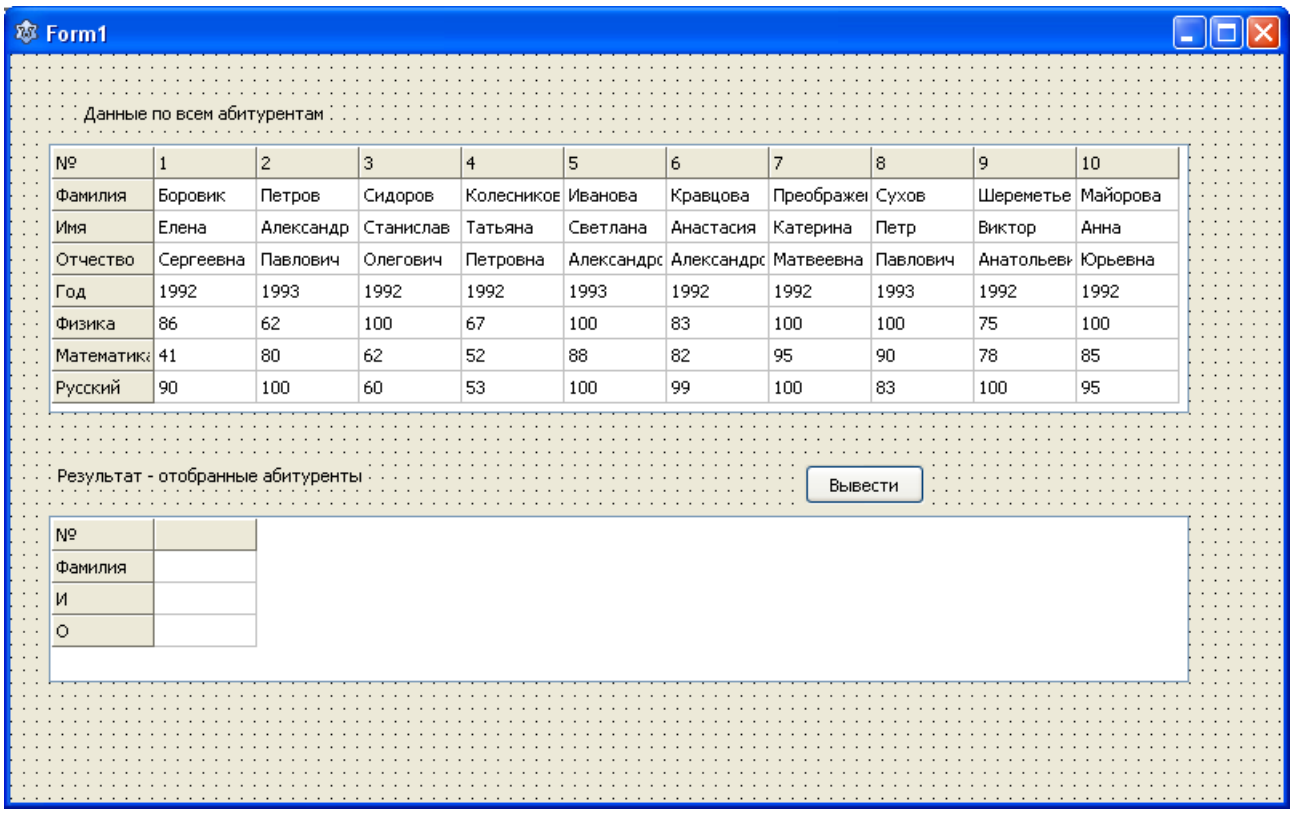
Настроим таблицу результатов (нижняя) установив значения свойств в инспекторе объектов:

RowCount = 4;

ColCount = 2;

В редакторе формы через контекстное меню stringGrid2 «Редактировать StringGrid» вносим первоначальные данные в шапку таблицы.

В результате форма имеет вид:



№	1	2	3	4	5	6	7	8	9	10
Фамилия	Боровик	Петров	Сидоров	Колесникое	Иванова	Кравцова	Преображе	Сухов	Шереметье	Майорова
Имя	Елена	Александр	Станислав	Татьяна	Светлана	Анастасия	Катерина	Петр	Виктор	Анна
Отчество	Сергеевна	Павлович	Олегович	Петровна	Александрс	Александрс	Матвеевна	Павлович	Анатольеви	Юрьевна
Год	1992	1993	1992	1992	1993	1992	1992	1993	1992	1992
Физика	86	62	100	67	100	83	100	100	75	100
Математика	41	80	62	52	88	82	95	90	78	85
Русский	90	100	60	53	100	99	100	83	100	95

№	
Фамилия	
И	
О	

Рис 6.3 — Окончательный дизайн формы — настроена таблица StringGrid2.

Из задания каждый абитуриент обладает набором данных, поэтому мы сгруппируем их в отдельную структуру запись и назовем ее TAbiturient. Опишем ее в разделе type, т. к. здесь описываются новые типы данных. 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, а через «:» его тип данных. Под Фамилию, Имя, Отчество — выбрали строковые поля, а под год рождения и оценки целые.

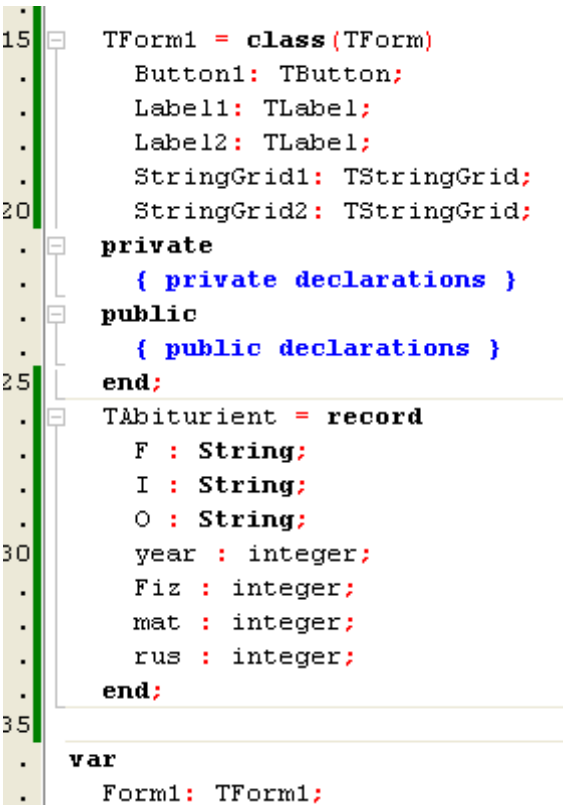


Рис 6.4 — Объявление структуры TAbiturient в разделе type.

Описывать новый тип данных необходимо после описания формы и до описания глобальных переменных в разделе var.

Данная структура описывает данные только для одного абитуриента, а по заданию их 10. Что бы не создавать 10 переменных — создадим массив, т. к. каждый абитуриент имеет однотипные персональные данные.

```
35  TMas = array [1..10] of TAbiturient;
```

Рис 6.5 — Объявление типа массива абитуриентов.

На данный момент создано два новых типа данных, что бы ObjectPascal выделил под них область данных необходимо определить переменные соответствующих типов данных. Нам понадобится два массива абитуриентов. В первом мы будем хранить всех абитуриентов, а во втором только тех, которые удовлетворяют критериям (отобранные абитуриенты).

Код представлен ниже.

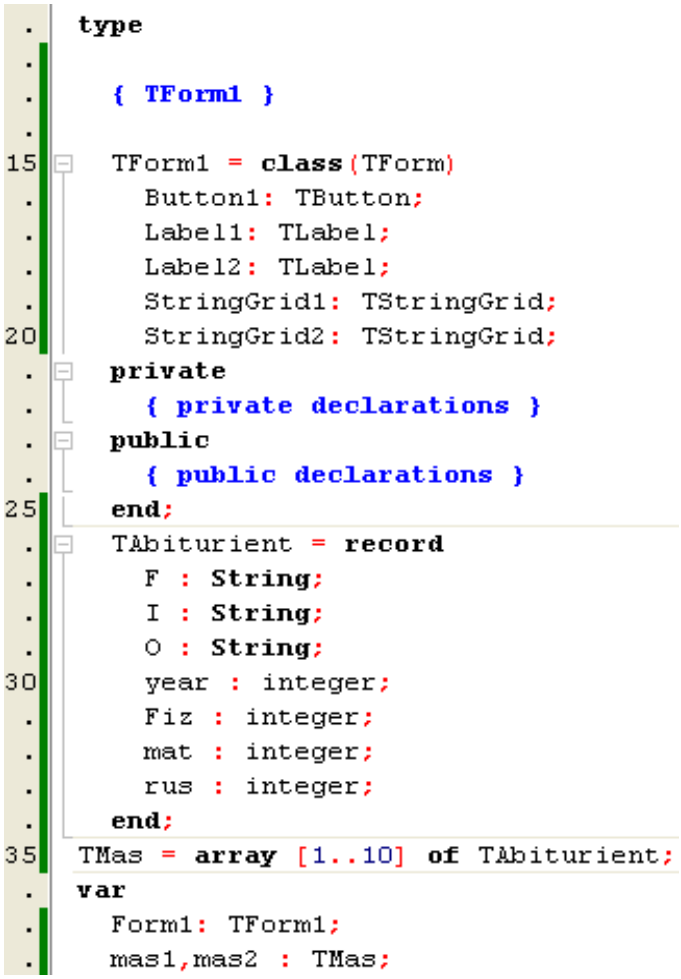


Рис 6.6 — Объявление типов: структуры TAbiturient и массива TMas; Определение переменных массивов mas1 и mas2.

Опишем обработчик кнопки «Вывести». Он выполняет:

1. Перенос данных StringGrid1 в mas1;
2. Из mas1 выбираем подходящих абитуриентов и копирует их в mas2;
3. Запоминаем количество скопированных элементов в mas2;
4. Выдает с первого по количество запомненных элементов из mas2 в StringGrid2 — искомые данные.

Для выдачи только один символа от имени и отчества (инициалы) воспользуемся функциями AnsiToUtf8(Utf8ToAnsi(mas2[i].I)[1]+'.'); Строки в среде программирования Lazarus хранятся в формате Utf8 — под каждый символ выделяется не один, а несколько байт (1-4 байта в зависимости к какому из национальных шрифтов символ принадлежит — подмножество Unicode). Переводим его в формат Ansi – отделяем первый символ, дописываем точку и

обратно преобразовываем в Utf8 формат.

Код обработчика представлен ниже:

```
45  procedure TForm1.Button1Click(Sender: TObject);
    var
      i : integer;
      n : integer;
    begin
50    for i:=1 to 10 do
      begin
        mas1[i].F:=StringGrid1.Cells[i,1];
        mas1[i].I:=StringGrid1.Cells[i,2];
        mas1[i].O:=StringGrid1.Cells[i,3];
55      mas1[i].year := StrToInt(StringGrid1.Cells[i,4]);
        mas1[i].Fiz := StrToInt(StringGrid1.Cells[i,5]);
        mas1[i].mat := StrToInt(StringGrid1.Cells[i,6]);
        mas1[i].rus := StrToInt(StringGrid1.Cells[i,7]);
      end;
60    N := 0;
      for i := 1 to 10 do
        if mas1[i].Fiz = 100 then
        begin
          N := N + 1;
65        mas2[N] := Mas1[i];
        end;
      StringGrid2.ColCount:= N+1;
      for i := 1 to N do
      begin
70      StringGrid2.Cells[i,0] := IntToStr(i);
        StringGrid2.Cells[i,1] := mas2[i].F;
        StringGrid2.Cells[i,2] := AnsiToUtf8(Utf8ToAnsi(mas2[i].I)[1]+'.');
        StringGrid2.Cells[i,3] := AnsiToUtf8(Utf8ToAnsi(mas2[i].O)[1]+'.');
      end;
75  end;
```

Рис 6.7 — код обработчика кнопки «вывести».

Запускаем программу на выполнение и получим примерно следующий результат.

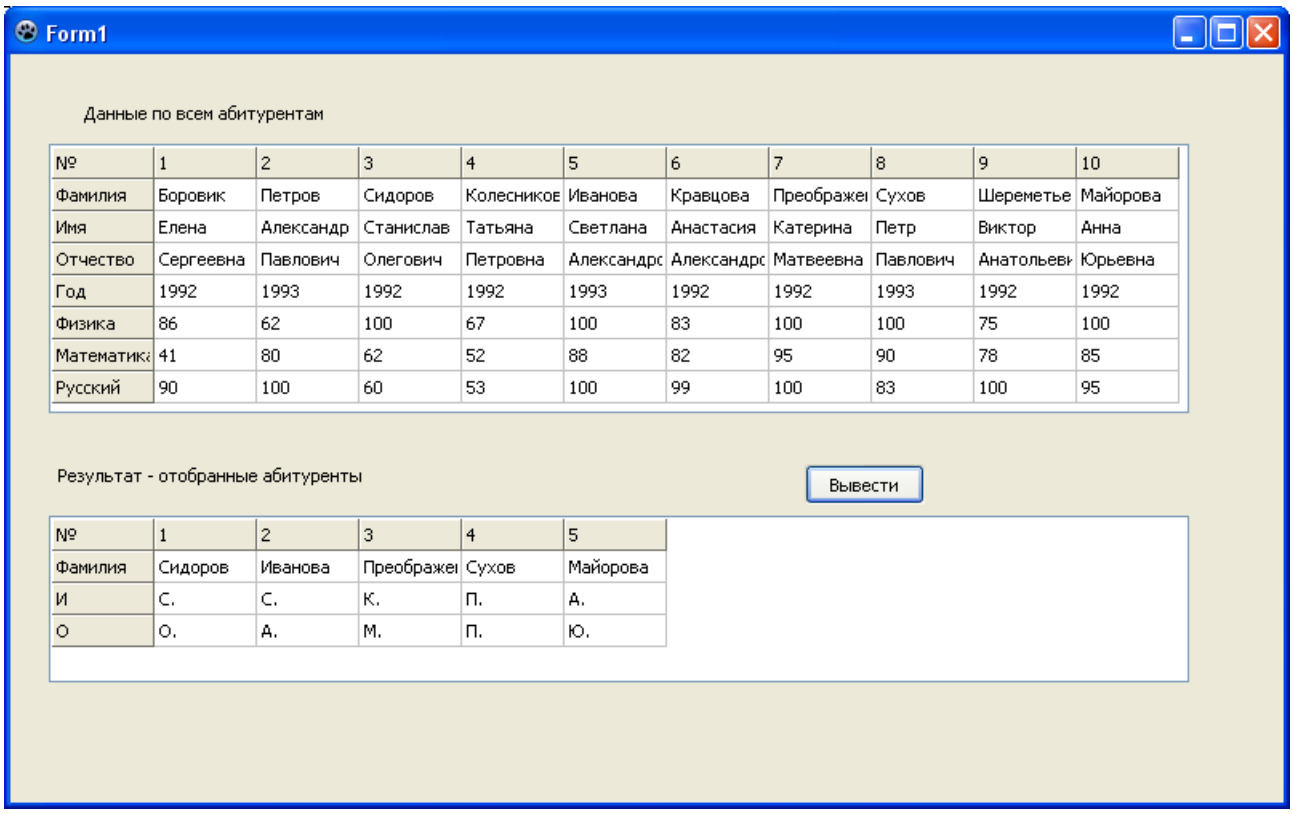


Рис 6.8 — Результат выполнения программы — абитуриенты удовлетворяющие условию.

Программа закончена. На основе данного примера необходимо создать индивидуальную работу. Необходимо заполнить массив абитуриентов индивидуальными данными и выдать во вторую таблицу только тех, которые удовлетворяют условию вашего индивидуального задания. Интерфейс программы меняется минимально (только вторая таблица — количество и наименование строк), а для вариантов со средними значениями — можно вывести и сами средние значения на форму.

Лабораторная работа №7

Тема: Двумерные массивы

Задание: Создать двухмерный массив (матрица) размером 4x4 элементов, заполнить его и выдать разницу между суммами главной и побочной диагоналей.

Разработаем интерфейс программы.

Для отображения двумерных массивов (матриц) можно использовать также компонент TStringGrid (Строковая таблица). Внешний вид формы представлен на рисунке 7.1.

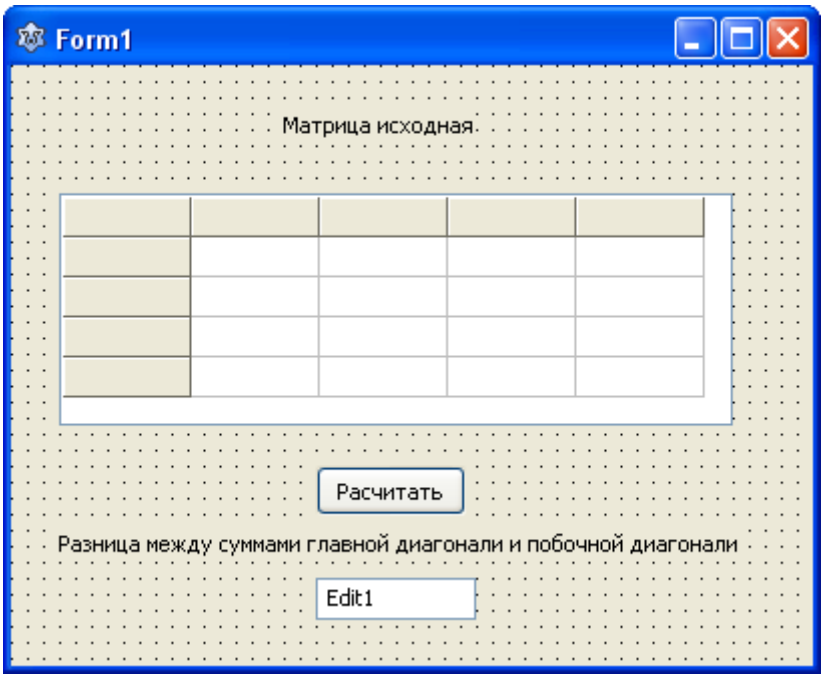


Рис 7.1 — Интерфейс программы.

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 Edit1 с компонента Edit1 (свойство Text). В результате получим форму.

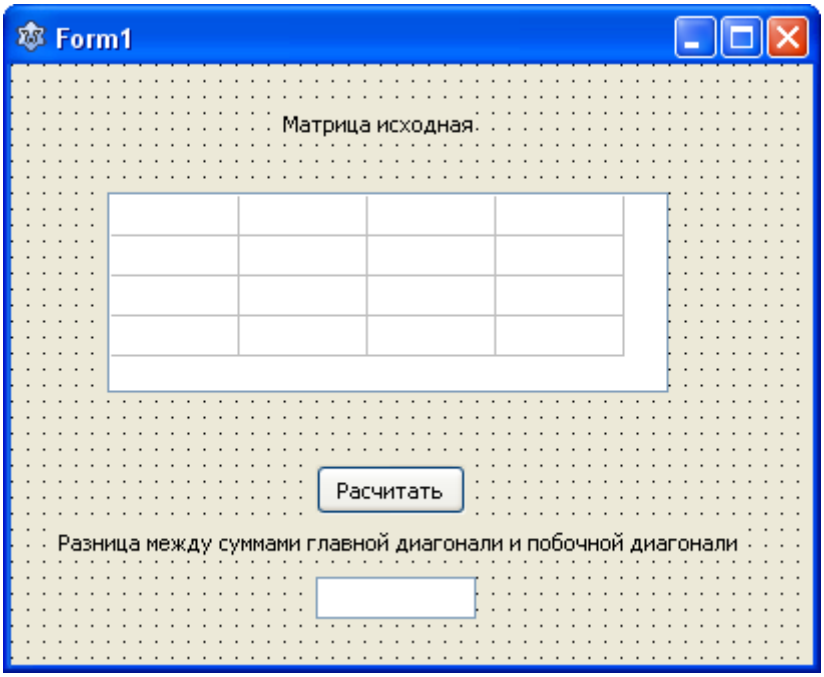


Рис 7.2 — Настройка компонентов интерфейса.

Сохраним проект и запустим на выполнение. Проверим — возможно ли вводить данные в таблицу.

После запуска программы введем несколько значений в таблицу:

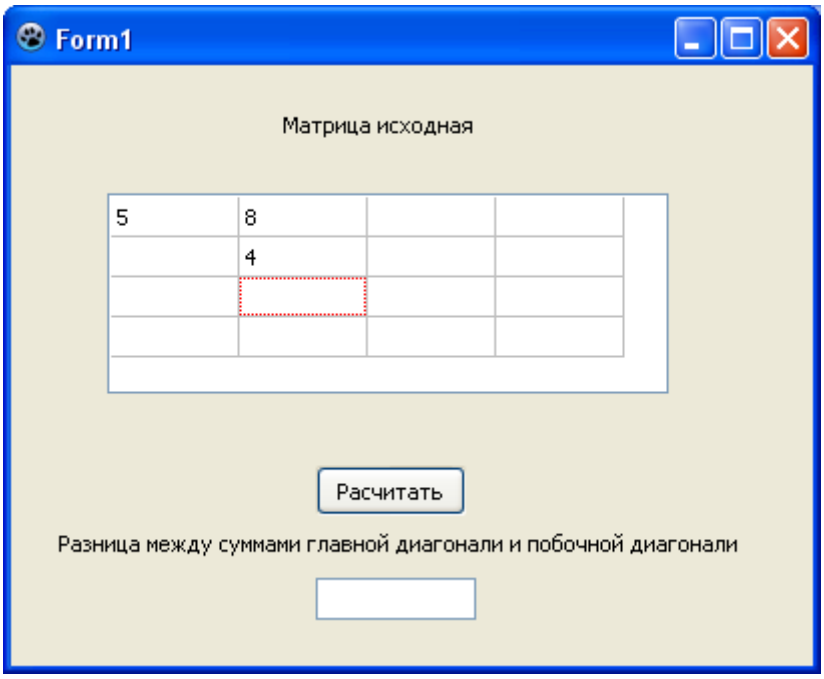


Рис 7.3 — Ввод данных в таблицу при выполнении программы.

Разработаем структуру данных для хранения двумерного массива — переменную опишем за формой (Form1) в глобальном разделе описания

переменных формы var.

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

Рис 7.4 — Описание массива.

Для создания двумерного массива необходимо указать диапазоны его индексов в квадрантных скобках через запятую. Если необходим 3-х, 4-х мерный массив то индексы (диапазоны индексов) указываются через запятую в квадратных скобках. Следует помнить, что с увеличением количество индексов резко возрастает количество элементов в матрице. Количество элементов равна произведению всех диапазонов. Если это произведение умножить на размер одного элемента в байтах — то можно рассчитать сколько потребуется оперативной памяти для хранения всей матрицы. В нашем случае 4x4x6 байт= 96 байт.

Переменная для хранения данных (двумерного массива вещественных чисел) готова. Необходимо написать процедуру рассчитывающую разницу между суммами главной и побочной диагоналей.

Создадим обработчик на кнопку «Рассчитать»:

```
{ 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;
```

Рис 7.5 — Обработчик кнопки «Рассчитать».

Для расчета результата — введем вспомогательные переменные:

i,j — индексы массивов — целые.

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 диагонали. Переменной R – присваиваем разность сумм диагоналей и выводим ее в поле данных Edit1.

Сохраняем проект и запускаем его на выполнение, проверяем его работоспособность. Пример работы программы представлен на рисунке.

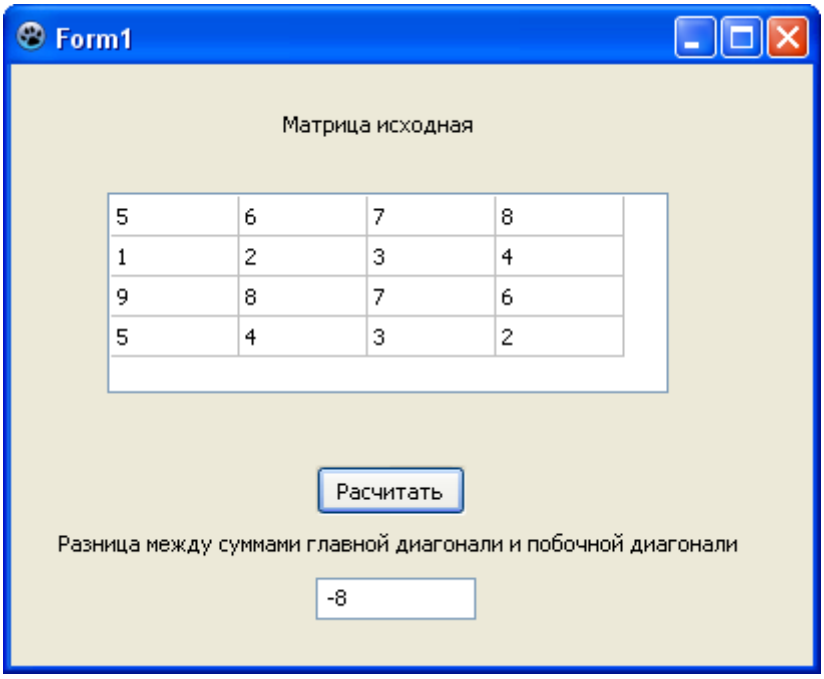


Рис 7.6 — Результат выполнения программы.

Программа готова. На основе данного примера необходимо выполнить свое индивидуальное задание, интерфейс программы меняется минимально.