



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

методическое пособие

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

Новочеркасск 2012

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 2013, 79 стр.

Цель методического пособия — ознакомить студентов с принципами написания программ на языке Object Pascal в среде программирования Lazarus применительно к строительным специальностям. В методическом пособии детально рассмотрены этапы программирования от элементарных программ, до применяемых в расчетах металлических конструкций зданий и сооружений. Дается общее представление методов реализации математических и физических моделей в Lazarus. Данный материал помогает освоить изучение курса «Информатика» и при выполнении курсового проекта, студентами строительных специальностей.

Оглавление

Общая информация о среде программирования Lazarus.....	3
Лабораторная работа № 1.....	8
Блок схемы.....	22
Лабораторная работа №2.....	26
Лабораторная работа №3.....	40
Лабораторная работа №4.....	49
Лабораторная работа №5.....	53
Лабораторная работа №6.....	59
Лабораторная работа №7.....	66
Лабораторная работа №8.....	70
Список рекомендованной литературы.....	78
Интернет ресурсы:.....	79

Общая информация о среде программирования Lazarus

Запуск программы Lazarus выбираем «Пуск→Все программы→Lazarus →Lazarus». При отсутствии дистрибутива, последнюю версию можно взять с официального сайта. Lazarus распространяется по лицензии GPL/LGPL - бесплатный для некоммерческого использования.

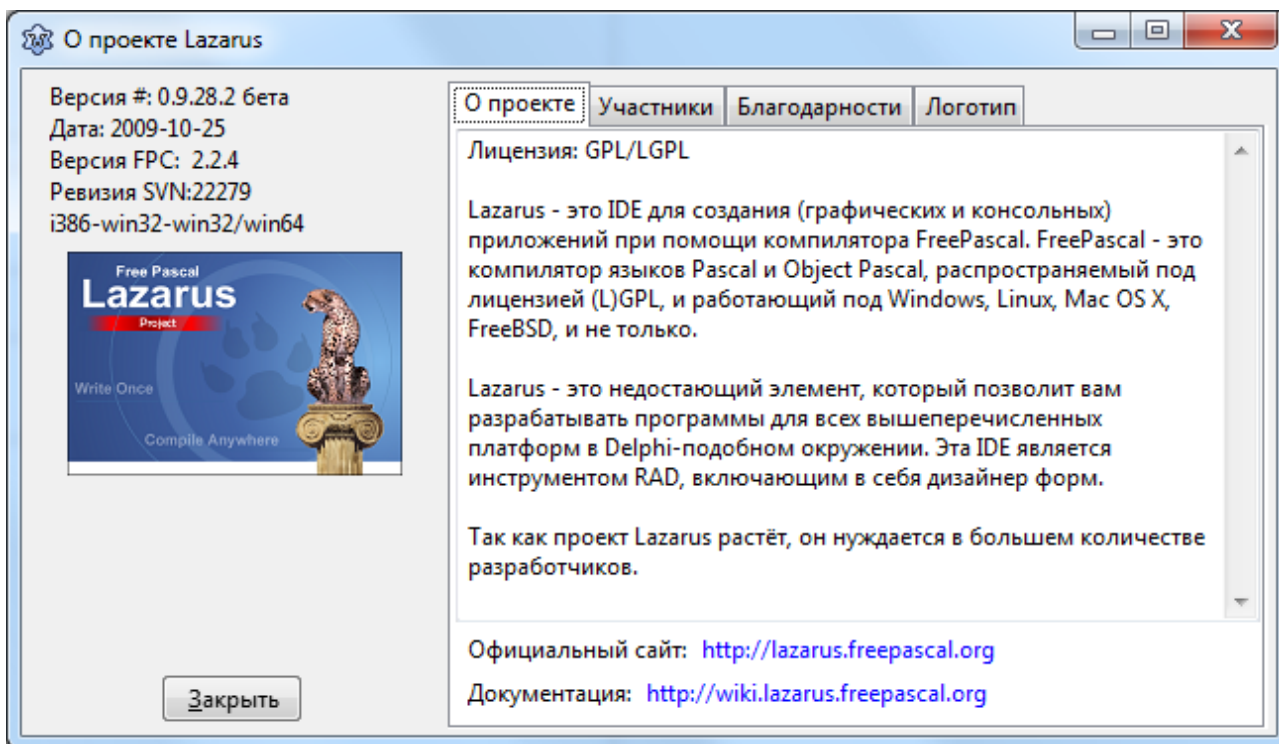


Рис. 1 - Проект Lazarus и официальный сайт

После запуска программы появятся несколько окон рис 2:

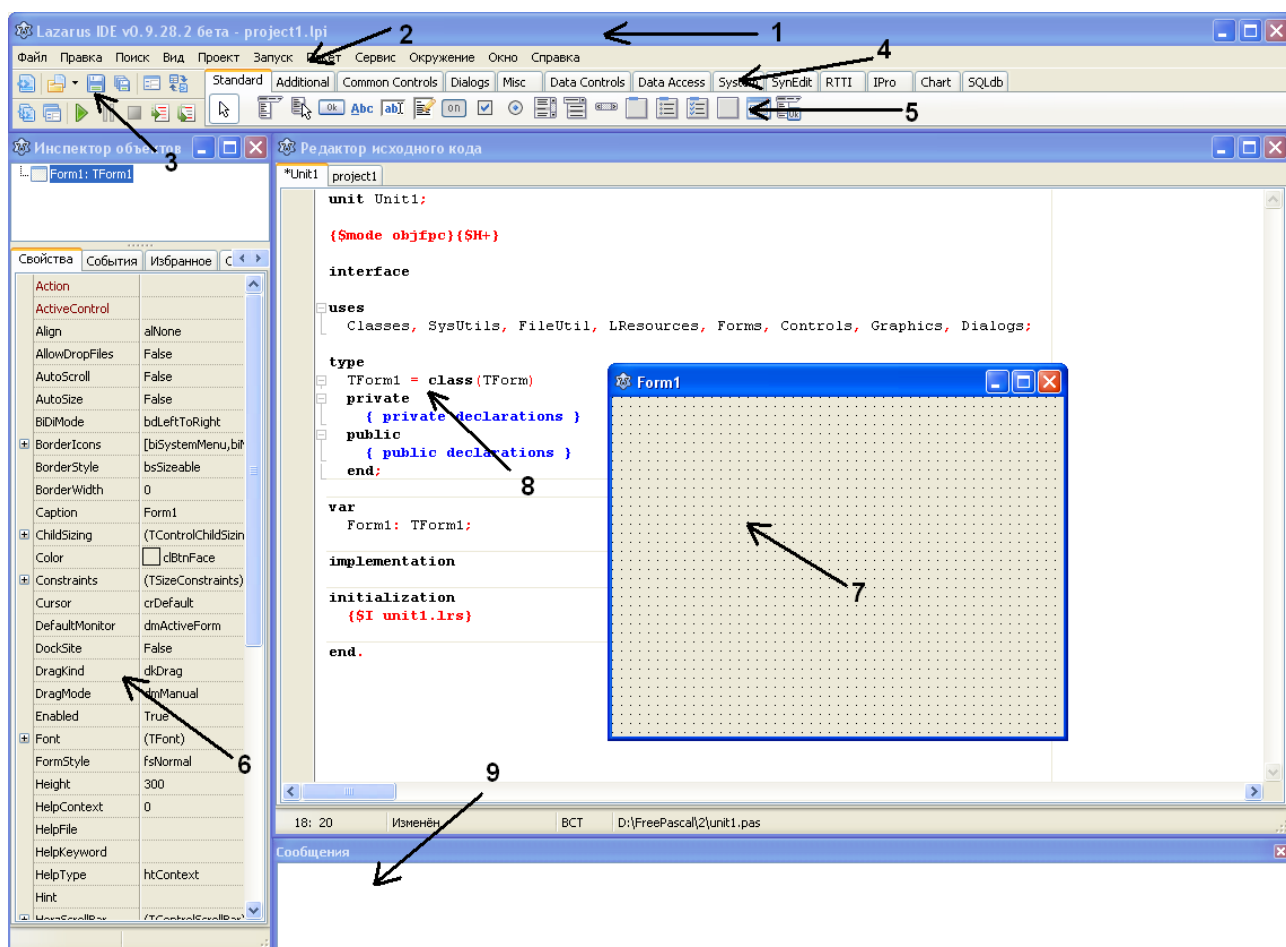


Рис. 2 - Основные окна среды программирования Lazarus

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».

Управления проектом:

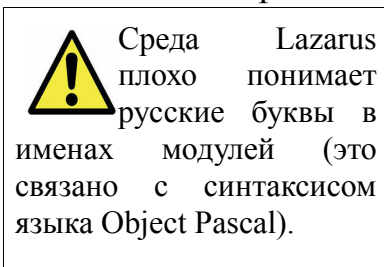
Назначение кнопок управление проектом рис 3.

Верхний ряд: слева — направо:

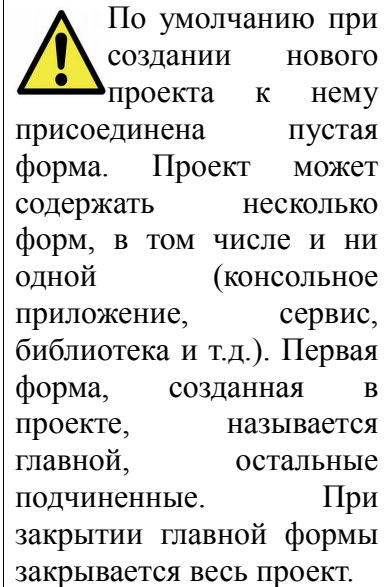
1. Создать модуль — создает новый модуль форму.

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.



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



5. Создать форму — создает новую форму программы и прикрепляет ее к проекту.

6. Переключить форму/модуль — осуществляет переход между окном формы и окном модуля (рис 1.3). Можно переключаться с помощью клавиши <F12>.

Нижний ряд кнопок слева — направо:

1. Показать модули — отображает список всех модулей проекта и позволяет переключиться на любой из них;

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

3. Запуск проекта — осуществляет сборку

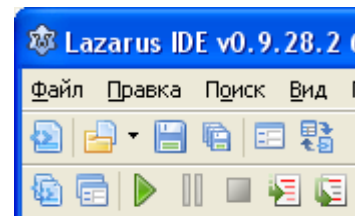
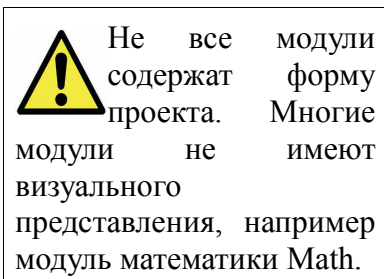


Рис. 3 - Основные кнопки управления проектом



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 — работу запущенного проекта можно приостановить — поставить на «паузу».

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



Можно зайти как в процедуры своего проекта, так и библиотечные функции Lazarus, но в системные процедуры Windows (API Windows) попасть не получится.

6. Шаг со входом — отладка проекта пошаговая с заходом во все процедуры и последовательная отладка этих процедур;

7. Шаг в обход — пошаговая отладка проекта без отладки вызываемых процедур и функций, каждая вызываемая процедура выполняется за один шаг

отладки;

Закладки компонентов — содержат наборы различных компонентов сгруппированные по различным признакам. В примерах использоваться



Выбранные компоненты имеют по краям черные прямоугольные маркеры, за которые можно перемещать объект по форме и изменять его размеры. Некоторые компоненты можно только перемещать.

компоненты из вкладок «Standard» и «Additional».

Компоненты — готовые настраиваемые блоки программ, которые можно устанавливать на форму и подключать к модулям. Для установки компонента на форму, необходимо один раз «кликнуть» на нем в панели компонентов, затем второй раз на форме — в том месте, где необходимо его разместить. Установленные



Свойства, как и типы данных, могут быть: целыми, вещественными, строковыми, логическими, множествами или сложными. События у компонентов создаются пустыми.

компоненты можно выбирать «кликнув» по нему или с помощью рамки выделения.

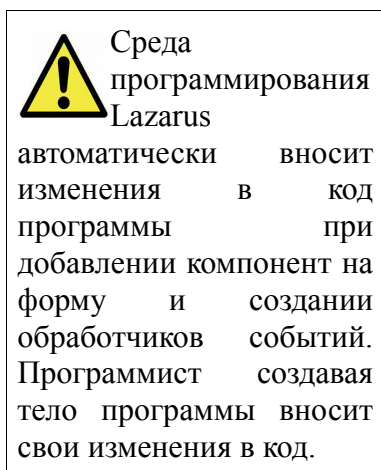
Инспектор объектов — форма позволяющая настроить свойства каждого компонента индивидуально. Инспектор объектов имеет несколько закладок.

Закладка «Свойство» отображает большинство

свойств объекта, хотя и не все. Свойство объекта это имя и значение. Левый столбец — имя свойства, правый — значение.

Закладка «События» — позволяет посмотреть список большинства событий, на которые может реагировать компонент, а также процедуры привязанные к каждому из событий.

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



Модуль программы — окно в которой содержится исходный код на языке Object Pascal.

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

Ошибки могут быть:

1. Синтаксические — когда 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.

Проект рекомендуется сохранять в индивидуальную папку, (проект — группа связанных файлов).

Для каждого проекта рекомендуется создавать следующую структуру:

Для доступа к проекту по сети разместим в папке общие документы (Мой компьютер → Общие документы), создадим папку с номером вашей группы (например «СФ 1-1а»), далее ваша фамилия (например «Иванов»), далее «1» для первой лабораторной, для второй «2» и т. д.

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 создания пустого проекта в среде программирования Lazarus выберем «Файл → Создать...» рис 1.1.

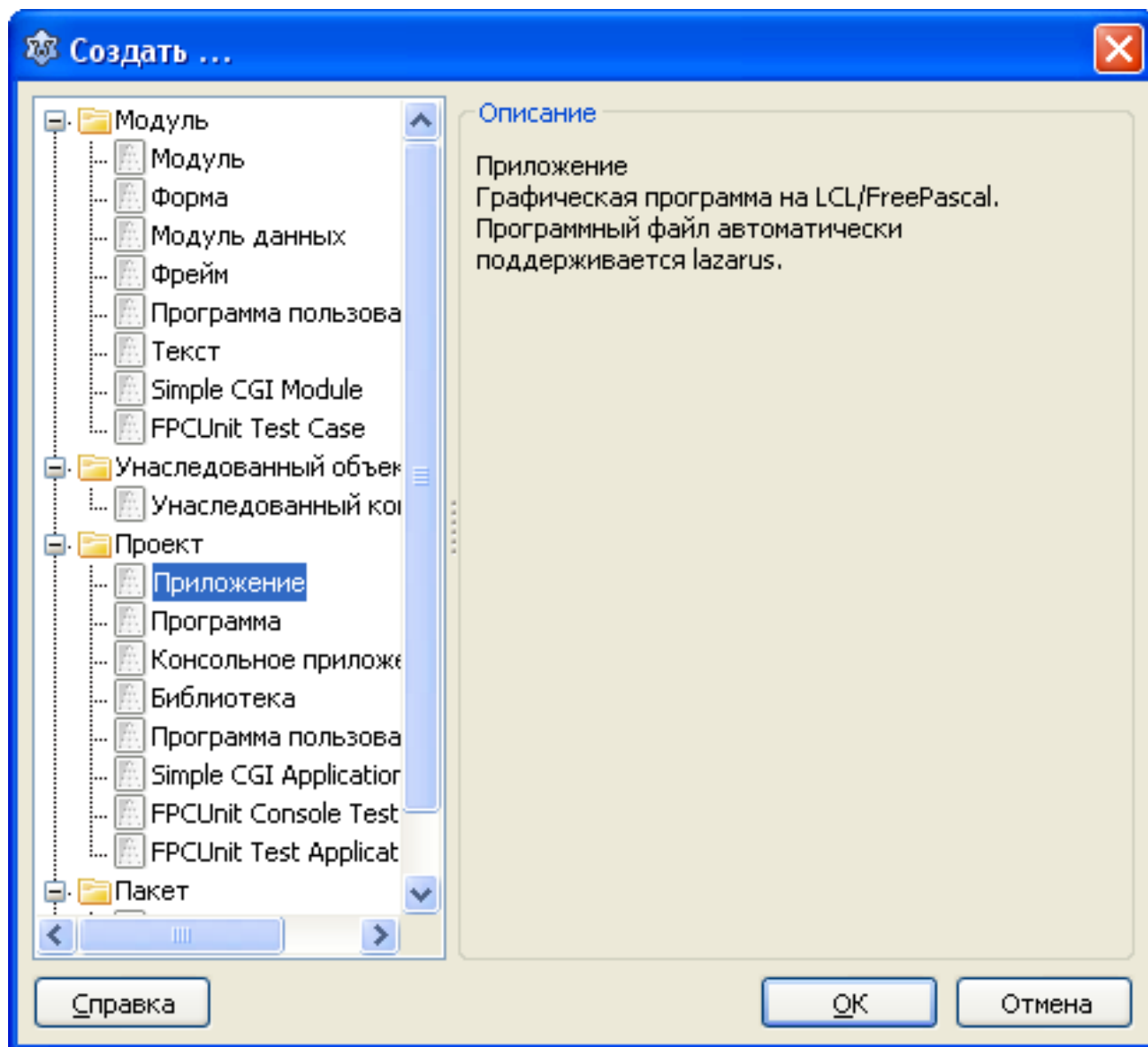


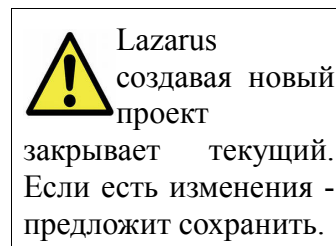
Рис. 1.1 - Создание нового проекта

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

➔ Нажмем кнопку «Ok».

➔ Сохраним пустой проект — «Файл → Сохранить все». Программа предлагает указать папку и имя файла.

➔ Нажимаем на кнопку «Сохранить».



Программа предложит (рис 1.2) преобразовать имя файла в нижний регистр — соглашаемся.

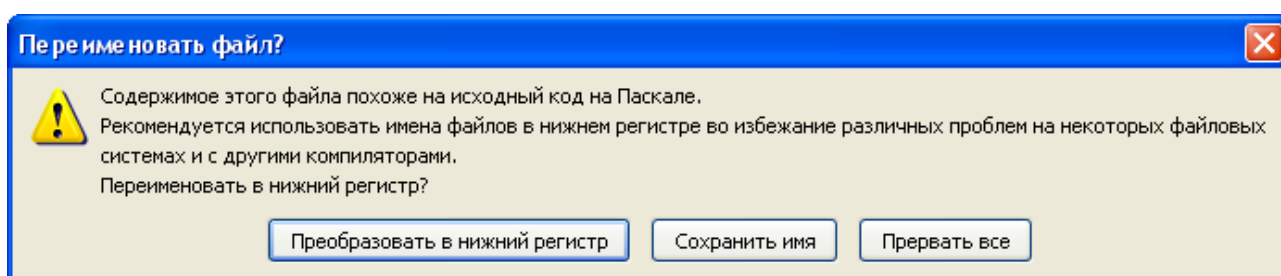


Рис. 1.2 - Предупреждение о преобразовании имени файла в нижний регистр

➔ Сохраняем проект в ту же папку.

Иконки «Сохранить» и «Сохранить все» становятся серыми (рис. 1.3), когда все изменения во всех модулях записаны на диск.

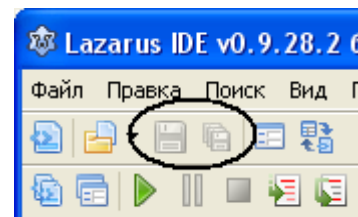
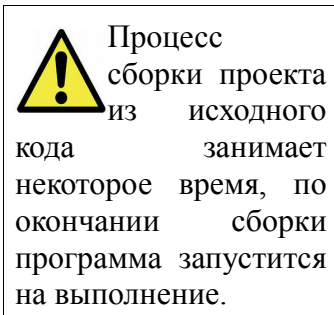


Рис. 1.3 - Проект сохранен на диске

➔ Запустим проект — для

этого нажмем на кнопку с зеленым треугольником.

На экране отобразилась форма (рис. 1.4), без точек. В панели программ Windows появилась новая программа с именем Project1.



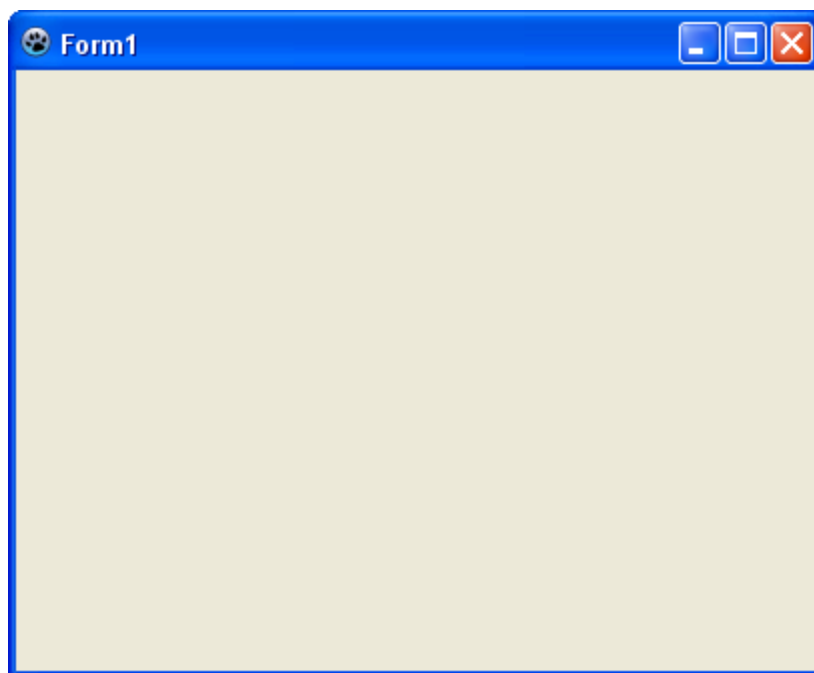


Рис. 1.4 - Пустая форма программы

➡ Закрываем программу (крестик в правом верхнем углу) и возвращаемся в среду программирования Lazarus.

После закрытия проекта появится окно (рис. 1.5) с информацией, что проект остановлен.

Вносить изменения в исходный код проекта рекомендуется только тогда, когда выполнение программы остановлено.

Отобразим форму программы — для этого нажмем «F12».

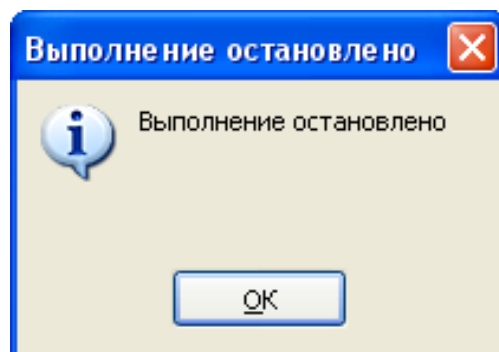


Рис. 1.5 - Предупреждение — программа остановлена

➡ Поместим на форму компонент «TMainMenu». Данный компонент находится на закладке «Standard» — самый первый (рис. 1.6).

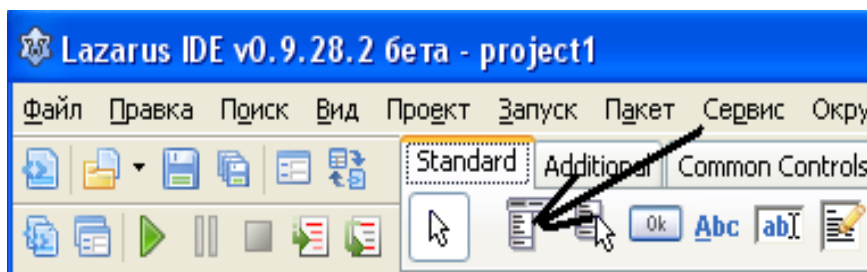


Рис. 1.6 - Выбор компонента из палитры компонентов

На форме отобразится иконка этого компонента (рис. 1.7).

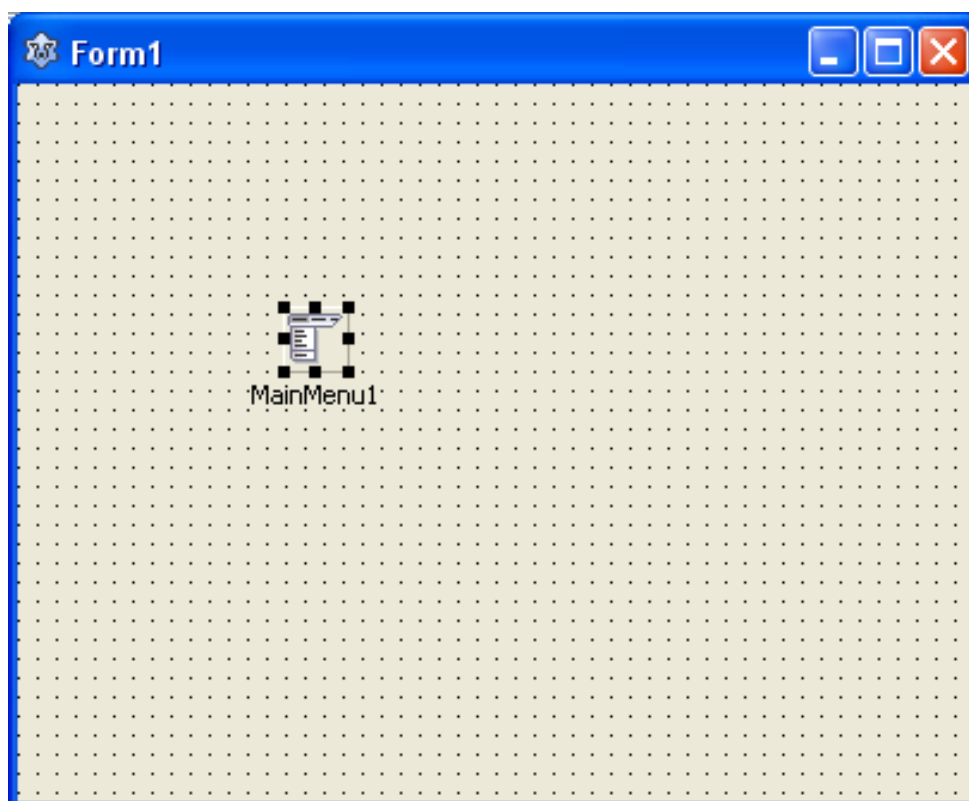


Рис. 1.7 - Установка компонента на форму

Компонент «TmainMenu» предназначен для создания главного меню программы.

➡ Откроем редактор главного меню (рис. 1.8): сделаем «двойной клик» на иконке главного меню, которая находится на форме программы (Ее только что добавили на форму).

Откроется «редактор меню»:

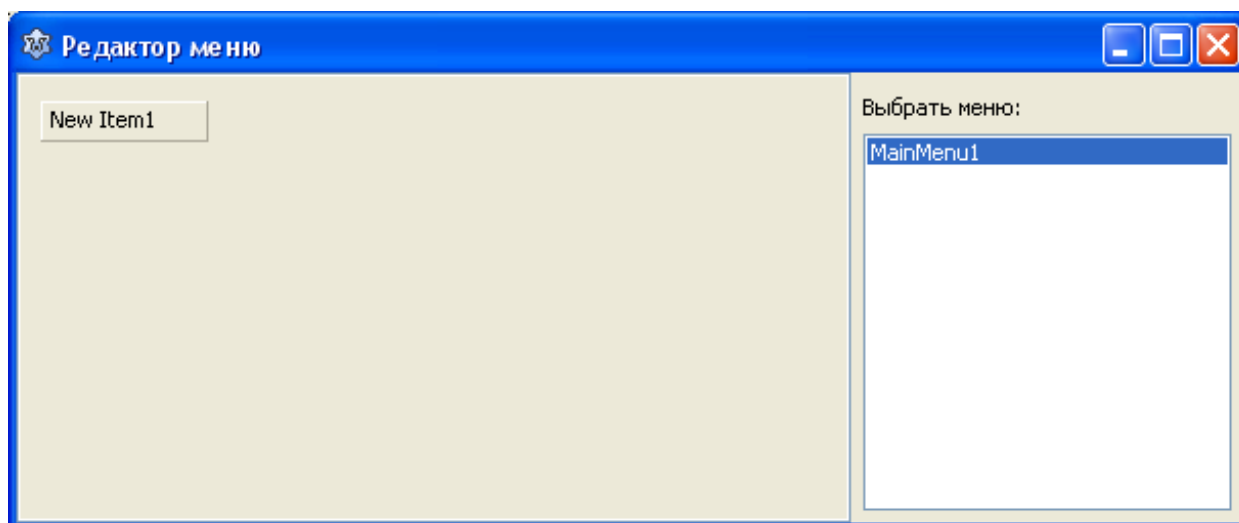


Рис. 1.8 - Внешний вид первого элемента меню в «Редакторе меню»

После открытия «Редактора меню» будет автоматически создан первый элемент меню с именем «New Item1». Переименуем его в «Файл» (рис. 1.9). 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».

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. 1.10).

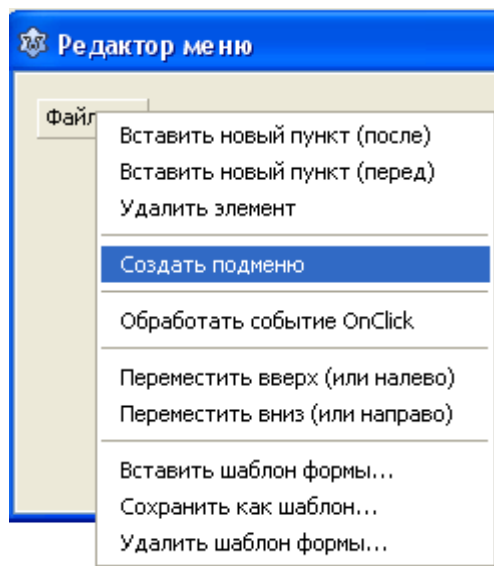


Рис. 1.10 - Создание подменю «Выход» для меню «Файл»

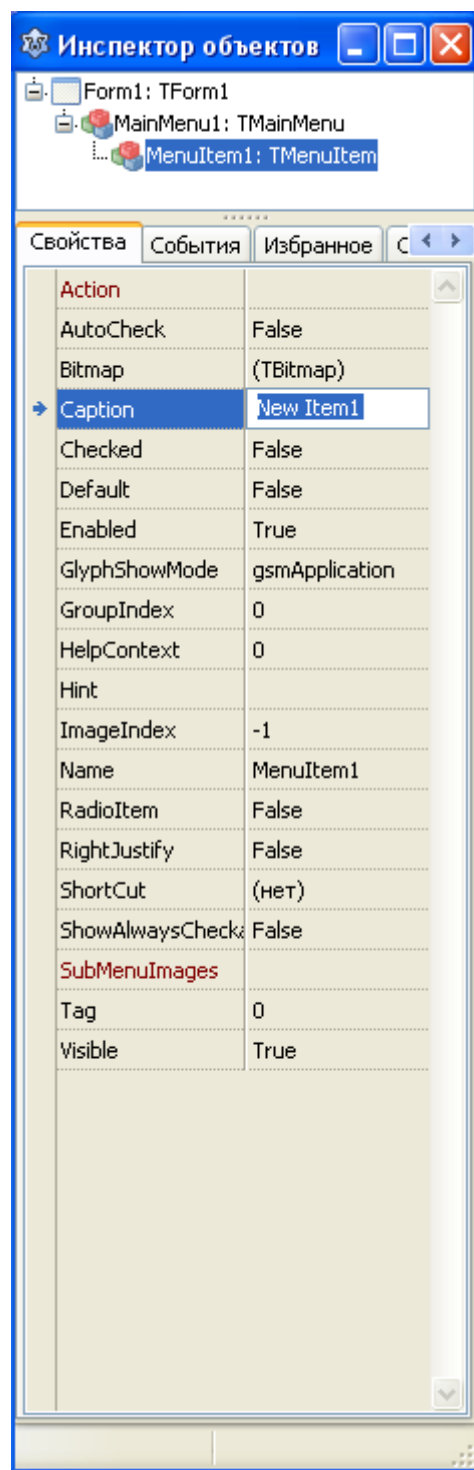


Рис. 1.9 - Переименование элемента меню из «New Item1» в «Файл»

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)» (рис. 1.11) - создадим новый элемент меню правее элемента «Файл».

➡ Переименуем его в «Вычислить».

Аналогично создаем пункты меню «Помощь» и «О программе» как показано на рис 1.12.

➡ Закрываем редактор меню (крестик у окна «Редактор меню» в правом верхнем углу), сохраняем проект и запускаем его на выполнение.

Проконтролируем наличие главного меню на форме запущенной программы.

Рекомендуется сохранять и запускать на выполнение программу при каждом существенном изменении программы, контролируя сделанные изменения.

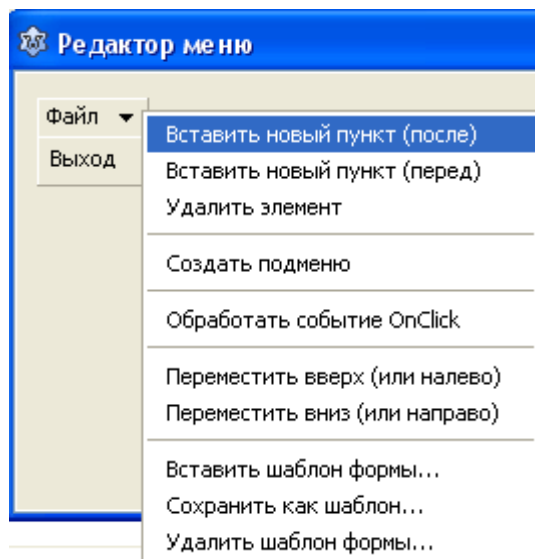


Рис. 1.11 - Создание меню «Вычислить»

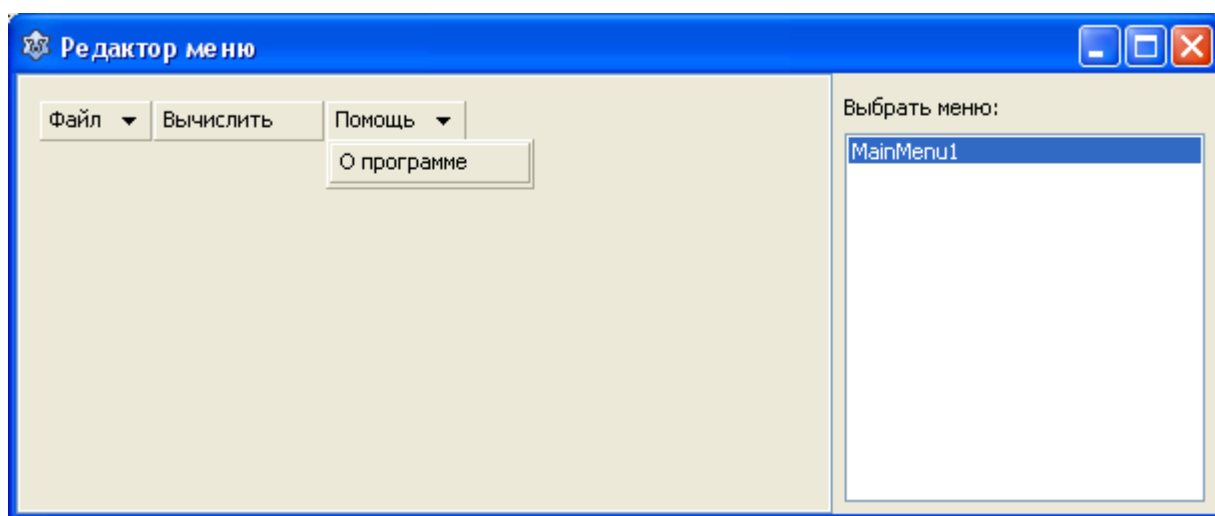


Рис. 1.12 - Создание остальных пунктов меню

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. 1.13:

3 компонента «TEdit»;

5 компонентов

«TLabel»;

1 компонент

«TButton»;

➡ Последовательно выберем каждый выставленный компонент «TLabel» и изменять у него свойство «Caption» в инспекторе

➡ Изменим это свойство у кнопки «Botton1».

В результате получим форму рис. 1.14.

➡ Выберем последовательно каждый компонент «Edit» и в «Инспекторе объектов» изменим свойство «Text» на «5», «7» и «<». В результате получим форму рис. 1.15.

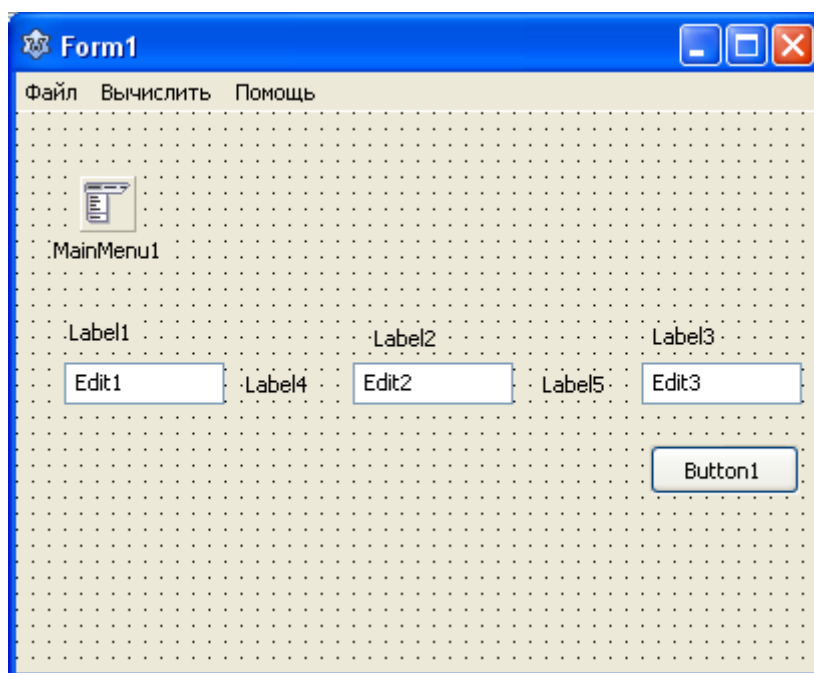


Рис. 1.13 - Расположение компонентов на форме

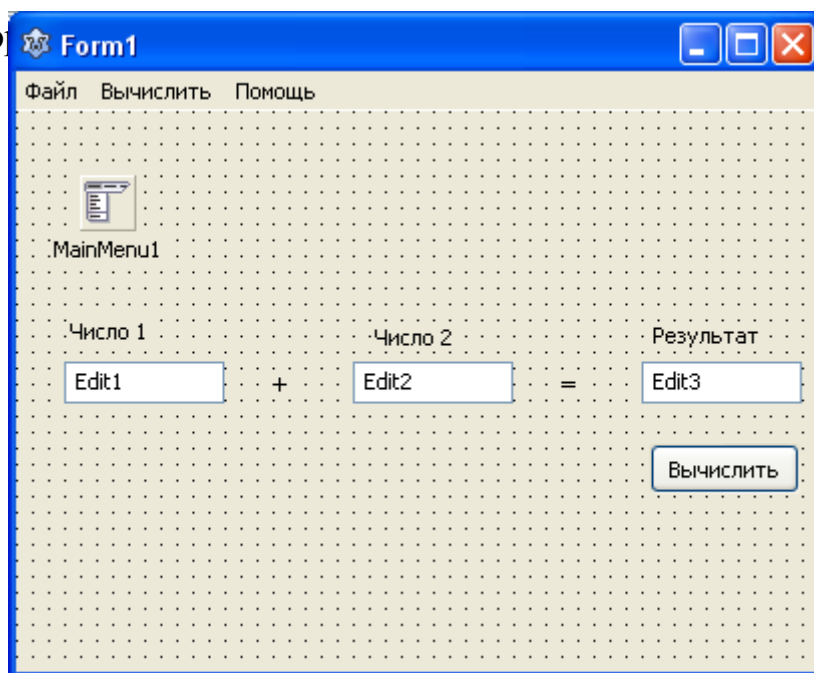


Рис. 1.14 - Переименование «Меток» и «Кнопки»

➡ Сохраним проект и запустим его на выполнение.

Форма изменит свой вид как на рисунке рис. 1.16.

На данный момент разработан полностью интерфейс программы, но пока нет функционального наполнения — программа не считает.

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

➡ Возвращаемся в среду программирования Lazarus, отображаем форму программы и выбираем пункт меню «Файл» → «Выход» (рис. 1.17).

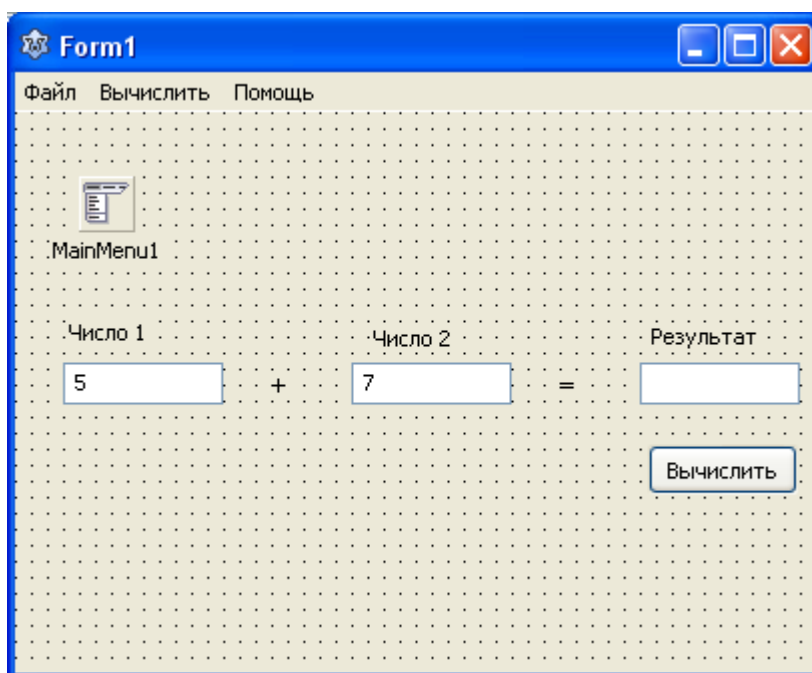


Рис. 1.15 - Изменение свойства у элемента «TEdit»

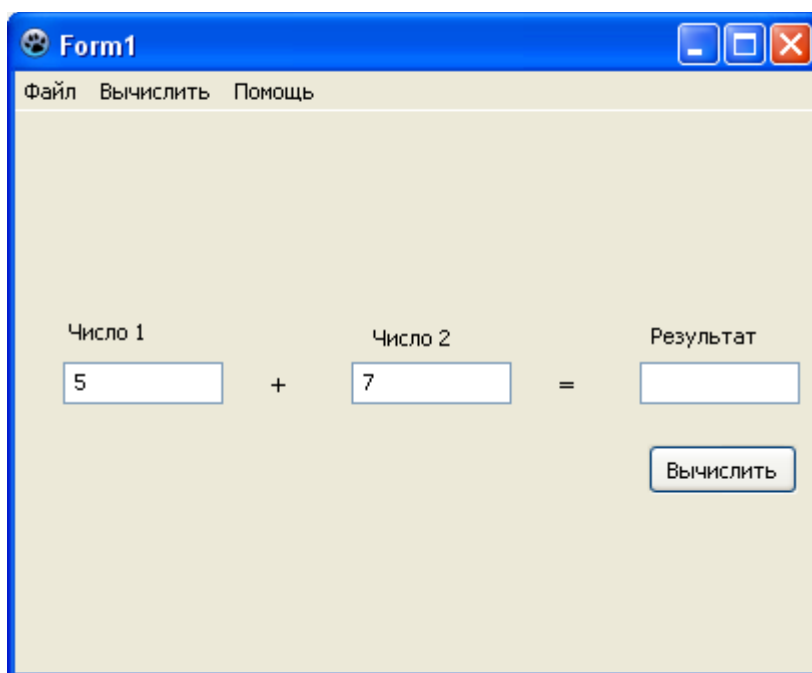


Рис. 1.16 - Внешний вид формы после сборки проекта

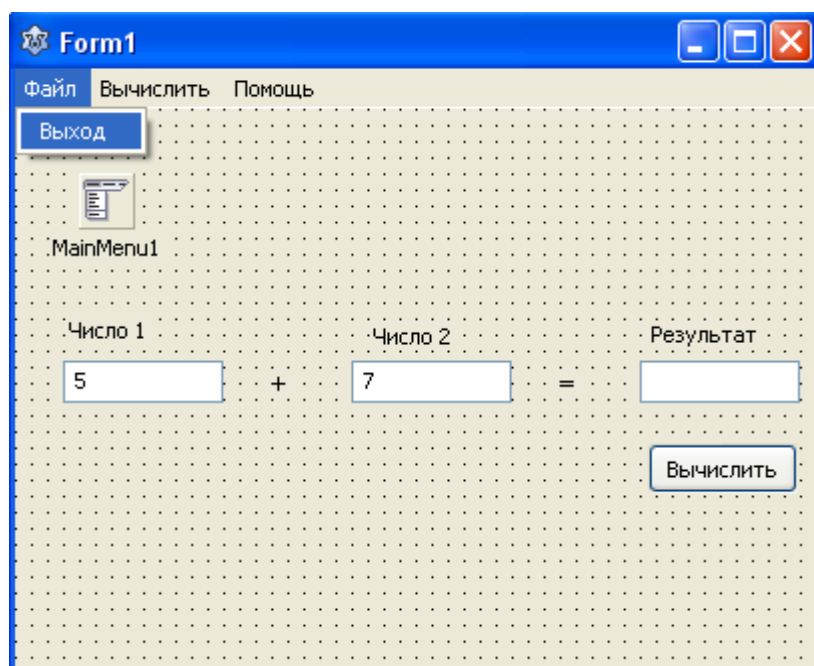


Рис. 1.17 - Привязка обработчика к меню «Выход»

В результате форма программы уйдет на задний план, а на передний план покажется «Редактор исходного кода» с заготовкой события (рис. 1.18).

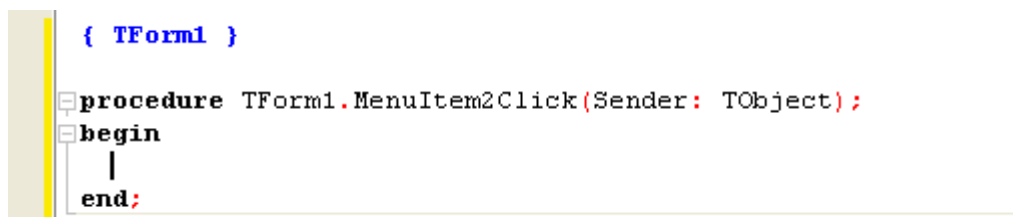


Рис. 1.18 - Заготовка процедуры для события «Выход»

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 рис. 1.19 в конце поставив «;».

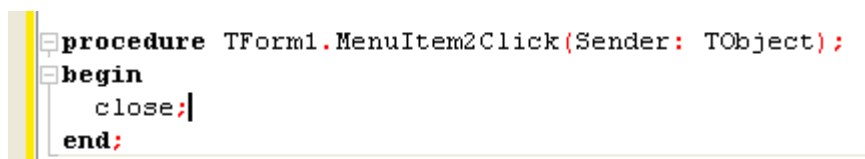


Рис. 1.19 - Добавление функции закрытия программы

➡ Отобразим форму программы «F12» и выберем пункт меню «Помощь»→«О программе».

➡ Напишем обработчик данного события (рис. 1.20). 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.');
```

Рис. 1.20 - Создание обработчика «О программе»

Текст сообщения необходимо писать в скобочках и одиночных кавычках.

➡ Сохраним проект и запустим его на выполнение.

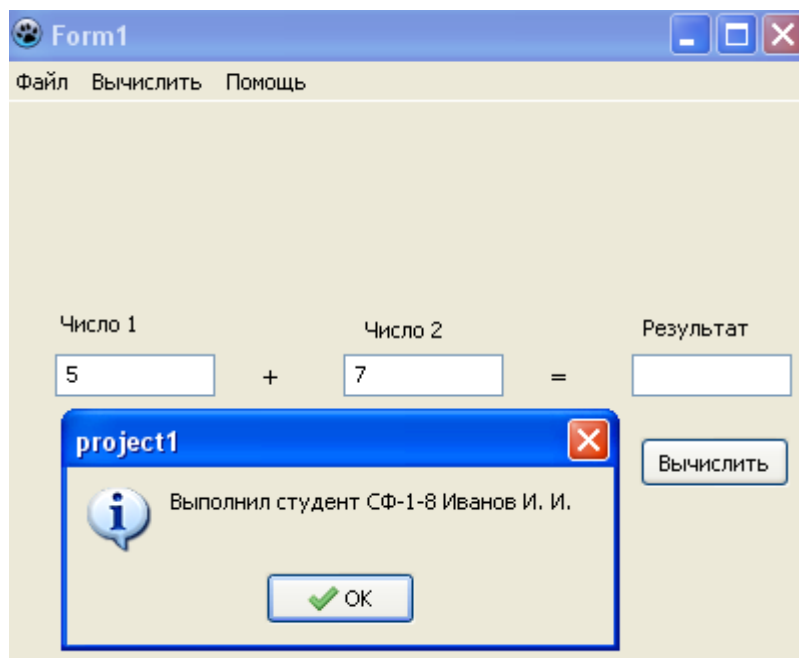


Рис. 1.21 - Проверка работоспособности обработчиков

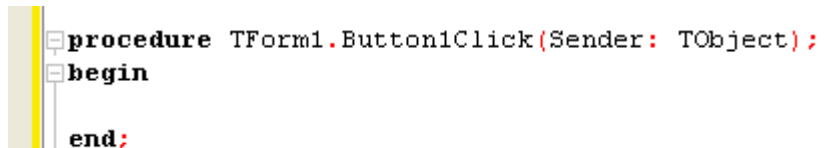
При выборе пункта меню «Помощь»→«О программе» должно отображаться следующее окно (рис. 1.21). При выборе «Файл»→«Выход» программа должна зарыться так же, как при нажатии на крестик в правом верхнем

углу программы.

Создана часть функциональности, теперь добавим процедуру вычисления суммы двух чисел.

➡ Закроем программу и на кнопке «Вычислить» сделаем двойной клик.

В результате будет создана заготовка (пустая процедура) обработчика нажатия на кнопку (рис. 1.22).



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

Рис. 1.22 - 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 «А»;
2. Получение данных со второго поля ввода данных, преобразование из текста в число и запись в переменную «В»;
3. Суммирование этих двух переменных и запись результата в переменную «С»;
4. Значение суммы из переменной «С» 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 длины. (Например: '123' + '456' = '123456', что не соответствует правилам математики). После того как получим сумму в переменной «С» необходимо ее обратно преобразовать в текст и записать в поле «Edit3». Процедура (обработчик события нажатия кнопки «Button1» на форме) имеет следующий вид рис. 1.23.

```
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. 1.23 - 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. 1.24).

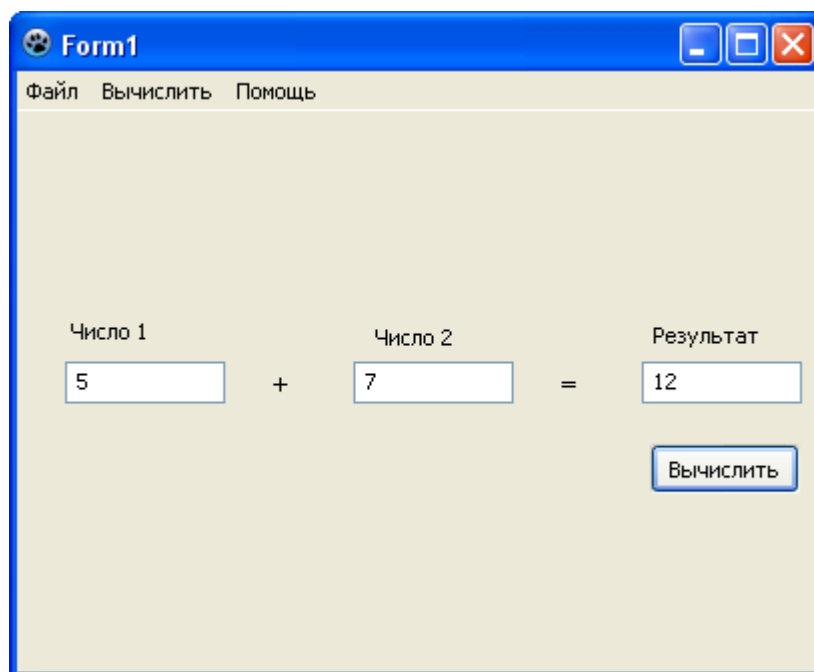


Рис. 1.24 - Пример расчета с помощью кнопки «Вычислить»

На данный момент при выборе пункта меню «Вычислить» — расчет не производится. Добавим эту возможность.

➡ Вернемся в «Редактор меню» Lazarus (рис. 1.25).

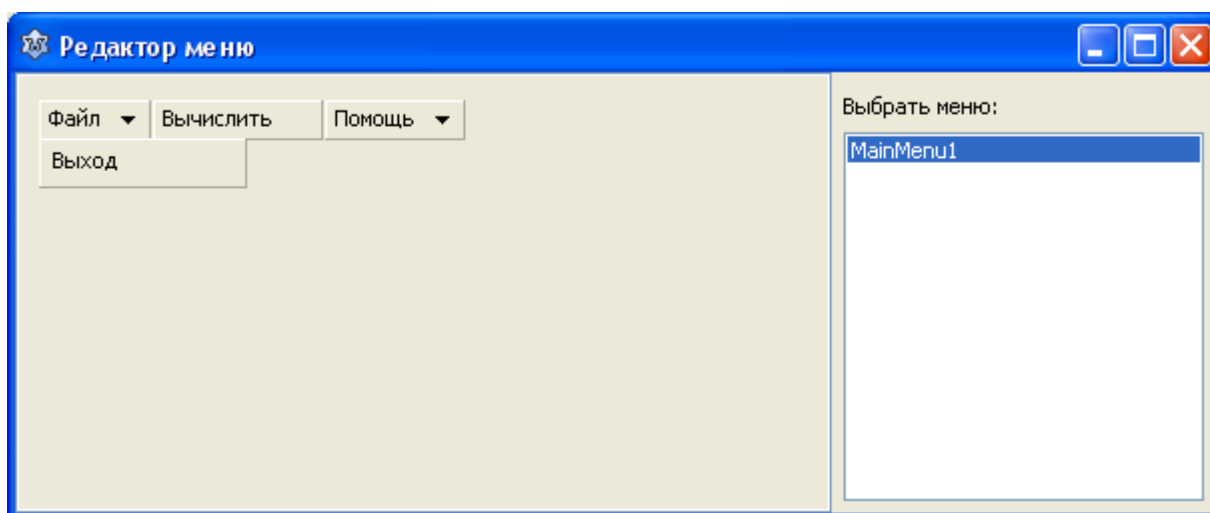


Рис. 1.25 - Выбор пункта меню «Вычислить» в «Редакторе меню»

➡ Один раз «кликнем» левой кнопкой «мыши» на пункте «Вычислить» и в «Инспекторе объектов» выбираем закладку «События», найдем событие «OnClick» (рис. 1.26)

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

➡ Сохраняем проект, запускаем, проверяем.

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

Проект готов.

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

Дополнительная информация по индивидуальным работам:

В результате операции деления «/» получается не целое значение, а вещественное. Для хранения вещественных чисел в языке Object Pascal используется тип данных Real. Для конвертирования строки текста в вещественное число используется функция StrToFloat, для конвертирования из вещественного числа в строку функция FloatToStr.

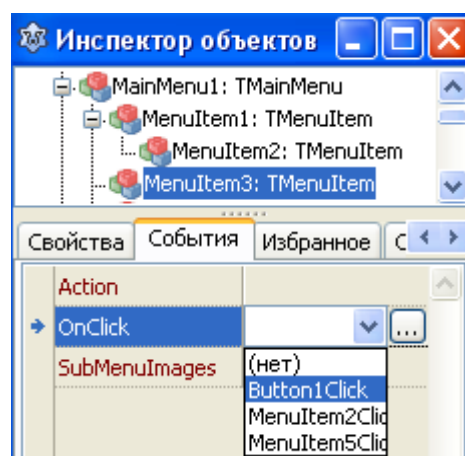


Рис. 1.26 - Привязка обработчика кнопки «Рассчитать» к меню «Вычислить»

Результат сложения (+) двух целых — целое значение.

Результат разности (-) двух целых — целое значение.

Результат умножения (*) двух целых — целое значение.

Для расчетов, в которых используется операция деления «/» - результат, в общем случае, может получиться дробным ($2/3 = 0.6666666666$), поэтому его нельзя присвоить целой переменной (integer). В таких случаях необходимо использовать операции округления/усечения/преобразования или записывать результат в вещественную (дробную) переменную. Для хранения дробных переменных в языке Object Pascal используется тип данных Real. Для преобразования чисел из строки в число и обратно используются функции FloatToStr и StrToFloat – аналогичные функциям IntToStr и StrToInt.



Модуль в языке Object Pascal это самостоятельная библиотека классов, процедур, функций, констант, типов и т. д., которая создана независимо от вашей программы и реализующая, часто повторяющиеся действия. В каждую программу подключена библиотека System — реализующая набор некоторых системных процедур и функций. Все подключенные библиотеки перечислены после слова Uses (в начале листинга) В этот список можно добавлять произвольные модули, дописывая их. Чем больше модулей подключено, тем больше функционал поддерживает программа и тем больше занимает программа места на диске и больше требует оперативной памяти для своего выполнения. Рекомендуется подключать только те модули, которые используются в программе. Уже подключенные модули — это форма, кнопки, системные утилиты, графика и т. п. необходимы для отображения формы и ее компонент на экране.

Для округления чисел можно использовать функцию Round — округление до целых. Для округления с другой степенью точности необходимо воспользоваться математикой. Также можно использовать функцию RoundTo из модуля Math. Функция RoundTo имеет два параметра: 1-й округляемое выражение; 2-й степень округления — целое число (при 0-м значении второго параметра — соответствует функции Round).

Примеры:

Round(22.2) = 22; RoundTo(22.2, 0) = 22

Round(34.357) = 34; RoundTo(34.357, -2) = 34.36

Round(1234.7)=1234; RoundTo(1234.7,3) = 1000

В заданиях со строками необходимо помнить, что в языке Object Pascal строки можно складывать операцией «+». Результат сложения тоже строка, состоящая из символов первой строки, а затем второй. Для определения длины строки

можно воспользоваться функцией `Length`, которая возвращает количество байт, занимаемое строкой. Строки содержат текст в определенной кодировке (`Win1251`, `cp866`, `KOI-8R`, `ANSI`, `UNICODE`, `UTF8` и т. д.). Первые 4 кодировки содержат по 1 байту на символ (число символов в кодировке 255), а последняя от 1 до 4 байт, в зависимости от принадлежности символа определенному набору. `UNICODE` предложен в 1991г и постоянно дополняется. Версия 6.2 вышедшая 2012г насчитывает 110 000 символов. Последние версии Lazarus для хранения текстовой информации используют `UTF8` – расширенный вариант `UNICODE`. Для изменения кодировки строк используется функции `AnsiToUtf8` и `Utf8ToAnsi`.

В заданиях с логическими выражениями рекомендуется воспользоваться типом данных `Boolean`. Это логический тип данных, позволяющий хранить два значения `True` и `False`. Для преобразования строки в логический тип данных можно воспользоваться функцией `StrToBool`, для обратного преобразования `BoolToStr`. Над логическими типами данных можно выполнять логические операции:

`and` — «и», `or` — «или», `xor` — «исключающее или», `not` — «отрицание».

A and B = R		
A	B	R
true	true	true
true	false	false
false	true	false
false	false	false

A or B = R		
A	B	R
true	true	true
true	false	true
false	true	true
false	false	false

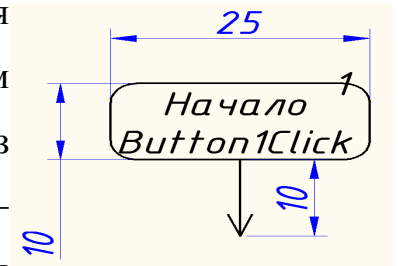
A xor B = R		
A	B	R
true	true	false
true	false	true
false	true	true
false	false	false

Not A = R	
A	R
true	false
false	true

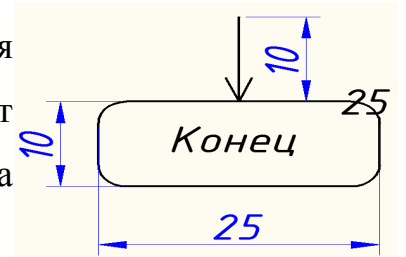
Блок схемы

Блок схемы составляются по листингу программы. Количество блок схем соответствует количеству процедур и функций в листинге. Количество входящих и исходящих стрелок у каждого элемента блок схемы строго определено. По окончании построения блок схемы элементы блок нумеруются слева-направо и сверху-вниз.

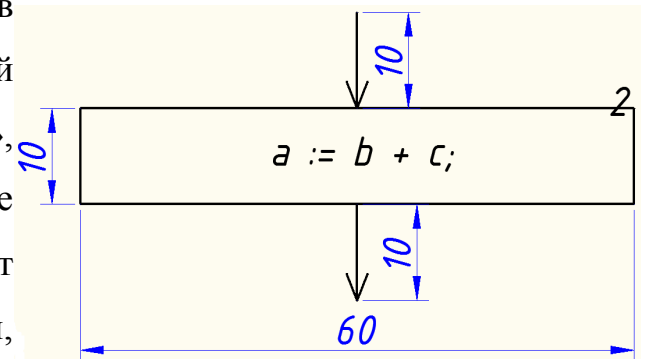
Элемент «Начало» - любая блок схема начинается этим элементом, он всегда присутствует в единственном экземпляре (процедура не может начать выполнение из неоткуда и у нее не может быть два или более начал — всегда только одно начало). Указывается имя процедуры и параметры, если они используются в теле процедуры.



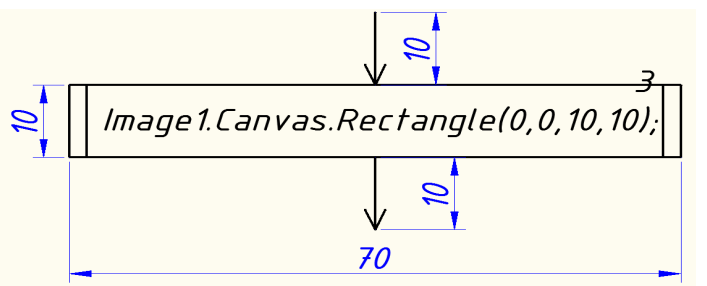
Элемент «Конец» - этим элементом заканчивается блок схема. В любой блок схеме всегда присутствует только один элемент «конец» (программа должна заканчивать свое действие, а не работать бесконечно).



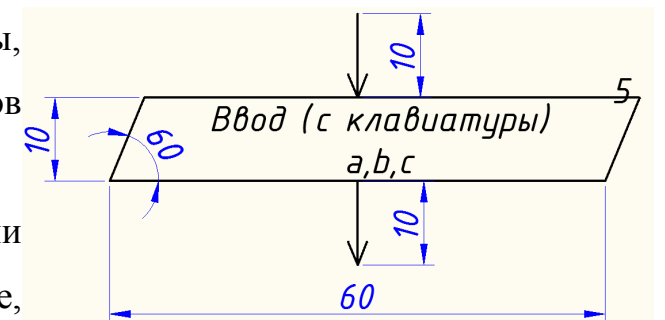
Элемент «Присваивание», если в программе какой либо переменной присваивается значение выражения «:=», то на блок схеме отображается в виде прямоугольника. Если присваиваний идет несколько подряд по листингу программы, то можно их объединить в один элемент блок схемы, записав последовательно каждый оператор присваивания в отдельной строчке.



Элемент «Вызов процедуры», если в программе вызывается какая либо процедура, то на блок схеме он отражается в виде прямоугольника с двойными вертикальными линиями.



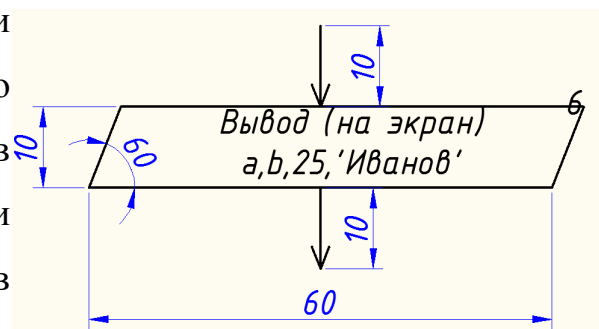
Если вызовов процедур идет несколько подряд по листингу программы, то можно их объединить в один элемент блок схемы, записав последовательно каждый вызов процедуры в отдельной строчке.



Элемент «Ввод данных», если программа получает информацию извне,

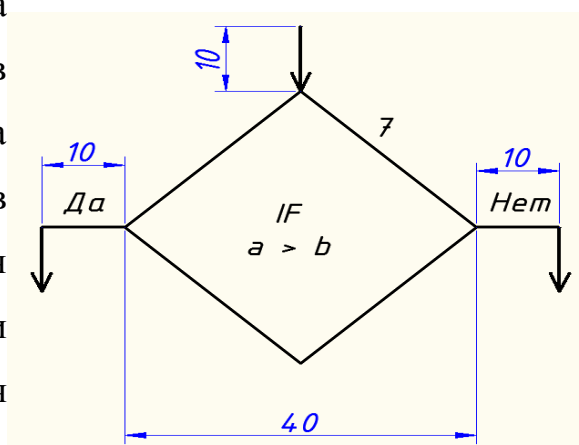
например от пользователя с клавиатуры или из файла, из сети или со сканера и т. п., то процесс получения информации оформляется на блок схеме в виде параллелограмма. Первая строка указывает, что информацию получает программа и указывается источник (файл, порт, адрес и т.п.), вторая строка перечисляет те переменные в которые будет внесена эта информация. Для разных источников получения информации необходимо формировать свой элемент блок схемы, для одного источника при последовательном получении нескольких данных возможно их объединение в один элемент блок схемы.

Элемент «Вывод данных», если программа выводит какие либо данные во внешние устройства (на экран, на принтер, в файл, в сеть и т. п.) то процесс выдачи информации оформляется на блок схеме в виде параллелограмма. Первая строка



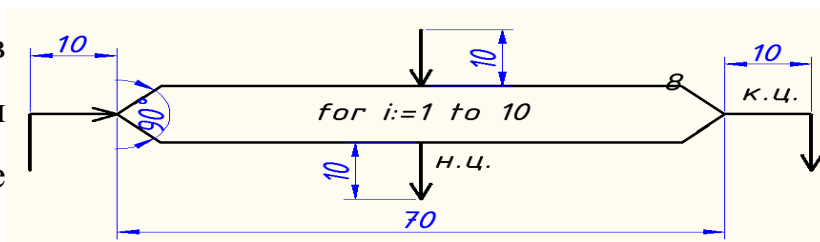
указывает что информация отправляется на внешний носитель и указывается приемник информации (файл, порт, адрес и т.п), вторая строка указывает те переменные значения которых будет отправлено, также можно использовать выражения и константы.

Элемент «Выбор», если программа должна выполнить различные действия, в зависимости от каких либо условий, то на блок схеме такой оператор оформляется в виде ромба. У элемента есть одна входящая стрелка и две исходящие. Исходящие стрелки нужно различать и над ними указывается надписи «да» - «нет» или «true» - «false» или «истина» - «ложь» или «1» - «0». Внутри ромба указывается условие выбора. Если



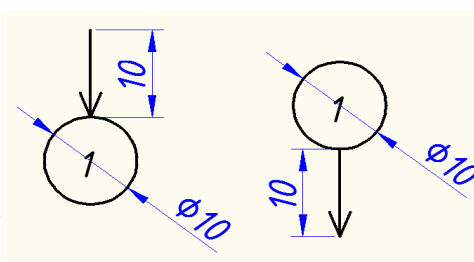
условие принимает истинное значение, то программа пойдет по стрелке с надписью «да», если условие ложно то по стрелке «нет».

Элемент «Цикл», если в программе используются операторы цикла (в языке Object Pascal их 3 — For,

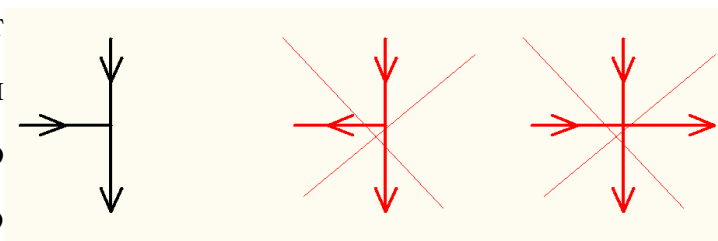


While, Repeat), то на блок схеме они обозначаются в виде шестиугольников. Элемент имеет две входящие и две исходящие стрелки. Исходящие стрелки именуются «начало цикла» и «конец цикла». По стрелке с названием «начало цикла» программа переходит на тело цикла, пока условие цикла выполняется. Действия выполняемые в теле цикла должны вернуться по одной из входящих стрелок на следующую итерацию цикла. По стрелке «конец цикла» программа переходит при невыполнении условия цикла, минуя тело цикла.

Если блок схема не помещается на листе то необходимо оформить разрыв блок схемы. Внизу листа указывается круг с входящей стрелкой и буквой или цифрой внутри круга. На следующем листе указывается такой же круг с исходящей стрелкой и такой же буквой. Это не элемент блок схемы, а средство оформления, поэтому он не нумеруется.



Стрелки на блок схеме могут сходиться, но не могут расходиться (программа не может выбрать какую либо из стрелок без каких либо



условий, программа не может пойти сразу по двум стрелкам и не пойти никуда). Стрелки не могут пересекаться — теряется однозначность блок схемы.

Лабораторная работа №2

Тема: Операторы выбора.

Задание: Необходимо отобразить рис 2.1 на форме программы и дать возможность пользователю вводить координаты точек. Программа проверяет и сообщает: точка с данными координатами принадлежит области или нет.

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

Работу сохраним в папке «2». Рекомендованный путь к проекту: «Мой компьютер\Общие документы\СФ-1-8а\Иванов\2».

Создадим интерфейс.

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

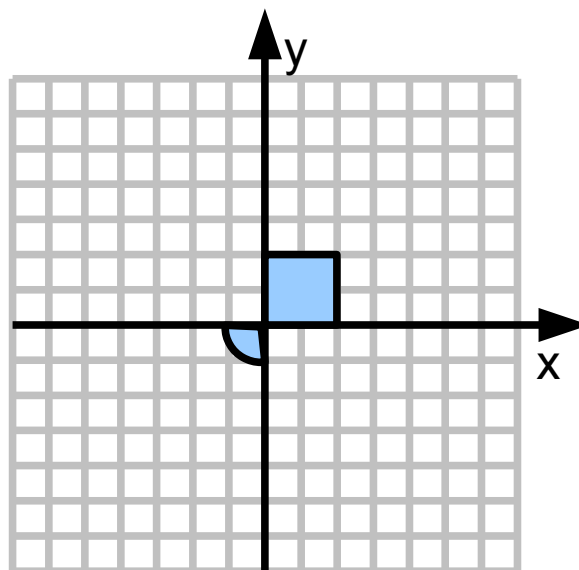


Рис. 2.1 - Задание

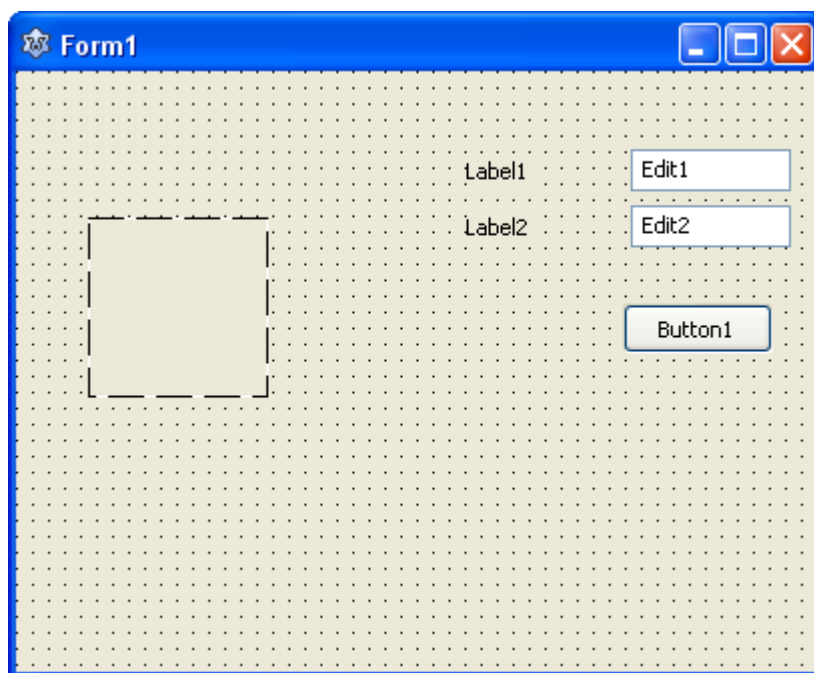


Рис. 2.2 - Расположение компонентов на форме

Слева на форме компонент TImage (прямоугольник из пунктирных линий), он находится на закладке компонент Additional. Справа: два TEdit, два TLabel и один TButton.

➡ Расставим компоненты, как на рис. 2.2.

➡ Изменим заголовки у компонентов TLabel и TButton (свойство Caption) и значения поля Text у компонентов TEdit. В результате получим форму (рис. 2.3).

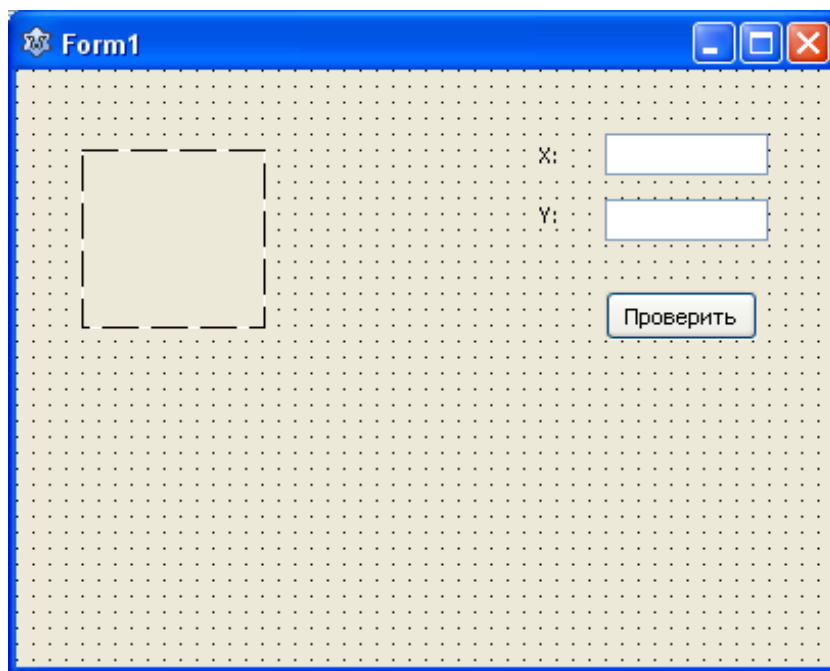


Рис. 2.3 - Переименование и настройка основных компонент

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

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

Система координат (рис. 2.4) компонента TImage, как и у всех других компонент, перевернута от обычной (которой вы привыкли

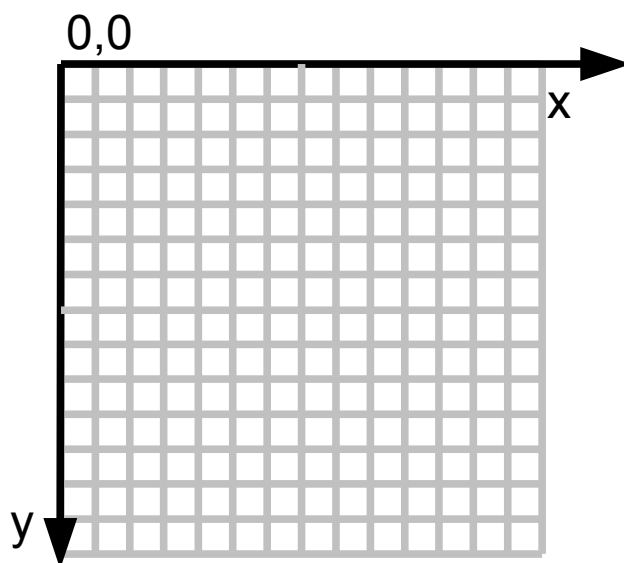


Рис. 2.4 - Система координат используемая у компонента TImage

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 (один пиксель — одна единица).

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 и перенесем центр координат в центр картинки. Масштаб выберем равным 20:1, а центр в точке с координатами $x=100$; $y=100$.

➡ Интерфейс создан — сохраняем проект и запускаем на выполнение.

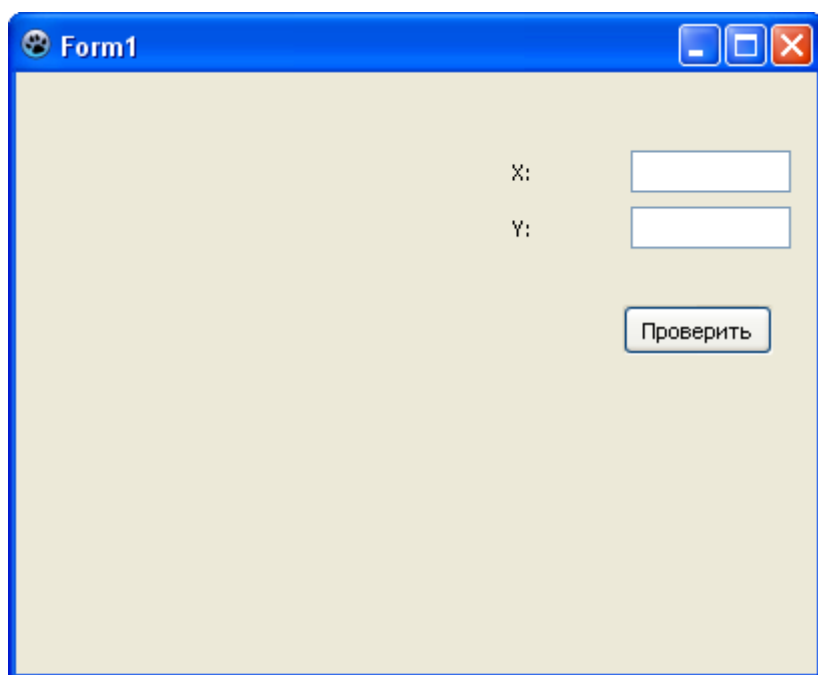



Рис. 2.5 - Внешний вид формы после сборки и запуска проекта

Компоненты «TLabel», «TEdit» и «TButton»

 Компонент «TImage» может отобразить на форме рисунок 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. Использовать функции примитивного рисования и последовательно создать рисунок из таких элементов как: линии, точки, дуги, окружности, эллипсы и т. д.

Первый способ лучше подходит для фотографий, а второй для схем.

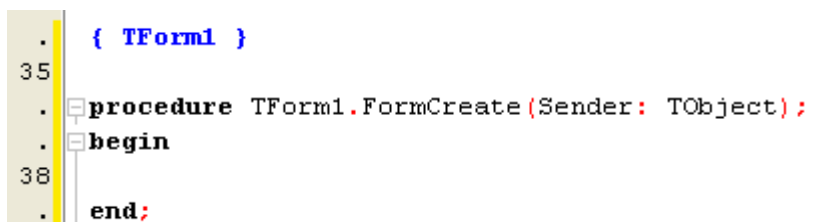
видны на форме, а компонент «TImage» на форме не виден (рис. 2.5) т. к. на нем ничего не нарисовано.

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 другие компоненты) нарисуем область из задания.

➡ Сделаем «двойной клик» в свободной области формы (где нет других компонентов) для обработчика создания формы (рис. 2.6):



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

Рис. 2.6 - 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 Image1. Свойство отвечающее за рисование примитивов называется Canvas.

Существует множество процедур у свойства Canvas, которые рисуют примитивные объекты. Например: для отображения линии нужно 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.

Перевод из физических в экранные $X_{\phi} = X_p \cdot 20 + 100$; $Y_{\phi} = -Y_p \cdot 20 + 100$

➡ Отобразим в центре рисунка окружность (рис. 2.7).

```
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 (рис. 2.8).

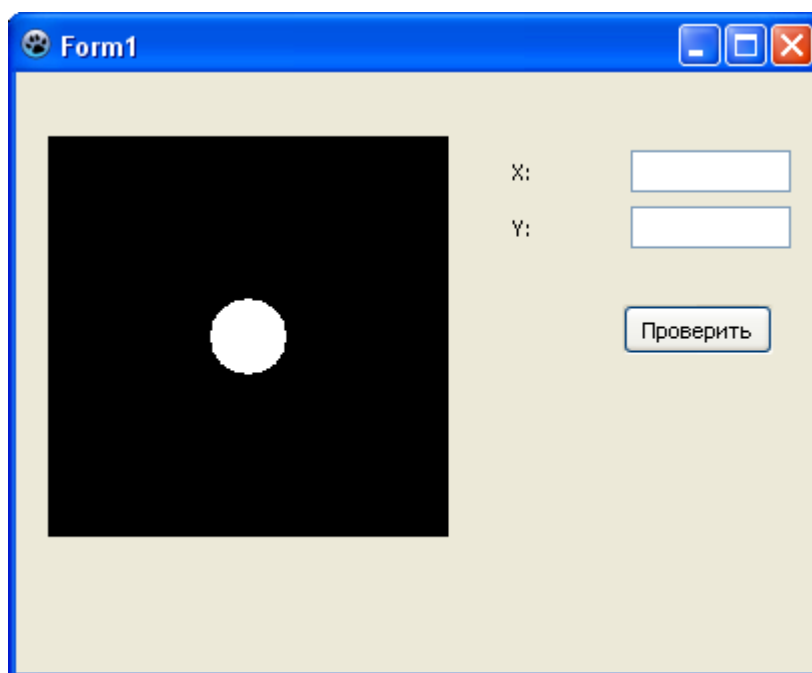


Рис. 2.8 - Окружность в центре Image

Обычно на белом фоне рисуют черные линии — нужно сменить фон.

По умолчанию «карандаш» черный, а «кисть» белая, 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 (рис. 2.9).

```

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 (рис. 2.10).

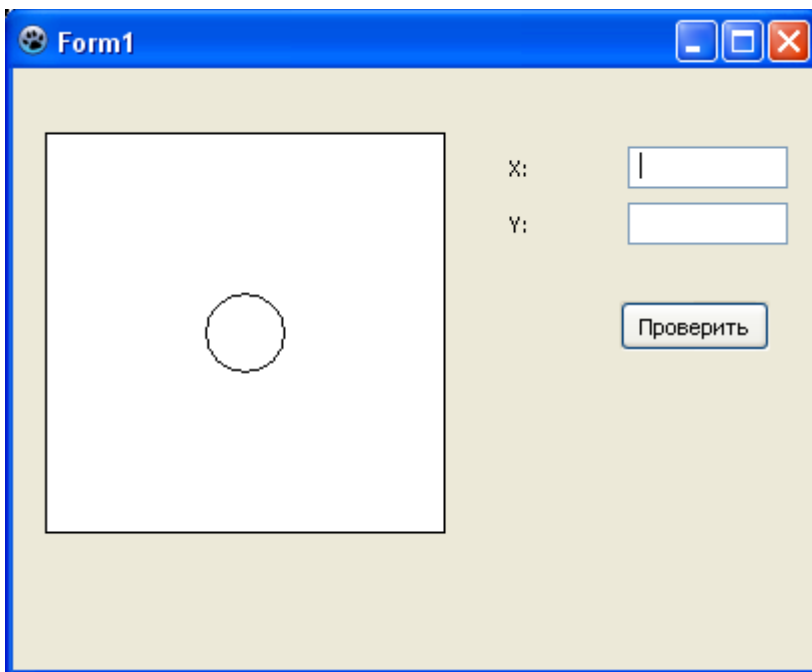


Рис. 2.10 - Отображение окружности на белом фоне

➡ Закрасим внутреннюю часть окружности в зеленый цвет — для информативности области. До рисования окружности назначим зеленый цвет



Цвета 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 часть цвета берется в диапазоне от 0 до 255 и чем больше число, тем сильнее выражен соответствующий цвет. Пример: для красного цвета надо красную часть повысить на максимум, а остальные убрать на минимум (255,0,0). В среде Lazarus присутствует константы цветов. Это цвета, которыми мы привыкли пользоваться в повседневной жизни. 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).

кисти (рис. 2.11).

➡ Сохраним проект и запустим на выполнение. В результате (рис.2. 12) получим следующую форму:

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 (рис. 2.11).

➡ Сохраним проект и запустим на выполнение.

```

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

```

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

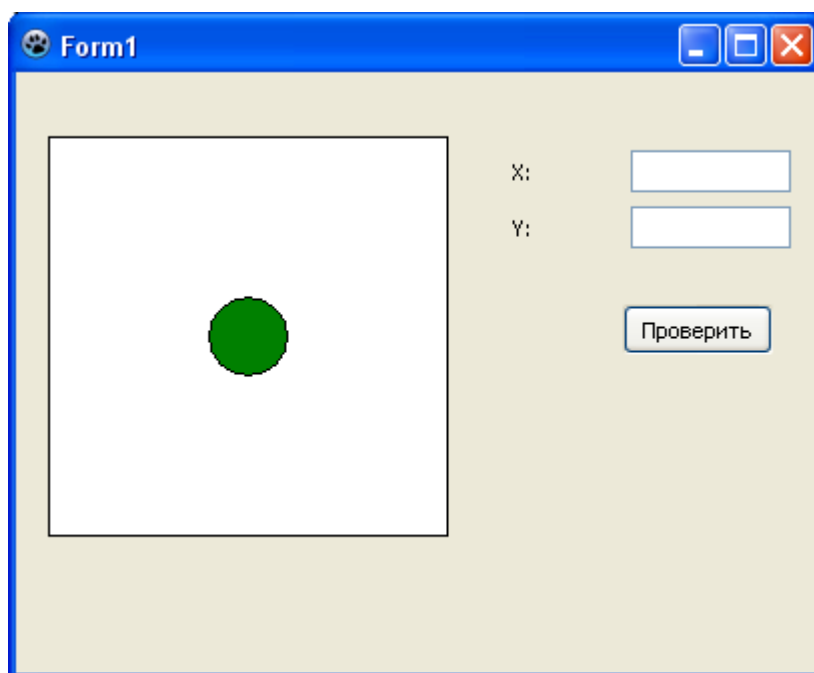


Рис. 2.12 - Зеленая окружность в центре области

```

. procedure TForm1.FormCreate(Sender: TObject);
. begin
.     Image1.Canvas.Rectangle(0,0,200,200);
.     Image1.Canvas.Brush.Color:=clGreen;
40  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);
45  end;

```

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

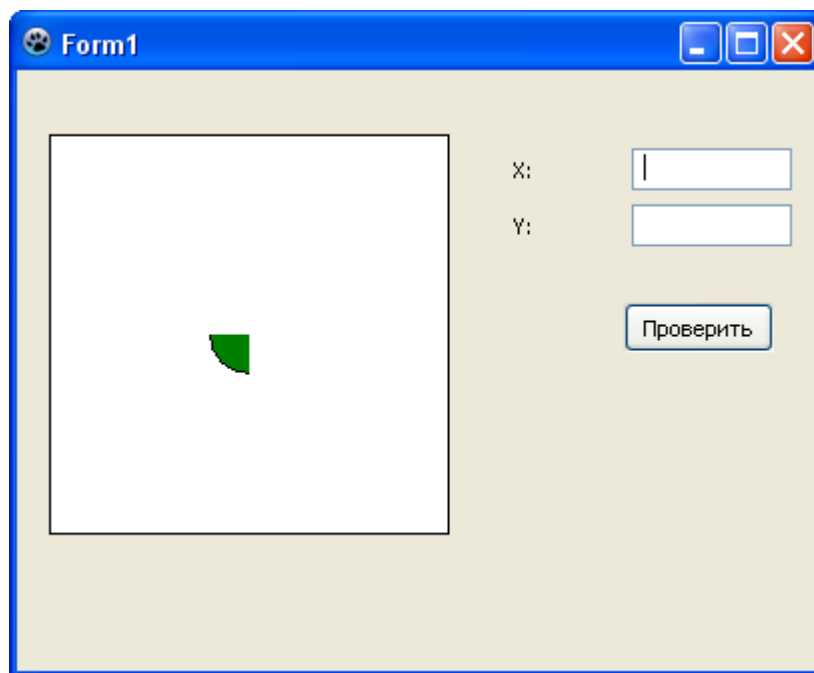


Рис. 2.14 - Четверть окружности

В результате от окружности осталась одна четверть (рис. 2.14). Приступим к рисованию второй части области — прямоугольник.

➡ Сменим кисть на зеленую, а карандаш на черный и нарисуем прямоугольник. Добавим следующий код (рис. 2.15).

```
. procedure TForm1.FormCreate(Sender: TObject);
. begin
.     Image1.Canvas.Rectangle(0,0,200,200);
.     Image1.Canvas.Brush.Color:=clGreen;
40  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);
45  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).

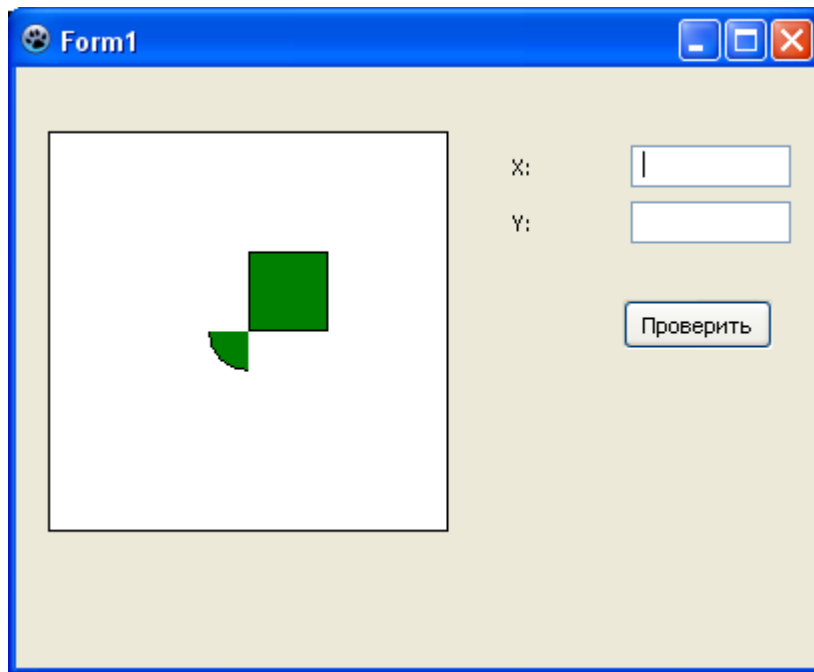


Рис. 2.16 - Четверть окружности и прямоугольник

➡ Осталось добавить оси координат, воспользуемся функцией `Line` (рис. 2.17).

```
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).

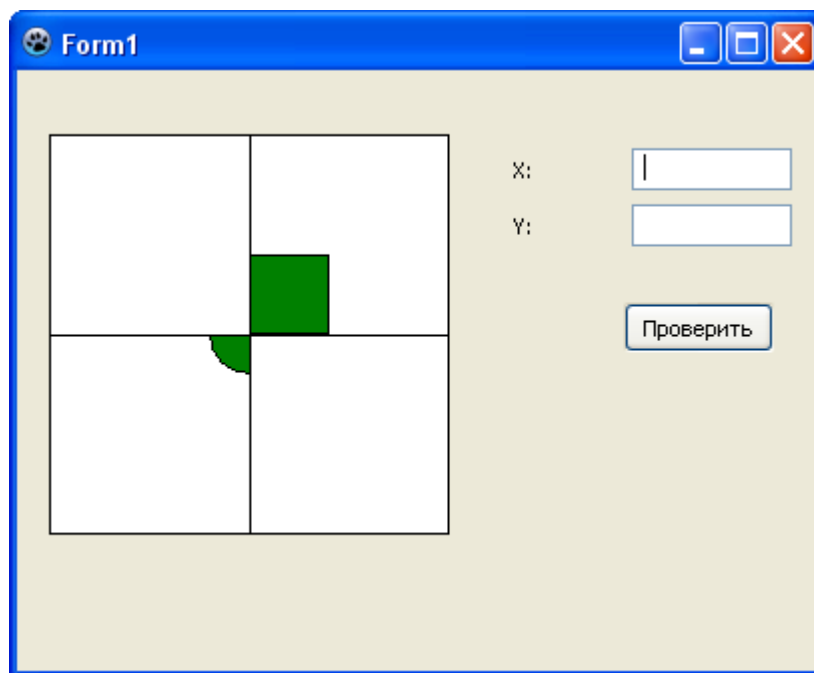


Рис. 2.18 - Полностью готовый рисунок с четвертью окружности, прямоугольником и системой координат

Программа рисует область, а проверять координаты пока не умеет. Добавим эту возможность в программу. Необходимо написать обработчик на кнопку «Проверить».

➡ Закроем программу и в редакторе формы сделаем двойной «клик» на кнопке «Проверить». В результате отобразится следующий код (рис. 2.19).

```
. 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 переменные X и Y значения из соответствующих полей (рис. 2.20).

```
. 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, но не произойдет 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 всей 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 проверки четверти окружности необходимо знать 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) 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



Оператор выбора «IF» может содержать один или два (как в данном случае) вложенных в него операторов. Первый оператор находящийся после слова «Then» выполнится если условие выполнится, а второй если условие не выполнится. Обо одновременно не могут выполниться, и ни один тоже не может выполниться. Хотя бы один из операторов (после слова «Then» или «Ealse») всегда выполняется, но какой зависит от условия.

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).

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

➔ Добавим код (рис. 2.21).

➔ Сохраним проект и запустим на выполнение. Опробуем проверку координат.

```

. procedure TForm1.Button1Click(Sender: TObject);
. var
55   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
60     ((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

поля введем координаты (например 1, 1) и нажмем кнопку проверить (рис. 2.22).

➔ 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

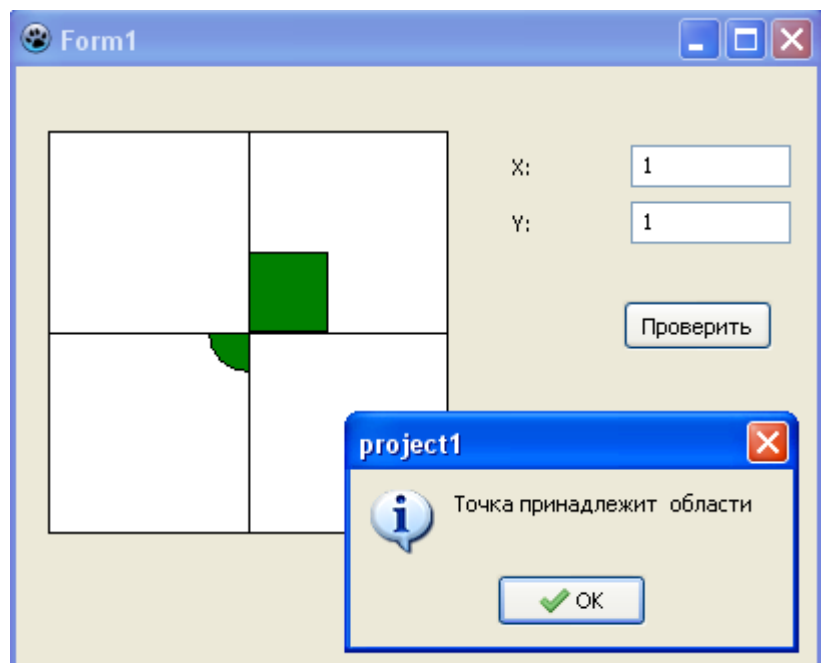


Рис. 2.22 - Проверка координат

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 «события» (рис. 2.23).

➡Список возможных привязываемых событий пуст, но справа есть кнопка «...» нажмем на нее и получим заготовку события OnMouseDown (рис. 2.24).

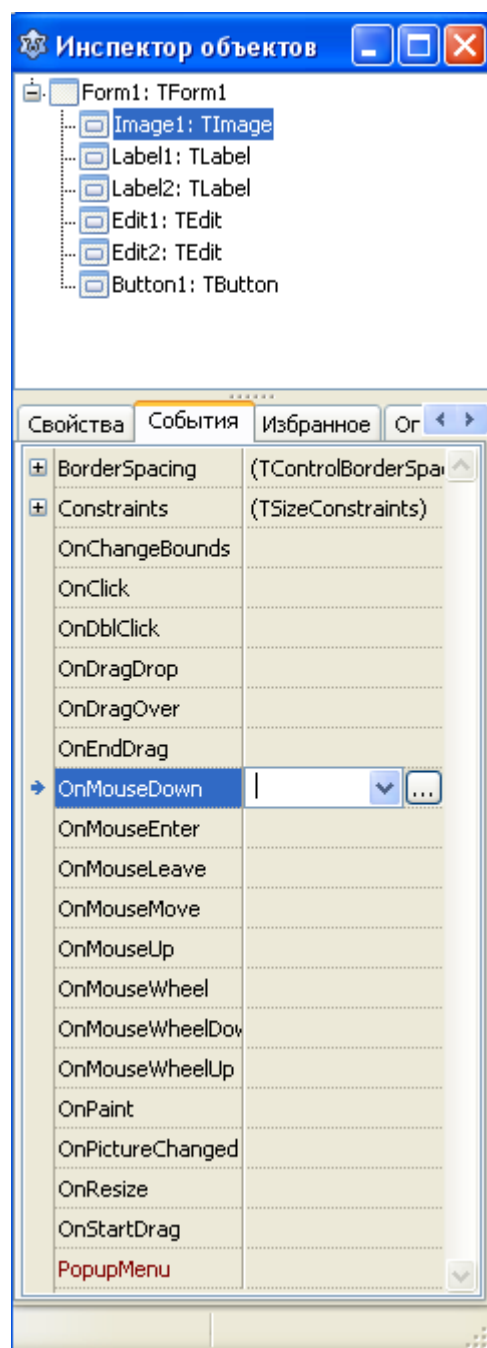


Рис. 2.23 - Привязка события «OnMouseDown» к «Image1»

```

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 параметра — координаты «мыши» в момент «клика». Они соответственно хранятся в переменных «X» и «Y».

➡ Напишем код между begin и end (рис. 2.25). 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;

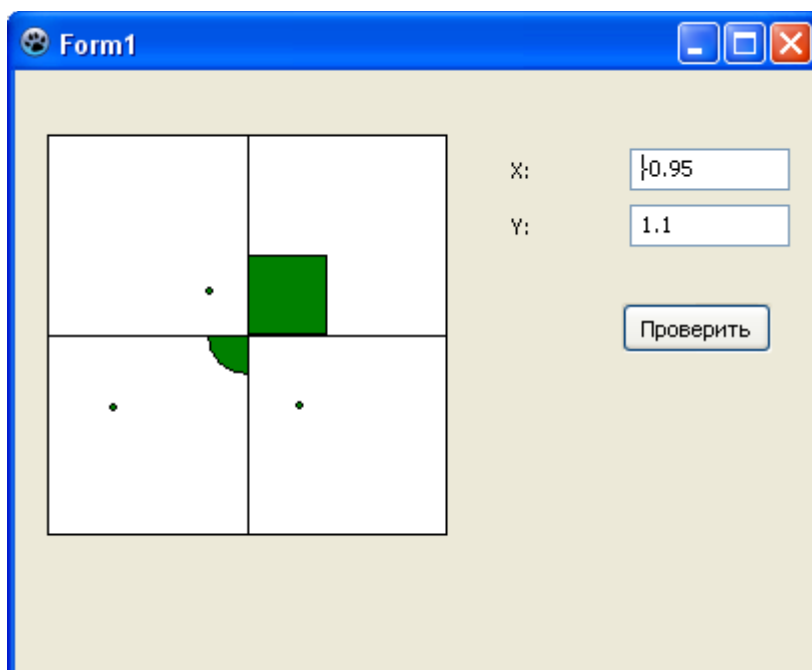
```

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.).

Работа над демонстрационным 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, расставив компоненты (рис. 3.1).

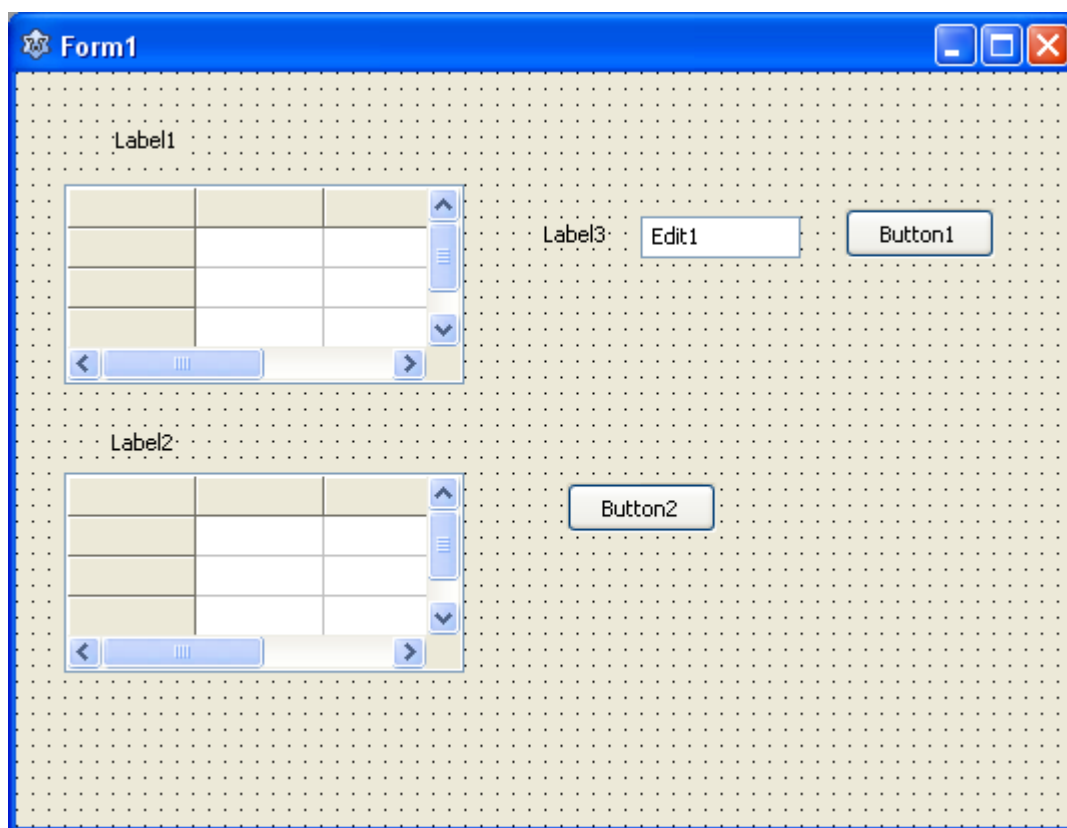


Рис. 3.1 - Расположение компонентов на форме

Настройка компонентов TLabel, TEdit и TButton рассмотрено в предыдущих примерах.

➡ На форму установим два компонента TStringGrid — таблица строк (один под другим, палитра компонент Additional).


➡ Установим для обеих таблиц в «инспекторе объектов» свойства FixedRows=0



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).

и FixedCols = 0, что бы избавиться от «шапки» у таблиц.

➡ Настроим «метки», «кнопки» и «поле ввода данных» — количество элементов как показано на рис 3.2.

 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.

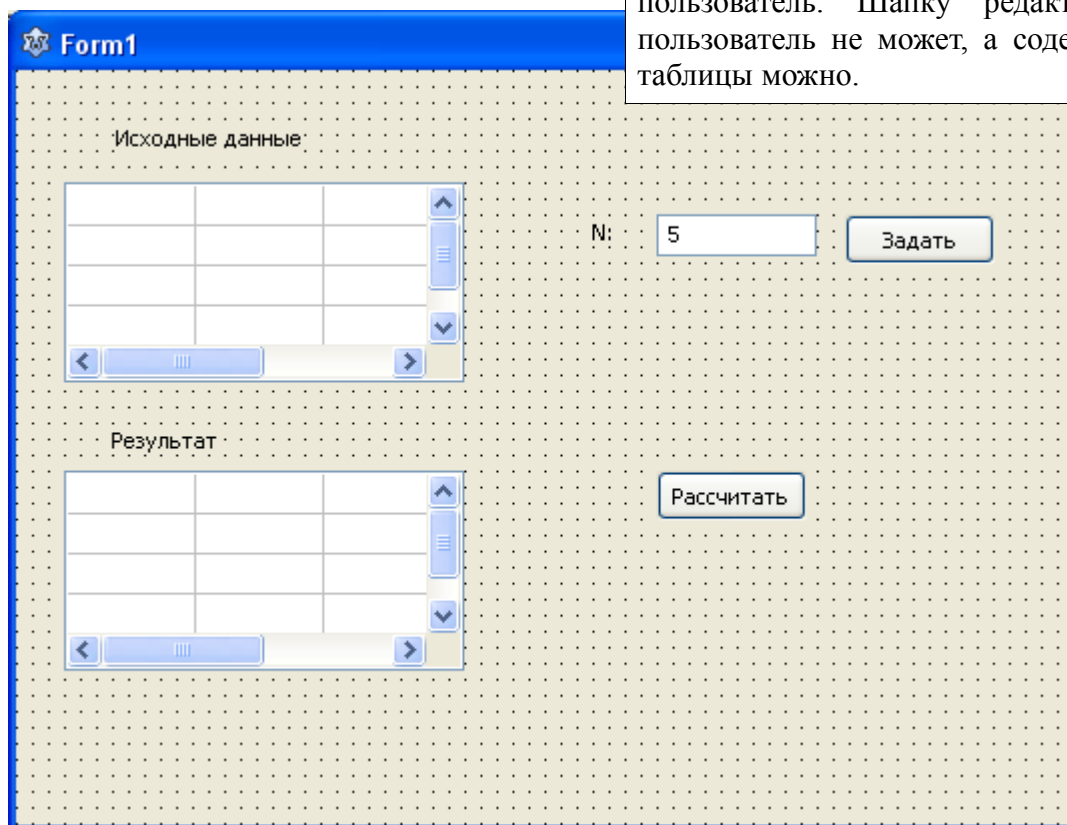


Рис. 3.2 - Настройка компонентов на форме

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).

Form1

Исходные данные

N: 5

Задать

Результат

Рассчитать

Рис. 3.3 - Настройка компонентов StringGrid

➡ Сохраним проект и запустим на выполнение (рис. 3.4).

Form1

Исходные данные

N: 5

Задать

Результат

Рассчитать

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.

➡ «Раскроем» список всех возможных элементов множества (для доступа к элементам множества), «кликнув» по знаку «+». Интересующий элемент множества goEditing — значений ячеек таблицы во время выполнения программы (рис. 3.5).

➡ Его необходимо внести в множество — установив значение true для разрешения редактирования ячеек таблицы.

➡ Настроим Options у обеих таблиц.

➡ Сохраним проект и запустим на

выполнение.

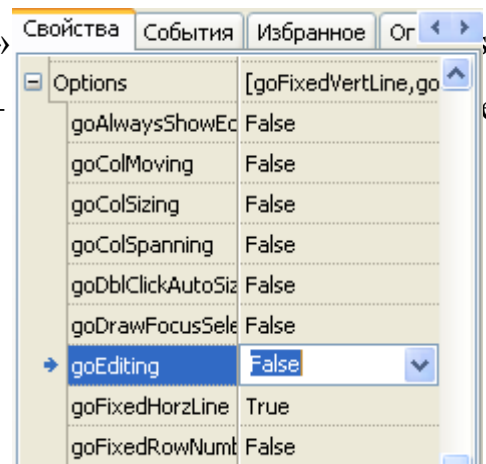


Рис. 3.5 - Изменение свойства Options → goEditing у обеих таблиц

The image shows a Windows application window titled 'Form1'. It contains two tables. The first table, labeled 'Исходные данные', has 2 rows and 5 columns with the following values:

7	9	1	444	555
66	8	0	333	666

 To the right of this table is a label 'N:' followed by a text box containing the value '5'. Below these is a button labeled 'Задать'.

 The second table, labeled 'Результат', also has 2 rows and 5 columns. The first row contains 'gggg' and the second row contains 'hhhh'. To the right of this table is a button labeled 'Рассчитать'.

Рис. 3.6 - Внесение данные в таблицы

Появилась возможность внесения текста в таблицы (рис. 3.6).

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 (рис. 3.7).

```
. procedure TForm1.Button1Click(Sender: TObject);  
. var  
40 i,n : integer;  
. begin  
    n := StrToInt(Edit1.Text);  
    StringGrid1.ColCount:=n;  
    for i:=0 to n-1 do  
45     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». Отобразится форма (рис. 3.8).

Form1

Исходные данные

1	2	3	4

N: 4

Задать

Результат

Рассчитать

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 математики известно- среднее значение это сумму всех элементов разделенная на количество.

Процедура (обработчик) для кнопки «Рассчитать» (рис. 3.9).

```

1 procedure TForm1.Button2Click(Sender: TObject);
2 var
3     i, n : integer;
4     sr : Real;
5 begin
6     sr := 0;
7     n := StrToInt(Edit1.Text);
8     for i:=0 to n-1 do
9         sr := sr + StrToFloat(StringGrid1.Cells[i,1]);
10    sr := sr / n;
11    StringGrid2.ColCount:=1;
12    for i:=0 to n-1 do
13        if StrToFloat(StringGrid1.Cells[i,1]) >= sr then
14            begin
15                StringGrid2.Cells[StringGrid2.ColCount-1,0] := IntToStr(StringGrid2.ColCount);
16                StringGrid2.Cells[StringGrid2.ColCount-1,1] := StringGrid1.Cells[i,1];
17                StringGrid2.ColCount:= StringGrid2.ColCount +1;
18            end;
19    StringGrid2.ColCount:= StringGrid2.ColCount -1;
20 end;

```

Рис. 3.9 - 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 которой 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:

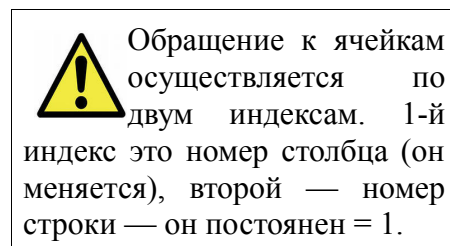
« sr := 0;» - обнуление счетчика суммирования.

«n := StrToInt(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. Перед суммированием преобразовываем значения ячеек из строк в числа — функция 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 выполнится 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(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.

➡ Сохраним проект и запустим на выполнение.

➡ Установим количество элементов в первой таблице, внесем в нее данные и рассчитаем вторую таблицу (рис. 3.10).

Данные для расчета необходимо вносить полностью и корректно. Количество столбцов должно быть целым и 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. Данный стиль программирования считается более дружелюбным к пользователю.

The screenshot shows a Windows application window titled 'Form1'. It contains two sections: 'Исходные данные' (Initial data) and 'Результат' (Result). In the 'Исходные данные' section, there is a table with 5 columns and 2 rows of data, and a text box labeled 'N:' containing the value '5'. Below the table is a button labeled 'Задать' (Set). In the 'Результат' section, there is a table with 2 columns and 2 rows of data, and a button labeled 'Рассчитать' (Calculate).

1	2	3	4	5
55	22	10	6	88

N: 5

Задать

1	2
55	88

Рассчитать

Рис. 3.10 - Пример работы программы

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

Для нахождения минимальных и максимальных значений можно воспользоваться — перебором всех значений таблицы с запоминанием наибольшего или наименьшего из них в отдельной переменной.

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 (рис. 4.1).

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

Вверху расположена таблица «StringGrid», внизу поле для вывода суммы элементов ряда и кнопка «Рассчитать». Таблицу настроим так же, как в 3-й работе, кнопку переименуем.

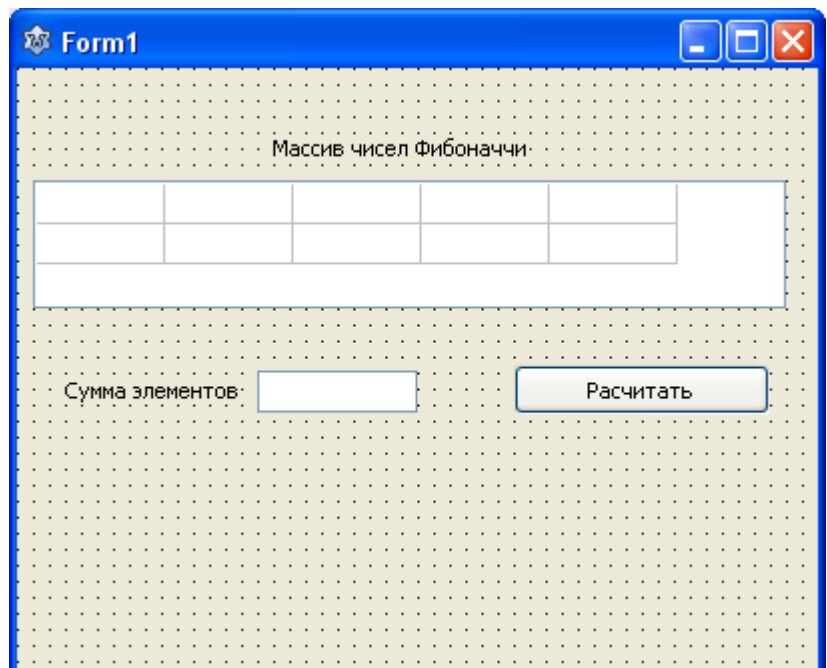


Рис. 4.1 - Интерфейс программы

Программа содержит один обработчик связанный с кнопкой «Рассчитать».

➡ Напишем процедуру обработчик, для кнопки «Рассчитать» (рис. 4.2).

```
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. Первоначальная установка количества столбцов и инициализация суммы.

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

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.

«с := 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 «Edit1».

➡ Сохраним проект и запустим на выполнение (рис. 4.3).

Рис. 4.3 - Форма для расчета ряда Фибоначчи

➡ Нажав на кнопку «Расчитать» получим следующие данные (рис. 4.4).

Массив чисел Фибоначчи

6	7	8	9	10
8	13	21	34	55

Сумма элементов 143

Расчитать

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 близка к формуле расчета следующего элемента в ряде чисел Фибоначчи — она так же оперирует предыдущими двумя значениями ряда).

Лабораторная работа №5

Тема: Массивы.

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

Начальные условия:

$$S_i = 2S_{i-2} - 3S_{i-1} + 5 \quad S_1 = 2 \quad S_7 = 4 \quad n = 16$$

type = int

Разработаем интерфейс программы.

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

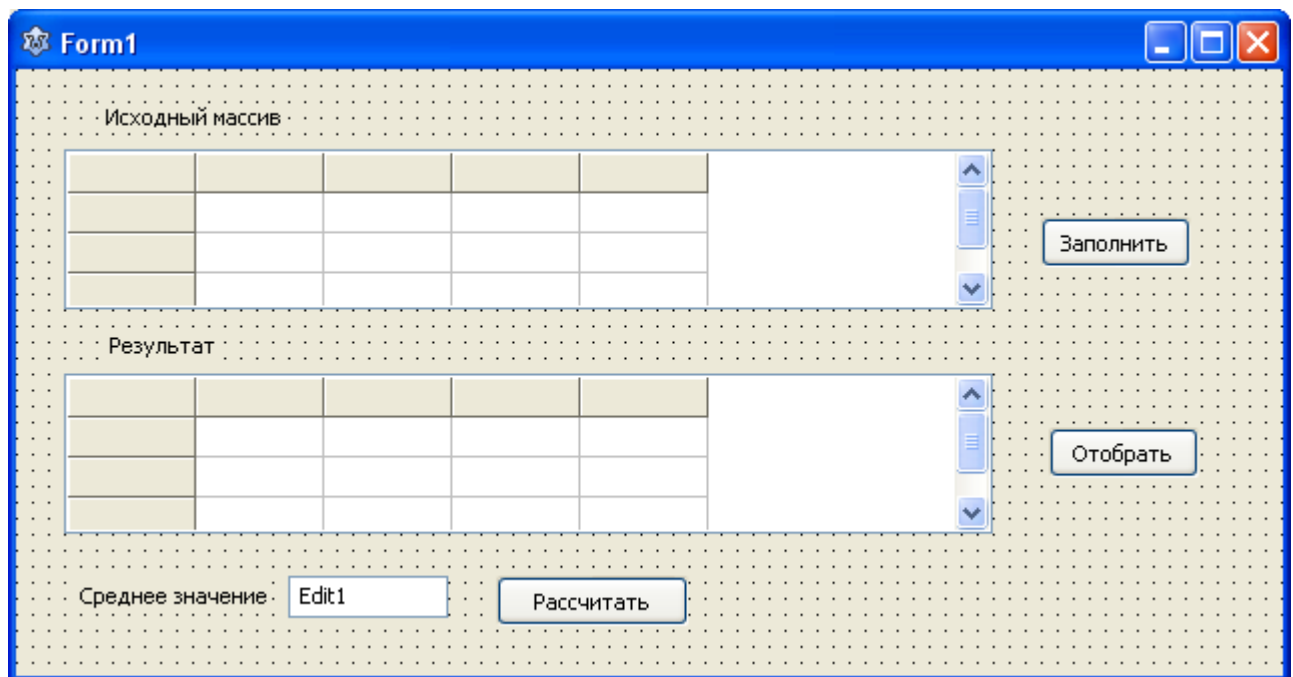


Рис. 5.1 - Интерфейс программы

➡ 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 (рис. 5.2):

FixedRows = 0

FixedCols = 0

RowCout =2

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

➡ Интерфейс готов, напомним обработчики событий.

Обращаться к ряду чисел будем неоднократно — вычислить его, подсчитывать среднее значение и выдавать часть элементов удовлетворяющих условию задания. Что бы 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 (рис. 5.3).

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.

```

35  var
36      Form1: TForm1;
      mas : array[1..16] of integer;
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 рис 5.4.

```
. { 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  
60       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).

Рис. 5.5 - Результат работы программы по кнопке «Заполнить»

➡ Вычислим среднее значение ряда - напомним 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 (рис. 5.6).

```

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 (рис. 5.7) и вычислим среднее значение ряда.

Исходный массив

1	2	3	4	5	6	7	8
2	4	-3	22	-67	250	-879	31

Результат

Среднее значение 3966274.1875

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 «Отобразить» (рис. 5.8).

```

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

Алгоритм отбора элементов массива, такой же, как в предыдущих работах.

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).

Form1

Исходный массив

1	2	3	4	5	6	7	8
2	4	-3	22	-67	250	-879	31

Заполнить

Результат

14	16
6407626	81278518

Отобрать

Среднее значение 3966274.1875

Рассчитать

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).

Form1

Данные по всем абитуриентам

Результат - отобранные абитуриенты

Вывести

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).

Form1

Данные по всем абитуриентам

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

Результат - отобранные абитуриенты

Вывести

Form
Left: 33
Width: 79

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 (рис. 6.3).

Данные по всем абитуриентам

№	1	2	3	4	5	6	7	8	9	10
Фамилия	Боровик	Петров	Сидоров	Колесников	Иванова	Кравцова	Преображен	Сухов	Шереметье	Майорова
Имя	Елена	Александр	Станислав	Татьяна	Светлана	Анастасия	Катерина	Петр	Виктор	Анна
Отчество	Сергеевна	Павлович	Олегович	Петровна	Александрс	Александрс	Матвеевна	Павлович	Анатолеви	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

Результат - отобранные абитуриенты

№	
Фамилия	
И	
О	

Вывести

Рис. 6.3 - Окончательный дизайн формы — настроена таблица StringGrid2

Каждый абитуриент обладает набором персональных данных, поэтому сгруппируем их в структуру — запись и назовем ее TAbiturient. Опишем ее в разделе type, т. к. здесь описываются новые типы данных (рис. 6.4). По принятым стилевым правилам оформления кода все типы имеют префикс «Т».

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 «:»

```

15 TForm1 = class(TForm)
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    StringGrid1: TStringGrid;
    StringGrid2: TStringGrid;
20
    private
        { private declarations }
    public
        { public declarations }
25
    end;
30
    TAbiturient = record
        F : String;
        I : String;
        O : String;
        year : integer;
        Fiz : integer;
        mat : integer;
        rus : integer;
35
    end;
40
    var
        Form1: TForm1;

```

Рис. 6.4 - Объявление структуры TAbiturient в разделе type

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.

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 (рис. 6.5).

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

Рис. 6.5 - Объявление типа массива абитуриентов

Объявлено два новых типа данных, что бы ObjectPascal выделил под них область памяти нужно определить переменные соответствующих типов данных. Необходимо два массива абитуриентов (рис. 6.6). В первом мы будем хранить всех абитуриентов, а во втором только тех, которые удовлетворяют критериям (отобранные абитуриенты).

Код представлен ниже.

➡ 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

```

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;

```

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

элементов из mas2 в StringGrid2 — искомые данные.

➡ Для выдачи только одного символа от имени и отчества (инициалы) воспользуемся функциями `AnsiToUtf8(Utf8ToAnsi(mas2[i].I)[1]+'.')`; Переводим его в формат Ansi — отделяем первый символ, дописываем точку и обратно преобразовываем в Utf8 формат.



Строки в среде Lazarus хранятся в формате Utf8 — под каждый символ выделяется не один, а несколько байт (1-4 байта в зависимости к какой из национальных кодировок символ относится — Unicode).

Код обработчика представлен ниже (рис. 6.7).

```

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).

Form1

Данные по всем абитуриентам

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

Результат - отобранные абитуриенты

Вывести

№	1	2	3	4	5
Фамилия	Сидоров	Иванова	Преображен	Сухов	Майорова
И	С.	С.	К.	П.	А.
О	О.	А.	М.	П.	Ю.

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 двумерный массив (матрицу) размером 4x4 элемента, 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 рис 7.1.

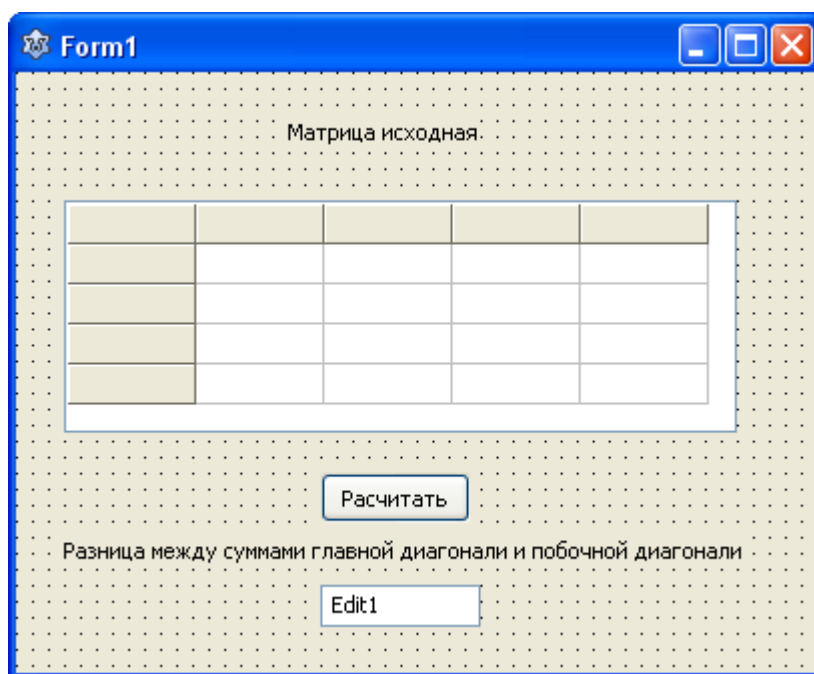


Рис. 7.1 - Интерфейс программы

➡ 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).

The screenshot shows a Windows form titled 'Form1' with a blue title bar. The form has a light gray background with a dotted grid. At the top, the text 'Матрица исходная:' is displayed. Below it is a 5x5 grid. A button labeled 'Расчитать' is positioned below the grid. At the bottom, the text 'Разница между суммами главной диагонали и побочной диагонали' is displayed above an empty text box.

Рис. 7.2 - Настройка компонентов интерфейса

➡ Сохраним проект и запустим на выполнение.

➡ Запустив программу введем несколько значений в таблицу (рис. 7.2).

The screenshot shows the same form as in Figure 7.2, but now it is running. The grid contains the following values:

5	8			
	4			

The cell containing '4' is highlighted with a red dotted border. The button 'Расчитать' and the text 'Разница между суммами главной диагонали и побочной диагонали' are still visible at the bottom.

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 (рис. 7.4).

```

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 квадратных 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» (рис. 7.5).



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 равно 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.

```

{ TForm1 }

35
procedure TForm1.Button1Click(Sender: TObject);
var
  i, j : integer;
  S1, S2, R : real;
40 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;
45  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];
50  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.

Пример работы программы представлен на рис. 7.6.

5	6	7	8
1	2	3	4
9	8	7	6
5	4	3	2

Расчитать

Разница между суммами главной диагонали и побочной диагонали

-8

Рис. 7.6 - Результат выполнения программы

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

Лабораторная работа №8

Тема: Процедуры и Функции

Задание: То же задания как в лабораторной №4 только расчет S_i - оформить в виде функции с параметрами (два параметра S_{i-1} и S_{i-2} . и создать процедуру в качестве параметров ей также передаются значения S_{i-1} и S_{i-2} - по значению, а третий параметр — рассчитанное значение S_i - передается по ссылке.

Напомним задание к 4-й работе: Написать программу, которая выдаст ряд чисел Фибоначчи пока их сумма (ряда) не превысит 100.

➡ Возьмем программу с 4-й лабораторной и скопируем ее в паку для 8-й.

Рис. 8.1 - Интерфейс программы

Если посмотреть на исходный код обработчика кнопки «Расчитать»

```

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;
.     b := c;
55     s := s + c;
.   end;
.   Edit1.Text:=IntToStr (S);
. end;

```

Рис. 8.2 - Исходный код кнопки "Рассчитать"

Разобравшись можно понять, что расчет последующего числа ряда Фибоначчи осуществляется в 52 строчке кода, а все остальное это вспомогательная часть программы служащая для вывода данных в таблицу, запоминания предыдущих двух чисел ряда Фибоначчи, подсчет суммы ряда и проверка на не превышение этой суммы числа 100.

Как видно из кода каждое последующее число Фибоначчи зависит только от двух предыдущих. Напишем функцию расчета следующего числа в ряде на основе двух предыдущих. и поместим ее выше обработчика кнопки «Рассчитать».

➡ Напишем заготовку пустой функции.

Любая функция начинается с ключевого слова function, далее идет ее имя — мы придумаем его самостоятельно,

```

35 function FibonacciF(a,b:integer):integer;
. begin
.
. end;

```

Рис. 8.3 - Пустая функция

оно не должно совпадать с ключевыми словами и именами уже зарезервированными под переменными и функциями. За именем в круглых скобках идут параметры функции — в нашем случае две переменные a и b целого

типа. Функция при вызове всегда возвращает какое то значение то тип возвращаемого значения описывается за скобками — в нашем случае целый. Тело функции (то что она делает) описывается между ключевыми словами `begin` и `end`. В нашем случае функция должна сложить переменные `a` и `b` и выдать сумму в качестве результата. Функция возвращает то значение, что мы присвоим переменной `Result`.

➡ Добавим расчет в нашу функцию. Код функции следующий:

➡ В обработчике кнопки заменим явный расчет следующего числа в ряде Фибоначчи на вызов нашей функции.

```
35 function FibonacciF(a,b:integer):integer;
. begin
.     Result := a + b;
. end;
```

Рис. 8.4 - Расчет следующего элемента ряда Фибоначчи, на основе предыдущих двух

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

Рис. 8.5 - Замена явного расчета чисел Фибоначчи на вызов функции

Следует обратить внимание на строчку 57 (ее номер поменялся т. к. вверху мы добавили несколько строк кода — функция расчета).

Если необходимо, что бы функция возвращала не одно значение, а

несколько необходимо результат возвращать через глобальные переменные (которые видны в функции и вызывающей программе, что является не безопасным) или в виде записи или класса или передавать переменные в процедуру не по значению, а по ссылке. При передачи данных по ссылке (с помощью ключевого слова `var` в описании параметров функции) любые изменения переменной внутри процедуры повлекут изменение этой переменной вне процедуры (фактически процедура и вызывающая программа работают с одной и той же областью памяти). Если параметры передаются по значению (как в нашем примере с функцией) то передаваемые переменные копируются во временную область памяти (стек вызова) и любое их изменение внутри функции не повлечет изменение в вызывающей процедуре. В примере используется копии переменных внутри функции, а когда выходим из функции то копии теряются.

➡ Добавим В 40 `procedure FibonacciP(a,b:integer;var R: integer);`
 программу процедуру. `begin`
 Она так же будет `end;`

Рис. 8.6 - Пустая процедура

рассчитывать значение следующего ряда Фибоначчи. Процедура в отличии от функции ничего не возвращает (явно) и не может использоваться в выражениях.

Необходимо вернуть результат — следующее число ряда Фибоначчи — мы воспользуемся передачей параметра по ссылке.

Любая процедура начинается с ключевого слова `procedure`, за которым идет имя процедуры (ограничение на выбор имени тот же, что у функции). За именем функции в круглых скобках идут параметры. Первые два параметра мы передаем по значению (переменные `a` и `b`), а третий параметр `R` — по ссылке (перед ним стоит ключевое слово `var`). Тело процедуры (то, что она делает) находится между словами `begin` и `end`.

➡ Напишем расчет переменной `R`.

➡ Напишем вызов процедуры в основной обработчик нашей программы, сразу же после вызова функции.

```
40 procedure FibonacciP(a,b:integer;var R: integer);
. begin
.   R := a + b;
. end;
```

Рис. 8.7 - Расчет следующего элемента ряда Фибоначчи, на основе предыдущих двух в виде процедуры

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

Рис. 8.8 - Замена явного расчета чисел Фибоначчи на вызов процедуры

В примере дважды рассчитывается каждое число ряда Фибоначчи разными способами.

➡ Напишем процедуру (назовем ее InitStringGrid) вывод в таблицу первых двух элементов ряда Фибоначчи. Процедура не имеет параметров, т. к. ее действия не зависят от каких либо переменных и первые

```
45 procedure InitStringGrid;
. begin
.   Form1.StringGrid1.ColCount:=2;
.   Form1.StringGrid1.Cells[0,0] := '1';
.   Form1.StringGrid1.Cells[0,1] := '1';
50   Form1.StringGrid1.Cells[1,0] := '2';
.   Form1.StringGrid1.Cells[1,1] := '1';
. end;
```

Рис. 8.9 - Процедура вывода первых двух элементов ряда Фибоначчи

два элемента ряда Фибоначчи всегда одинаковые 1 и 1. Данная процедура постоянно обращается к таблице (StringGrid1), которая находится на форме

(Form1).

Процедуры и функции наиболее интенсивно работающие с классом рекомендуется вносить в этот класс. Их можно разместить в одном из трех разделов класса с различной защитой:

private - ограничен доступ методами класса, который содержит объявление поля, процедуры или функции.

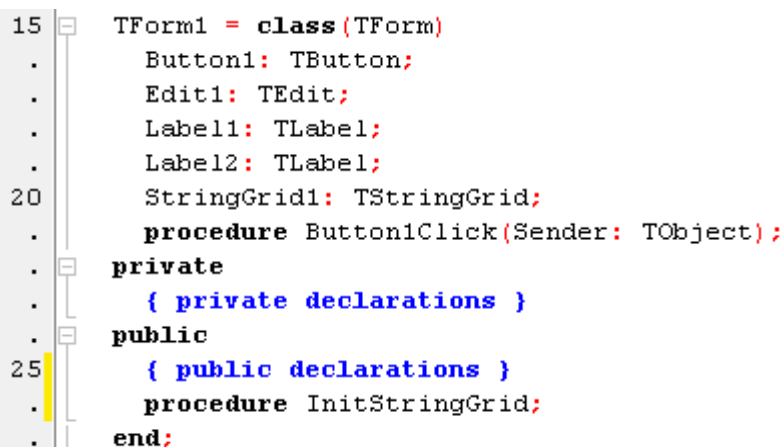
protected - доступны только методам самого класса, а также любым его потомкам, независимо от того, находятся ли они в том же модуле или нет.

public — видны всем классам, процедурам и функциям в любом модуле.

Поместим процедуру `InitStringGrid` в раздел **public** – сделав ее общедоступной. В данном примере не играет роли в кой из разделов ее помещать, т. к. программа состоит из одного модуля и одного класса.

➡ Добавим в класс объявление процедуры `InitStringGrid` (строка 26).

Саму процедуру также необходимо изменить — добавить префикс перед именем процедуры - имя класса (`TForm1`), к которому она относится, а в теле



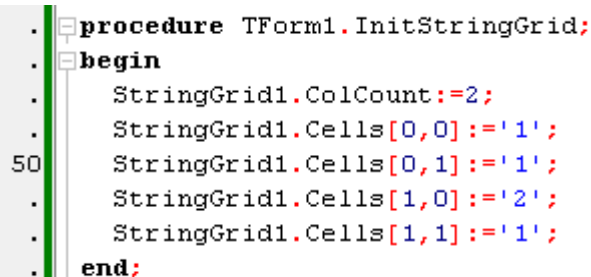
```

15  TForm1 = class(TForm)
    .   Button1: TButton;
    .   Edit1: TEdit;
    .   Label1: TLabel;
    .   Label2: TLabel;
20   StringGrid1: TStringGrid;
    .   procedure Button1Click(Sender: TObject);
    . private
    .   { private declarations }
    . public
25   { public declarations }
    .   procedure InitStringGrid;
    . end;
  
```

Рис. 8.10 - Добавление процедуры класса

процедуры необходимо убрать явное обращение к `Form1`. Данная процедура стала процедурой класса и она знает о всех методах и полях своего класса.

➡ Заменим в основной процедуре расчета вывод первых двух элементов ряда Фибоначчи на вызов процедуры.



```

    . procedure TForm1.InitStringGrid;
    . begin
    .   StringGrid1.ColCount:=2;
    .   StringGrid1.Cells[0,0] := '1';
50  StringGrid1.Cells[0,1] := '1';
    .   StringGrid1.Cells[1,0] := '2';
    .   StringGrid1.Cells[1,1] := '1';
    . end;
  
```

Рис. 8.11 - Процедура класса

```

55 procedure TForm1.Button1Click(Sender: TObject);
. var
.   a,b,c : integer;
.   s : integer;
. begin
60   InitStringGrid;
.   a:=1;
.   b:=1;
.   s := a+b;

```

Рис. 8.12 - Замена явного вывода в StringGrid на вызов процедуры

➡ Напишем

процедуру добавляющую столбец в таблицу и выводящую номер столбца и значение, переданное в эту процедуру. Назовем ее OutputToStringGrid. Данная процедура интенсивно работает с формой — внесем ее в класс.

```

15 TForm1 = class(TForm)
.   Button1: TButton;
.   Edit1: TEdit;
.   Label1: TLabel;
.   Label2: TLabel;
20   StringGrid1: TStringGrid;
.   procedure Button1Click(Sender: TObject);
. private
.   { private declarations }
. public
25   { public declarations }
.   procedure InitStringGrid;
.   procedure OutputToStringGrid( R : Integer);
. end;

```

Рис. 8.13 - Добавление процедуры OutputToStringGrid в класс TForm1

Напишем процедуру вывода элементов ряда Фибоначчи.

```

. procedure TForm1.OutputToStringGrid( R : Integer);
. begin
.   StringGrid1.ColCount:= StringGrid1.ColCount+1;
.   StringGrid1.Cells[StringGrid1.ColCount-1,0] := IntToStr(StringGrid1.ColCount);
60   StringGrid1.Cells[StringGrid1.ColCount-1,1] := IntToStr(R);
. end;

```

Рис. 8.14 - Процедура вывода следующего ряда Фибоначчи в таблицу

➡ Заменим в основной процедуре расчета вывод в таблицу на вывод через процедуру OutputToStringGrid.

➡ Сохраним программу и запустим на выполнение — проверим ее работоспособность.

```

. while s < 100 do
.   begin
.     c := FibonacciF(a,b);
75   FibonacciP(a,b,c);
.     OutputToStringGrid(c);
.     a := b;
.     b := c;
.     s := s + c;
80   end;

```

Рис. 8.15 - Применение процедуры OutputToStringGrid

The screenshot shows a Windows application window titled "Form1". Inside the window, the text "Массив чисел Фибоначчи" (Fibonacci array) is centered. Below it is a table with two rows of numbers. The first row contains the numbers 5, 6, 7, 8, 9, and 10. The second row contains the numbers 5, 8, 13, 21, 34, and 55. Below the table is a horizontal scrollbar. At the bottom of the window, there is a label "Сумма элементов" (Sum of elements) followed by a text box containing the value "143". To the right of the text box is a button labeled "Расчитать" (Calculate).

5	6	7	8	9	10
5	8	13	21	34	55

Сумма элементов: 143

Расчитать

Рис. 8.16 - Внешний вид программы после расчета

Внешний вид программы и выдаваемые результаты не поменялись, а внутренняя структура существенно изменилась и стала более наглядной и простой в понимании. Рекомендуется выделять в отдельные процедуры и функции, те части кода, которые по смыслу выполняют совместную работу.

Список рекомендованной литературы

1. **Симонович С.В.** Информатика: базовый курс, учеб. пособие для вузов / под ред. С. В. Симоновича; 2-е изд. - Питер, СПб.: 2008. - 640 с. – ISBN 978-5-94723-752-8
2. **Ремнев А. А.** Курс Delphi для начинающих: полигон нестандартных задач, с компакт-диском / Федотова С. В.; - СОЛОН-Пресс, М.: 2007. - 360 с. – ISBN 5-98003-241-X
3. **Меняев М. Ф.** Информатика и основы программирования: учеб. пособие для вузов / 2-е изд., перераб. и доп. - Омега-Л, М.: 2006. - 458 с. – ISBN 5-365-00151-6
4. **Чиртик А. А.** Delphi / Борисок В. В., Корвель Ю. И.; - Питер, СПб.: 2007. - 400 с. - ISBN 978-5-91180-219-6
5. **Емельянов В. И.** Основы программирования на Delphi : учеб. пособие для вузов / Воробьев В. И., Тюрина Т. П.; под ред. В.М.Черненкого; - Высшая школа, М.: 2005. - 231 с. – ISBN 5-06-004869-1
6. **Прищепов М. А.** Программирование на языках Basic Pascal и Object Pascal в среде Delphi: учеб. пособие / Севернёва Е. В., Шакирин А. И.; под ред. М. А. Прищепова; - ТетраСистемс, Минск: 2006. - 320 с. – ISBN 985-470-396-7
7. **Зубов В. С.** Object Pascal. Практикум в среде Delphi/ - МЭИ, М.: 2004. - 272 с. - ISBN 5-7046-0886-8
8. **Архангельский А. Я.** Программирование в Delphi: учебник по классическим версиям Delphi/ - Бином-Пресс, М.: 2006. - 1152 с. – ISBN 5-9518-0152-4
9. **Хомоненко А. Д.** Самоучитель Delphi/ Гофман В. Э.; - БХВ-Петербург, СПб.: 2003. - 576 с. – ISBN 5-94157-384-7

10. **Бобровский С. И.** Delphi 7: учеб. Курс / - Питер, М.-СПб.: 2004. -736 с. – ISBN 5-8046-0086-9
11. **Баженова И. Ю.** Delphi 7: Самоучитель программиста, учеб.-справ. Пособие / - КУДИЦ-ОБРАЗ, М.: 2003. - 448 с. – ISBN 5-93378-072-3

Интернет ресурсы:

<http://www.ПГСГиФ.рф/>

<http://www.pgsgif.ru/>

<http://www.pgsgif.ru/node/17> — Информатика 2-й семестр.

<http://www.freepascal.ru/>

<http://lazarus.freepascal.org/>

<http://delphigl.com/>

<http://www.delphi-manual.ru/>

<http://www.delphilab.ru/>