МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

МЕТОДИЧЕСКОЕ ПОСОБИЕ

К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

В СРЕДЕ DELPHI

Донецк ДонНТУ 2007

УДК 681.3.06(071)

Методическое пособие к выполнению лабораторных работ в среде Delphi/ Сост.: Л. В. Незамова, И. Ю. Анохина. – Донецк, ДонНТУ, 2007. – 52 с.

В методическом пособии приведены справочные сведения по языку Object Pascal в Delphi 3, рассмотрена технология визуального программирования, приведены примеры решения типовых задач.

Методическое пособие можно использовать в качестве основы курсов лекций, в которых изучаются соответствующие программные продукты. Пособие предназначено для студентов, аспирантов, преподавателей.

Авторы: Л.В. Незамова, ассистент
И.Ю. Анохина, доцент

Рецензент

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Середина 90-х годов XX столетия ознаменована массовым переходом на 32-разрядные операционные системы семейства Windows. На смену структурному программированию пришло объектно-ориентированное программирование. Такой скачок послужил основой для создания многочисленных технологий.

В свете этих нововведений компания Borland представила на суд программистской общественности новый программный продукт. Delphi - этим именем был назван новый программный продукт с феноменальными характеристиками.

Delphi - это комбинация нескольких важнейших технологий:

- **Высокопроизводительный компилятор в машинный код.**
  Компилятор, встроенный в Delphi, обеспечивает высокую производительность, необходимую для построения приложений.

- **Объектно-ориентированная модель компонент.**
  Основной упор этой модели в Delphi делается на максимальном использовании кода. Это позволяет разработчикам строить приложения весьма быстро из заранее подготовленных объектов, а также дает им возможность создавать свои собственные объекты для среды Delphi.

- **Визуальное (а, следовательно, и скоростное) построение приложений из программных прототипов.**
  Среда Delphi включает в себя полный набор визуальных инструментов для скоростной разработки приложений.

Программы в Delphi пишут на языке Object Pascal. Язык Object Pascal является дальнейшим объектно-ориентированным развитием широко распространенного языка Turbo Pascal. Однако если компилятор Turbo Pascal существовал в качестве самостоятельного программного продукта для среды DOS, то Object Pascal является неотъемлемой частью системы визуального проектирования Delphi.

1 СТРУКТУРА СРЕДЫ ПРОГРАММИРОВАНИЯ DELPHI

После запуска Delphi 3 на экране компьютера можно увидеть (см. рис. 1.1):

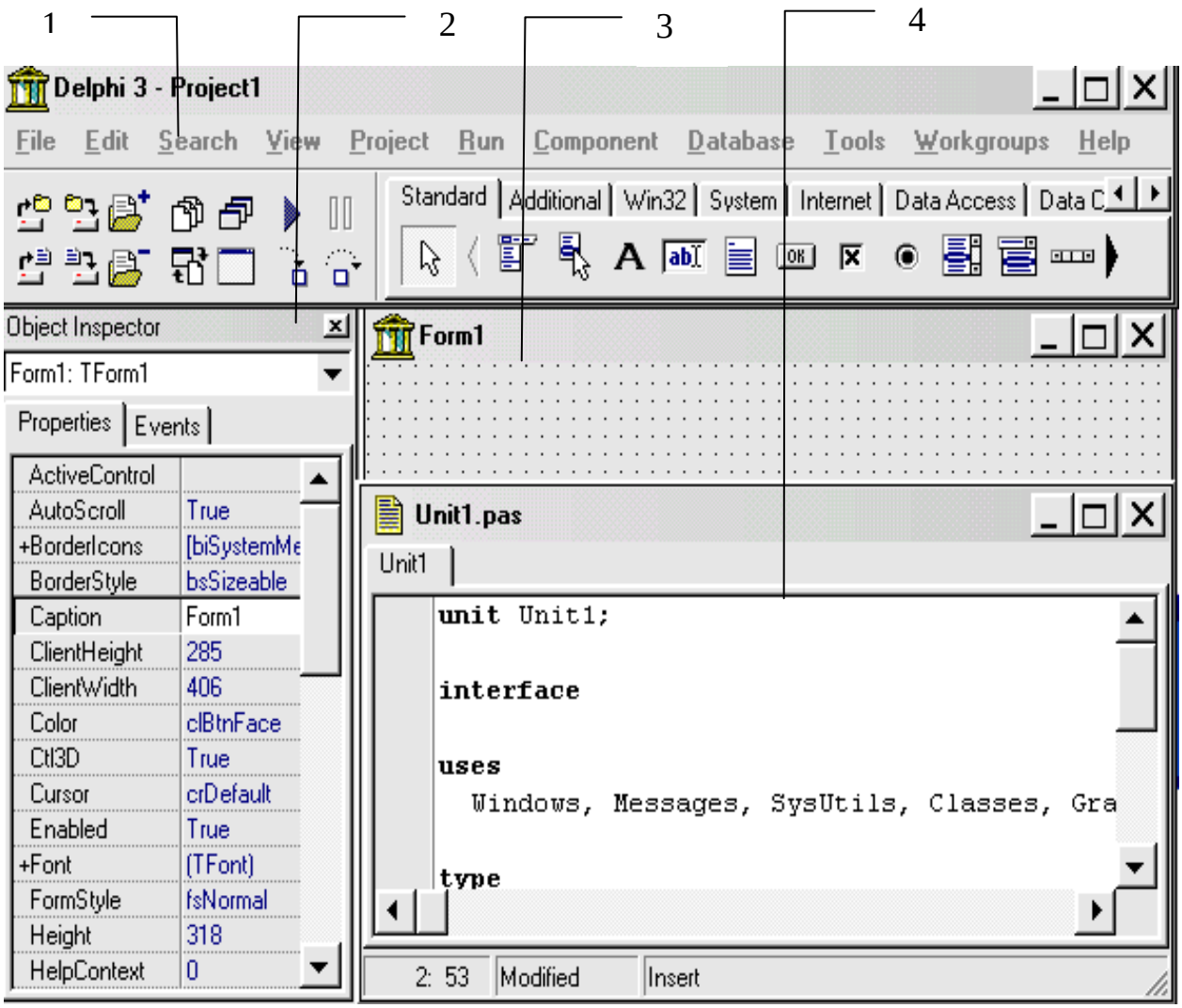

Рис. 1.1 – Окна и панели Delphi 3

Главные составные части среды программирования:

1 - главное окно осуществляет основные функции управления проектом создаваемой программы и включает в себя:
- главное меню, обеспечивающее доступ к командам среды программирования;
- панели инструментов, обеспечивающие быстрый доступ к часто используемым командам главного меню;
- палитру компонентов, содержащую набор объектов, которые можно добавлять в форму

2 - окно инспектора объектов, для изменения свойств компонентов;

3 - окно формы, которое используется для размещения компонентов Delphi;

4 - окно редактора кода, предназначенное для создания и редактирования кода программы.

1.1 Главное окно

1.1.1 Главное меню

Меню (рис. 1.2) предоставляет быстрый и гибкий интерфейс к среде Delphi, потому что может управляться по набору “горячих клавиш”. Это удобно еще и потому, что здесь используются слова или короткие фразы, более точные и понятные, нежели иконки или пиктограммы. Можно использовать меню для выполнения общих задач типа открытия и закрытия файлов, управления отладчиком или настройкой среды программирования. Назначение разделов и подразделов меню приведено в приложении А.

Рис. 1.2 – Главное меню

1.1.2 Панели инструментов

Кнопки панели инструментов выполняют много из того, что можно сделать через меню. Если задержать мышь над любой из иконок на панели инструментов, то появится подсказка, объясняющая назначение данной иконки.

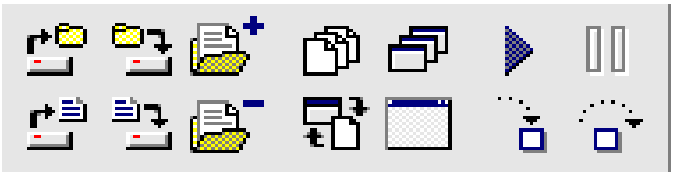

Рис. 1.3 – *Панели инструментов*

1.1.3 Палитра компонентов

Под компонентом понимается некий функциональный элемент, содержащий определенные свойства и размещаемый в окне формы. Для размещения элемента на форме требуется щелкнуть на нужной вкладке палитры компонентов, затем на кнопке с пиктограммой соответствующего компонента и после этого щелкнуть в окне формы. Если вы забыли, на какой странице расположен конкретный компонент, выберите пункт Component List (Компоненты) из меню View (Вид), и на экране появится список компонентов в алфавитном порядке.

Не каждый компонент виден на форме во время запуска программы. Например, размещение на форме компонента MainMenu приводит к появлению в разрабатываемом приложении меню, но соответствующая пиктограмма во время запуска программы не отображается. Размеры невидимого компонента в процессе разработки не изменяются, он всегда отображается в виде пиктограммы.

Основная палитра компонентов Delphi имеет двенадцать страниц.

Standard. Большинство компонентов на этой странице (рис. 1.2) являются аналогами экранных элементов самой Windows: меню, кнопки, полосы прокрутки и т.д. Имена и назначение компонентов этой страницы приведены в приложении Б.

Рис. 1.2 – Страница Standard палитры компонентов

Additional. Эта страница содержит более развитые компоненты. Например, компонент Outline удобен для отображения информации с иерархической структурой, MediaPlayer позволит программам воспроизводить звук, музыку и видео. Данная страница также содержит компоненты, главное назначение которых — отображение графической информации. Компонент Image загружает и отображает растровые изображения, а компонент Shape, добавит в форму окружности, квадраты и т.д.

System. На этой странице представлены компоненты, которые имеют различное функциональное назначение, в том числе компоненты, поддерживающие стандартные для Windows технологии межпрограммного обмена данными посредством OLE.

Win32. Эта страница содержит компоненты, позволяющие созданным с помощью Delphi программам использовать такие нововведения в пользовательском интерфейсе 32-разрядной Windows, как просмотр древовидных структур, просмотр списков, панель состояния, присутствующая в интерфейсе программы

Windows Explorer (Проводник), расширенный текстовый редактор и др.

Dialogs. Страница, Dialogs предоставляет программам Delphi простой доступ к стандартным диалоговым окнам (окна для операций над файлами, выбора шрифтов, цветов и т.д.).

Data Access и Data Controls. Delphi использует механизм баз данных компании Borland для организации доступа к файлам баз данных различных форматов.

Win 3.1. На этой странице, находятся компоненты Delphi 1.0, возможности которых перекрываются аналогичными компонентами Windows 95.

Internet. Страница предоставляет компоненты для разработки приложений, позволяющих создавать HTML-файлы непосредственно из файлов баз данных и других типов, взаимодействующих с другими приложениями для Internet.

Samples. Страница содержит компоненты разного назначения.

ActiveX. Эта страница содержит компоненты ActiveX, разработанные независимыми производителями программного обеспечения: сетка, диаграмма, средство проверки правописания.

QReport. Компоненты этой страницы обеспечивают простой способ создания отчетов по результатам выборки данных из базы данных.

Midas и Decision Cube. Компоненты этой страницы реализуют специальный доступ к многомерным наборам данных.

1.2 Окно инспектора объектов

Т. к. каждый компонент является объектом, то можно изменить его вид и поведение с помощью Инспектора Объектов. Информация в Инспекторе Объектов меняется в зависимости от объекта, выбранного на форме.

Инспектор Объектов состоит из двух страниц, каждую из которых можно использовать для определения поведения данного компонента. Первая страница - это список свойств, вторая - список событий. Совокупность свойств отображает видимую сторону компонента: его размеры, цвет, шрифт и цвет надписи и т.д.; совокупность событий – его поведенческую сторону.

Строки таблицы выбираются щелчком мыши и могут отображать простые или сложные свойства. Простые свойства определяются единичным значением – числом, строкой символов. Сложные свойства определяются совокупностью значений. Слева от имени таких свойств указывается значок “+” и двойной щелчок на имени свойства раскроет список составляющих сложного свойства.

1.3 Окно формы

Дизайнер Форм в Delphi прост в использовании. Дизайнер Форм первоначально состоит из одного пустого окна, которое Вы заполняете всевозможными объектами, выбранными на Палитре Компонент.

1.4 Окно редактора кода

В создаваемой программе вручную можно запрограммировать последовательность действий, которые будут выполняться при нажатии на некоторую кнопку, расположенную на форме. Чтобы создать первоначально пустую подпрограмму, вызываемую при нажатии на эту кнопку, надо дважды щелкнуть на ней мышкой. Delphi откроет окно редактора кода, автоматически сгенерирует нужный текст и разместит курсор в том месте, где можно начать описание нужного алгоритма. Например, первоначально созданная подпрограмма, для некоторой кнопки может иметь вид (рис. 1.3):

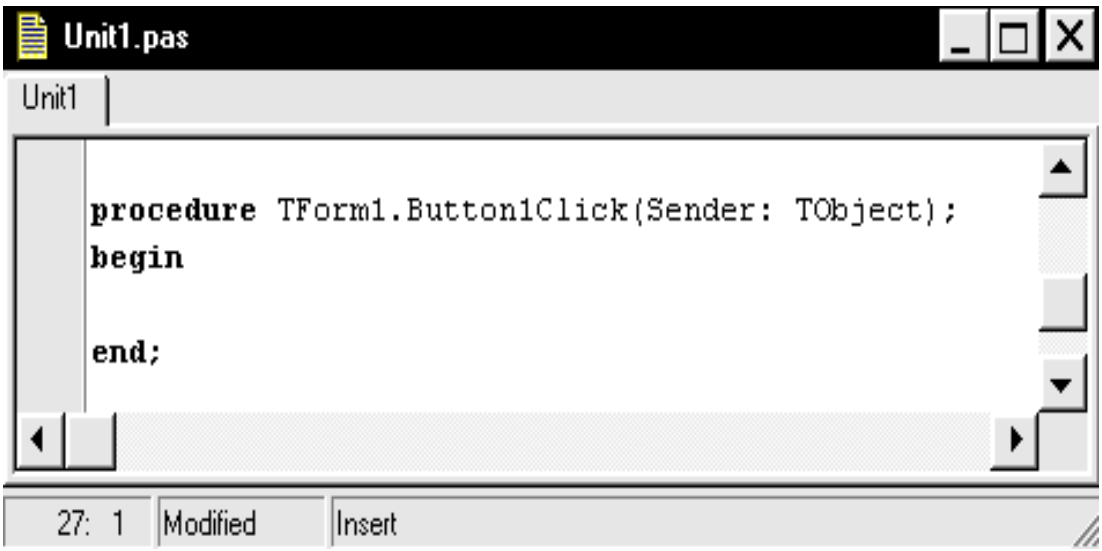

Рис. 1.3 – Окно редактора кода, содержит заготовку обработчика события

Обработчик события представляет собой процедуру и имеет имя. Имя состоит из двух частей, разделенных между собой точкой. Первая часть представляет собой имя класса создаваемой формы. Вторая часть может быть создана программистом или автоматически сформирована Delphi, если программист не ввел свое имя. В том случае, если имя сформировано автоматически, то оно будет состоять из имени компонента и имени события без предлога On (см. рис. 1.3). Тело процедуры ограничено зарезервированными словами begin…end, между которыми можно разместить свои операторы. Некоторые из используемых операторов могут изменять значения свойств компонентов, расположенных на форме. Т.е. изменять значения свойств компонентов можно не только при помощи Инспектора Объектов, но и в процессе выполнения программы.

Между содержимым окна формы и окна редактора кода существует неразрывная связь, которая контролируется Delphi. Заготовка для обработчика событий создается автоматически и эту заготовку можно наполнить конкретным содержанием, записать программный код (подпрограмму), которая будет реализовывать некоторый алгоритм. При этом **следует помнить, нельзя удалять из текста программы те строки, которые вставила туда среда Delphi**.

2 ФАЙЛЫ ПРОЕКТА DELPHI

2.1 Основные файлы проекта

Проект Delphi состоит из форм, модулей, установок параметров проекта, ресурсов и т.д. Вся эта информация размещается в файлах. Многие из этих файлов автоматически создаются Delphi при разработке приложения. Кроме того, компилятор также создает файлы.

Когда Вы проектируете приложение, то в папке в которой Вы сохранили проект, Delphi создаст следующие файлы:

- Файл проекта (.DPR)
  Этот файл хранит информацию о формах и модулях. Содержит он основной код программы, ссылки на все окна (формы) проекта и относящиеся к ним модули. В нем также содержится код инициализации. Имеет одноименное название с проектом.
- Файл модуля (.PAS)
  Содержит текст, который Вы видите в окне редактора кода так называемого модуля программы.
- Файл формы (.DFM)
  Используется для хранения информации о внешнем виде формы. Каждому файлу формы соответствует файл модуля (.PAS)
- Файл ресурсов (.RES)
  Содержит используемую проектом пиктограмму и прочие ресурсы.
- Файл (.DSK)
  Содержит информацию о состоянии рабочего пространства.
- Файл конфигурации проекта(.CFG)
  В этом файле хранятся установки проекта.

Группа файлов, которые создаются компилятором

- Исполняемый файл (.EXE)
  Откомпилированная программа. Сохраняется автоматически при запуске проекта на выполнение. Обновляется в момент компиляции. Полностью самостоятельное приложение.
- Объектный файл модуля (.DCU)
  Откомпилированный файл модуля, который компонуется в окончательный исполняемый файл
- Динамически присоединяемая библиотека (.DLL)
  Файл создается, если Вы проектируете свою собственную библиотеку.

2.2 Структура модуля

Любая программа в Delphi состоит из файла проекта (главного модуля - файла с расширением DPR) и одного или нескольких модулей (файлы с расширением PAS). Файлы проекта создаются автоматически самой системой программирования Delphi и не предназначены для редактирования пользователем. Именно по этой причине файл проекта имеет особое расширение и обычно не показывается в окне кода. Операторы исполнимой части обеспечивают инициализацию приложения и вывод на экран стартового окна.

Помимо главного модуля каждая программа включает как минимум один модуль формы, который содержит описание стартовой формы приложения и поддерживающих ее работу процедур.

Модули – это программные единицы, предназначенные для размещения фрагментов программ. С помощью содержащегося в них программного кода реализуется поведенческая сторона программы.

Каждый модуль в общем случае имеет структуру:

```
unit <имя модуля>;

interface    // Открытый интерфейс модуля

{Сюда могут помещаться списки подключаемых модулей, объявления типов, констант, переменных, функций и процедур, к которым будет доступ из других модулей.}

implementation // Реализация модуля

{Сюда могут помещаться списки подключаемых модулей, объявления типов, констант, переменных, функций и процедур, к которым не будет доступа из других модулей. Тут же должны быть реализации всех объявленных в разделе interface функций и процедур, а также могут быть реализации любых дополнительных, не объявленных ранее функций и процедур.}

initialization // не обязательный
{Операторы, выполняемые один раз при первом обращении к модулю}

finalization // не обязательный
{Операторы, выполняемые при любом завершении работы модуля}

end.
```

Пример содержимого модуля после загрузки среды Delphi .

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs;
type
  TForm1 = class (TForm)

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

Implementation

{$R *.DFM}
end.
```

Весь текст сформирован Delphi, но в отличии от файла проекта содержимое модуля можно изменять, придавая программе нужную функциональность.

Начинается модуль словом **unit**, за которым следует имя модуля.

Модуль может состоять из четырех разделов: интерфейса, реализации, инициализации и завершающей части.

В раздел **interface** перечислены (после слова uses) стандартные модули, используемые данным модулем, а также находится сформированное Delphi описание типа формы, которое следует за словом type.

Раздел **Implementation** содержит объявления локальных переменных, процедур и функций**,** поддерживающих работу формы. Вначале раздела реализации располагается директива {$R *.dfm}, указывающая компилятору, что в раздел реализации надо вставить команды установки значений свойств формы, которые находятся в файле с расширением .dfm

За директивой {$R *.dfm} располагаются описания процедур обработки событий формы, здесь можно поместить описание своих процедур и функций, которые могут вызываться из процедур обработки событий.

Раздел **initialization** включает в себя операторы, которые выполняются только один раз при первом обращении программы к данному модулю.

Раздел **finalization** включает в себя операторы, которые выполняются только один раз при любом завершении программы: нормальном или аварийном.

Инициирующая и завершающая части являются необязательными.

Текст модуля доступен как Delphi, так и Вам. Delphi автоматически вставляет в текст модуля описание любого добавленного к форме компонента, а также создает заготовки для обработчиков событий. При написании программного кода соблюдайте правило: **нельзя удалять строки, которые вставлены не Вами, а Delphi.**

2.3 Элементы программы

Элементы программы – это минимальные неделимые ее части, несущие в себе определенную значимость для компилятора. К элементам программы относятся:

- зарезервированные слова;
- идентификаторы;
- типы;
- константы;
- переменные;
- метки;
- подпрограммы;
- комментарии.

Зарезервированные слова – это английские слова, указывающие компилятору на необходимость выполнения определенных действий, т.е. существует некоторый набор служебных слов, назначение которых в языке строго определено (зарезервировано) и которые не могут быть использованы с другой целью.

Зарезервированные слова используются: для описаний (const - констант, var – переменных), некоторых операций (div – целочисленного деления), для обозначения операторов (if, then, else – оператора условного перехода.) и т.д.

Идентификаторы (имена) служат для обозначения различных объектов и отличаются от служебных слов тем, что назначаются по следующему правилу. Идентификаторы состоят из латинских букв, арабских цифр, знака подчеркивания и должны начинаться с буквы. Длина должна быть не более 64 символа.

Тип определяет множество допустимых значений, которые может иметь тот или иной объект, а также множество допустимых операций, которые применимы к нему. Кроме того, тип определяет также и формат внутреннего представления данных в памяти ПК.

Символьный тип

Символьный тип называется Char. Значением символьного типа является множество всех символов ПК. Символьный тип занимает один байт в памяти и может содержать 255 возможных значений символов, что соответствует стандартной кодировке ANSI. Для кодировки в Windows используется код ANSI (назван по имени American National Standard Institute – американского института стандартизации, предложившего этот код).

Целочисленный тип

Целые типы используются для хранения и преобразования целых чисел (чисел, не имеющие дробной части). Названия целых типов, их длина и диапазон возможных значений приводится в табл.2.1

Таблица 2.1 – Целые типы

Название	Значение	Длина, байт
Byte	0…255	1
ShortInt	-128…127	1
SmallInt	-32768…32767	2
Word	0…65535	2
Integer	-2147483648…2147483647	4
LongInt	-2147483648…2147483647	4
Cardinal	0…2147483647	4

Вещественный тип

Вещественные типы предназначены для представления действительных чисел, т.е. чисел состоящих из целой и дробной части. Информация о переменных вещественного типа приводится в табл. 2.2

Таблица 2.2 – Вещественный тип

Название	Значение	Количество значащих цифр	Длина, байт
Real	$2{,}9 * 10^{-39} \ldots 1{,}7 * 10^{38}$	11…12	6
Single	$1{,}5 * 10^{-45} \ldots 3{,}4 * 10^{38}$	7…8	4
Double	$5{,}0 * 10^{-324} \ldots 1{,}7 * 10^{308}$	15…16	8
Extended	$3{,}4 * 10^{-4951} \ldots 1{,}1 * 10^{4932}$	19…20	10
Comp	$-2^{-63} \ldots 2^{63} - 1$	19…20	8
Currency	± 922337203685477,5807	19…20	8

Последние два типа применяются для финансовых арифметических операций.

Тип Real оставлен для совместимости с ранними версиями Delphi и Pascal. Использование типов Single, Double, Extended эффективно, если в состав ПК входит арифметический сопроцессор.

Для работы с целыми и вещественными данными могут использоваться арифметические операции и математические стандартные функции представленные в Приложении В.

Строковый тип

В языке программирования Pascal максимальная длина строки String имела 255 символов. В Delphi оставлена такая строка, но называется она ShortString. Длина строки String меняется от 0 до 2 Гбайт. Данные в строковых, как и в символьных переменных, помещается в кавычки, которые отделяют данные от команд программы.

Переменная является идентификатором, обозначающим некоторую

область памяти, в которой хранится значение переменной. Это значение может изменяться во время выполнения программы. Переменные должны быть объявлены в разделе объявления переменных (в разделе var).

Объявление переменной имеет вид:

```
var < список идентификаторов переменных > : < тип >;
```

Например:

```
var
   i, j : integer;
  x, y : string;
```

Комментарии – тексты, поясняющие программу, но не влияющие на ход ее выполнения. Комментарии это любая последовательность символов, заключенная в фигурные скобки, например:

```
{это комментарий}
```

Такой комментарий может занимать любое число строк.

Комментарием может быть также последовательность символов, стоящая после пары символов // и до конца строки, например:

```
// это комментарий
```

Метки позволяют изменить естественный ход выполнения программы. Метки должны быть перечислены в разделе объявления меток (в разделе label). Оператор в программе можно выделит, поставив перед ним метку. Метка от оператора отделяется двоеточием.

Пример использования метки.

```
Label         // раздел описания меток
  2;
begin
    .....
    Goto 2;    // передать управление оператору, помеченному меткой 2
    ......
2:  ......      // оператору, идущему за меткой, будет передано управление
end;
```

Раздел label может отсутствовать, если в программе нет меток.

Константы могут быть обычными или именованными, т.е. константы в программе могут быть представлены либо непосредственно своим значением, либо именем. Обычная константа - это число, символ, строка или логическое

выражение. Именованная константа в отличие от обычной имеет имя. Т.е. в программе вместо указания значения константы можно использовать ее имя. Программы с группированными константами становятся более наглядными и эффективными. Именованные константы описывают в разделе описания констант (раздел const).

В общем виде именованные константы записывают следующим образом:

```
< имя константы > = < значение >;
```

Например:

```
Const
    M = 256.3 ;
    Head = ‘ Таблица ’ ;
```

Именованная константа может быть определена и с помощью константного выражения.

```
< имя константы > = < константное выражение >;
```

Например:

```
Const
    nalog = 0.13*100 ;
    name = ‘ Таблица ’ + ‘ исходных данных ‘ ;
```

Подпрограммы это специальным образом оформленные фрагменты программы. Назначение программы в целом – решение всей задачи, назначение подпрограммы – решение отдельных подзадач. Применение подпрограмм позволяет уменьшить число повторений одной и той же последовательности операторов, что позволяет уменьшить размер программы в целом и делает ее более наглядной за счет расчленения программы на группы независимых фрагментов.

Имеется два вида подпрограмм: процедуры и функции. Функция отличается от процедуры тем, что ее идентификатор можно использовать в выражениях вместе с переменными и константами, т.к. функция имеет выходной результат определенного типа.

3. СОЗДАНИЕ ОКОННЫХ ПРИЛОЖЕНИЙ

3.1 Алгоритм линейной структуры

Пример 1. Вычислить: $y = 2x^2 + x$

Блок-схема алгоритма вычисления представлена на рис. 3.1

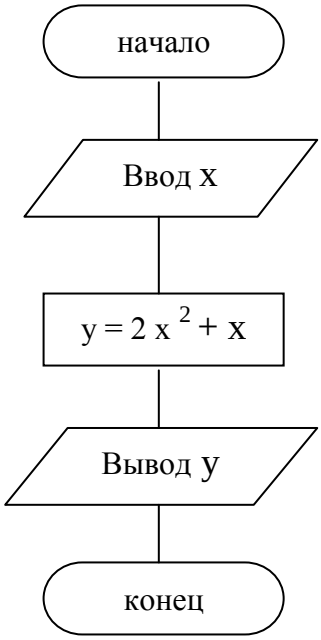

Рис. 3.1 – Блок-схема алгоритма вычисления

Окно формы, реализующей заданный алгоритм вычисления, представлен на рис. 3.2

Рис. 3.2 – Окно формы

Для создания такой формы и написания программного кода, реализующего указанный алгоритм вычисления $y = 2x^2 + x$, выполним действия, приведенные ниже.

1. Создадим папку C:\STUDENTS\PRIM1. Запустим Delphi 3.

2. Откроем новое приложение с помощью команды главного меню File - New Application.

3. На форме Form1 расположим компоненты (см. рис. 3.3) и определим их свойства.

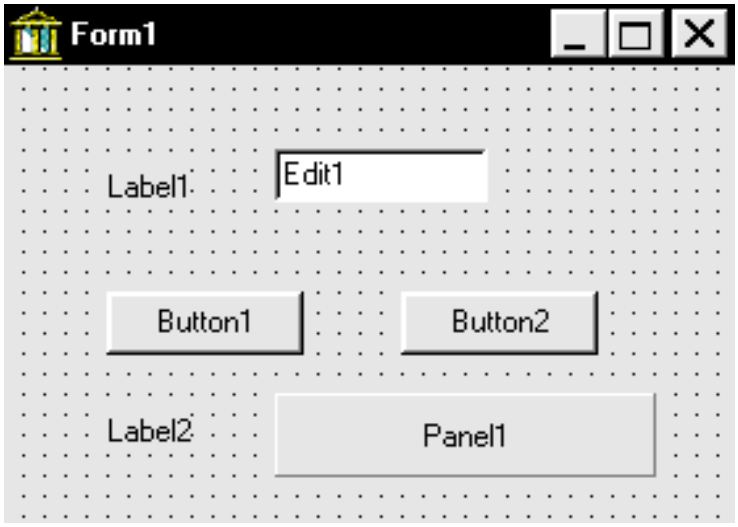

Рис. 3.3 – Размещение компонентов на форме

Все используемые компоненты расположены на странице Standard Палитры компонентов. Для размещения элемента на форме требуется щелкнуть на нужной вкладке палитры компонентов, затем на кнопке с пиктограммой соответствующего компонента и после этого щелкнуть в окне формы. После того, как все необходимые элементы разместим на форме, перейдем к формированию внешнего вида будущей программы.

Label1 и Label2 – метки – компоненты, предназначенные для отображения текстовой информации. Эти элементы будем использовать для подписи окна ввода переменной x, и панели для вывода результата переменной y. Текстовая информация для этих компонентов записывается в Свойстве Caption. Выберем объект Label1, щелкнув по нему 1 раз левой кнопкой мышки, и в Инспекторе Объектов, в свойстве Caption для этого элемента запишем введите x. То же самое выполним с элементом Label2, в свойстве Caption для него запишем – вывод y.

Edit1 – строка ввода - компонент, предназначенный для ввода символьной строки. Элемент Edit1 будем использовать для ввода значения x. Данные в этот элемент вводятся в виде символьной строки, а математические действия

производят над числами, а не над символами, поэтому в программе необходимо **предусмотреть преобразование строки в число.** Свойство Text является основным для этого элемента. Для улучшения внешнего вида программы, зададим в качестве значения этого свойства, пустую строку, т. е. очистим свойство Text.

Button1, Button2 – кнопки – компоненты, предназначенные для формирования события при нажатии на кнопку. Основное свойство для этого компонента Caption. Для кнопки Button1 в свойстве Caption запишем Расчет, для кнопки Button2 в свойстве Caption запишем Выход.

Panel1 – панель – компонент, предназначенный для вывода результата. Основное свойство для этого компонента Caption. В свойстве Caption кнопки Button1 запишем Расчет, а для кнопки Button2 в свойстве Caption запишем Выход.

4. После установления значений указанных свойств, можно переустановить и значения некоторых других свойств компонентов в соответствии с дизайном формы.

В приведенном примере, в свойстве Font (Шрифт), для компонентов Label1 и Label2 установим начертание – полужирный, размер - 12. Для выделения группы элементов, к которой будут применены действия, используют клавишу Shift. Для того, чтобы установить характеристики шрифта, выберем щелчком мыши свойство Font в Инспекторе Объектов и щелкнем по кнопке с тремя точками, появившейся в правой колонке. На экране появится окно Выбора шрифта, в котором можно выбрать соответствующие установки. Для компонентов Button1 и Button2 установим начертание – курсив, размер - 10. Для компонентов Edit1 и Panel1 установим начертание – полужирный, размер – 14.

5. Переходим к этапу написания кода. На форме расположены две кнопки: Расчет и Выход. Чтобы подключить к соответствующей кнопке ее программный код нужно выделить соответствующую кнопку и щелкнуть два раза левой кнопкой мыши. В результате, окно редактора кода станет активным, и будет содержать заготовку обработчика события (см. рис. 3.4)

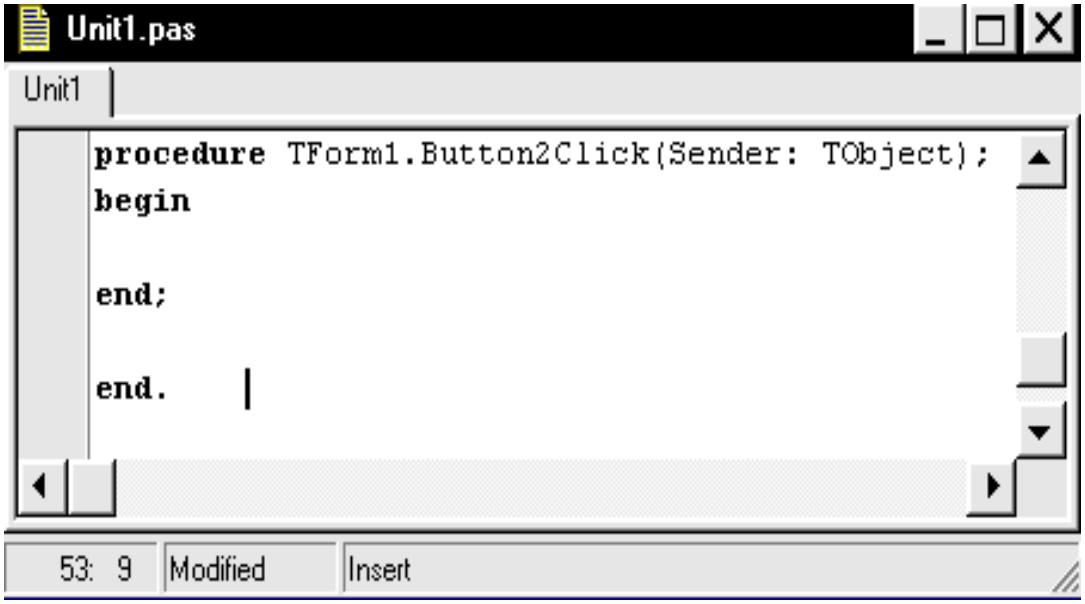

Рис. 3.4 – Окно редактора кода

Программный код для кнопки Выход содержит только одну команду Close.

Программа, написанная для кнопки Расчет, содержит следующий программный код

```
……..
Procedure TForm1.Button1Click (Sender: TObject);
  var
  x, y: integer;
begin
   x:= strtoint(Edit1.Text);
   y:= 2*sqr(x)+x;
   Panel1.Caption:=inttostr(y);
end;
……..
```

В разделе описания переменных описаны переменные x и y как целые (тип integer). Свойство Text является основным для компонента Edit1 и предназначено для ввода символьной информации, в данном примере для ввода переменной x. Функция StrToInt преобразует строку в целое число (т. е. преобразует переменную из строковой в числовую переменную целого типа). Если необходимо было бы преобразовать строку в действительное число (т. е. преобразовать переменную из строковой в числовую переменную вещественного типа), то использовали бы функцию StrToFloat. Далее следует сама функция

$y=2x^2+x$, реализованная средствами языка Object Pascal в виде y:=2*sqr(x)+x. Полученное значение необходимо вывести на экран. Для этого используется компонент Panel (панель) и его основное свойство Caption. Т. к. полученное значение y является числовым, а значением свойства Caption является символьная строка, то производят преобразование числовой переменной целого типа в символьную, с помощью функции IntToStr.

6. После написания программного кода, программу надо сохранить. Для сохранения программы выполните команды File – Save All. В появившемся диалоговом окне, в поле адреса, надо найти свою папку PRIM1 (полный путь C:\STUDENTS\PRIM1). После того как нужная папка открыта, в поле Имя файла следует ввести имя сохраняемого модуля (по умолчанию это имя Unit1). После этого появится диалоговое окно сохранения проекта и следует ввести имя проекта (по умолчанию проекту будет присвоено имя Project 1).

7. Сохранив проект, запустите его на выполнение, нажав клавишу F9 или же выбрав команды меню Run – Run. В случае возникновения ошибок, ошибки необходимо исправить, еще раз сохранить свою работу и запустить проект на выполнение.

8. В случае отсутствия ошибок появится окно формы, в котором вы сможете ввести исходные данные и получить результат (см. рис. 3.2).

Текст модуля Unit1.pas, сформированного пользователем и средой Delphi3

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs, ExtCtrls, StdCtrls;
Type
  TForm1 = class(TForm)
   Label1: TLabel;
```

```
    Edit1: TEdit;
    Label2: TLabel;
    Button1: TButton;
    Button2: TButton;
    Panel1: TPanel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.DFM}
procedure TForm1.Button1Click(Sender: TObject);
  var
  x,y:integer;
begin
   x:= strtoint(Edit1.Text);
   y:= 2*sqr(x)+x;
   Panel1.Caption:=inttostr(y);
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
  close
end;
end.
```

Рис. 3.5 - Линейный вычислительный процесс

3.2 Разветвляющийся вычислительный процесс

В разветвляющемся вычислительном процессе проверяется некоторое условие и в зависимости от того, выполняется оно или нет, вычисления идут по одной либо другой ветви.

Пример 2. Вычислить значение переменной y, при заданном значении x.

$$y = \begin{cases} x + 4, \text{если } x \geq 5 \\ x^2 \quad , \text{если } x < 5 \end{cases}$$

Блок-схема алгоритма вычисления представлена на рис. 3.6

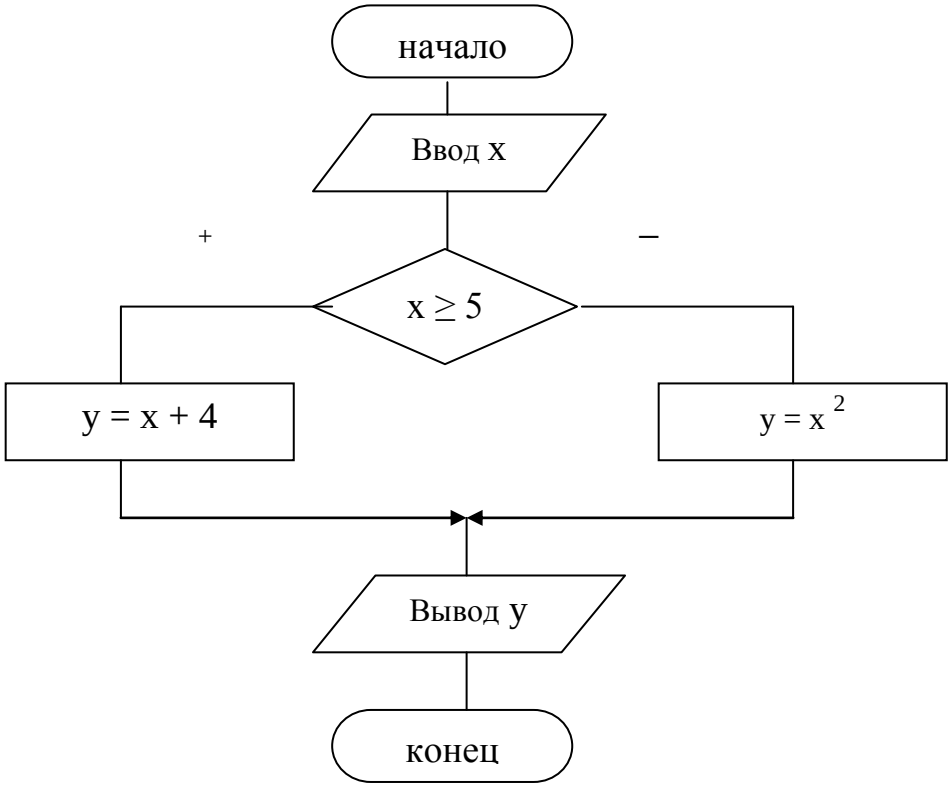

Рис. 3.6 – Блок-схема алгоритма (пример 2)

В программе, для реализации разветвляющегося вычислительного процесса используют **оператор if.** Оператор имеет следующий вид:

If <логическое выражение> then <оператор1> else <оператор2>;

Оператор if вычисляет значение логического выражения. Если значение логического выражения истинно (условие выполняется), то выполняется оператор1, следующий за словом then. Если значение логического выражения ложно (условие не выполняется), то выполняется оператор2, следующий за словом else.

В состав условного оператора может входить только один оператор. Если в какую-либо ветвь разветвления требуется вставить несколько операторов, то они объединяются в один, **составной оператор**, для чего в начале группы этих операторов записывается слово begin, а в конце слово end.

Составной оператор имеет следующий вид:

```
begin
    <оператор 1>;
    <оператор 2>;
        …
    <оператор N>
end
```

Оператор безусловного перехода goto. Оператор goto производит передачу управления к оператору, помеченному указанной меткой. В общем виде оператор записывается следующим образом:

goto < метка >;

Метка – это произвольный идентификатор либо целое число без знака. Метка должна быть описана в разделе описания меток, прежде чем на нее будет поставлена ссылка в теле программы. Метка может стоять только перед одним оператором и отделяться от него двоеточием.

Окно сообщений **ShowMessage** выводит простейшее сообщение для пользователя. Заголовок отображаемого окна совпадает с именем выполняемого файла приложения. В общем виде записывается следующим образом, например:

ShowMessage (‘нет решения’);

Элементы управления, располагаемые на форме можно скрывать или же наоборот отображать. Свойство **Visible** определяет, отображается ли элемент на экране. Если Visible установлено true, то элемент будет отображен, а если false, то скрыт.

Форма реализующей заданный алгоритм вычисления (пример 2) содержит те же компоненты, что и форма представленная на рис. 3.2. Отличие состоит в том, что программа, написанная для кнопки Расчет, будет содержать следующий программный код

```
……
procedure TForm1.Button1Click(Sender: TObject);
  var
  x,y: integer;
begin
   x:= strtoint(Edit1.Text);
   if x>=5 then
        y:= x+4
           else
         y:=sqr(x);
    Panel1.Caption:=inttostr(y)
end;
……
```

Запустив программу на выполнение и введя исходные, получим следующие результаты (см. рис. 3.7).

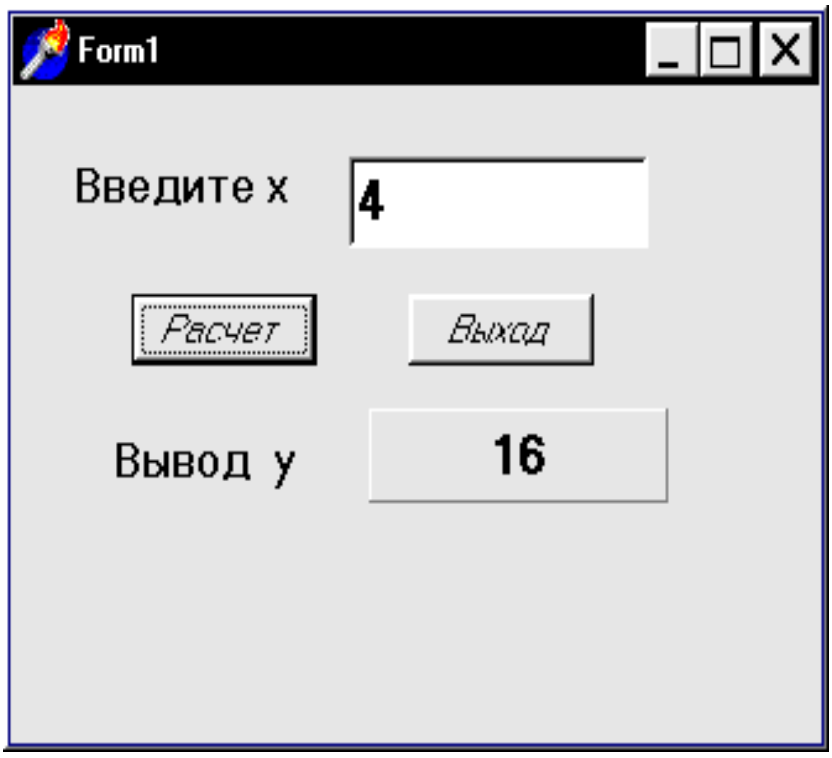

Рис. 3.7 – Окно формы

Пример 3. Вычислить значение переменной y, при заданном значении x

$$y = \frac{2x^2}{x}$$

При составлении блок-схемы к заданной задаче, необходимо предусмотреть ограничение – делить на 0 нельзя.

Блок-схема алгоритма вычисления представлена на рис. 3.8

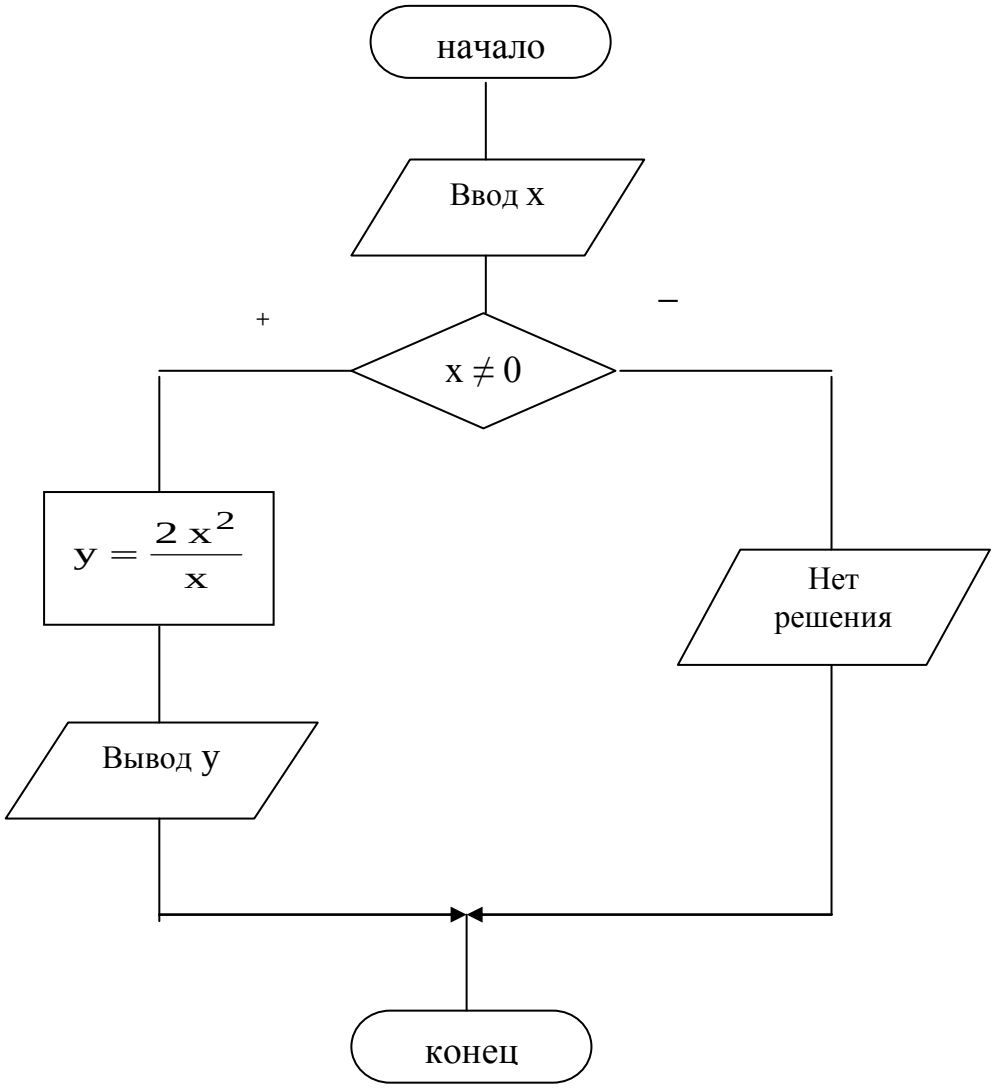

Рис. 3.8 – Блок-схема алгоритма (пример 3)

Форма реализующей заданный алгоритм вычисления содержит те же компоненты, что и форма представленная на рис. 3.2. Программа, написанная для кнопки Расчет, содержит следующий программный код

```
procedure TForm1.Button1Click(Sender: TObject);
  var
   x:integer; y:real;
begin
   x:= strtoint(Edit1.Text);
   if x<>0 then
    begin
       Label2.Visible:=true;
       Panel1.Visible:= true;
       y:= 2*sqr(x)/x;
       Panel1.Caption:=floattostr(y);
      end
          else
       begin
         Label2.Visible:=false;
         Panel1.Visible:=false;
         showmessage(' нет решения')
       end
end;
```

Запустив программу на выполнение и введя исходные данные (число отличное от 0), получим следующие результаты (см. рис. 3.9).

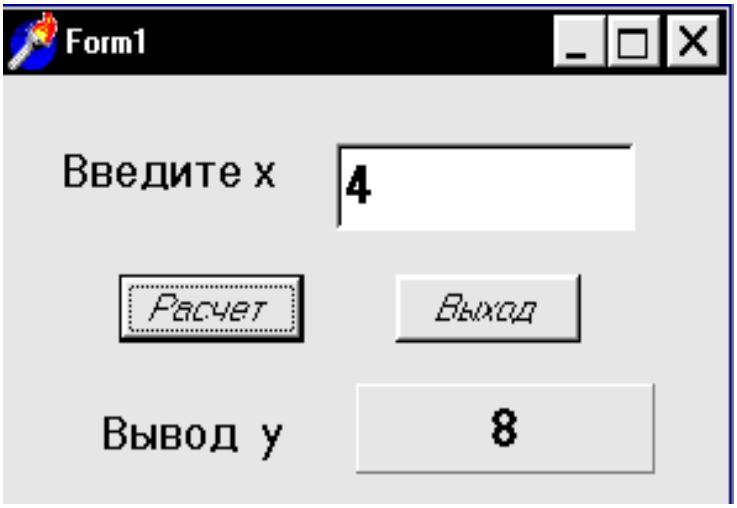

Рис. 3.9 – Окно формы (x≠ 0)

Запустив программу на выполнение и введя исходные данные - число равное 0, получим следующие результаты (см. рис. 3.10). В этом случае целесообразно скрыть некоторые компоненты: панель результата и надпись перед ней. Т. к. делить на 0 нельзя, то программа выдаст сообщение - Нет решения, чтобы закрыть это сообщение необходимо щелкнуть по кнопке OK в этом окне.

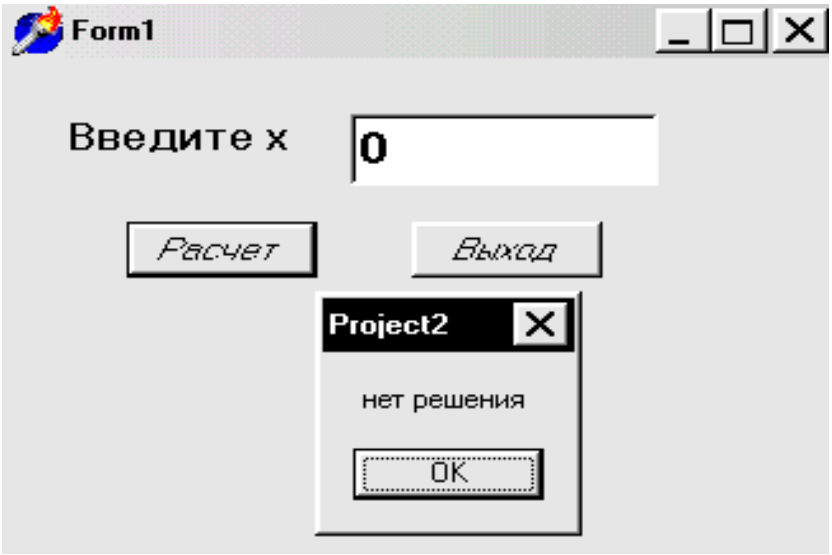

Рис. 3.10 – Окно формы (x=0)

3. 3 Циклический вычислительный процесс

Циклическим является вычислительный процесс, содержащий группу многократно повторяющихся операторов, т.е. цикл. В языке Object Pascal возможна организация трех видов циклов:

- с предусловием (while);
- с постусловием (repeat);
- с параметром (for).

Достаточно часто приходится сталкиваться с циклическими вычислительными процессами, когда число **повторений цикла неизвестно**, а задано некоторое условие его окончания (или продолжения). Для программной реализации таких вычислительных процессов в языке Паскаль существует два типа операторов: оператор цикла с предусловием и оператор цикла с постусловием. Рассмотрим организацию цикла с постусловием и предусловием на примере 4.

Пример 4. Исходные данные: -2 ≤ x ≤ 8 , Δ x =1. Вычислить значение y

$$y = \begin{cases} x + 4, \text{если } x \geq 5 \\ x^2 \quad , \text{если } x < 5 \end{cases}$$

3.3.1 Оператор цикла с предусловием

Оператор цикла с предусловием имеет следующий формат:

while <логическое выражение> do <оператор>;

Вычисляется значение логического выражения. Если это значение истинно, то выполняется оператор, стоящий после слова do. Если условие ложно с самого начала, то оператор не выполняется ни разу. Если в цикле нужно выполнить не один оператор, а несколько, то их следует заключить в операторные скобки begin … end, т.е. использовать составной оператор.

Блок-схема алгоритма вычисления представлена на рис. 3.11

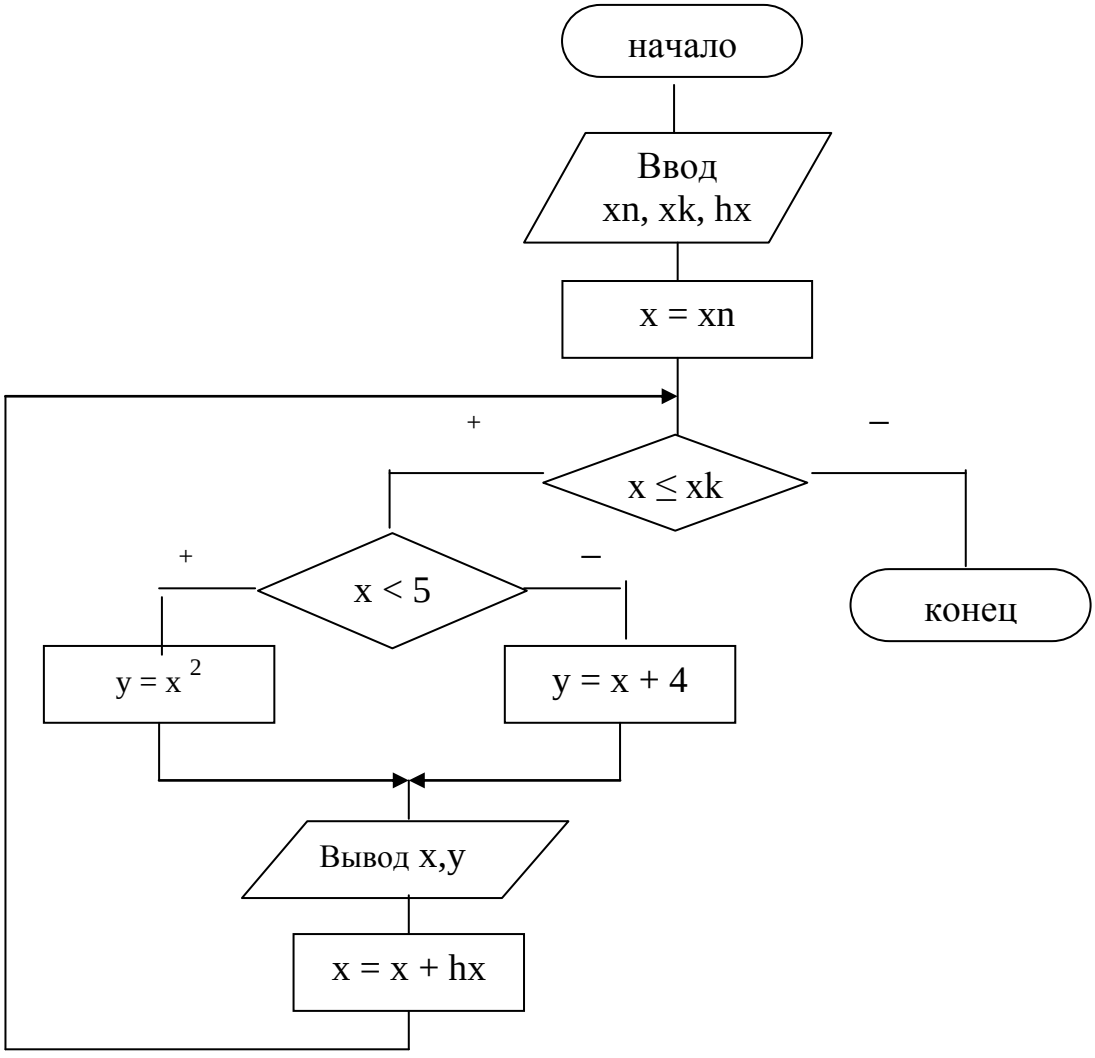

Рис. 3.11 – Блок-схема алгоритма (цикл с предусловием)

Форма, которую необходимо создать для этого примера, представлена на рис. 3. 12

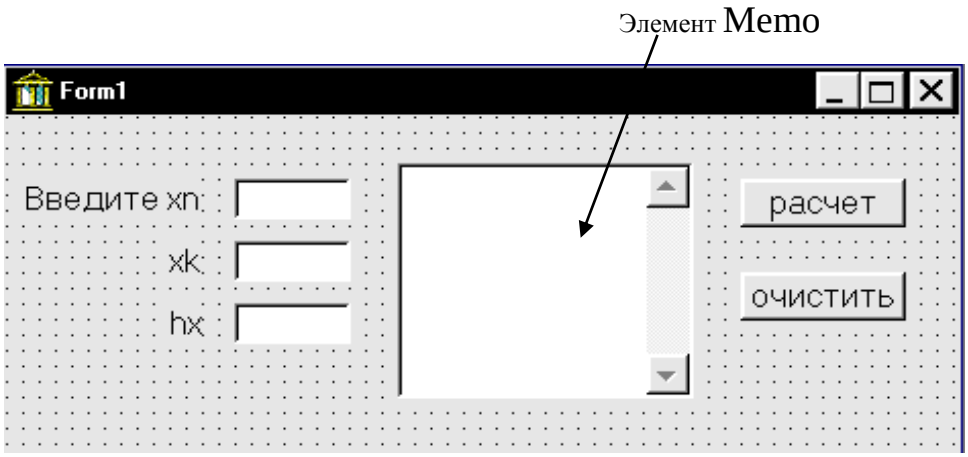

Рис. 3.12 – Размещение элементов на форме

В этом примере, используем уже известные компоненты: Edit, Label, Button и ранее не используемый компонент Memo. Текстовый редактор Memo может содержать в отличие от строки ввода Edit не одну, а любое количество строк. Элемент Memo используем для вывода пары значений x и y, причем значение x меняется в интервале от –2 до 8. Метод Add позволяет добавить новую строку (Lines) в этом компоненте.

Программа, написанная для кнопки Расчет, содержит следующий программный код

```
procedure TForm1.Button1Click(Sender: TObject);
var
  xn,xk,hx,x,y:integer;
  s,d:shortstring;
begin
    xn:=strtoint(edit1.text);
    xk:=strtoint(edit2.text);
    hx:=strtoint(edit3.text);
      x:=xn;
     while x<=xk do
       begin
         if x<5 then y:=sqr(x)
                else y:=x+4;
         str(x:6,s);
         str(y:6,d);
               Memo1.Lines.Add(s+d);
          x:=x+hx
       end
end;
```

Программный код для кнопки Очистить

```
procedure TForm1.Button2Click(Sender: TObject);
begin
 Memo1.Clear
end;
```

……

Запустив программу на выполнение и введя исходные данные, получим следующие результаты (см. рис. 3.13).

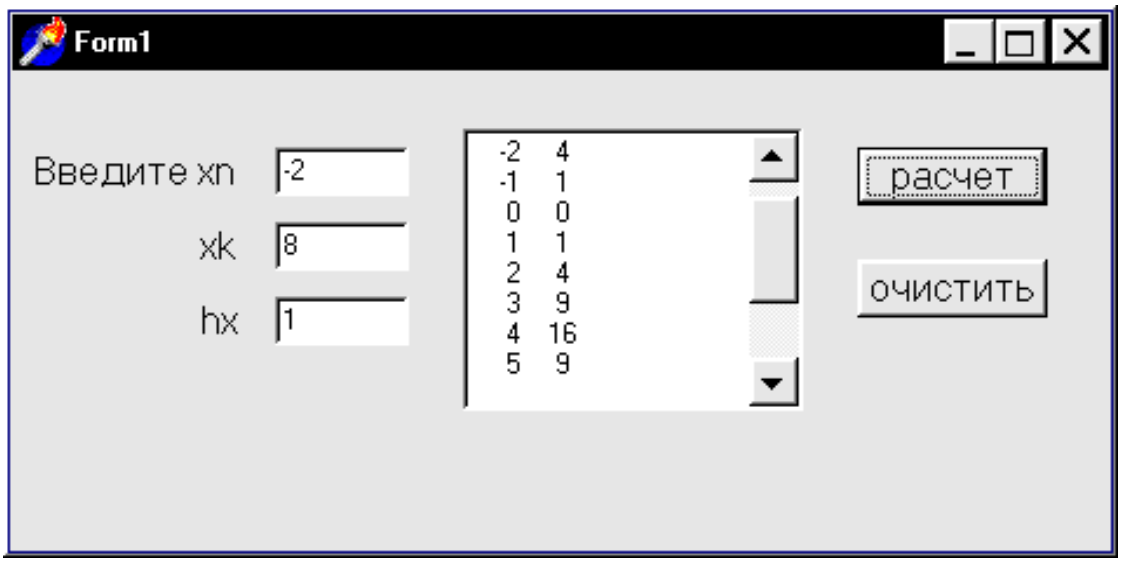

Рис. 3.13 – Окно формы (пример 4)

3.3.2 Оператор цикла с постусловием

Оператор цикла с постусловием имеет следующий формат:

```
Repeat
  <оператор 1>;
  <оператор 2>;
  <оператор 3>
until <логическое выражение>;
```

Выполняется оператор следующим образом. Вначале выполняется группа операторов и затем проверяется логическое выражение. Если оно истинно, то цикл прекращается иначе выполняется следующая итерация цикла. В тело цикла оператора repeat может входить произвольное количество операторов.

Блок-схема алгоритма вычисления представлена на рис. 3.14

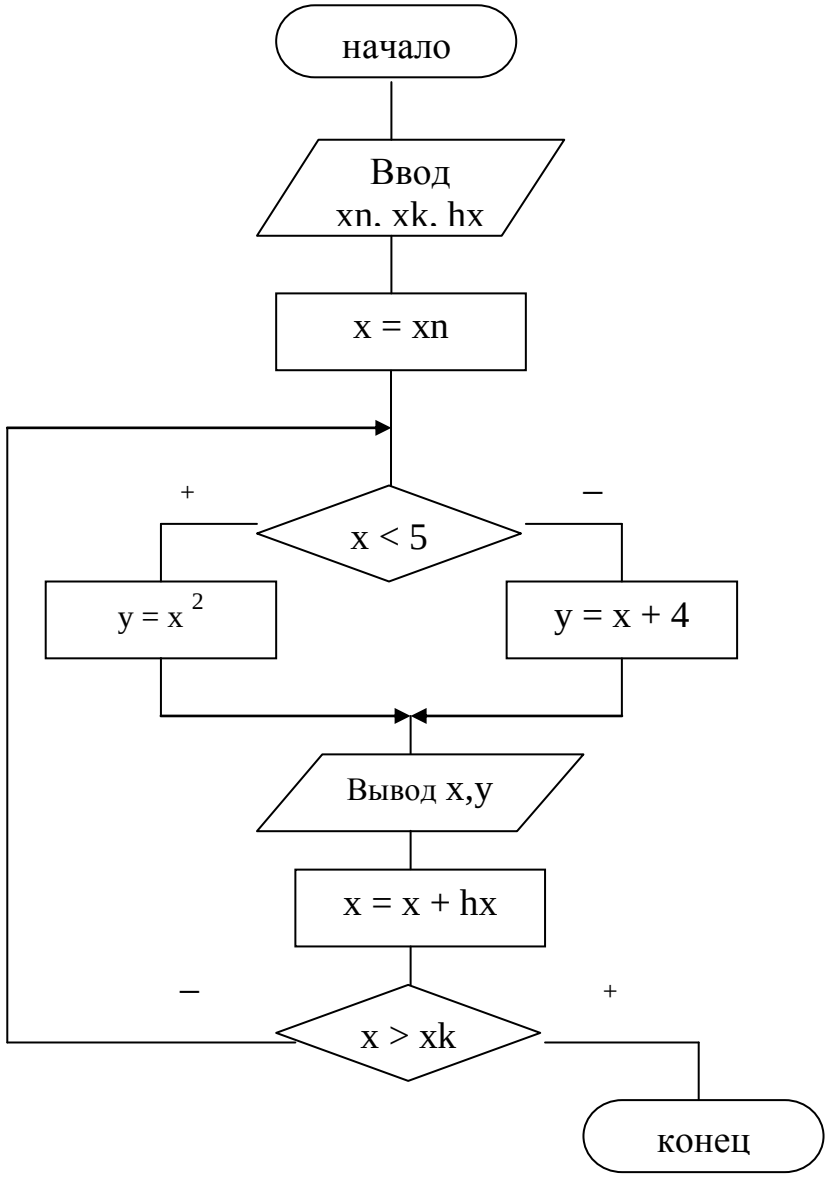

Рис. 3.14 – Блок-схема алгоритма (цикл с постусловием)

Форма, которую необходимо создать для этого примера, представлена на рис. 3. 12

Программа, написанная для кнопки Расчет, содержит следующий программный код.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  xn,xk,hx,x,y:integer;
  s,d:shortstring;
begin
    xn:=strtoint(edit1.text);
```

```
    xk:=strtoint(edit2.text);
    hx:=strtoint(edit3.text);
     x:=xn;
    repeat
       if x<5 then y:=sqr(x)
                 else y:=x+4;
       str(x:6,s);
       str(y:6,d);
   Memo1.Lines.Add(s+d);
        x:=x+hx
     until x>xk
end;
```

Запустив программу на выполнение и введя исходные данные, получим те же результаты, что и в предыдущем примере (см. рис. 3.13).

3.3.3 Оператор цикла с параметром

Оператор цикла **for** организует выполнение последовательности операторов **заранее известное число раз**. Число повторений тела цикла в этом случае подсчитывается с помощью специальной переменной (счетчика), для которой известны начальное и конечное значение, шаг ее изменения. Переменную-счетчик часто именуют параметром цикла, а сам цикл – цикл с параметром.

Существуют два варианта оператора:

- с увеличением счетчика

```
for <параметр цикла>:= <начальное значение>
    to <конечное значение> do <оператор>  ;
```

- с уменьшением счетчика

```
for <параметр цикла>:=<начальное значение>
   downto<конечное значение>do<оператор>;
```

Оператор for действует следующим образом.

В начале счетчику присваивается начальное значение. Затем значение счетчика сравнивается с конечным значением. Далее, пока счетчик меньше или

равен конечному значению (в первом варианте) или больше или равен конечному значению (во втором варианте), выполняется очередная итерация цикла. В противном случае выполняется выход из цикла.

При использовании в цикле служебного слова to значение параметра цикла увеличивается, при downto – уменьшается.

Если в цикле нужно выполнить не один оператор, а несколько, то их следует заключить в операторные скобки begin … end, т.е. использовать составной оператор.

Пример использования цикла с параметром для примера 4 (см. стр. 34).

```
procedure TForm1.Button1Click(Sender: TObject);
var
  xn,xk,hx,x,y,i,n:integer;
  s,d:shortstring;
begin
    xn:=strtoint(edit1.text);
    xk:=strtoint(edit2.text);
    hx:=strtoint(edit3.text);
        n:= trunc((xk-xn)/hx)+1;
        x:=xn;
    for i:=1 to n do
      begin
          if x<5 then y:=sqr(x)
                 else y:=x+4;
            str(x:6,s);
            str(y:6,d);
              Memo1.Lines.Add(s+d);
              x:=x+hx
      end
  end;
```

3.4 Массивы

В общем случае массив представляет собой фиксированное количество упорядоченных однотипных компонентов (элементов), снабженных индексами. Массив может быть одномерным или многомерным.

Чтобы задать массив, используется зарезервированное слово array, после которого следует указать в квадратных скобках тип индекса (индексов) компонентов, затем, после зарезервированного слова of, тип самих компонентов.

Ниже приведены два способа объявления одних и тех же массивов.

……1-й способ объявления……..

```
type
< имя типа > = array [ < тип индекса (индексов) >] of < тип >;
```

Например:

```
Type
        stroka = array [1..7] of real;
        matrix = array [1..5, 1..4] of integer;
```

Здесь stroka – массив, состоящий из семи вещественных чисел, matrix – двумерный массив, состоящий из пяти строк и четырех столбцов.

Введя тип массива, можно затем задать переменные этого типа.

Для описанных выше типов можно задать, например, следующие переменные:

```
    Var
        a1, a2 : stroka;
        p : matrix;
```

……2-й способ объявления ……

Определить переменную как массив можно и непосредственно при описании этой переменной, без предварительного описания типа массива. Например:

```
Var
  a1, a2 : array [1 .. 7] of  real;
  matrix : array [1..5, 1..4] of integer;
```

В Object Pascal допускается использование типизированных констант. Они задаются в разделе объявления констант :

<идентификатор> : <тип> = <значение>

<идентификатор> — идентификатор константы ;

<тип> — тип константы;

<значение> — значение константы.

Типизированным константам можно присваивать другие значения в ходе выполнения программы. В качестве начального значения типизированной константы-массива используется список констант, отделенных друг от друга запятыми, список заключается в круглые скобки, например:

```
Const
    schet : array [1..6] of integer = ( 4, 5, 0,-7 ,5, 2);
```

При объявлении многомерных констант-массивов множество констант, соответствующих каждому измерению, заключается в дополнительные круглые скобки и отделяется от соседнего множества запятыми, например:

```
Const
    Matrix : array [1..3, 1..5] of integer =
            ((1, 2, 5, 0,8),
             (5,-8, 7, 1,-9),
             (7, 4, -5, 2,8));
```

Использование типизированных констант может быть полезным при работе с массивами, имеющими начальные значения (исходные данные, справочные значения, паспортные данные).

3.4.1 Компонент StringGrid (страница Additional)

Компонент StringGrid (см. рис. 3.15) представляет собой таблицу, содержащую строки. Эти таблицы используют только для чтения или редактирования. Таблица может иметь полосы прокрутки, заданное число первых строк и столбцов может быть фиксированным и не прокручиваться, т.е. в таблице можно задать заголовки строк и столбцов, постоянно присутствующие в окне компонента. На пересечении строк и столбцов находятся ячейки. Каждая

ячейка может содержать символьную строку. Нумерация строк и столбцов таблицы начинается с нуля. Координаты каждой ячейки таблицы задается парой чисел, первое из которых является номером столбца, а второе – номером строки. Например, ячейка с координатами (2,3) расположена в третьем столбце и четвертой строке.

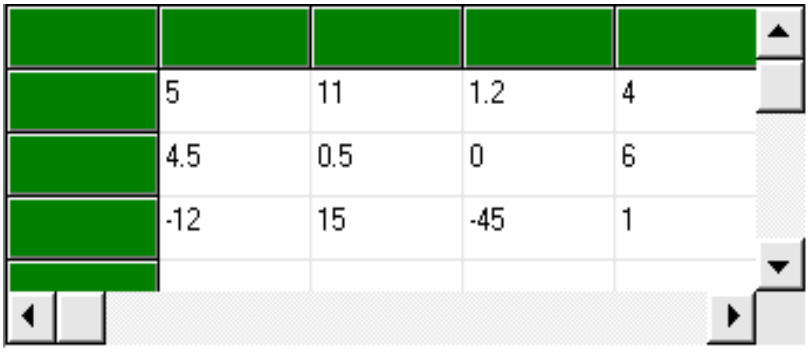

Рис. 3.15 – Пример компонента StringGrid

Основные свойства компонента StringGrid

ColCount, RowCount – определяет соответственно число столбцов и строк.

FixedCols, FixedRows – число фиксированных, не прокручиваемых столбцов и строк. По умолчанию задается один фиксированный столбец и одна фиксированная строка соответственно.

FixedColor – цвет фона фиксированных ячеек.

ScrollBars – определяет наличие в таблице полос прокрутки.

Cells – определяет двумерный массив символьных строк, каждая из которых принадлежит ячейке, находящейся в столбце Col и строке Row.

Options – определяется множеством, определяющим многие свойства таблицы: наличие разделительных вертикальных и горизонтальных линий, фиксированных ячеек, возможность для пользователя изменять с помощью мыши размеры столбцов и строк, перемещать столбцы и строки, **возможность редактировать содержимое таблицы** и многое другое:

goRowSizing — высота строк таблицы может изменяться;

goColSizing — ширина столбцов таблицы может изменяться;

goEditing – ячейки могут редактироваться.

Пример 5. В целочисленной матрице A (4,5) найти сумму элементов. Исходная матрица задана множеством значений (рис 3.16).

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

Рис. 3.16 – Исходная матрица A

Для решения поставленной задачи сформируем форму, расположив следующие элементы на форме (см. рис. 3.17): кнопку Расчет - для реализации алгоритма вычисления суммы элементов, элемент StringGrid - для отображения элементов матрицы, элемент Panel – для вывода результата, элемент Label – для вывода надписи. С помощью Инспектора Объектов свойствам компонента StringGrid зададим следующие свойства:

ColCount – 5,
RowCount – 4,
FixedCols – 0,
FixedRows – 0,
Options – goEditing – False.

Текст модуля, реализующего заданный алгоритм.

```
unit Unit1;
interface
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms, Dialogs, Grids, StdCtrls, ExtCtrls;
type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    Button1: TButton;
    Label1: TLabel;
    Panel1: TPanel;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);

 private
    { Private declarations }

 public
    { Public declarations }
  end;
var
```

```
  Form1: TForm1;
  i,j:integer;
 const
A:array[0..4,0..3]of integer=
((1,2,3,4),
(7,8,8,9),
(4,5,6,7),
(5,6,9,1),
(4,5,7,3));
implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
for j:=0 to 4 do
for i:=0 to 3 do
stringgrid1.Cells[j,i]:=inttostr(a[j,i]);
end;
procedure TForm1.Button1Click(Sender: TObject);
var
  s:integer;
begin
   s:=0;
   for j:=0 to 4 do
   for i:=0 to 3 do
     s:=s+a[j,i];
    Panel1.Caption:=inttostr(s);
end;
end.
```

Запустив программу на выполнение, получим следующие результаты.

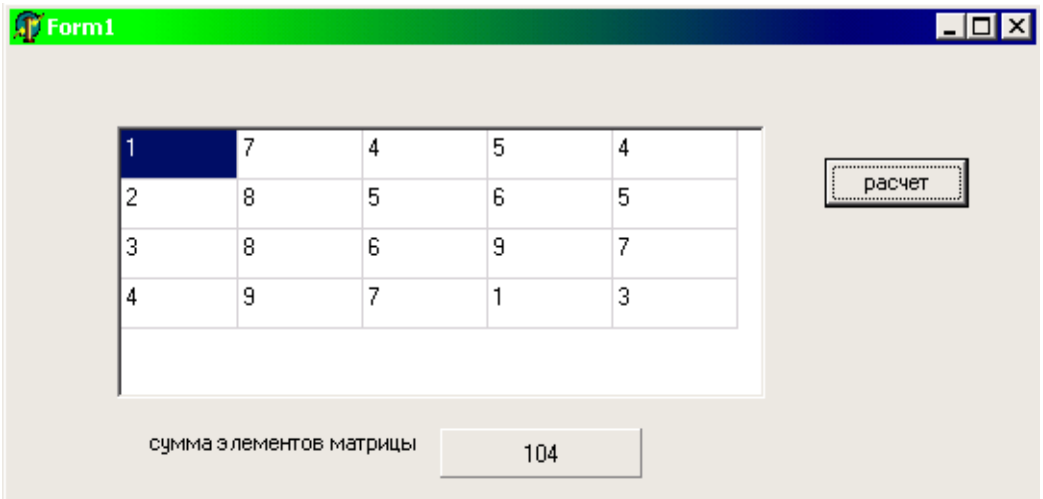

Рис. 3.17 – Окно формы (результаты расчета примера 5)

Пример 6. В целочисленной матрице A (4,4) найти сумму элементов. Элементы матрицы вводить с клавиатуры. Реализовать поставленный алгоритм с помощью **Cells.**

Для решения поставленной задачи сформируем форму, расположив следующие элементы на форме (см. рис. 3.18): кнопку Расчет - для реализации алгоритма вычисления суммы элементов, кнопку Очистить – для очистки содержимого матрицы, элемент StringGrid - для отображения элементов матрицы, элемент Panel – для вывода результата, элемент Label – для вывода надписи. С помощью Инспектора Объектов свойствам компонента StringGrid зададим следующие свойства:

ColCount – 4,
RowCount – 4,
FixedCols – 0,
FixedRows – 0,
Options – goEditing – True.

Текст модуля, реализующего заданный алгоритм.

```
unit Unit1;

interface

uses
Windows, Messages, SysUtils, Classes, Graphics,
Controls, Forms, Dialogs, Grids, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    Button1: TButton;
    Panel1: TPanel;
    Label1: TLabel;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);

  private
    { Private declarations }

  public
    { Public declarations }
  end;
```

```
const
   m:integer=3;
var
  Form1: TForm1;
  i,j:integer;
implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var
    S:integer;
begin
     s:=0;
   for j:=0 to m do
     for i:=0 to m  do
       S:=S+strtoint(StringGrid1.Cells[J,I]);
       Panel1.caption:=inttostr(s);
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
    for j:=0 to m do
     for i:=0 to m  do
       StringGrid1.Cells[J,I]:=' ';
       Panel1.caption:=' ';
end;
end.
```

Запустив программу на выполнение и введя исходные данные, получим следующие результаты (см. рис. 3.18)

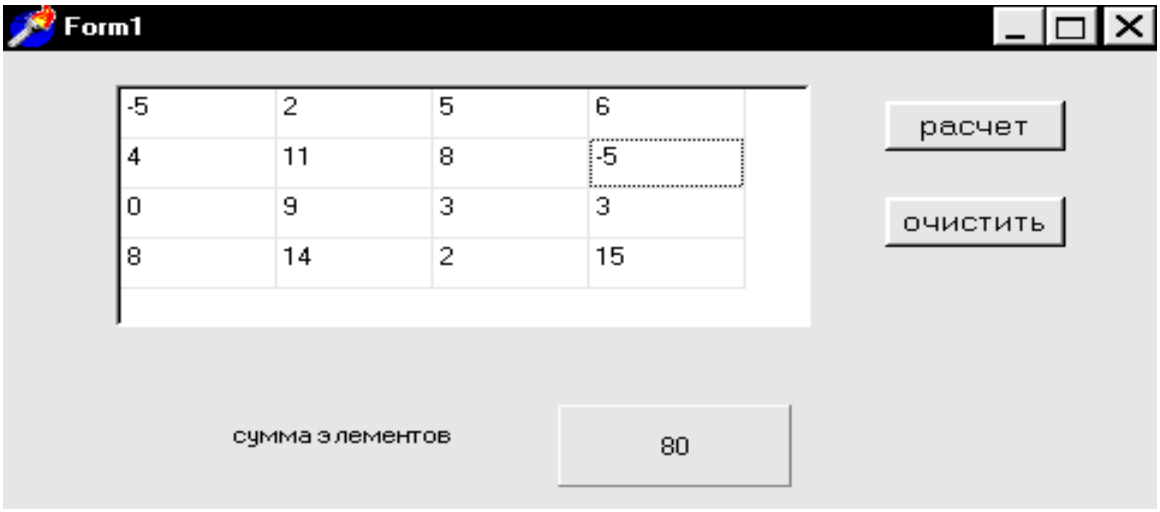

Рис. 3.18 – Окно формы (результаты расчета)

Пример 7. В целочисленной матрице A (4,5) поменять местами 3-й и 5-й столбец. Элементы матрицы вводить с клавиатуры. Реализовать поставленный алгоритм с помощью **Cells.**

Для решения поставленной задачи сформируем форму, расположив соответствующие элементы на форме (см. рис. 3.19).

Текст модуля, реализующего заданный алгоритм.

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs, Grids, StdCtrls, ExtCtrls;
type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    Button1: TButton;
    Button2: TButton;
    StringGrid2: TStringGrid;
    Label1: TLabel;
    Label2: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
const
   n:integer=4;
   m:integer=3;
var
  Form1: TForm1;
```

```
  i,j:integer;
implementation
{$R *.DFM}
procedure TForm1.Button1Click(Sender: TObject);
var
    r:string;
  begin
    { копирование }
   for j:=0 to n do
     for i:=0 to m  do
         StringGrid2.Cells[J,I]:=StringGrid1.Cells[J,I];
     {обмен}
   for i:=0 to m do
      begin
         r:=StringGrid2.Cells[2,I];
         StringGrid2.Cells[2,I]:=StringGrid2.Cells[4,I];
         StringGrid2.Cells[4,I]:=r;
      end
  end;
procedure TForm1.Button2Click(Sender: TObject);
begin
    for j:=0 to n do
     for i:=0 to m  do
       begin
        StringGrid1.Cells[J,I]:=' ';
        StringGrid2.Cells[J,I]:=' ';
       end;
end;
end.
```

Запустив программу на выполнение и введя исходные данные, получим следующие результаты (см. рис. 3.19)

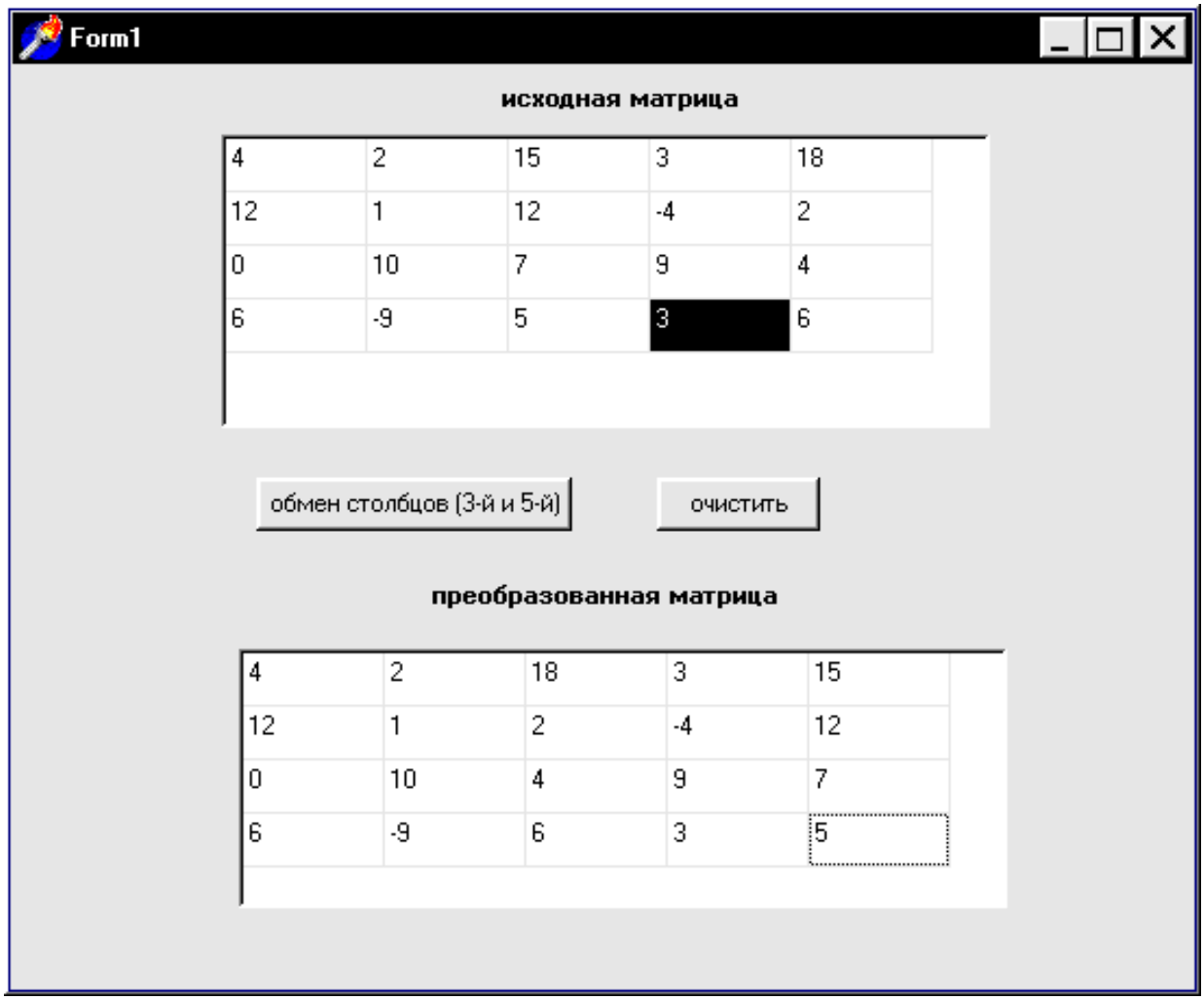

Рис. 3.19 – Окно формы (исходная матрица, преобразованная)

СПИСОК ЛИТЕРАТУРЫ

1. Архангельский А.Я. Object Pascal в Delphi.- М.: Бином, 2002, 384 с.
2. Дарахвелидзе П.Г., Марков Е.П. Delphi – среда визуального программирования. – СПб.: BHV – Санкт-Петербург, 1996, 352 с
3. Джон Манчо, Дэвид Р. Фолкнер. Delphi: Пер. с англ. – М.: БИНОМ, 1995, 464с.
4. Фаронов В. В. Delphi 3. Учебный курс. – М.: Нолидж, 1998, 400 с.

ПРИЛОЖЕНИЕ А

СИСТЕМА МЕНЮ

Пункт меню “File”

New	Открывает доступ к репозитарию Delphi
New Application	Создает новую программу
New Form	Создает новую форму и подключает ее к проекту
New Data Module	Создает новый модуль данных и подключает его к проекту
Open	Открывает ранее созданную форму
Reopen	Вызывает список ранее загружавшихся проектов и форм для выбора и повторной загрузки
Save	Сохраняет активную форму
Save As	Сохраняет активную форму под другим именем
Save Project As	Сохраняет файл проекта под другим именем
Save All	Сохраняет файл проекта и все открытые модули
Close	Закрывает текущую форму
Close All	Закрывает все открытые файлы
Use Unit	Вставляет в текущую форму ссылку на другой модуль
Add to Project	Добавляет к проекту ранее созданную форму
Remove from Project	Удаляет из проекта форму
Print	Печатает активную форму или модуль
Exit	Прекращает работу Delphi

Пункт меню “Edit”

Undelete	Отменяет последнее изменение проекта
Redo	Восстанавливает последнее изменение проекта
Cut	Вырезает выбранный компонент формы или фрагмент теста и помещает его в буфер
Copy	Копирует в буфер выделенные компоненты формы или фрагменты текста модуля
Paste	Извлекает из буфера и копирует компоненты на форму или текст в модуль
Delete	Удаляет выделенные компоненты или фрагмент текста
Select All	Выделяет все компоненты формы или весь текст модуля
Align To Grid	Привязывает выделенные компоненты к сетке
Align	Вызывает окно выравнивания выделенных компонент
Size	Вызывает окно изменения размеров выделенных компонентов
Scale	Масштабирует выделенные компоненты

Пункт меню “Search”

Find	Ищет фрагмент текста и подсвечивает его, если он найден
Find In File	Ищет фрагмент текста во всех файлах проекта или только в открытых файлах или, наконец, во всех файлах текущего каталога
Replace	Ищет и заменяет фрагмент текста
Search Again	Повторяет поиск или поиск и замену
Incremental Search	Ищет текст по мере его ввода – сначала первую букву, затем две первых буквы и т.д.
Go to Line Number	Перемещает курсор на строку с указанным номером от начала файла
Find Error	По адресу ошибки периода прогона программы отыскивает фрагмент кода, связанный с ее возникновением
Browse Symbol	Подыскивает характеристики символа программы по его имени (опция доступна только после успешного прогона программы)

Пункт меню “View”

Project Manager	Показывает окно менеджера проекта
Project Source	Показывает текст файла проекта
Object Inspector	Показывает окно Инспектора Объектов
Alignment Palette	Показывает окно палитры выравнивания компонентов
Browser	Показывает окно броузера объектов
Breakpoints	Показывает окно точек останова
Call Stack	Показывает окно стека
Watches	Показывает окно наблюдения за переменными/выражениями
Threads	Показывает окно статуса потоков команд
Modules	Показывает окно модулей проекта
Component List	Показывает окно для выбора компонентов
Window List	Показывает окно открытых окон проекта
Toggle Form/Unit	Переключает активность из окна формы в окно кода программы и обратно
Units	Показывает окно модулей
Forms	Показывает окно форм
Type Library	Показывает окно библиотеки типов (используется при разработке компонентов для применения вне Delphi)
New Edit Window	Открывает новое окно с кодом текущего модуля
Speedbar	Прячет или показывает панель инструментальных кнопок
Component Palette	Прячет или показывает палитру компонентов

Пункт меню “Project”

Add To Project	Добавляет файл к проекту
Remove From Project	Удаляет файл из проекта
Add To Repository	Помещает проект в репозитарий
Compile	Компилирует модули, которые изменились с момента предыдущей компиляции
Build All	Компилирует все модули проекта и создает исполняемую программу
Syntax Check	Проверяет синтаксическую правильность программы
Information	Показывает информацию о вашей программе
Options	Показывает диалоговое окно установки параметров проекта

Пункт меню “Run”

Run	Компилирует программу и осуществляет ее прогон
Parameters	Указывает командную строку запуска программы
Step Over	В отладочном режиме выполняет текущую строку кода и не прослеживает работу вызываемых подпрограмм
Trace Info	В отладочном режиме выполняет текущую строку кода и прослеживает работу вызываемых подпрограмм
Trace To Next Source Line	Программа выполняется до ближайшего от текущего положения курсора исполняемого оператора
Run To Cursor	В отладочном режиме выполняет программу и останавливается перед выполнением кода в строке с текстовым курсором
Execution Point	Отображает в окне кода оператор, на котором было прервано выполнение программы
Program Pause	Приостанавливает прогон отлаживаемой программы
Program Reset	Прекращает прогон программы и восстанавливает режим конструирования программы
Add Watch	Добавляет переменную или выражение в окно наблюдения
Add Breakpoint	Добавляет точку останова
Evaluate/Modify	Открывает окно проверки/изменения переменных

ПРИЛОЖЕНИЕ Б

ПАЛИТРА КОМПОНЕНТОВ СТРАНИЦЫ STANDARD

MainMenu Главное меню программы. Компонент способен создавать и обслуживать сложные иерархические меню.

PopupMenu Вспомогательное меню. Позволяет создавать всплывающие меню. Этот тип меню появляется по щелчку правой кнопки мыши.

Label Метка. Этот компонент используется для размещения в окне не очень длинных однострочных надписей.

Edit Текстовый редактор. Предназначен для ввода и/или отображения одной текстовой строки.

Memo Многострочный текстовый редактор. Используется для ввода и/или отображения многострочного текста.

Button Командная кнопка. Позволяет выполнить какие-либо действия при нажатии кнопки во время выполнения программы.

CheckBox Независимый переключатель. Отображает строку текста с маленьким окошком рядом. В окошке можно поставить отметку, которая означает, что что-то выбрано.

RadioButton Зависимый переключатель. Позволяет выбрать только одну опцию из нескольких.

ListBox Список выбора. Содержит список предлагаемых вариантов (опций).

ComboBox Комбинированный список выбора. Представляет собой комбинацию списка выбора и текстового редактора.

Scrollbar Полоса прокрутки. Появляется автоматически в объектах редактирования.

GroupBox Группа элементов. Используется для группировки нескольких связанных по смыслу компонентов.

RadioGroup Группа зависимых переключателей. Предназначен для отображения состояния и выбора одного из взаимоисключающих переключателей.

Panel Панель. Служит для объединения нескольких компонентов или вывода результата.

ПРИЛОЖЕНИЕ В.

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ. МАТЕМАТИЧЕСКИ СТАНДАРТНЫЕ ФУНКЦИИ.

Бинарные арифметические операции

Обозначение	Операция	Типы операндов	Тип результата	Пример
+	сложение	Integer, real	Integer, real	X + Y
-	вычитание	Integer, real	Integer, real	X - Y
*	умножение	Integer, real	Integer, real	X * Y
/	вещественное деление	Integer, real	Real	X / Y
div	целочисленное деление	Integer	Integer	I div 4
mod	остаток от деления целых чисел	Integer	Integer	I mod 3

Математические стандартные функции

Обозначение	Операция	Типы операндов	Тип результата
Abs(x)	абсолютное значение	Integer, real	Integer, real
ArcTan(x)	арктангенс	Integer, real	Real
Cos(x)	косинус	Integer, real	Real
Sin(x)	синус	Integer, real	Real
Exp(x)	экспонента	Integer, real	Real
Int(x)	целая часть аргумента	Integer, real	Integer
Ln(X)	натуральный логарифм	Integer, real	Real
Pi	число: 3.1415926535897932385		Real
Round(x)	ближайшее целое	Integer, real	Integer
Sqr(x)	квадрат аргумента	Integer, real	Integer, real
Sqrt(x)	квадратный корень	Integer, real	Real
Trunc(x)	отсечение дробной части вещественного числа	Integer, real	Integer

МЕТОДИЧЕСКОЕ ПОСОБИЕ

К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

В СРЕДЕ DELPHI

(учебно-методическое пособие для студентов всех специальностей)

Составители: Незамова Лариса Викторовна
Анохина Инна Юрьевна

Донецкий национальный технический университет
83000, г. Донецк-00, ул. Артема, 58