Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Институт повышения квалификации
и переподготовки кадров

Кафедра «Информатика»

**С. А. Чабуркина, Н. С. Емельянченко**

# ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ НА ЯЗЫКАХ ВЫСОКОГО УРОВНЯ

**КУРС ЛЕКЦИЙ**
**по одноименной дисциплине**
**для слушателей специальности 1-40 01 73**
**«Программное обеспечение информационных систем»**
**заочной формы обучения**

**Гомель 2013**

УДК 004.421+004.43(075.8)
ББК 32.973.26-018.1я73
Ч-12

*Рекомендовано кафедрой «Информатика»*
*ГГТУ им. П. О. Сухого*
(*протокол № 5 от 27.11.2012 г.*)

Рецензент: канд. физ.-мат. наук, доц. каф. «Информационные технологии»
ГГТУ им. П. О. Сухого *О. А. Кравченко*

**Чабуркина, С. А.**

Ч-12 Основы алгоритмизации и программирования на языках высокого уровня : курс лекций по одноим. дисциплине для слушателей специальности 1-40 01 73 «Программное обеспечение информационных систем» заоч. формы обучения / С. А. Чабуркина, Н. С. Емельянченко. – Гомель : ГГТУ им. П. О. Сухого, 2013. – 130 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http:library.gstu.by/StartEK/. – Загл. с титул. экрана.

Рассматриваются три основных типа алгоритмов: линейные, разветвляющиеся и циклические, алгоритмы обработки одномерных и двухмерных статических и динамических массивов, программирование с использованием подпрограмм и алгоритмы обработки символьной информации. В курсе лекций рассмотрены вопросы создания приложений в системе программирования Delphi. Приведены методы разработки интерфейса приложений с использованием библиотеки визуальных компонентов.

Для слушателей специальности 1-40 01 73 «Программное обеспечение информационных систем» заочной формы обучения ИПК и ПК.

**УДК 004.421+004.43(075.8)**
**ББК 32.973.26-018.1я73**

## Тема 1. Понятие алгоритма

### 1.1. Свойства алгоритмов и формы их представления

Процесс решения задачи с использованием вычислительной техники можно представить последовательностью действий:

- постановка задачи;
- разработка алгоритма;
- программирование;
- тестирование.

*Алгоритм* – это точный набор инструкций, описывающих порядок действий исполнителя для достижения результата решения задачи за конечное время.

*Алгоритм* – это точная последовательность инструкций, определяющая вычислительный процесс, ведущий от исходных данных к искомому результату за конечное число шагов.

Алгоритм не содержит ошибок, если он даёт правильные результаты для любых допустимых исходных данных. Если исходные данные недопустимы, то в алгоритме должна быть предусмотрена защита от них (деление на 0).

Основные свойства алгоритма:

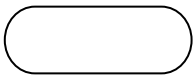
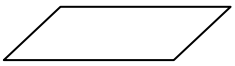
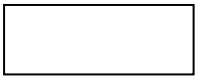
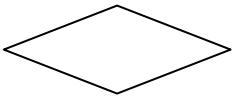
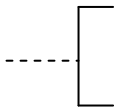
- *определенность* – однозначность выполнения составляющих алгоритм шагов;
- *результативность* – это получения результата за конечное число шагов;
- *массовость* – это применение алгоритма ко всему классу однотипных задач;
- *дискретность* – это возможность разбиения на элементарные операции, выполнение которых не вызывает затруднения.

Основные формы представления алгоритмов:

- словесно-формульное описание;
- алгоритмическая запись на условном языке (псевдокод);
- графические схемы алгоритмов (ГСА).

В графических схемах алгоритмов могут использоваться следующие графические элементы или блоки, представленные в таблице 1.1

Таблица 1.1

| N п/п | Графический элемент | Назначение |
|---|---|---|
| 1 |  | Начало и конец алгоритма |
| 2 |  | Ввод и вывод данных, обмен данными с внешними устройствами |
| 3 |  | Преобразование данных, любые вычисления |
| 4 |  | Условный блок, проверка условия |
| 5 |  | Предопределенный процесс, обращение к подпрограмме |
| 6 |  | Комментарий к алгоритму |
| 7 |  | Соединители |

## 1.2. Линейные вычислительные алгоритмы

Линейным называется алгоритм, в котором все указанные действия выполняются один раз в том порядке, в котором они записаны.

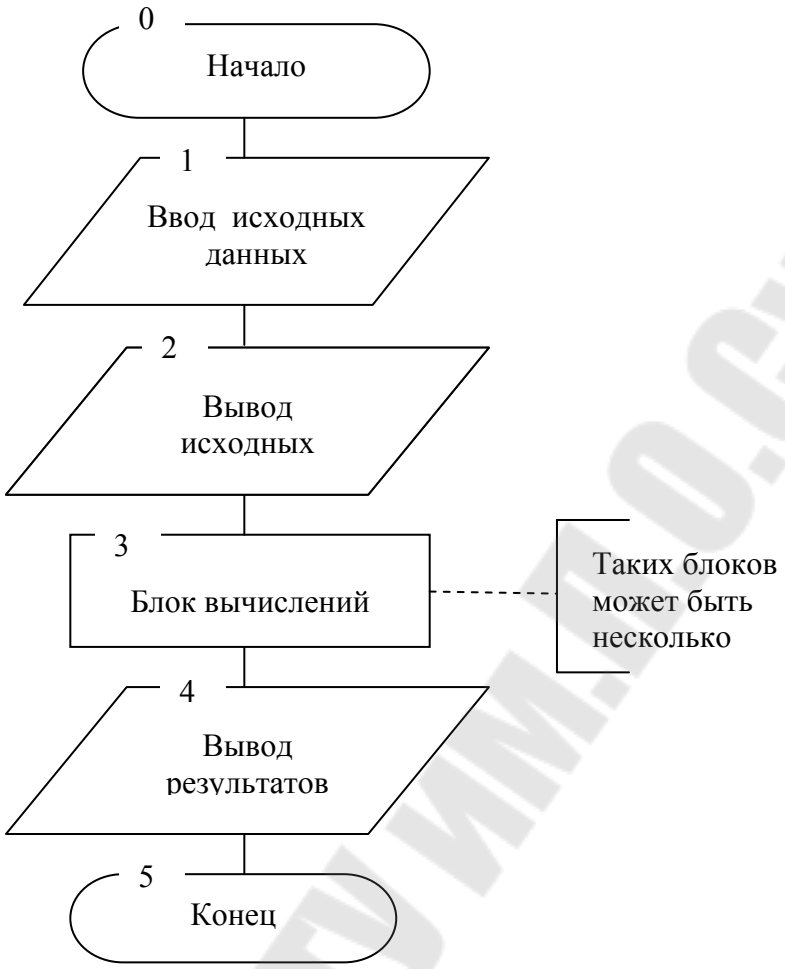



*Рис 1.1.* Общая графическая схема алгоритма линейной расчетной задачи

**Пример 1.1**: Вычислить площадь треугольника со сторонами a,b,c.

$$S = \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)} \qquad p = \frac{a+b+c}{2}$$

Исходные данные: a, b, c. Результат: S.

Промежуточный результат: p.

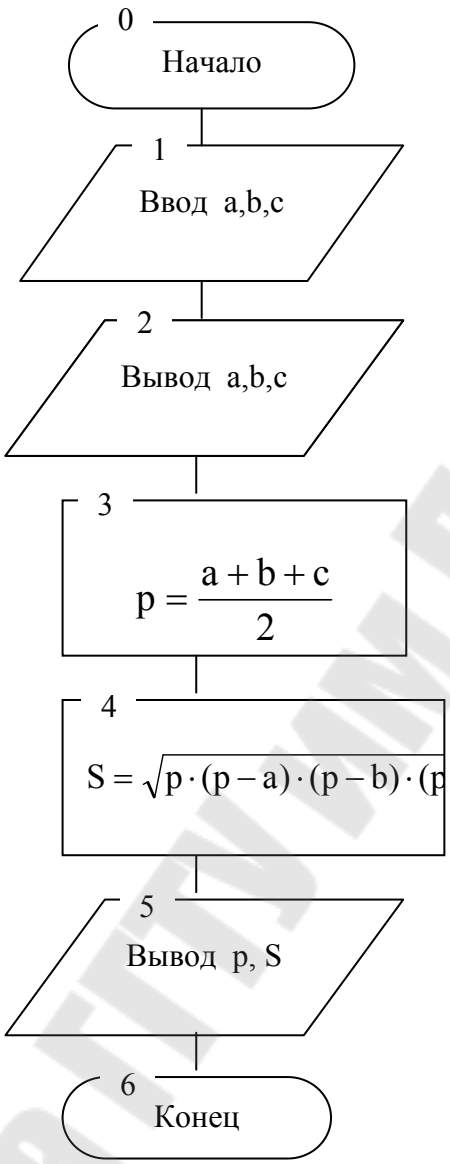



*Рис 1.2.* Графическая схема алгоритма примера 1.1

**Тесты** – это наборы исходных данных с известными результатами, с помощью которых выполняется проверка работоспособности (правильности работы) программы.

**Тест для примера:**

Исходные данные a=3, b=4, c=5  Ожидаемый результат: p=6, S=6

# Тема 2. Основные элементы языка программирования Delphi

## 2.1. Общие сведения о системе программирования

Система программирования Delphi относится к классу инструментальных средств ускоренной разработки программ (RAD – Rapid Application Development). Ускорение разработки программ достигается за счет 2-х характерных свойств Delphi:

- визуального конструирования форм;
- широкого использования библиотеки визуальных компонентов (VCL – Visual Component Library).

При разработке интерфейса программы используется специальное окно, которое называется окном формы. Оно заполняется визуальными компонентами, реализующими различные интерфейсные элементы (кнопки, списки, меню, окна, полосы прокрутки и т. п.).

После размещения на форме очередного компонента Delphi автоматически вставляет в связанный с формой модуль ссылку на этот компонент и корректирует специальный файл описания формы.

На этапе выполнения разработанная программа, как все Windows-приложения, работает в окне, полностью совпадающем с окном формы.

Языком программирования в СП Delphi является Object Pascal, начиная с версии 7 язык называется Delphi.

Delphi относится к языкам, реализующим идеи объектно-ориентированного программирования (ООП).

Основная идея ООП состоит в том, что программное приложение должно строиться из взаимодействующих объектов, каждый из которых имеет свои специфические качества (свойства), поведение (определяемое методами) и события, на которые он реагирует. Каждый объект включает в себя данные и способы обработки этих данных в виде программного кода.

Все визуальные компоненты являются объектами и имеют соответствующие каждому свойства, методы и события.

## 2.2. Элементы языка Delphi

Алфавит языка включает буквы, цифры, специальные символы, зарезервированные слова и идентификаторы пользователя. Прописные и строчные буквы не различаются, если только они не входят в состав символьных и строковых констант.

К буквам относят латинские строчные буквы от a до z, заглавные от A до Z и _.

Цифры – это арабские цифры от 0 до 9.

Специальные символы: (***, /, +, - , [ ] , { }** ). К специальным символам относятся пары символов **<= ; >= ; <> ; := , (* , *)**, их нельзя разделять пробелом.

Символ пробел является разделителем идентификаторов, зарезервированных слов, констант. Несколько пробелов рассматриваются как один (это не относится к строковым константам).

Зарезервированные слова указывают компилятору на необходимость выполнения определенных действий, они не могут изменяться и использоваться в качестве идентификаторов.

Это слова:

begin, end, for, if, var, real и т.д.

Идентификаторы – это имена, которые используются в программе для обозначения констант, переменных, типов, объектов и т.д.

Правила образования имен (идентификаторов):

- имя состоит из последовательности латинских букв, цифр и знака **_;**
- первым символом имени может быть только буква и **_;**
- имена могут иметь произвольную длину;
- нельзя использовать в именах русские буквы и пробелы;
- имя должно быть информативным.

## 2.3. Описание констант

Константы – это именованные элементы программы, значения которых определяются при их описании и при выполнении программы не меняются.

Описание (объявление) констант:

const

<идентификатор> = <значение>;

Тип константы определяется её значением.

Как значение константы могут использоваться:

- целые числа;
- вещественные числа с использованием десятичной точки (10.7) или в экспоненциальной форме (1.72E02);
- логические константы (true, false);
- символьные константы ('A' ≠ 'a');
- строковые константы ('ABC').

Примеры описания констант:

```
const

    X=3.5; L=true; Str= 'ГГТУ';
```

### 2.4. Описание переменных

Переменные - это именованные элементы программы, которые в процессе выполнения программы могут принимать различные значения в соответствии с указанным при их объявлении типом. Каждой переменной выделяется место в ОЗУ.

Все переменные до их использования в разделе операторов должны быть объявлены в разделе описаний.

Описание переменных

```
var
   <идентификатор>: <тип>;
var
   a, b: real; c : byte;
```

Типы рассматриваются компилятором как образцы для создания констант, переменных, функций.

Тип данного определяет:

1. диапазон возможных значений;
2. объём выделяемой памяти и форму представления данных в ней;
3. действия, которые можно совершать над данными.

Таблица 2.1

Основные типы данных

| Название типа | Выделяемая память (байт) | Диапазон значений | Примечания |
|---|---|---|---|
| Byte | 1 | 0….255 | Целые типы |
| Word | 2 | 0….65535 | |
| SmallInt | 2 | 32768…..32767 | |
| Integer | 4 | -2 147 483 648…. 21 474 836 47 | |
| Real | 8 | $\pm 10^{-324} \ldots 10^{308}$ 15 – 16 знач. цифр | Вещественные типы |
| Extended | 10 | $\pm 10^{-4951} \ldots 10^{4932}$ 19 – 20 знач. цифр | |
| Boolean | 1 | true, false | Логический тип |

Вещественные числа обрабатываются математическим сопроцессором всегда в формате Extended, а затем результат преобразуется к нужному типу.

## 2.5. Арифметические выражения

Арифметические выражения служат для вычисления целого или вещественного значения. Они состоят из констант, переменных, функций, знаков арифметических операций и круглых скобок.

Арифметические операции в порядке убывания приоритета:

* , / , div, mod, + , -

Порядок выполнения действий при вычислении значения арифметического выражения определяется приоритетом операций и круглыми скобками.

Операции div и mod применяется только к целочисленным операндам, результат – целое число.

Div – целочисленное деление;

Mod – остаток от целочисленного деления.

7 mod 3 = 1  7 div 3 = 2

В арифметических выражениях могут использоваться следующие математические функции:

Таблица 2.2

| Математическая запись | Функция Delphi | Тип аргумента и результата |
|---|---|---|
| $\|x\|$ | abs(x) | Целый и вещественный тип |
| $x^2$ | sqr(x) | |
| $\sqrt{x}$ | sqrt(x) | |
| ln x | ln(x) | |
| $e^x$ | exp (x) | |
| sin x | sin (x) | |
| cos x | cos(x) | Вещественный тип |
| arctg x | arctan (x) | |
| π | pi | |
| целая часть числа | int(x) | |
| дробная часть числа | frac(x) | |

Все эти функции находятся в модуле System, который подключается к программе автоматически. Аргументы функции заключаются в круглые скобки.

Аргументы функций sin(x), cos(x) и результат функции arctan(x) указываются в радианах. Для возведения в степень используются функции exp(x) и ln(x).

$x^y=e^{y\ln(x)}$  exp(y*ln(x))  Для $\sqrt[3]{x}$  exp(ln(x)/3)

В Delphi есть возможность использовать множество математических функций из модуля Math, подключив его к своей программе. Для этого его нужно добавить в список подключаемых модулей. В этом случае можно использовать функции: arccos(x), arcsin(x), tan(x), log10(x).

Для возведения числа в степень используются функции модуля Math:

Power(x, y) – возведение вещественного числа x в вещественную степень y. Нельзя возвести отрицательное вещественное число в вещественную степень !

Для $\sqrt[3]{x}$ используется функция Power(x, 1/3)

IntPower (x, y) - возведение вещественного числа x в целую степень y.

Для $x^3$ sqr(x)*x ; x*x*x IntPower (x, 3).

### 2.6. Оператор присваивания

С его помощью можно присвоить переменной значение некоторого выражения.

Синтаксис оператора:

<идентификатор>: = <выражение>

Примеры:

a: = b+c*d;

i:=i+1;

Выполнение: вычисляется значение выражения, стоящего справа от знака присваивания, а затем полученное значение присваивается переменной стоящей слева от знака := (то есть записывается в то место памяти, которое отведено для переменной после ее описания).

Идентификатор и выражение должны иметь одинаковые типы.

Исключение: вещевой переменной можно присвоить целое значение, но не наоборот.

В операторе присваивания могут использоваться свойства компонентов, как переменные соответствующего типов.

Пример:

Вычислить $y = \dfrac{\cos^3 x + \sqrt[5]{|x + tg2x|}}{2e^{\sqrt{x}}}$

В программе:

```
y:=(IntPower(cos(x),3)+Power(abs(x+tan(2*x)),1/5))/(2*exp(sqrt(x)))
```

## Тема 3. Структура программы (проекта, приложения) Delphi

### 3.1. Элементы программы

Любая программа в Delphi состоит из проекта (головного файла проекта) и одного или нескольких модулей, структура которых создается автоматически при работе в интегрированной среде разработки (ИСР). Не рекомендуется изменять и удалять автоматически созданные части программы.

**Проект** создаётся автоматически при создании нового приложения (и при загрузке в Delphi7). Он представляет собой основную программу, которая имеет следующий вид:

```
Program project1;
  uses
  Forms,
     unit1 in unit1.pas          {Form1},
     // unit2 in unit2.pas          {Form2},   для следующих форм
  {$R *.RES}
  Begin
    Application. Initialize;                    инициализация приложения
    Application. CreateForm (TForm1,   Form1);          создание формы
 // Application. CreateForm (TForm2,   Form2);   для следующих форм
    Application.Run;                  запуск приложения до закрытия окна
  End.
```

Имена project1, unit1 могут изменяться при сохранении программы.

Обычно проект не выводится на экран во время работы с программой, но его можно вывести с помощью команды ИСР *Project – View Source.*

Проект сохраняется в файле с типом dpr и имеет по умолчанию имя Project1. Для первой формы автоматически создаётся файл формы с именем Unit1.dfm и модуль Unit1.pas. В них содержится описание пустой формы. При проектировании интерфейса на форме будут размещены различные визуальные компоненты. Delphi автоматически поместит описания этих компонентов в модуль (в класс TForm1), а описание их свойств – в файл формы. Процедуры

пользователя для обработки событий, на которые должны реагировать ВК формы, размещаются в модуле, связанном с формой. Для всех дополнительно создаваемых форм проекта создаются файлы форм (Unit2.dfm и т.д.) и модулей (Unit2.pas и т.д).

**Модуль** представляет собой текстовый файл, предназначенный для хранения функционально связанных частей программы. С помощью программного кода модулей реализуется алгоритм решения задачи. В модуле размещаются объявления констант, переменных, типов, тексты процедур и функций. Нельзя запустить модуль на выполнение. Подключение всех модулей и создание форм выполняется в основной программе проекта.

Модуль, связанный с первой пустой формой имеет следующий вид:

```
unit   unit1;  // заголовок модуля
interface     // раздел интерфейса
{ здесь можно разместить списки подключаемых модулей,
 описание      всех типов,  констант, переменных, классов, заголовки подпрограмм, к которым будет доступ из всех программ и модулей, подключающих  этот модуль}
Uses Windows, Messages,…, Math;
type TForm1 = class (TForm)
// TForm – класс пустых форм, TForm1 – класс форм с визуальными компонентами
{здесь будут объявления всех визуальных компонентов первой формы}
private
 {Private declaration}          частные описания, доступны всем процедурам модуля
public
  { public declaration}    общие описания, доступны всем модулям программы
Var
  Form1: TForm1;   // Form1 – экземпляр класса TForm1
implementation   // раздел реализации
 { здесь можно разместить списки подключаемых модулей, описание всех типов,  констант, переменных, классов, процедур и функций, к которым будет доступ из всех процедур  только этого
```

модуля, а также текст подпрограмм, имена которых перечислены в разделе интерфейса}

{$R *.DFM}

{здесь будут размещены тексты процедур и функций пользователя, включая процедуры обработки событий}
end.

В проекте и модулях могут находиться директивы компилятора, указывающие компилятору на выполнение определенных действий: {$R *. DFM} – подключение файлов форм; {$R *. RES} – подключение файла ресурсов.

**Процедура** – это логически законченная и специальным образом оформленная часть программы, имеющая собственное имя. Процедура описывается один раз, а вызываться для выполнения может многократно из различных мест программы. Процедура может выполняться в результате наступления некоторого события (например, щелчок мышью по кнопке).

Структура процедуры:

Заголовок процедуры

Procedure <имя > [(список формальных параметров)];

<разделы описаний (объявлений)> Тело процедуры
begin
<раздел операторов>
end;

Все идентификаторы, используемые в разделе операторов, должны быть объявлены (перечислены) в разделе описаний.

В разделе операторов с помощью операторов языка программирования указывается последовательность действий, реализующих разработанный алгоритм.

Операторы, разделы описаний и описания однотипных элементов заканчиваются символом “;”.

В любом месте программных элементов могут находиться комментарии. Многострочный комментарий заключается в {} или в (* *), комментарий до конца строки размещается за символами //.

### 3.2. Файлы проекта Delphi.

Для каждого проекта создается несколько файлов, поэтому необходимо для хранения файлов каждого приложения создавать отдельную папку.

Для каждого проекта создаются файлы:

- Project1. dpr. Имя Project1 можно изменить при сохранении проекта командами *Save Project As* или *Save All.* При открытии существующего проекта необходимо открывать именно этот файл.
- Project1. exe - исполняемый файл создается в результате компиляции и компоновки проекта. Может запускаться на выполнение без среды Delphi.
- Project1. res служебные файлы , создаются
- Project1. cfg автоматически
- Project1. dof

В TurboDelph создаются еще три дополнительных файла в виде html-кода, с данными компиляции, компоновки, расположения файлов. Файл Project1.bdsproj является главным файлом (менеджером) проекта, его можно открывать при открытии проекта, но без Project1.dpr программа выполняться не будет.

Для каждой формы создается 3 файла, имена этих файлов должны отличаться от имени проекта.

- Unit1.pas. Имя Unit1 может быть изменено при сохранении модуля и формы, с помощью команд: *File – Save As* или *File – Save All*
- Unit1.dfm – файл формы
- Unit1.dcu – результат компиляции модуля.

Необходимыми для сохранения является файлы: Project1. dpr Unit1. dfm Unit1. Pas.

Unit1.~dfm, Unit1.~pas – предыдущие (перед последним изменением) версии соответствующих файлов, в Turbo Delphi создается папка _history, с несколькими предыдущими версиями.

Файл формы текстовый, его можно просмотреть в ИСР с помощью команды КМ формы View As Text, возврат к форме с помощью команды View As Form (Alt+F12).

### 3.3. Технология разработки программ с использованием визуальных средств проектирования

Процесс разработки программы в Delphi состоит из двух этапов, на каждом из которых часть действий выполняется вручную до загрузки Delphi, а часть в ИСР.

Первый этап – создание интерфейса приложения (конструирование форм).

Второй этап – разработка и реализация алгоритма решения задачи.

На первом этапе:

до работы в ИСР необходимо определить:

- все формы приложения, их иерархию;
- необходимые визуальные компоненты и их размещение на формах;
- события, которые будут управлять работой программы.

В ИСР конструирование формы осуществляется путем выбора необходимых компонентов, размещения их на форме и установки их свойств с помощью вкладки Properties (свойства) Инспектора Объектов.

ВК можно перемещать по форме, изменять их размеры и удалять, как любые объекты Windows.

На втором этапе:

до работы в ИСР необходимо:

- разработать алгоритмы всех процедур;
- написать программный код на языке Delphi для всех алгоритмов;
- установить события, в результате наступления которых будут реализованы разработанные алгоритмы.

В ИСР необходимо выделить ВК, для которого нужно написать процедуру обработки события, на вкладке Events выбрать нужное событие (например, OnClick – щелчок мышью) и выполнить двойной

щелчок в поле, рядом с названием этого события. Delphi автоматически размещает заготовку процедуры обработки этого события в модуле и переходит в окно редактора кода, в котором необходимо записать текст процедуры обработки данного события.

Имя процедуры создается автоматически из имени класса формы, имени визуального компонента и названия события без префикса On.

Например, для обработки события щелчок по первой кнопке на первой форме создается процедура с именем TForm1.Button1Click.

После разработки интерфейса и записи кода всех процедур обработки событий (все это режим проектирования) необходимо перейти к выполнению программы.

### 3.4. Имена, свойства и методы объектов

Каждый визуальный компонент (ВК) принадлежит к определенному классу объектов. Класс – это специальный тип, который содержит поля (данные), свойства этих данных и методы их обработки. Каждый визуальный компонент – это отдельный экземпляр реализации класса, имеющий имя. Название класса начинается с буквы Т(тип), а имена экземпляров этого класса состоят из названия класса без буквы Т и порядкового номера отдельного экземпляра реализации класса. Свойство Name определяет имя, под которым к компоненту будет обращение в программе.

TForm1 – класс всех форм с размещенными ВК
Form1 – экземпляр класса TForm1

Форма является объектом – контейнером для всех ВК, расположенных на данной форме. Свойство Caption определяет заголовок формы.

При работе с формами можно использовать процедуры обработки событий OnCreate (возникает при создании формы, один раз в начале программы), OnActivate (при активизации окна, передаче форме фокуса ввода), OnShow (при появлении окна на экране).

Свойства ВК могут устанавливаться на этапе конструирования форм или в программе (динамически). В Turbo Delphi все свойства и события в ИО разбиты на категории, при этом они могут повторяться в разных категориях. Можно упорядочить свойства и события по

категориям или по именам в алфавитном порядке, используя КМ вкладки ИО.

Для обращения к свойству в программе используется составное имя, состоящее из имен ВК и свойства, разделенных точкой. Объекты могут быть вложенными, в этом случае их имена тоже разделяются точками.

Обращение к свойству имеет вид:

[<объект-контейнер>].<объект>.<свойство>

Пример:

Form1.Caption:= 'Моя первая программа' (например, в обработчике события OnCreate)

Обращение к методам:

[<объект-контейнер>].<объект>.<метод> [(параметры)]

Пример:

Form1.Show

### 3.5. Визуальные компоненты, используемые при создании простых приложений.

Для установки или изменения свойств любого ВК на этапе проектирования интерфейса, его нужно выделить или на форме, или в окне структуры, или найти в списке объектов Инспектора Объектов.

**Tlabel – метка** (категория Standard)

Используется для размещения в формах различных текстовых надписей. Основное свойство компонента Caption: String содержит текст надписи.

В программе: Form1.label1.Caption:= 'ГГТУ'

Если процедура находится в модуле unit1, то имя формы Form1можно не указывать.

Свойство Transparent: Boolean определяет прозрачность метки (группа Miscellaneous – разнообразные). При значении True – сквозь

метку будут видны расположенные на ней компоненты (например для размещения текста на графических объектах), при значении False – пространство метки закрашивается цветом, задаваемым свойством Color.

**TEdit - редактируемое однострочное поле** (Standard)

Используется для ввода и вывода редактируемого текста. Текст может занимать только одну строку. Основное свойство Text:String содержит текст, который находится в окне компонента (группа Localizable).

Свойство ReadOnly:Boolean определяет можно ли редактировать текст на этапе выполнения программы. При значении True - текст редактировать нельзя *(для результатов работы)*, False - текст редактировать можно (для исходных данных).

Свойство PasswordChar:Char определяет символ, который заменяет любой вводимый символ текста. Используется для ввода паролей.

Для очистки поля ввода можно использовать метод Clear. Например, Edit1.Clear.

**TButton - кнопка** (Standard)

Свойство Caption определяет надпись на кнопке. Щелчок по кнопке во время выполнения программы вызывает процедуру обработки события OnClick.

Заголовок такой процедуры имеет вид:

Procedure TForm1.Button1Click (Sender: TObject);
Sender – объект, источник события для запуска процедуры.

Для создания процедуры обработки события OnClick для кнопки можно выполнить двойной щелчок по этой кнопке.

**TBitBtn** (Additional)

Это аналог кнопки Button, на которой кроме текста может располагаться рисунок. Свойство Caption определяет надпись на

кнопке. Свойство **Glyph** задает рисунок на кнопке. Свойство Kind определяет одну из 11 стандартных кнопок (OK, YES, NO).

**TMemo** (Standard)

Многострочное редактируемое текстовое поле, предназначено для ввода, вывода и редактирования многострочного текста. Свойство ReadOnly (Input) определяет возможность редактирования текста на этапе выполнения. Текст хранится в свойстве Lines и представляет собой нумерованный набор строк, нумерация начинается с нуля. Для вывода текста в программе можно использовать метод ADD, который добавляет строку в конец текста.

Memo1.Lines.ADD ('Группа ЭПП-11')

Если есть строки, хотя бы пустые:
Memo1.Lines[0]:= 'Иванов Дмитрий'
Memo1.Lines[1]:= 'Группа ЗИС-11'

Для того, чтобы ВК не выделялся на форме, а был виден только текст, можно установить его свойства Color = clBtnFace (в цвет формы), BorderStyle = bsNone (без границы).

**TImage - изображение** (Additional)

Определяет файл с изображением, содержимое которого будет выведено в ВК. Тип файла может быть bmp, jpeg и др. Имя файла задается основным свойством Picture (кнопка Load).

Свойство Stretch = True разрешает изменять размеры изображения так, чтобы оно полностью заполнило область ВК.

Свойство AutoSize = True позволяет изменять размеры ВК изображения так, чтобы в него полностью поместилось изображение.

Метод LoadFromFile позволяет загрузить в компонент изображение из файла на этапе выполнения программы.

Синтаксис метода LoadFromFile (<полное имя файла>)

Пример:
Image1.Picture.LoadFromFile ('Y:\Общие\Фото\bmw.jpeg')

Если файл находится в одной папке с проектом, то указывается только имя файла (bmw.jpeg).

Общие свойства компонентов:

- Font – определяет шрифт текста;
- Top, Left – задают координаты левого верхнего угла компонента на форме; для формы, относительно экрана;
- Height – высота ВК;
- Width – ширина ВК;
- Color – цвет фона ВК;
- Visible : Boolean – позволяет при необходимости во время выполнения программы любой ВК сделать невидимым (при значении свойства False) или видимым (при значении свойства True)
- Enabled : Boolean – определяет возможность активизации компонента. При значении свойства False компонент недоступен. Надписи на таких компонентах отображаются серым цветом.
- Hint – задает текст всплывающей подсказки при наведении курсора мыши на ВК на этапе выполнения; при этом для вывода подсказки нужно установить значение свойства ShowHint = True.

## Тема 4. Программирование линейных вычислительных алгоритмов

### 4.1. Ввод данных

Ввод - это передача информации с внешнего устройства (клавиатура, диск) в ОЗУ для дальнейшей обработки.

Ввод данных осуществляется обычно в результате обработки свойств ВК. Данные вводятся с клавиатуры и размещаются в ВК.

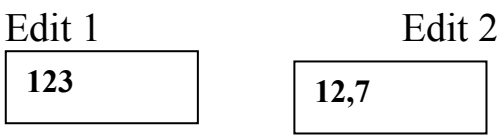
При реализации алгоритмов обработки данных будет использовано свойство компонента, содержащее эти данные. Для ввода значений переменных целого и вещественного типов можно использовать компонент Edit.

Данные, вводимые с клавиатуры, определяются его свойством Text.

Свойство Text имеет строковый тип, поэтому при вводе вещественных и целых чисел необходимо использовать функции преобразование типов StrToFloat (S), StrToInt (S).

Аргумент этих функций имеет строковый тип, результат соответственно вещественный или целый.

Пример:




```
Var
    m:byte;
    b:real;
Begin
   m:= StrToInt (Edit1.Text);
   b:= StrToFloat (Edit2.Text);
    ……
```

При вводе чисел текст, вводимый с клавиатуры, не должен содержать никаких символов кроме цифр, ведущего знака «+» или «-» и десятичной запятой для вещественных чисел.

Если текст содержит другие символы, включая пробелы или пустую строку, выполнение программы завершается аварийно, выводится сообщение об ошибке, а затем окно проекта. Для повторного выполнения или редактирования программы ее сначала необходимо завершить с помощью команды *Run – Program Reset* или кнопки *Reset* (Turbo Delphi).

### 4.2. Вывод данных

Вывод – это передача информации из ОЗУ на внешнее устройство (монитор, принтер, диск).

Для вывода данных можно использовать ВК Label, Edit, Memo. Для преобразования целого или вещественного значения в строку необходимо использовать функции IntToStr(x) и FloatToStr (x).

```
Edit1.Text:=IntToStr (m);
Label1.Caption:=FloatToStr (b);
```

Если выводимые данные не предназначены для редактирования (результаты), то при их выводе в компонент Edit нужно запретить

возможность редактирования, установив для свойства ReadOnly значение True.

Для объединения нескольких строк в одну при их выводе используется операция сцепления строк (+).

Примеры:

Label1. Caption: = 'x='+IntToStr(x);

Memo1.Lines.Add ('Сумма =' + FloatToStr (Sum));

Программным путем можно создать метку, содержащую несколько строк текста:

label2.Caption:= 'ГГТУ' + #13+ 'ИПК и ПК';

Для вывода различных предупреждающих или информационных сообщений можно использовать процедуру: ShowMessage (<текст сообщения>.

Пример: ShowMessage ('Ошибка ввода');

### 4.3. Реализация линейного алгоритма

**Пример 4.1.** Вычислить площадь треугольника со сторонами a, b, c.

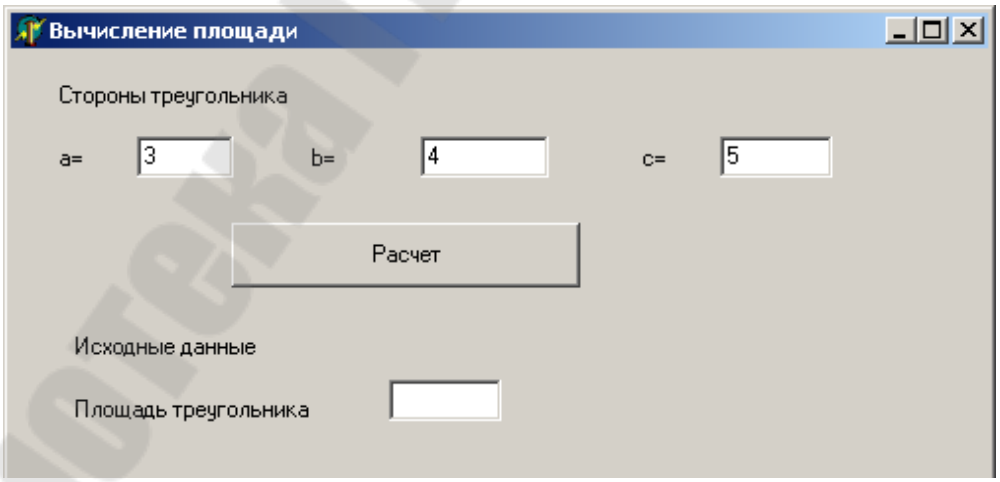



*Рис 4.1.* Вид окна проекта (главного окна)

Таблица 4.1

**Таблица используемых визуальных компонентов**

| Элемент интерфейса | ВК | Свойство | Значение свойства (в ИО) |
|---|---|---|---|
| Заголовок окна | Form1 | Caption | Вычисление площади |
| Стороны треугольника | Label1 | Caption | Стороны треугольника |
| a= | Label2 | Caption | a= |
| Поле ввода a | Edit 1 | Text | |
| b= | Label3 | Caption | b= |
| Поле ввода b | Edit2 | Text | |
| c= | Label4 | Caption | c= |
| Поле ввода c | Edit3 | Text | |
| Кнопка Расчет | Button1 | Caption | Расчет |
| Исходные данные | Label5 | Caption | Исходные данные |
| Поле вывода исходных данных | Label6 | Caption | |
| Площадь треугольника | Label7 | Caption | Площадь треугольника |
| Поле вывода площади | Edit4 | Text<br>Read Only | <br>True |

Таблица 4.2

**Таблица используемых событий**

| Визуальный компонент | Событие | Имя процедуры обработки события | Реализуемый алгоритм |
|---|---|---|---|
| Button 1 | OnClick | TFrom1.Button1Click | Расчет площади треугольника |

**Текст процедуры**

Тест процедуры обработка события щелчок по кнопке “Расчет” реализует разработанный алгоритм.

```
Procedure T From1.Button1Click (Sender:TObject);
  Var
       a,b,c: byte;
       p,s; real;
  Begin
       a:= StrToInt (Edit1.Text);
       b:= StrToInt (Edit2.Text);
       c:= StrToInt (Edit3.Text);
       Label6.Caption:= 'a='+IntToStr(a)+ ' b='+ IntToStr(b)+
                          ' c='+IntToStr(c);
       p:=(a+b+c)/2;
       S:=sqrt(p*(p-a)*(p-b)*(p-c))
       Edin4.Text:=floatToStr(S)
  End
```

**Тест**

Исходные данные: a=3 b=4 c=5 Ожидаемый результат S=6

## 4.4. Разработка интерфейса приложений, использующих несколько форм

**Создание новой формы**

Любой проект имеет хотя бы одну форму, которая появляется на экране в начале выполнения программы и называется главной формой. Для создания новой формы и подключения ее к проекту на этапе проектирования используется команда *File – New – Form (Delphi for Win32)*. Вид каждой формы конструируется так же, как и главной (с помощью размещения на ней ВК и создания для них процедур обработки событий, которые помещаются в модуль связанный с формой).

**Методы работы с несколькими формами**

Для работы с несколькими формами используется следующие методы:

Show – показывает форму в немодальном режиме. Если до обращения к методу формы не было на экране, то выводит форму на экран. Если форма была на экране – делает ее активной и передает ей фокус.

Пример: Form2.Show

ShowModal – показывает форму в модальном режиме. Отличие от Show состоит в том, что никакие действия с другими окнами не возможны, пока модальное окно не будет закрыто. Модальные окна обычно требуют от пользователя принятия каких-либо решений или выводят информационные сообщения. Никакие операторы процедуры не выполняются, пока не закончится выполнения метода ShowModal, и модальное окно не будет закрыто.

Пример: Form2.ShowModal

Close – закрывает окно. Для главного окна (формы) завершает работу приложения.

Hide – скрывает окно. Отличается от Close возвращаемым результатом для модальных окон.

Для вывода на экран закрытого или скрытого окна (кроме главной формы) используются методы Show и ShowModal.

SetFocus – передает форме или указанному ВК фокус ввода, до использования метода форма должна быть выведена на экран. После использования методов Show и ShowModal фокус ввода передается форме автоматически.

**Фокус ввода**

Формы могут содержать несколько ВК, в которые нужно вводить данные с клавиатуры. Для того, что бы выделить ВК, в который будут вводиться данные, можно передать ему фокус ввода в программе. (В противном случае придется щелкнуть мышью в нужном ВК)

В поле Edit появится текстовый курсор. На кнопке появится пунктирная рамка. Это означает, что кнопка связана в этот момент с

клавиатурой, нажатие клавиши Enter аналогично щелчку по кнопке. Компоненту Label фокус ввода не передается.

Если фокус ввода передан форме, то форма становится активной и выводится поверх всех окон. При первом открытии формы установка фокуса ввода определяется свойством TabOrder ВК формы. Значение этого свойства устанавливается автоматически в виде целого числа, начиная с нуля, в порядке создания ВК, но может быть изменено в ИО. Тот элемент, у которого TabOrder=0, первым на форме получает фокус ввода. Использование клавиши Tab приводит к переходу к ВК с TabOrder =1,2,3…. Использование метода SetFocus изменяет этот порядок. При повторной передаче фокуса ввода ранее открытой форме он устанавливается на том ВК, с которым последним работали на этой форме (обычно на кнопке, щелчок по которой запустил процедуру, открывшую другую форму).

Пример**:** Form2.Edit1.SetFocus.

**Работа с визуальными компонентами разных форм**

При работе с несколькими формами можно в процедуре модуля одной формы открыть вторую форму и обращаться к ВК этой формы.

Для этого необходимо:

1. Подключить к модулю первой формы модуль второй. Для этого в модуле первой формы необходимо указать:

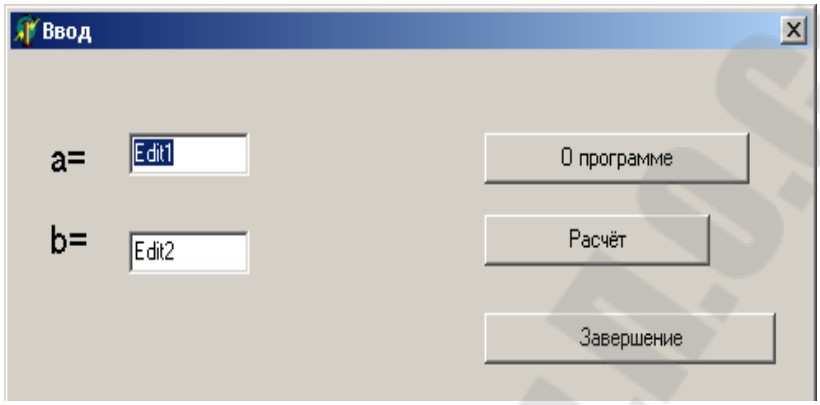
```
implementation

    uses <имя модуля 2>;
```

По умолчанию Unit2, Unit3.. или то имя, которое было дано второму модулю при сохранении. Подключение можно сделать автоматически, используя при открытом первом модуле команду *File - Use Unit,* и выбрать из появившегося списка нужный модуль.

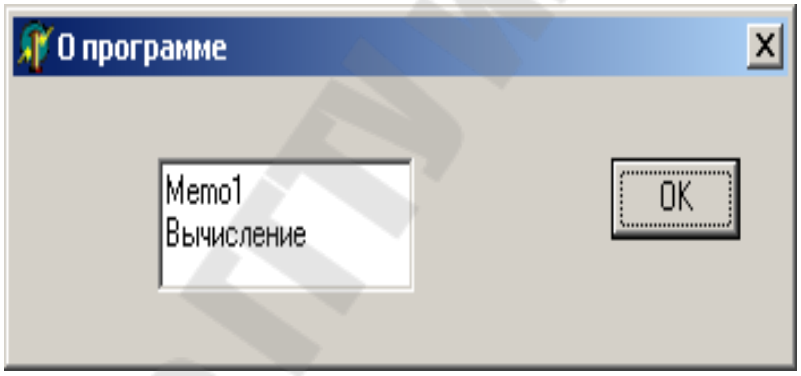
2. Использовать полные имена для свойств и методов ВК с указанием имени формы, в которой они находятся.

Пример: Form2.Label3:=’Привет’;

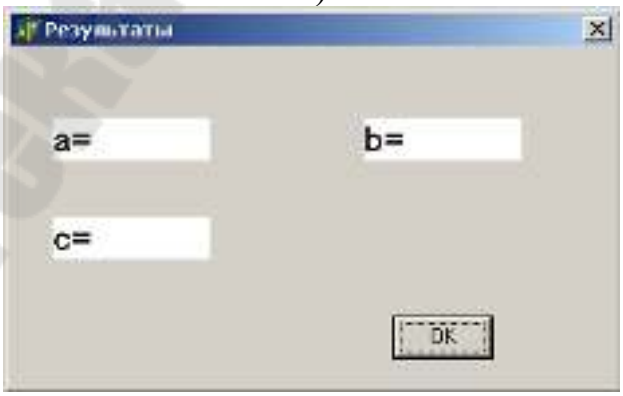
**Пример 4.2.** Вычислить гипотенузу **c** прямоугольного треугольника с катетами **a** и **b**, исходные данные и результат вывести в модальное окно.




а)




б)




с)

*Рис 4.2.* Вид окон проекта для: а) формы 1; б) формы 2; с) формы 3

**Unit Unit1;**

…….
implementation
  uses  Unit2, Unit3;
……

- Procedure  TForm1.Button1Click(…);
  Begin
    Form2.ShowModal
  End,
- Procedure  TForm1.Button2Click(…);
  Var
  a,b,c: real;
  Begin
  a:= StrToFloat (Edit1. text);
  b:= StrToFloat (Edit2. text);
  c:=Sqrt(Sqr(a)+sqr(b));
  //Если форма не модальная, то можно написать:
  Form3.Show;  Но не ShowModal !!
  Form3.Label 1.Caption:= ' a='+…..  устанавливается
  …….  независимо от того
  Form3.Label 3.Caption:= 'c='+…..  видна ли форма на экране
  Form3.ShowModal
  End;

- Procedure  TForm1.Button3Click(…);
    Begin
      Close   (Form1.Close)
    End;

**Unit Unit2;**

……..
implementation
  Uses Unit1;

- Procedure TForm2.Button1Click (…);
  Begin
    Close    (Form2.Close)
    Form1.Edit1.Set Focus;

End

**Unit Unit3;**

……
implementation
Uses Unit1;

- Procedure TForm3.Button1Click(..);
  Begin
  Close;
  Form1.Edit1.Text:= '';
  Form1.Edit 2.Clear;
  Form1.Edit1.Set Focus;
  End;

### 4.5. Обработка исключительных ситуаций (исключений)

Во время работы программы могут возникнуть такие ситуации, когда программа не может выполняться в соответствии с алгоритмом. Например, данные не введены или введены символы, которые нельзя представить как целые или вещественные числа, в случае деления на 0 и т.д.

Такие ситуации называются исключительными (исключениями), при их возникновении программа завершается аварийно и выдается системное сообщение об ошибке.

В Delphi имеется возможность избежать аварийного завершения программы, если использовать для обработки исключений защищенные блоки.

Для того, что бы эти блоки выполнялись, необходимо запретить Delphi завершать программу аварийно при возникновении ошибок. Для этого в среде Delphi7 необходимо выполнить команду *Tools – Debugger Options* и на вкладке Language Exceptions

Снять флажок в переключателе Stop on Delphi Exceptions.

Если флажок снят, но нет обработки исключений с помощью защищенного блока, то программа не завершится аварийно, а только выводится сообщение об ошибке в модальном окне.

В Turbo Delphi выполнить команду *Tools – Options.* Найти в списке *Options:*

*Debugger Options – Borland Debugger – Language Exceptions.*

Снять флажок  в переключателе Notify on language exceptions.

**Защищенный блок**

Для обработки исключений используется защищенный блок вида:

```
try
  < блок операторов 1 >
except
   <блок операторов 2 >
end;
```

Блок операторов 1 – это группа операторов, при выполнении которых могут возникнуть исключения.

Блок операторов 2 – это группа операторов, которые необходимо выполнить, если исключение возникло.

Выполнение защищенного блока:

Выполняется блок операторов 1. Если ошибки не возникло, то происходит переход к оператору, расположенному за end, если возникла исключительная ситуация, то выполняется блок операторов 2, а затем переход к оператору, следующему за end.

В любой части защищенного блока могут находиться другие защищенные блоки.

**Пример 4.3.** Ввести значения переменных a,b. Если они не введены или введены с ошибками, вывести об этом сообщение и повторить, при необходимости, их ввод.

```
flag:=0;
try
  a:=StrToFloat(Edit1.Text);
  try
    b:=StrToFloat(Edit2.Text);
     // < основная часть процедуры>
    except
        ShowMessage ('Ошибка при вводе значения b');
        Edit2.Text:= '';
        Edit2.SetFocus;
        flag:=1
```

```
  end;
 except
        ShowMessage ('Ошибка при вводе значения a');
        Edit2.Text:= '';
        Edit2.SetFocus;
        flag:=1
end;
```

Можно использовать переменные типа флаг или переключатель для определения наличия ошибки ввода, проверять его после защищенного блока, а затем писать основную часть процедуры

```
if flag=0 then
      <основная часть >;
```

## Тема 5. Алгоритмизация и программирование разветвляющихся алгоритмов

### 5.1. Общие сведения

Разветвляющимся называется алгоритм, в котором в зависимости от заданного условия может выполняться или не выполняться та или иная последовательность действий (в программе операторов). Каждый возможный путь вычислений называется ветвью алгоритма.

Для реализации разветвляющегося алгоритма в Delphi есть два оператора:

If – условный оператор, Case – оператор выбора.

Эти операторы влияют на порядок выполнения других операторов программы.

Для записи условий в операторе if используются логические выражения.

### 5.2. Логические выражения

Логические выражения имеют тип Boolean и могут принимать одно из двух значений: True или False.

Логические выражения состоят из арифметических выражений, операций отношения и логических операций.

Операции отношения (= ; <> ; < ; <= ; > ; >=) выполняют сравнение двух операндов и определяют истинно выражение (его значение true) или ложно (его значение false).

Логические операции (not; and; or; xor**)** используются для образования сложных логических выражений. Операнды логических операций должны иметь логический тип.

Данные для примеров: a:=3; b:=7;

**Not** (логическое отрицание).

Синтаксис: not (<логическое выражение>)

**And** (логическое И).

Синтаксис: (<логическое выражение 1>) and (<логическое выражение 2>)

**Or** (логическое ИЛИ).

Синтаксис: (<логическое выражение 1>) or (<логическое выражение 2>)

**Xor** (исключающее ИЛИ).

Синтаксис: (<логическое выражение 1>) xor (<логическое выражение 2>)

Таблица 5.1

Результат логических операций

| Операнд 1 | Операнд 2 | Not | And | Or | Xor |
|---|---|---|---|---|---|
| true | true | false | true | true | false |
| true | false | | false | true | true |
| false | true | true | false | true | true |
| false | false | | false | false | false |

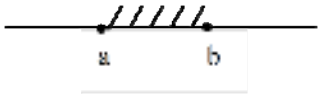
Приоритет логических операций выше, чем операций отношения, поэтому необходимо использовать круглые скобки для указания порядка действий при вычислении значения логического выражения.

Примеры:
a:=5; b:=7;

not (a<b) – False (a=7) and (b>3) – False
(a=7) or (b>3) – True (a=7) xor (b>3) – True

Примеры: Записать логическое выражения, которое имеет значение true, если:

1. значение переменной x принадлежит полуинтервалу (a;b]

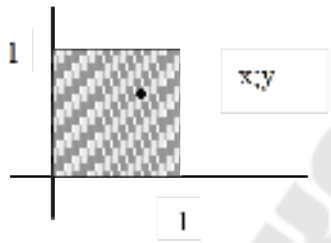



(x>a) and (x<=b)

2. значение переменной x находится вне полуинтервала [a;b)


(x<=a) or (x>b)

3. Точка с координатами (x; y) находится внутри квадрата со стороной 1



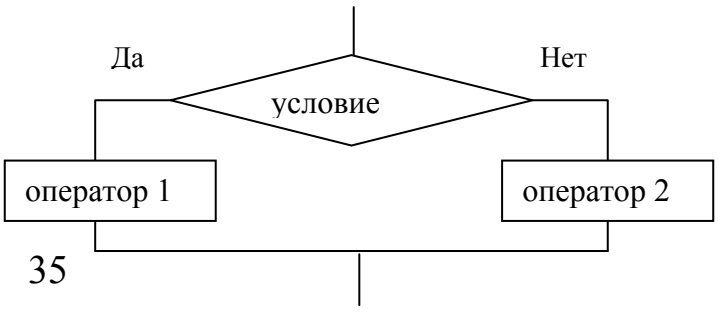

(x>=0) and (x<=1) and (y>=0) and (y<=1)

### 5.3. Условный оператор If

Имеет две формы: полную и сокращённую.

- полная форма (1)

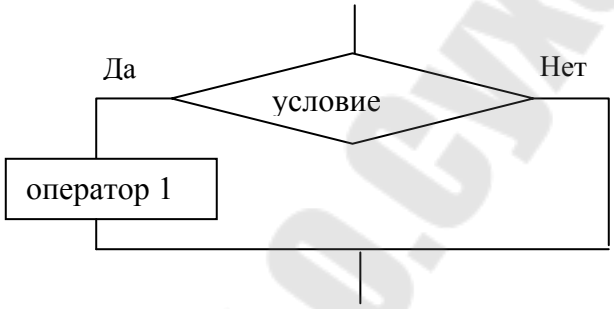
if <условие>
then
< оператор1>



else
< оператор2>;

- сокращенная форма (2)

if <условие>
then
<оператор1>;




Для записи условий используются логические выражения.

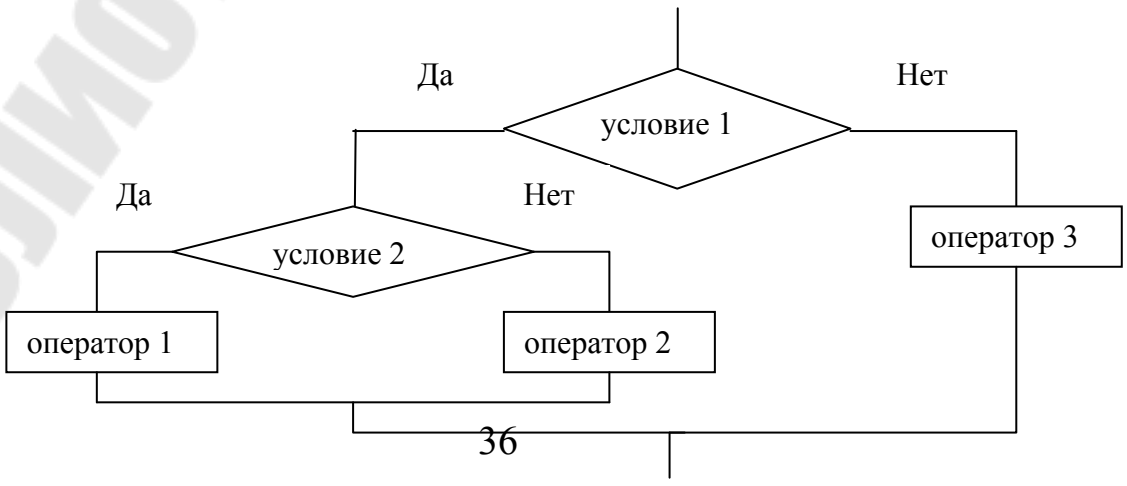
Пример: If (x>=10) and (x<=20)
then
ShowMessage ('Значение x в интервале [10,20] ')
else
ShowMessage ('Значение x вне интервале [10,20] ');

Выполнение оператора:

Вычисляется значение логического выражения в условии.

1. Если значение логического выражения true, то выполняется оператор 1, если false - оператор 2
2. Если значение логического выражения true, то выполняется оператор 1, если false – то оператор никаких действий не производит, и программа переходит к выполнению следующего за If оператора.

Операторы If могут быть вложены в другие операторы If.



В процедуре:

```
if  <условие 1>
        then
            if < условие 2>
                then
                    <оператор 1>
                else
                    <оператор 2>
        else
            <оператор 3>;
```

Каждое else соответствует тому then, который ему непосредственно предшествует.

Перед else “;” не ставится.

### 5.4. Составной оператор

Это последовательность из произвольного числа операторов, заключенных в операторные скобки begin . . . .end.

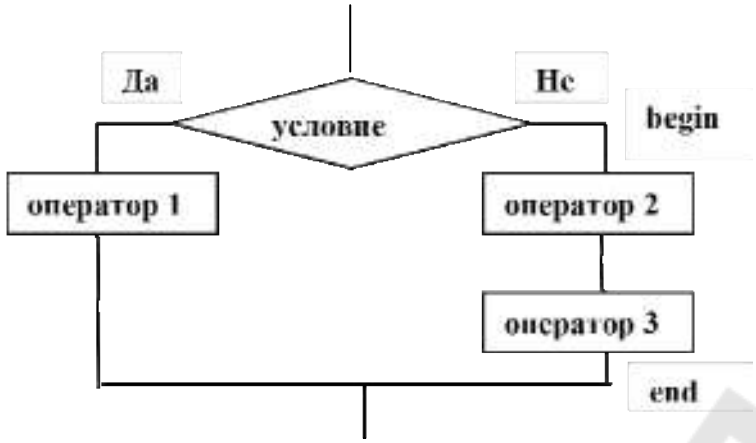
Синтаксис:

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

Перед end символ “;” может не ставиться.

Составной оператор может использоваться в любом месте программы, где допускается запись оператора. Он применяется, если по какой-либо ветви условия (в программе после слов Then или Else) необходимо записать не один оператор, а несколько.

Пример:




```
If  <условие>
   then<оператор 1>
   else
      begin
        <оператор 2>
        < оператор 3>
      end;
```

## 5.5. Оператор выбора Case

```
case  <выражение> of
    <список констант выбора 1>: <оператор1;>
    <список констант выбора 2>: <оператор2;>
    ……..
    else  <оператор>;
end;
```

Графическая схема алгоритма:

Выражение может иметь любой порядковый тип (byte, integer, boolean, char), но не может иметь тип real, string. Тип выражения должен совпадать с типом констант выбора. Оператор может быть простым или составным. Константы в списке разделяются запятыми и “**. .**” для диапазона. Часть else <оператор> может отсутствовать.

**Пример 5.1.** Определить к какому времени года относится введенный номер месяца.

```
case nom_mes of
    1,2,12 : ShowMessage (' Зима');
    3..5 : ShowMessage ('Весна');
    6..8 : ShowMessage ('Лето');
    9..11 : ShowMessage ('Осень')
  else ShowMessage ('Номер месяца введен неправильно')
end;
```

## 5.6. Визуальные компоненты, используемые в разветвляющихся алгоритмах

**TCheckBox –** флажок (вкладка Standard).

Используется для указания пользователем своего выбора типа *да/нет*. На форме может быть несколько таких компонентов, при этом состояние любого из них никак не зависит от состояния остальных.

Свойство Caption определяет текст, находящийся рядом с переключателем.

Свойство Checked**:**Boolean, содержит значение, соответствующее выбору пользователя (True – флажок установлен, False – флажок не установлен).

**TRadioButton** – переключатель (вкладка Standard).

Используется для выбора одного из нескольких взаимоисключающих значений.

Свойство Checked**:**Boolean определяет состояние переключателя. В компонент контейнер помещается по крайней мере

два таких переключателя. Если один из них включен (Checked= True), то остальные выключены (Checked= False).

Свойство Caption определяет текст, находящийся рядом с переключателем.

**TRadioGroup** (вкладка Standard) – группа переключателей, контейнер для размещения переключателей класса TRadioButton.

Свойства ВК определяют:

Caption – заголовок группы;

Columns**:** Integer – количество столбцов в группе;

ItemIndex **:** Integer – содержит номер (значение индекса) установленного переключателя, нумерация начинается с нуля; по умолчанию равен -1, т.е. ни один переключатель не установлен.

Items **:** TStrings – содержит список строк с поясняющим текстом, находящимся рядом с переключателями; количество строк в списке определяет количество переключателей в группе (хранится в свойстве. Items.Count).

**TListBox** - список и **TComboBox**– комбинированный (раскрывающийся) список (вкладка Standard)

Свойства Columns, ItemIndex и Items имеют то же назначение, что и у компонента TRadioButton. Свойство ItemIndex содержит индекс элемента, имеющего фокус ввода (выбранного).

**Пример 5.2.** Определить какое задание необходимо выполнить и открыть соответствующую форму.

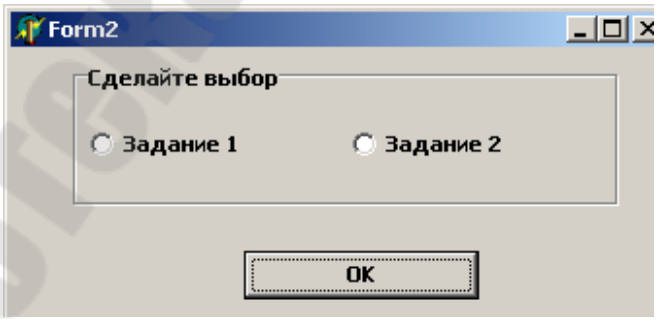



*Рис. 5.1.* Вид окна проекта

Выполнение: создать ВК **RadioGroup1.** Установить для него свойства:

- Caption: Сделайте выбор
- Columns: 2
- Items: Задание 1
  Задание 2

```
Procedure TForm1.Button1Click (Sender:TObject);
  var nom:byte;
begin
   nom:=RadioGroup.ItemIndex;
case nom of
    0: Form2.Show;
    1: Form3.Show;
    else ShowMessage ('Задание не выбрано')
end;
```

**Пример 5.3.** Вычислить значение функции и определить номер формулы, по которой она вычислялась. Вид функции:

$$y = \begin{cases} \sqrt{|x|}, \ \textit{если} \quad -5 \le x < 1 & (1) \\ x^3, \ \textit{если} \quad x > 1; x \ne 10 & (2) \\ 2 * x^2 + 1 \quad - \textit{в остальных случаях} & (3) \end{cases}$$

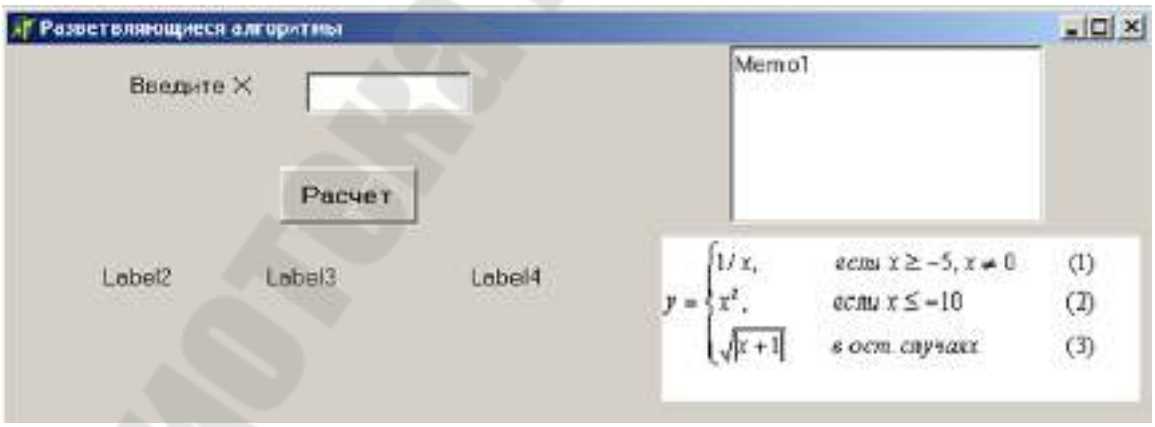



*Рис. 5.2.* Вид окна проекта

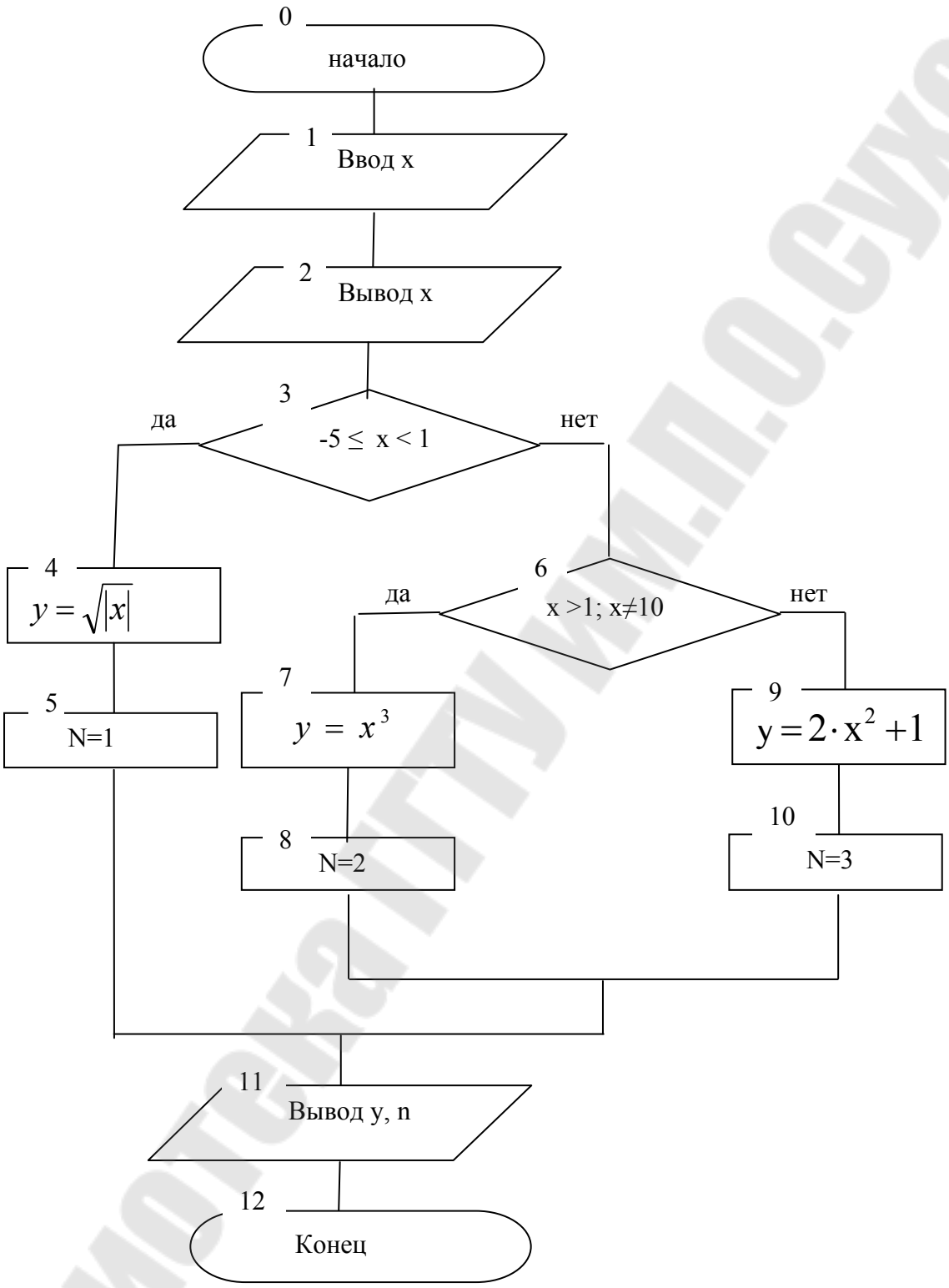



*Рис. 5.3.* Графическая схема алгоритма примера 5.3

**Текст процедуры:**

```
Procedure TForm1.Button1Click (Sender:TObject);
 var
     x, y : real;
     n : byte;
 begin
     x := StrToFloat (Edit1.Text);
     Label2.Caption:= 'x='+FloatToStr (x);
     If (x>= -5) and (x<1)
            then
                begin
                     y:= sqrt(abs(x));
                     n:= 1
                end
            else
                 If (x>1) and (x<>10)
                     then
                       begin
                          y:= sqr (x)*x;
                          n:=2
                       end
                      else
                                begin
                                   y:= 2*sqr (x)+1;
                                   n:=3
                                end
      Label 3.Caption := 'y=' + FloatToStr(y);
      Label 4.Caption := 'N формулы =' + IntToStr(n)
      //  Memo2.Lines.Add ('x='+FloatToStr(x)+' y='+FloatToStr(y)+' N='+
                                IntToStr(n))
 End;
```

**Тесты**

Для задач с разветвляющимися алгоритмами тесты должны быть в каждом интервале вычисляемой функции и в каждой точке ветвления (значение аргумента, при котором изменяется выражение для вычисления функции)

Таблица 5.1.

Тесты для примера 5.3

| x | y | N формулы |
|---|---|---|
| -6 | 73 | 3 |
| -5 | 2,24 | 1 |
| 0 | 0 | 1 |
| 1 | 3 | 3 |
| 2 | 8 | 2 |
| 10 | 201 | 3 |
| 11 | 1331 | 2 |

# Тема 6. Алгоритмизация и программирование циклических алгоритмов

## 6.1. Общие сведения

Циклическим называется алгоритм, в котором некоторая последовательность действий может выполняться несколько раз в зависимости от заданного условия. Повторяющаяся последовательность действий (в программе операторов) называется телом цикла.

Условие, от выполнения которого зависит число повторений цикла, включает в себя, по крайней мере, одну переменную, которая называется параметром или переменной цикла. Эта переменная должна обязательно изменяться в теле цикла.

Циклические алгоритмы реализуются в программе с помощью операторов цикла.

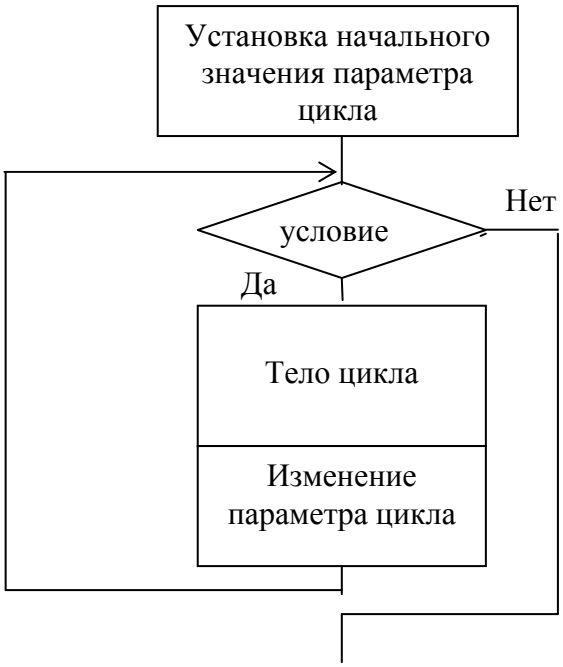
В Delphi три оператора цикла:

**While** – оператор цикла с предусловием;

**For** – оператор цикла с параметром;

**Repeat** – оператор цикла с постусловием.

## 6.2. Оператор цикла с предусловием While




В программе:

заголовок цикла

⇩

**While** <условие> **do**

*begin*

<тело цикла>;

*end:*

*Рис 6.1*. Графическое представление оператора

В условии записывается логическое выражение. Тело цикла – простой или составной оператор.

Т.к. в теле цикла обычно находится более одного оператора, то после заголовка цикла записывается begin, а в конце тела цикла – end (составной оператор).

При использовании оператора While необходимо правильно устанавливать начальное значение и изменение параметра цикла, иначе цикл может выполняться бесконечно. Для выхода из зациклившейся программы используется команда *Run – Program Reset*.

**Пример 6.1.** Вывести значения квадратов четных чисел от 2 до 20.

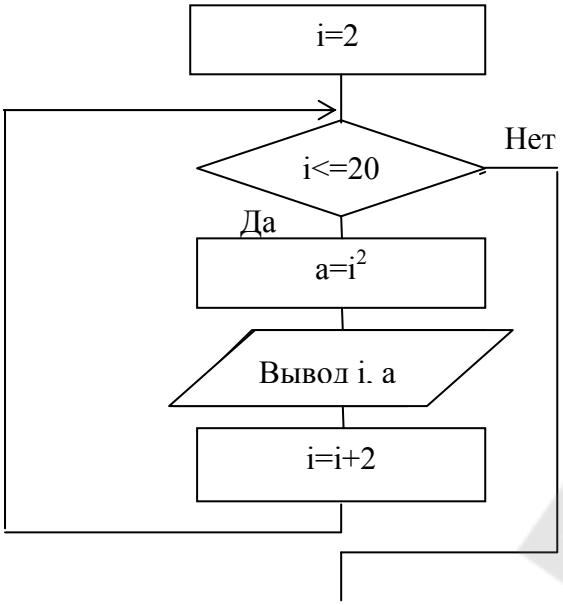



*Рис. 6.2.* Графическая схема алгоритма примера 6.1

Фрагмент процедуры:

```
i:=2:
While i <= 20 do
    begin
      a:=sqr(i);
      Memo1.Lines.Add ('квадрат'+IntToStr(i)+ '='+IntToStr(a));
      i:=i+2
    end;
```

### 6.3. Оператор цикла с параметром For

Оператор For является частным случаем цикла с предусловием. Любой циклический алгоритм можно реализовать с помощью оператора While, но там, где можно использовать оператор For, лучше использовать именно его.

Оператор имеет две формы.

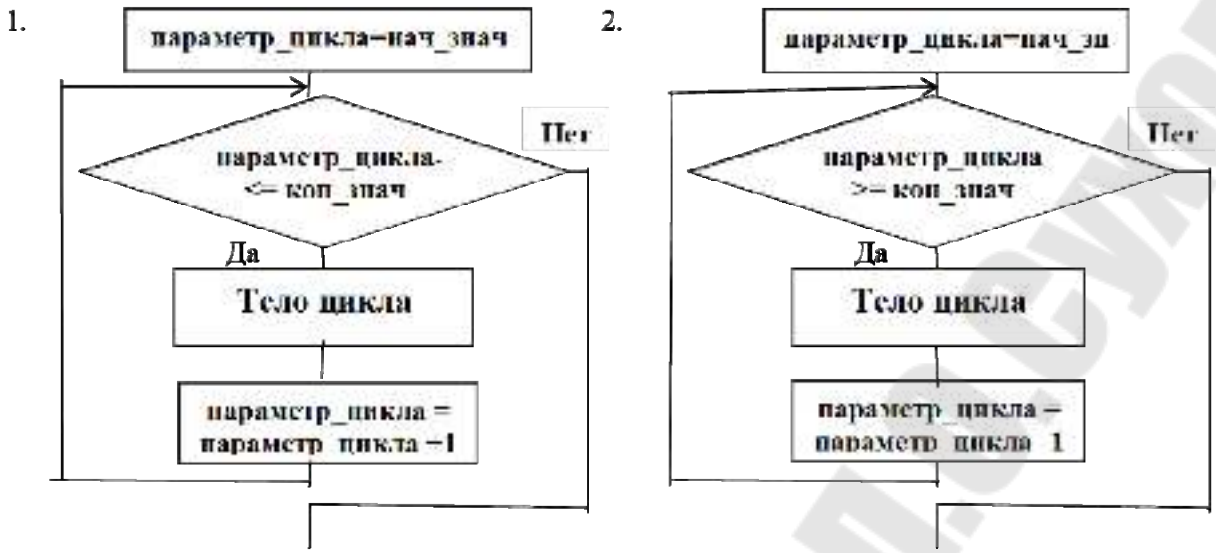



*Рис. 6.3.* Графическое представление оператора

В программе: Заголовок цикла

1. For параметр_цикла := нач_знач to кон_знач do
   <тело цикла>;

2. For параметр_цикла := нач_знач downto кон_знач do
   <тело цикла>;

Параметр цикла – переменная целого (порядкового, но не вещественного типа). Тело цикла – простой или составной оператор. Нач_знач, кон_знач могут быть константами, переменными и выражениями того же типа, что и параметр цикла. При выполнении оператора For операторы тела цикла повторяются для всех значений параметра цикла от нач_знач до кон_знач с шагом 1 для первой формы и -1 – для второй. Если в первой форме нач_знач > кон_знач, а во второй нач_знач < кон_знач, то тело цикла не выполняется ни разу, если нач_знач = кон_знач, то один раз.

Особенности цикла For:

1. установка начального, конечного значения и изменение параметра цикла происходит автоматически в соответствии с заголовком цикла;
2. запрещено изменять параметр цикла, его начальное и конечное значение в теле цикла;

3. значение параметра цикла после выхода из цикла не определено и не может использоваться в дальнейших вычислениях;

4. цикл For используется тогда, когда:

- параметр цикла – переменная целого типа;
- шаг параметра цикла +1 или -1;
- число повторений цикла можно определить до начала выполнения цикла.

В противном случае используется цикл While.

**Пример 6.2.** Вывести целые числа и их квадраты а) от 1 до 20; б) от 20 до 1.

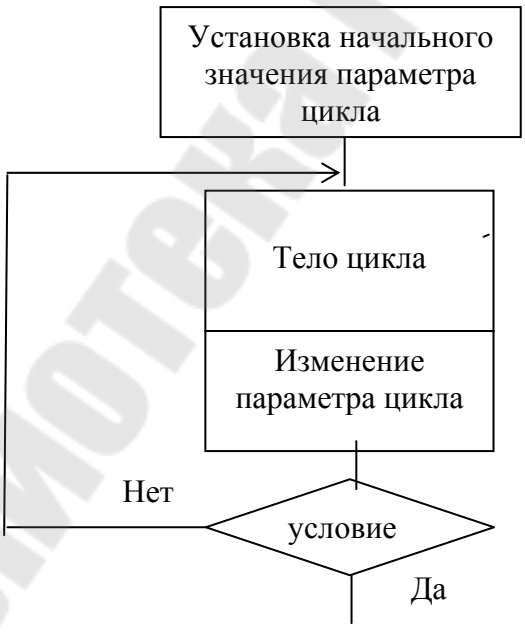
а)

```
For i:=1 to 20 do
        Memo1.Lines.Add (‘ квадрат ‘+IntTostr(i)+’= ‘+ IntTostr(i*i))
 Memo1.Lines.Add((‘ квадрат ‘+IntTostr(i)+’= ‘+IntTostr(i*i))
```

б)

```
For i:=20 downto 1 do
    Memo1.Lines.Add ('квадрат '+ IntTostr (i) + '= ' + IntTostr (i*i))
```

## 6.4. Оператор цикла с постусловием Repeat




В программе:

```
Repeat
  <тело цикла>
until <условие>;
```

*Рис. 7.4.* Графическая схема алгоритма

Особенности цикла Repeat:

1. Тело цикла повторяется хотя бы один раз.
2. Тело цикла повторяется до тех пор, пока значение логического выражения в условии False, выход из цикла при значении логического выражения в условии True.
3. В теле цикла может быть любое число операторов без операторных скобок begin…end. Конец цикла Until <условие>.

**Пример 6.3.** Вычислить сумму четных чисел от 2 до 20.

```
i:=2;  s:=0;
Repeat
     s:=s+sqr(i);
     i:=i+2
until i > 20;
```

## 6.5. Визуальный компонент TStringGrid

Таблица строк (текстовая таблица, вкладка Additional)

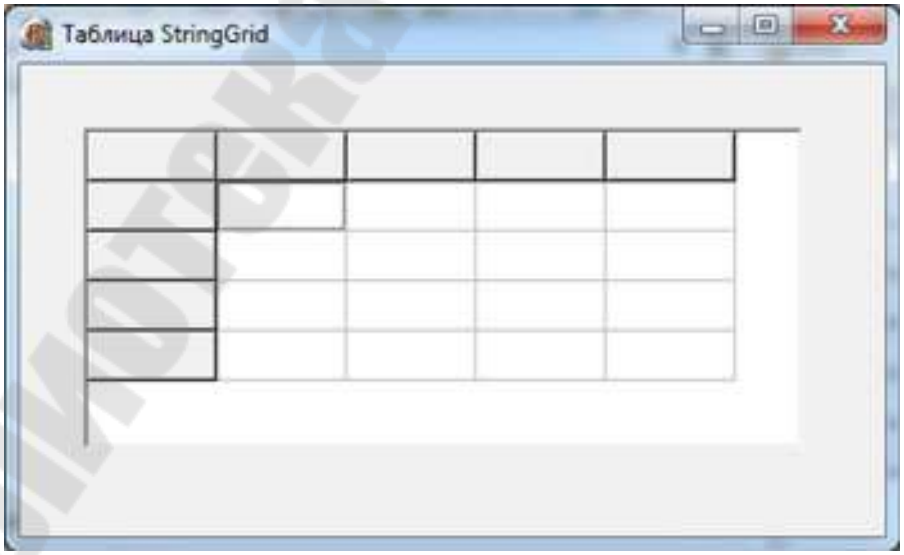



*Рис. 6.5.* Визуальный компонент TStringGrid

ВК предназначен для создания таблицы, в ячейках которой находятся текстовые строки. Используя функции преобразования типов, таблицу можно применить и для работы с числовой информацией. Таблица используется для ввода и вывода данных.

Таблица делится на 2 части: фиксированную и рабочую. Фиксированная часть обычно используется для вывода заголовков строк и столбцов и содержит 1 строку и 1 столбец. Если рабочая часть не помещается полностью в окне компонента, то появляются полосы прокрутки. Фиксированная часть постоянно остается на экране, может изменяться только её содержимое – заголовки строк и столбцов.

Основные свойства ВК:

Размер таблицы:

- ColCount (количество столбцов);
- RowCount (количество строк).

Фиксированная часть таблицы:

- FixedCols – число столбцов в фиксированной части;
- FixedRows – число строк в фиксированной части;
- FixedColor – цвет фона фиксированной части.

Значения свойств ColCount и RowCount должны быть по крайней мере на 1 больше, чем, соответственно, значения свойств FixedCols и FixedRows.

- Col, Row –содержат № столбца и строки ячейки, имеющей фокус ввода;
- DefaultColWidth (по умолчанию 54 пикселя) – ширина ячейки таблицы;
- Option – содержит параметры таблицы.

Значение параметра goEditing=True позволяет изменять данные в ячейках на этапе выполнения программы (для ввода данных из таблицы), goTabs=True позволяет перемещаться по ячейкам таблицы с помощью клавиши Tab.

Основное свойство таблицы Cells [№ столбца, № строки] определяет содержимое ячейки с заданными номерами столбца и

строки, нумерация начинается с 0, включает фиксированную и рабочую часть.

Пример: Cells [0,0]:=’№ п/п’;  Cells [2,3]:=FloatToStr(x);

### 6.6. Визуальный компонент TChart


*Рис. 6.6.* Визуальный компонент TChart

ВК **TChart** (диаграмма**,** вкладка Additional).предназначен для графического представления числовых данных. Все свойства компонента устанавливаются в диалоговом окне **Editing Chart**, которое имеет несколько вкладок. Окно можно открыть с помощью команды *Edit Chart* контекстного меню ВК или использовать вкладку свойств Инспектора Объектов. Свойство SeriesList открывает вкладку Series (серии), свойство Title – вкладку Titles (заголовки).

ВК может содержать несколько диаграмм различных видов. Для построения каждой из них нужно:

- на этапе проектирования на вкладке *Series* с помощью кнопки **Add** добавить пустую серию, которая будет содержать данные, отображаемые с помощью диаграммы, и выбрать ее тип. Каждая серия имеет имя SeriesList [N], где N – порядковый номер серии, начиная с нуля;
- если необходимо, можно очистить серию от старых данных с помощью метода Clear (в противном случае новые точки могут добавляться к старым данным).
- в программе с помощью метода **AddXY** вставить в серию данные.

Например, для построения графика функции y=f(x) при ее табулировании:

- на этапе проектирования создать компонент Chart1, добавить одну серию типа Line (убрать флажок 3D);
- в программе после ввода исходных данных очистить серию от старых данных с помощью метода Chart1.SeriesList[0].Clear;
- в цикл после вычисления значения y необходимо добавить оператор Chart1.SeriesList[0].AddXY(x,y).

### 6.7. Оператор присоединения With

Для того, чтобы не указывать несколько раз имя ВК при установке нескольких его свойств, можно использовать оператор присоединения.

*With* <имя ВК> *do*
*Begin*
<операторы>
*End;*

В операторах имена свойства записываются без указания имени ВК.

Пример:

```
With Edit1 do
  Begin
    Text :=FloatToStr(s);
    ReadOnly:= True
  End,
```

### 6.8. Табулирование функции

Табулирование функции – это вычисление значений функции для ряда значений аргумента. Аргумент может быть задан в виде набора произвольных значений (массива) или в виде набора чисел от некоторого начального значения до конечного значения с фиксированным шагом.

Результаты табулирования функции обычно выводятся в таблицу.

**Постановка задачи:** Вычислить значение функции y = f(x) для значений аргумента изменяющихся от $x_{нач}$ до $x_{кон,}$ с шагом $\Delta x$.

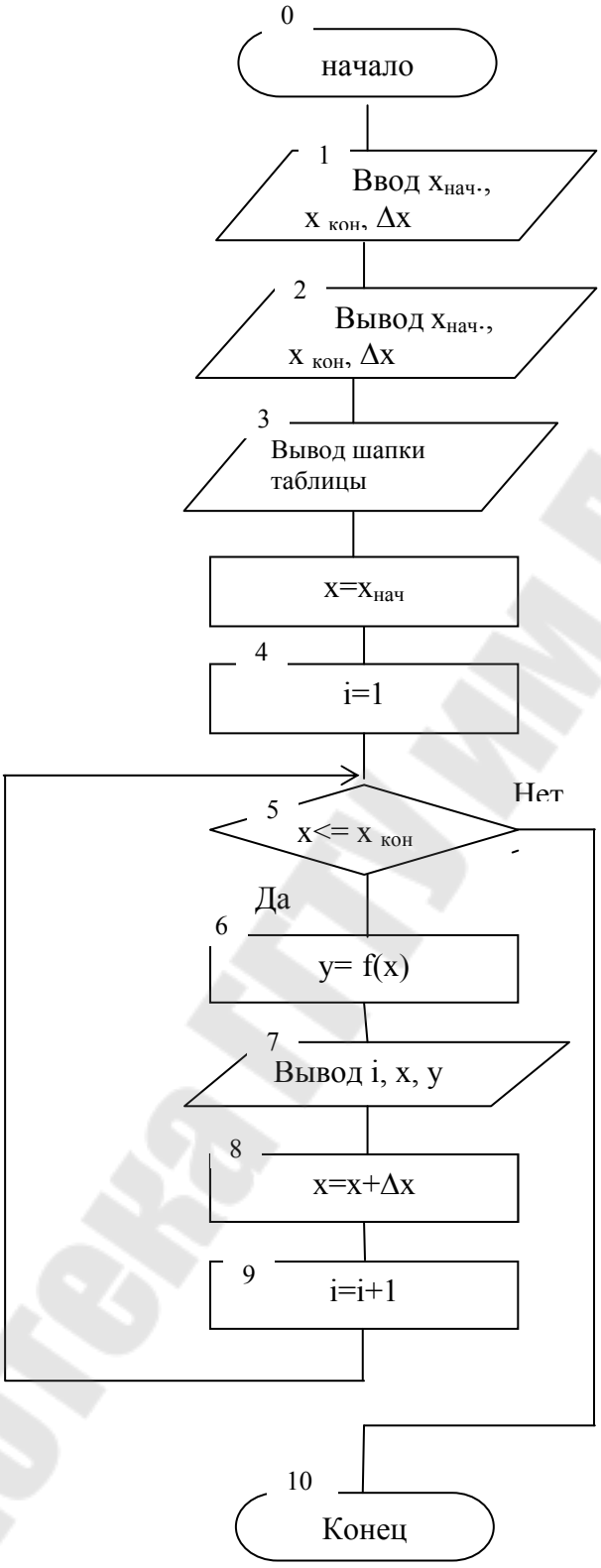



*Рис. 6.7.* Графическая схема алгоритма задачи табулирования функции

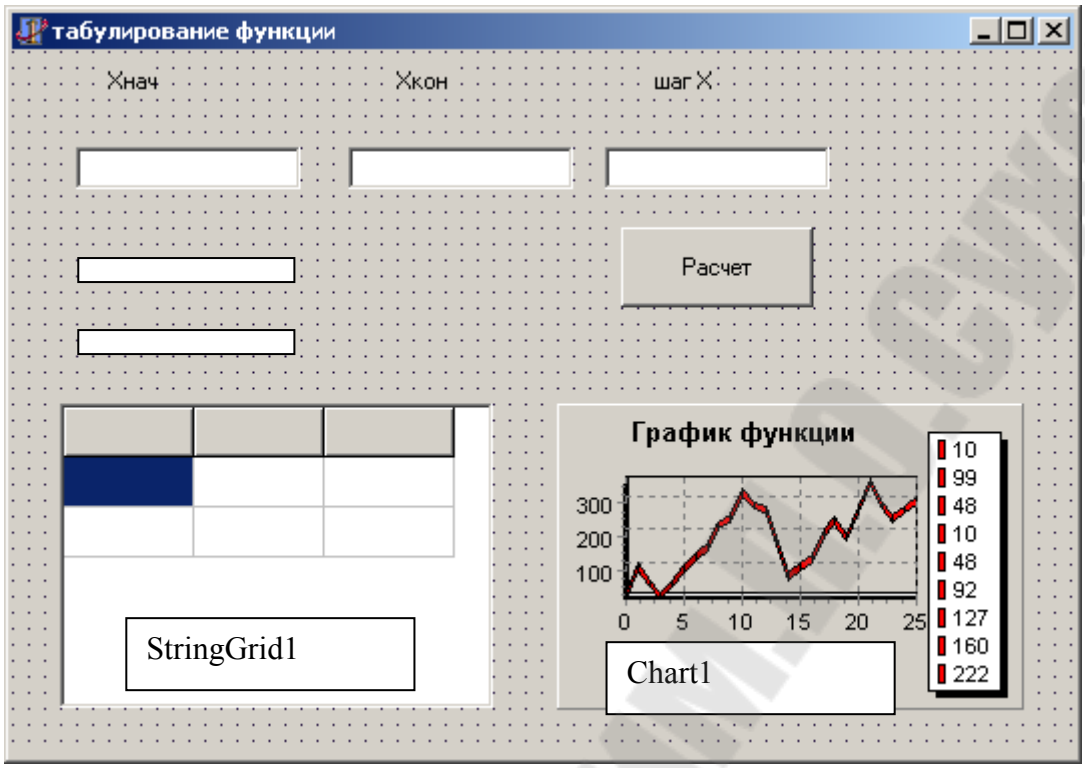



*Рис. 6.8.* Вид окна проекта для задачи табулирования функции

В Инспекторе Объектов устанавливаются свойства ВК StringGrid1 для вывода таблицы результатов:

ColCount (количество столбцов) =3

RowCount (количество строк) будет изменяться в зависимости от исходных данных.

В ИО = 2 (min). Точное значение устанавливается в программе после ввода исходных данных $к=[(x_{кон}-x_{нач})/\Delta x] +1+1$(для фиксированной части);

FixedCols = 0

FixedRows =1 – строка с “шапкой таблицы”

Visible = False – не будет видно пустой таблицы при вводе исходных данных (при использовании одной формы)

Процедура табулирования функции:

```
Procedure TFrom1.Button1Click  (Sender:Tobject);
var
    xn , xk , dx, x, y:real;
    i:byte;
Begin
```

```
xn:=StrToFloat(Edit1.Text);
xk:=StrToFloat(Edit2.Text);
dx:=StrToFloat(Edit3.Text);

// Можно выводить результаты на второй форме!

Label 4. Caption := 'таблица значений функции'
Label 5.Caption:= ' xn' +FloatToStr(xn)+…..;
Chart1.SeriesList [0].Clear;
Chat1.Visible:=True;
With StringGrid1 do
  Begin
     Cells[0,0]:=  '№ n/n';
     Cells[1,0]:= 'x';
     Cells[2,0]:= 'y';
       Rowcount:= trunc((xk-xn)/dx)+2;
     Visible:=True
  End;
x:=xn; i=1;
While x<=xk do
  Begin
     y=x*x;
     StringGrid1.cells[0,i]:=IntTostr(i);
     StringGrid1.cells[1,i]:=FloatTostr(x);
     StringGrid1.cells[2,i]:=FloatTostr(y);  Форматный вывод!
     Chart1.SeriesList[0].AddXY(x,y);
     x:=x+dx ; i=i+1
  End
End;
```

### 6.9. Форматный вывод числовых данных

При выводе вещественных чисел с помощью функции FloatToStr выводится столько десятичных разрядов, сколько их получается при вычислении значения функции. Для вывода необходимого пользователю числа десятичных разрядов используются функции FloatToStrF, FormatFloat и процедура Str. При этом число преобразуется в строку с использованием элементов форматирования.

**Функция FloatToStrF** имеет следующий синтаксис:

FloatToStrF (x, <формат>, p, q), где

x – это переменная или выражение вещественного типа;

<формат> - название одного из определенных в Delphi форматов.

Если используются форматы ffFixed или ffGeneral, то p - это общее количество десятичных цифр в представлении числа, q – количество цифр в дробной части. Если p или q меньше, чем количество цифр в значении числа, то число округляется. В формате ffGeneral очень большие и очень маленькие числа выводятся в экспоненциальной форме (0.25E-08).

Примеры:

StringGrid1.Cells[2,i]:= FloatToStrF (y,ffFixed,4,1) 2,87 ⟶ 2,9
StringGrid1.Cells[2,i]:= FloatToStrF (y,ffGeneral,5,2)

**Функция FormatFloat** имеет следующий синтаксис:

FormatFloat (<формат>, x), где

x – это переменная или выражение вещественного типа;

<формат> - строковая константа, в которой могут использоваться символы спецификаторы:

**0** – определяет поле для цифры; если в данной позиции в значении x есть цифра, то она выводится, если нет, то выводится ноль;

**#** – определяет поле для цифры; если в данной позиции в в значении x есть цифра, то она выводится, если нет, то не выводится ничего, отличается от 0 тем, что начальные и конечные нули не выводятся;

**. (точка)** – определяет поле для разделителя целой и дробной части числа;

**, (запятая)** – определяет поле для разделителя тысяч.

Символы, которые выводятся в полях, определяемых символами «**.**» и «**,**» устанавливаются в ОС Windows, обычно это запятая и пробел.

Примеры:

StringGrid1.Cells[2,i]:= FormatFloat (‘#00.00’,y)

6276,847 ⟶ 6276,85
0,5 ⟶ 00,50

StringGrid1.Cells[2,i]:= FormatFloat (‘#,##0.0##’,y)

6276,847 ⟶ 6 276,847
0,5 ⟶ 0,5

**Процедура Str (X:p:q, S) –** преобразование вещественного значения X в строку S по указанному формату, p – это общее количество символов в представлении числа в строке, q – число символов в дробной части числа.

Пример:

X:=15.7 (вещественное)
Str (X:6:2 , S) ; _15,70

**Пример 7.4.** Вычисление значения функции с помощью ее разложение в ряд Тейлора

**Постановка задачи.** Для заданного значения аргумента **x** и заданной точности **ε** вычислить значение функции **y=f(x)** двумя способами:

- с помощью встроенных функций Delphi;
- с помощью разложения функции в ряд Тейлора. Вывести в ВК TStringGrid номер очередного члена ряда n, его значение $a_{n,}$ , текущей суммы членов ряда $\sum_{i=1}^{n} a_i$ .

Вывести значение аргумента, значение функции (двумя способами) и количество просуммированных членов ряда.

Вычислить значения функции y=ln(1-x)

$$\ln(1-x) = -\sum_{n=1}^{\infty} \frac{x^n}{n} = -\left( x + \frac{x^2}{2} + \frac{x^3}{3} + \ldots \right) \qquad -1 \le x \le 1$$

$$\frac{x_n}{x_{n-1}} = \frac{x^n \cdot (n-1)}{n \cdot x^{n-1}} = \frac{x \cdot (n-1)}{n} \qquad x_n = \frac{x_{n-1} \cdot x \cdot (n-1)}{n} \qquad x_1 = -x$$

```
procedure TForm1.Button1Click(Sender: TObject);
 var
   x,x1,s,eps:real;
   i,n:byte;

begin
   x:=StrToFloat(Edit1.Text);
   eps:=StrToFloat(Edit2.Text);
   Label5.Caption:='x='+FloatToStr(x)+'   eps='+FloatToStr(eps);
   Edit3.Text:= FloatToStr(ln(1-x));
   if x<eps
    then
      begin
        s:=-x;
        n:=1;
      end
    else
     begin
       Label4.Caption:='Таблица значений функции';
       With    StringGrid1  do
        begin
         Visible:=True;
         Cells[0,0]:='    N п/п ' ;
         Cells[1,0]:='        x' ;
         Cells[2,0]:='        s'  ;
        end;
      x1:=x;  ;i:=1;  s:=0;
      while x1>eps do
       begin
        s:=s+x1;
        StringGrid1.Cells[0,i]:=IntToStr(i);
```

```
      StringGrid1.Cells[1,i]:=FloatToStr(-x1);
      StringGrid1.Cells[2,i]:=FloatToStr(-s);
      StringGrid1.RowCount:=i+1;
      i:=i+1;
      x1:=x1*x*(i-1)/i;
     end;
     n:=i-1;
    end;
  Edit4.Text:= FloatToStr(-s);
  Edit5.Text:= IntToStr(n);
end;
```

## Тема 7. Обработка одномерных массивов

### 7.1. Общие сведения

В Delphi четыре структурированных типа: массивы, записи, множества, файлы. Каждый элемент этих типов может также иметь структурированный тип. Допускается произвольная глубина вложенности типов, однако длина любого из них в памяти не должна превышать 2 Гб.

Одномерный массив – это упорядоченная последовательность величин одного типа, имеющих одно имя, но различающихся индексами.

Индекс – это выражение целого (порядкового типа), определяющее положение отдельной величины в последовательности. Каждая отдельная величина называется элементом массива.

### 7.2. Описание массива

Описание **типа** массива (статического) может задаваться одним из двух способов в виде:

array [<список одного или нескольких индексных типов>] of
< тип элементов>

В качестве индексных типов можно использовать любые порядковые типы. Обычно в качестве индексного типа используется

тип-диапазон. Размерность массива (количество индексов) ограничено только суммарной длиной представления массива в памяти – 2 Гб.

Описание одномерного массива:

1 способ (раздел описания переменных):

```
var
   <идентификатор>: array [Nнач..Nкон] of  < тип элементов>;
```

Nнач, Nкон – это минимальное и максимальное значение индекса, обязательно константы. Обычно Nнач=1, тогда Nкон это максимальное число элементов в массиве.

Пример

```
 var
   x: array [1..20] of real;
```

2 способ (раздел описания типов и раздел описания переменных):

```
type
   <имя типа>=array [Nнач..Nкон] of  < тип элементов>;
var
    <идентификатор>:<имя типа>;
```

Пример:

```
type
   mas=array [1..20] of  real;
var
   x,y:mas;
```

Обращение к элементу массива:

<идентификатор массива> [<индекс>]

x[3] – третий элемент массива x

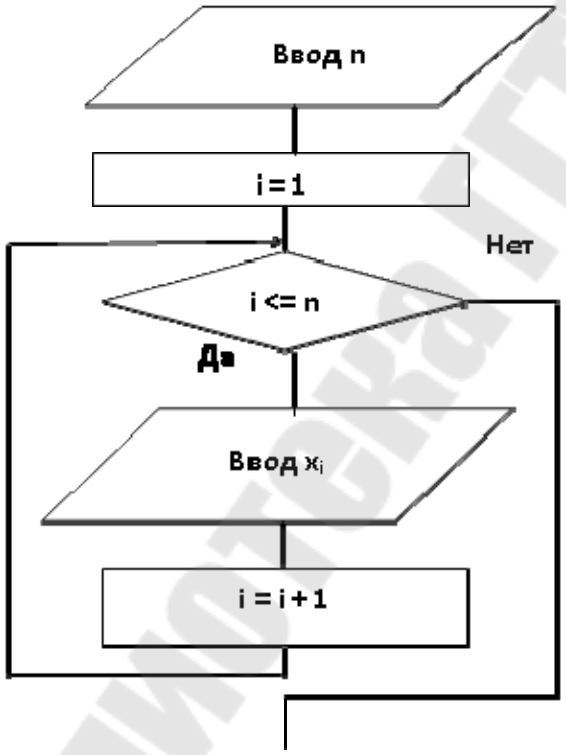
x[i+2] – элемент массива x с индексом i+2 (i при этом должно иметь значение).

Свойства элементов массива:

- все элементы массива имеют один тип:
- номера элементов изменяются от Nнач до Nкон с шагом 1:
- число используемых элементов может быть меньше, чем число элементов в описании массива.

Если работа с одним массивом выполняется в нескольких процедурах, то описание массива и числа элементов нужно разместить до всех процедур в разделе implementation. Если обработка массива выполняется в нескольких модулях, то описание массива нужно разместить в разделе interface одного из модулей и подключать его к остальным модулям в программе.

### 7.3. Ввод массива




n – число элементов в массиве
i – номер элемента в массиве
$x_i$ – элемент массива с номером i

*Рис 7.1.* Графическая схема алгоритма процедуры ввода массива

**Реализация ввода  массива на языке Delphi**

**Способ 1.**


*Рис 7.2.* Вид окна проекта

После щелчка по кнопке Ввод в процедуре обработки этого события вводятся число элементов и сами элементы. Для ввода элементов используется компонент StringGrid1. Свойства этого ВК, устанавливаемые в ИО:

- RowCount=1;
- ColCount= max числу элементов в описании массива;
- FixedRows=0;
- FixedCols=0;
- Options.GoEditing=True;
- Options.GoTabs=True.

можно установить:

- DefaultColWidth ~34 (по умолчанию 54 пикселя) ширина ячейки таблицы.

Текст процедуры:

```
n:=StrToInt(Edit1.Text);
StringGrid1ColCount:=n;
for i:=1 to n do
    x[i]:=StrToFloat(StringGrid1.Cells[i-1,1]);
```

**Способ 2.**

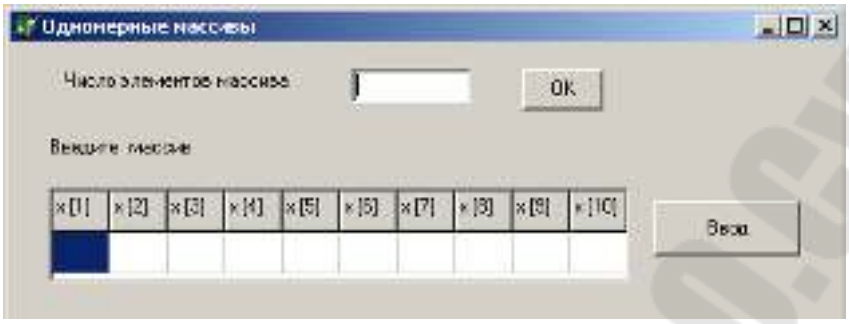



*Рис 7.3.* Вид окна проекта

Количество элементов массива вводится после щелчка по кнопке ОК, в соответствующей процедуре обработки события.

В этой процедуре необходимо

- проверить ввод количества элементов с помощью защищенного блока;
- установить число столбцов в StringGrid1 равное количеству элементов массива (ColCount=n);
- ВК StringGrid1 сделать видимым;
- установить фокус ввода на первую ячейку StringGrid1.

Первую строку StringGrid1 можно сделать фиксированной с заголовками элементов массива, во второй вводить элементы.

Ввод элементов в ОЗУ выполняется после щелчка по кнопке Ввод.

Способ 3.

На форме одна кнопка для ввода элементов массива. Для действий, выполняемых с помощью кнопки ОК во втором способе, создается процедура обработки события onChange для ВК Edit1 (TForm1.Edit1Change).

```
Procedure TForm1.Edit1Change (Sender: TObject);
    var
      i:byte;
    begin
```

```
try
  n:=StrToInt(Edit1.Text);
  With StringGrid1 do
   begin
    ColCount:=n;
    Visible:=True;
    Col:=0;
    Row:=1;
    SetFocus;
 except
   ShowMessage('Введите n правильно')
end;
```

## 7.4. Вывод массива

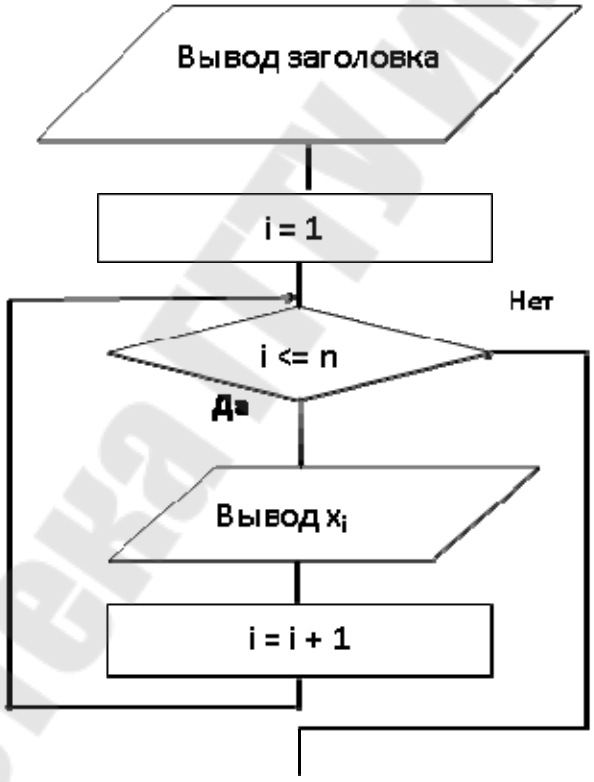



*Рис 7.4.* Графическая схема алгоритма процедуры вывода массива

Вывод массива выполняется в ВК StringGrid. Он может находиться на той же форме, где выполнялся ввод или на новой форме. Свойства этого ВК, устанавливаемые в ИО:

- RowCount =1; (=2, если с заголовками элементов)
- ColCount = 2, а затем устанавливается в программе равное числу элементов после ввода массива;
- FixedRows=0; (=1, если с заголовками элементов)
- FixedCols=0;
- Visible = False, если массив выводится на той же форме, где был ввод.
- DefaultColWidth ~34

Фрагмент процедуры (после ввода):

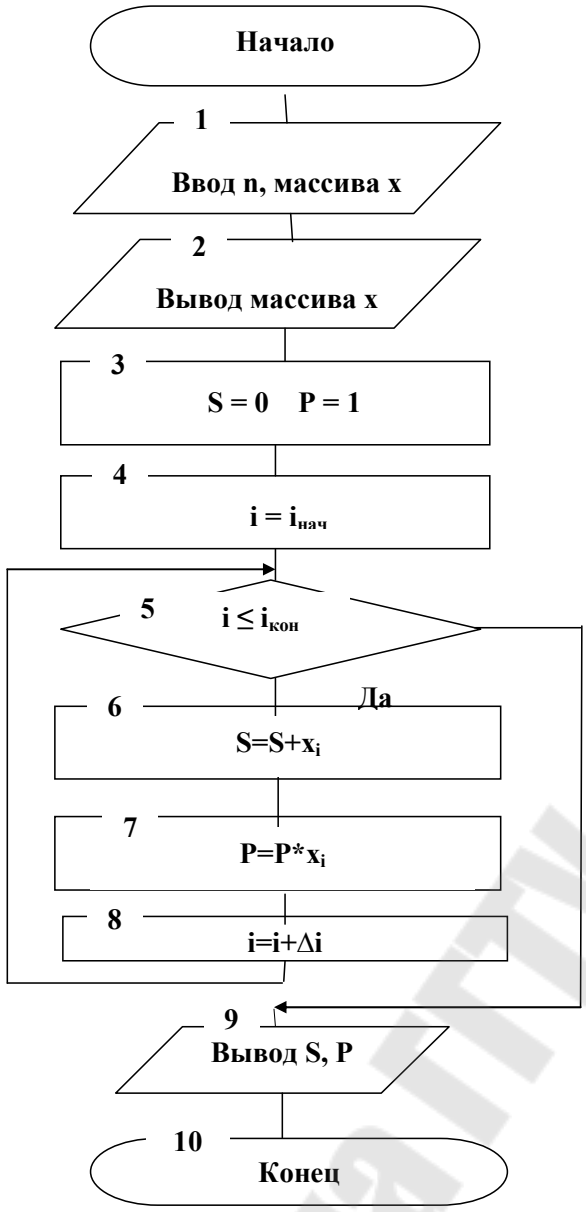
```
label3.Caption:='Исходный массив';
With StringGrid2 do
 begin
      ColCount=n;
      Visible:=True;    (если на одной форме)
       for i:=1 to n do
         Cells[i-1,1]:=FloatToStr(x[i]);
 end;
```

## 7.5. Типовые алгоритмы обработки одномерных массивов

### 7.5.1. Вычисление суммы и произведения элементов, находящихся на разных местах в массиве

Вычисление суммы и произведения всех элементов:

S = 0  S = S + x[i]  для i = 1 .. n
P = 1  P = P * x[i]  для i = 1 .. n




$i_{нач}$ – минимальное значение номера элемента массива

$i_{кон}$ – минимальное значение номера элемента массива

Δi – шаг изменения индекса

*Рис 7.5.* Общая графическая схема типового алгоритма

**Пример 7.1.** Вычислить сумму квадратов элементов, находящихся в массиве на местах с номерами, кратными трем.

```
S:=0;
i:=3;
While i<=n do
  begin
    S:=S+sqr (x[i]);
    i:=i+3
  end;
```

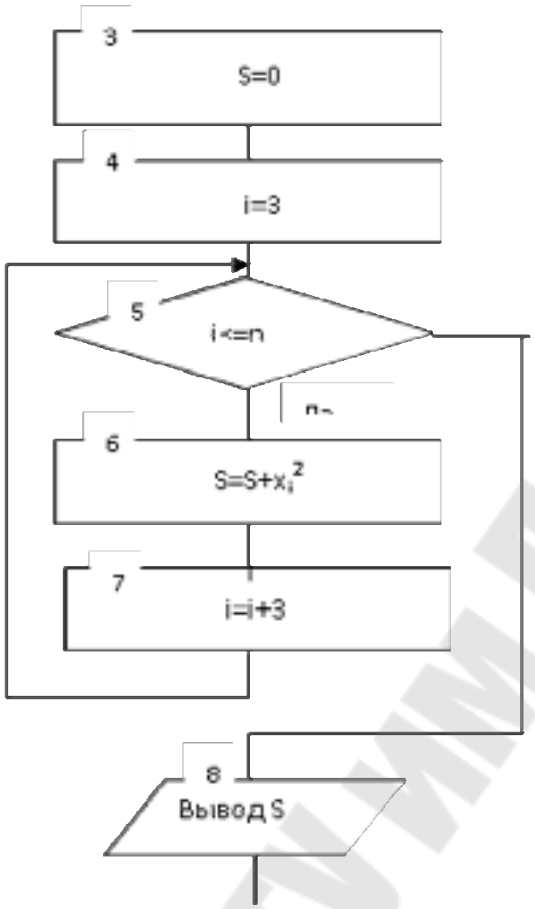



*Рис 7.6.* Графическая схема алгоритма примера 7.1

**Пример 7.2.** Вычислить произведение элементов, находящихся в массиве на местах с третьего по седьмое включительно.

```
if n<7
  then   <вывод сообщения >
  else
   begin
     P:= 1;
    For i:=3 to 7 do
       P:=P*x[i];
    <вывод P>
   end;
```

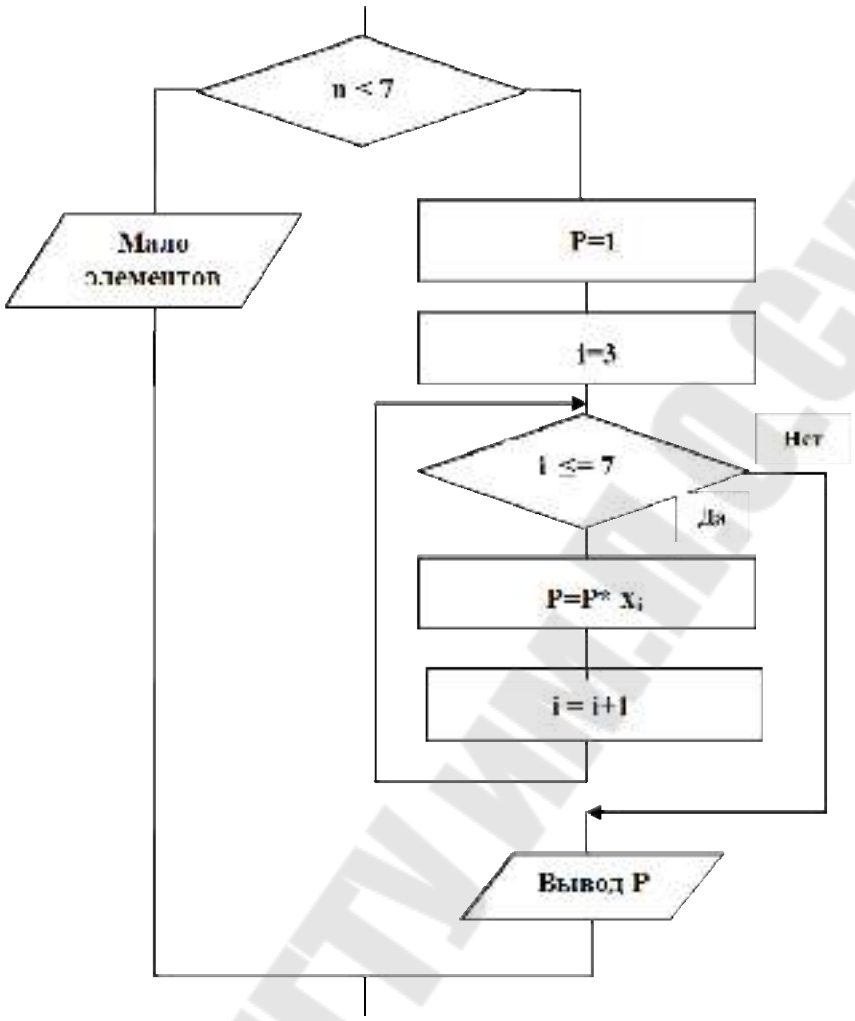



*Рис 7.7.* Графическая схема алгоритма примера 7.2

### 7.5.2. Вычисление суммы, произведения и количества элементов, удовлетворяющих заданному условию и находящихся на разных местах в массиве

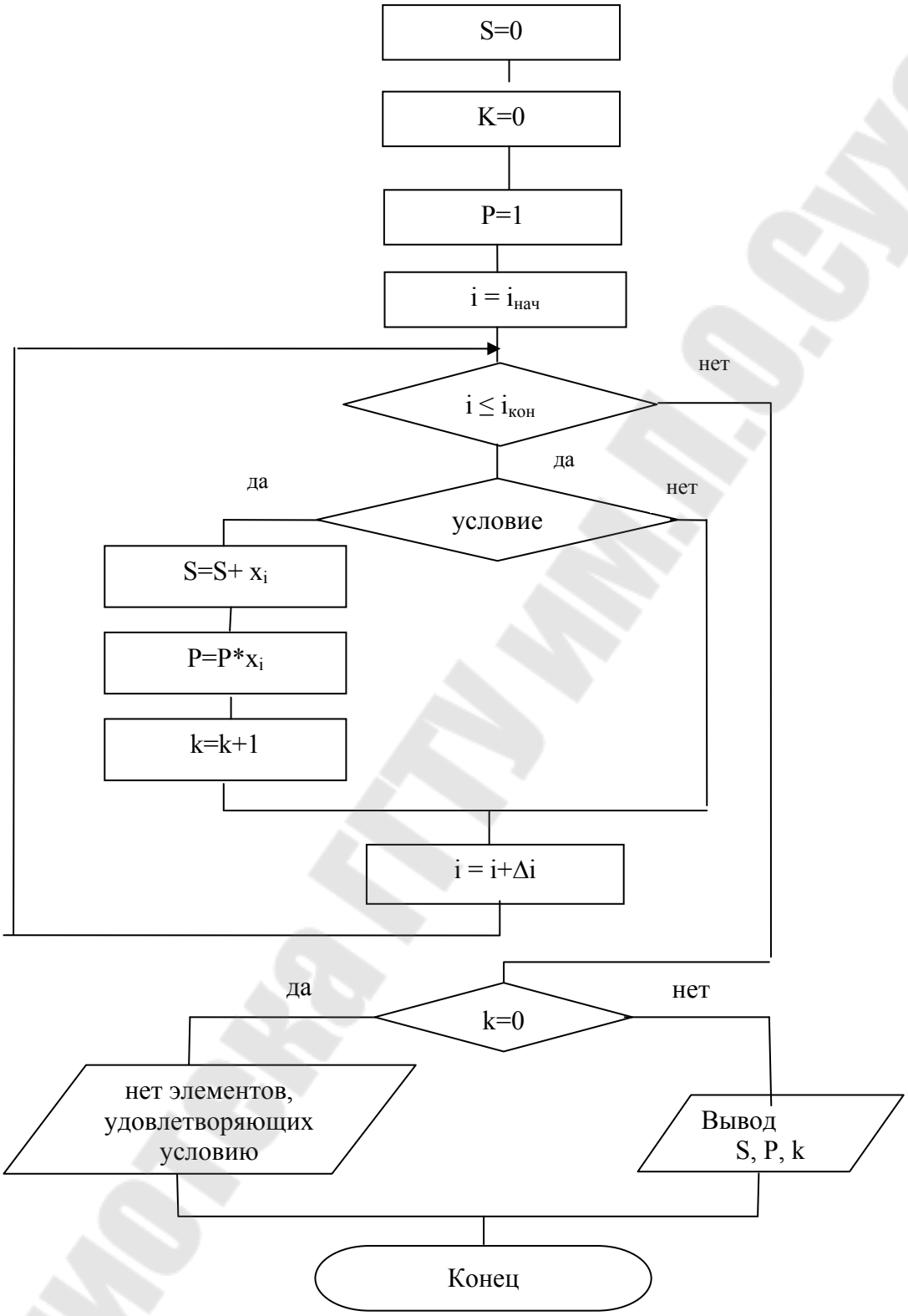



*Рис 7.8.* Общая графическая схема типового алгоритма

**Пример 7.3**. Вычислить количество элементов, больших заданного числа **a** и находящихся в массиве на местах с четными номерами.

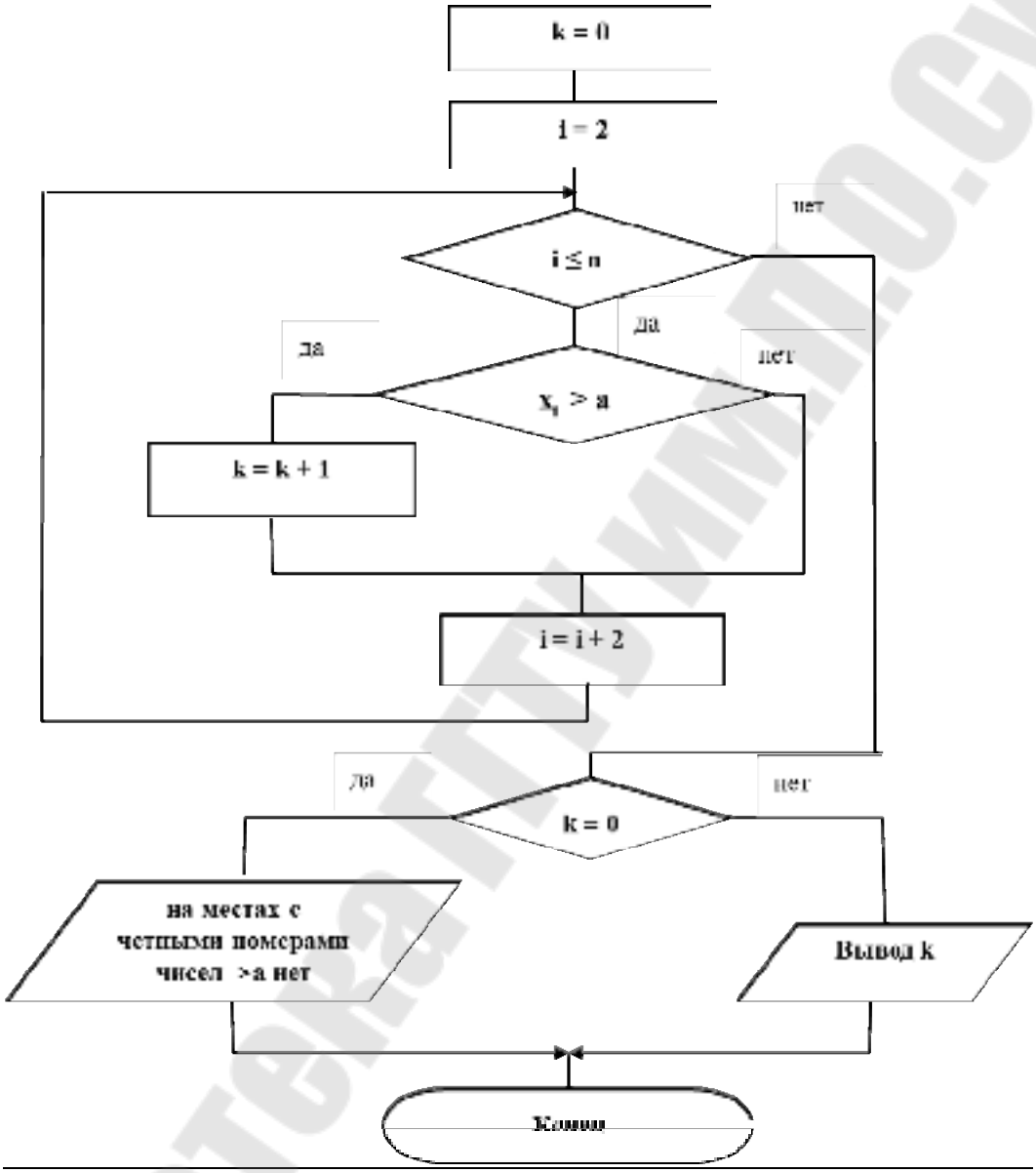



*Рис 7.9.* Графическая схема алгоритма примера 7.3

Фрагмент процедуры

```
k := 0;
i := 2;
While i <= n do
```

```
    begin
      if x[i] > a
         then k := k + 1;
      i := i + 2
    end;
  If k = 0
    then Label5.Caption:= ' Нет элементов на четных местах > a'
    else Label5.Caption:= ' Количество элементов на местах с
  четными номерами > a =' + IntToStr (k);
```

**Пример 7.4.** Вычислить среднее арифметическое положительных элементов массива, находящихся в массиве на нечетных местах, и произведение элементов, расположенных в массиве вне интервала [a,b).

Текст процедуры:

```
procedure TForm1.Button1Click(Sender: TObject);   щелчок по кнопке
Расчет
var
  x:array [1..10] of real;
  n,K,i,l:byte;
  Sr,P,S,a,b:real;
begin
   <ввод n,x>
   <ввод a,b>
   <вывод x>
   <вывод a,b>
   S:=0;
   K:=0;
   i:=1;
   while i<=n
    begin
      if x[i]>0 then
        begin
          S:=S+x[i];
          K:=K+1
```

```
        end;
      i:=i+2
    end;
  If K= 0 Then
      Label4.Caption:='нет положительных эл-в на нечетных местах'
    Else
      begin
        Sr:=S/K;
       Label4.Caption:='Среднее арифметич. положит. эл-в на нечетных
местах=' + FloatToStr(Sr);
      end;
// вторая часть
P:=1;
l:=0;
for i:=1 to n do
     if (x[i]< a) or (x[i]>=b)   then
       begin
         P=P*x[i];
         l:=1
       end;
If l = 0 Then
      Label5.Caption:='нет элементов вне интервала'
    Else
      Label5.Caption:= 'Произведение элементов вне интервала =' +
FloatToStr(P);
end;
```

**Тесты**

Для первой части

1. x ( -1 0 -2 -2 0 4 ) на нечетных местах нет положительных элементов
2. x ( 2 2 4 -6 3 0 ) на нечетных местах все элементы положительные, среднее арифметическое=(2+4+3)/3= 3
3. x (1 2 -1 3 7 5 -5 ) на нечетных местах часть элементов положительные, среднее арифметическое = (1+7)/2=4

Для второй части интервал [a, b):

1. [-4, 5)  нет элементов вне интервала
2. [5, 10)  все элементы вне интервала
   произведение = 2*2*4*-6*3*0=0
3. [-4,5)  часть элементов вне интервала
   произведение = 7*5-5= -175

### 7.5.3. Перестановка местами и замена элементов массива

Для перестановки местами двух элементов массива необходимо объявить дополнительную переменную того же типа, который имеют элементы массива.

```
var
  x:array [1..10] of real;
  p:real;
```

**Пример 7.5.** Поменять местами второй и четвертый элементы массива.

```
p:=x[2];
x[2]:=x[4];
x[4]:=p;
```

**Пример 7.6.** Поменять местами первый и последний элементы массива.

```
p:=x[1];
x[1]:=x[n];      n – число элементов массива, номер
                      последнего элемента.
x[n]:=p;
```

**Пример 7.7.** Заменить положительные элементы массива на последний элемент массива.

```
for i:=1 to n-1 do
      if x[i]>0 then
x[i]: =x[n];
```

**Пример 7.8.** Поменять местами предпоследний элемент массива с элементом с заданным номером **k**.

```
if (k>=1) and (k<=n)
  then
    if k = n-1
      then ShowMessage ('Обмен не нужен, т.к. …..')
      else
        begin
          p:=x[n-1];
          x[n-1]:=x[k];
          x[k]:=p;
        end;
```

Заменить предпоследний элемент на элемент с заданным номером **k**

```
x[n-1]:=x[k];
```

### 7.5.4. Поиск минимальных и максимальных элементов массива и определение их номеров

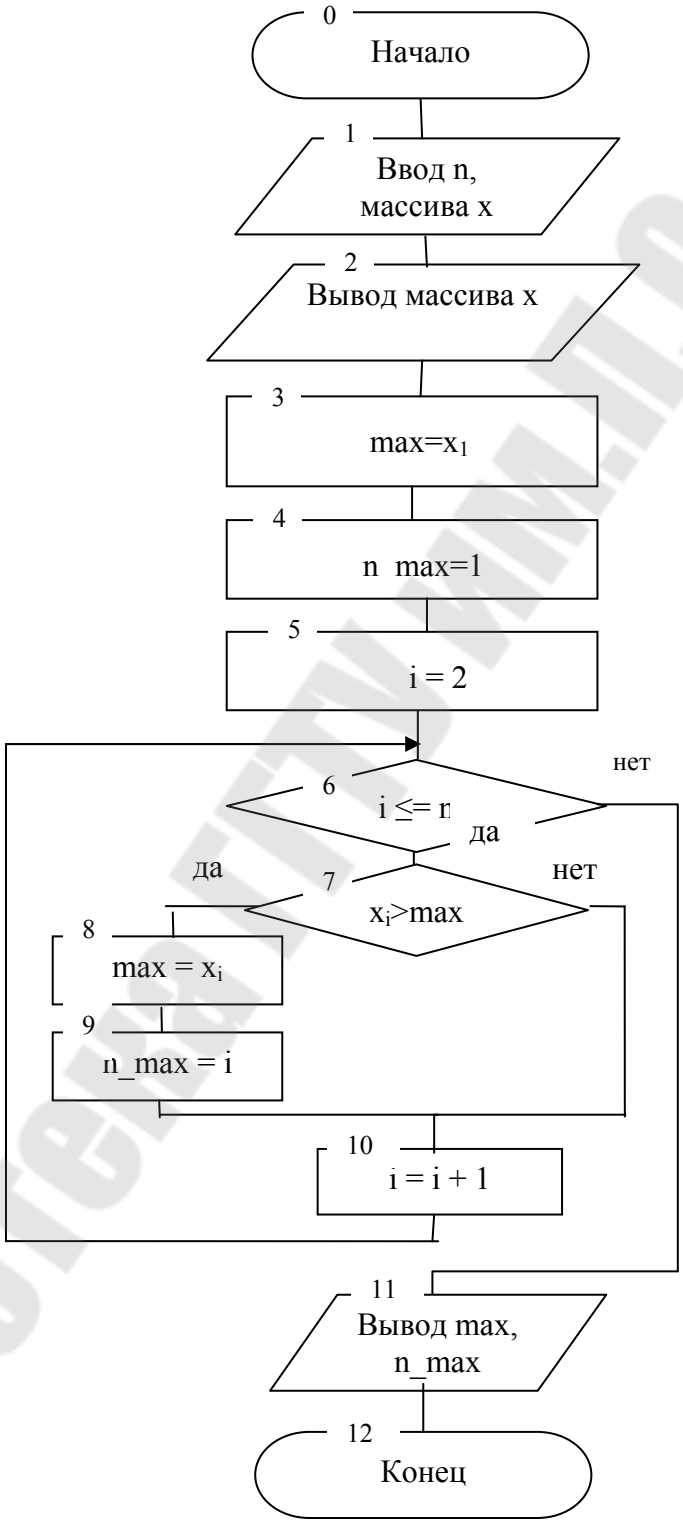
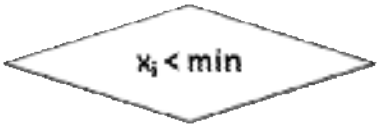


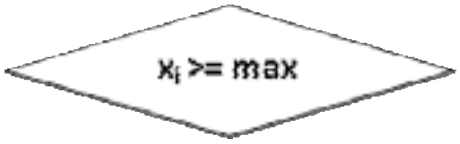

*Рис 7.10.* Общая графическая схема типового алгоритма

**Замечания к алгоритму:**

1. При поиске минимального элемента блок **7** должен иметь вид:




2. Для того, чтобы найти номер последнего из равных максимальных элементов, блок **7** должен иметь вид:




**Тесты:**

1. минимальный элемент находится на первом месте в массиве
2. минимальный элемент находится на последнем месте в массиве
3. минимальный элемент находится в середине массива
4. зависит от конкретных условий задачи

**Пример 7.9.** Найти минимальный из элементов массива с номерами, кратными трем, и поменять его местами со следующим элементом массива.

```
min:=x[3];
n_min:=3;
while i<=n do
  begin
     if x[i] < min then
      begin
        min:=x[i];
        n_min:=i
      end;
   i:=i+3
end;
```

```
if n_min = n
  then
    ShowMessage (‘минимальный элемент - последний’)
  else
    begin
        p:=x[n_min];                    x[n_min]:=x[n_min+1];
        x[n_min]:=x[n_min+1];           x[n_min+1]:= min
        x[n_min+1]:=p
        <вывод массива x>
    end;
```

### 7.5.5. Формирование новых массивов

**Постановка задачи**: из исходного массива перенести в новый массив элементы, удовлетворяющие некоторому условию, не меняя их взаимного расположения.

**Тесты:**

1. новый массив не сформирован, т.к. в исходном массиве нет элементов, удовлетворяющих заданному условию
2. новый массив сформирован из всех элементы исходного массива, т.к. они все удовлетворяют условию
3. новый массив сформирован из части элементов исходного массива, удовлетворяющих заданному условию

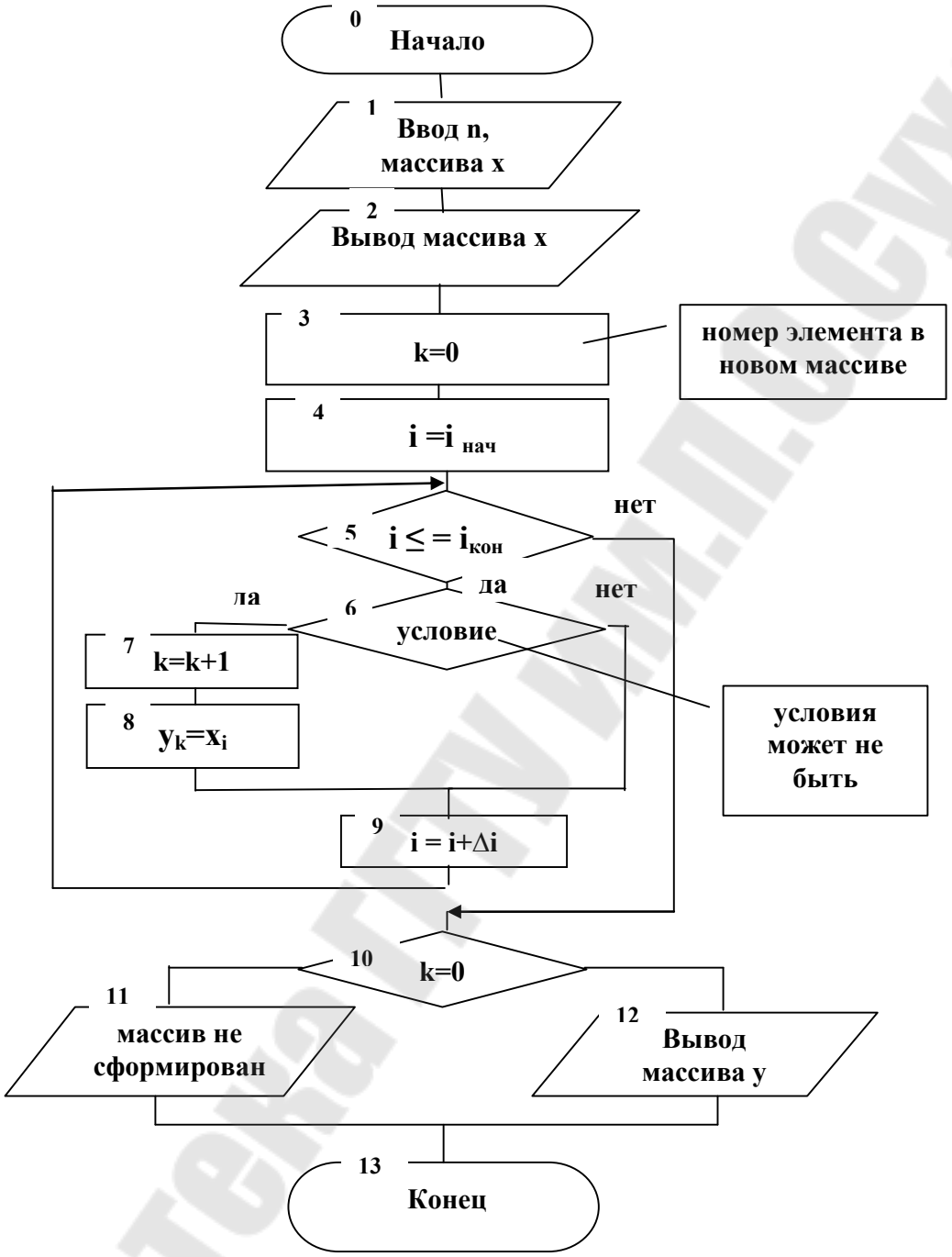



*Рис 7.11.* Общая графическая схема типового алгоритма

**Пример 7.10**. Даны два массива **a** и **b**. Переписать из них элементы в массив **c** по следующему правилу: из массива **a** квадраты элементов, находящихся в массиве на местах с четными номерами, из массива **b** элементы, которые не больше, чем последний элемент массива a.

**Фрагмент процедуры:**

```
k:=0;
i:=2;
while i<=n
   begin
      k:=k+1
      c[k]:=sqr(a[i]);
      i:=i+2
  end;
for i:=1 to m do
  if b[i] <= a[n] then
    begin
       k:= k +1;
      c[k]:=b[i]
   end;
  if  k = 0
    then  label4.Caption:='новый массив не сформирован'
  else
    for i:=1 to k do
       StringGrid3.Cells [i-1,0]:=c[i];
```

**Пример 7.11.** Если в массиве больше трех нулей, то сформировать новый массив из положительных элементов исходного массива, находящихся в исходном массиве перед максимальным элементом. Если задание выполнить невозможно, вывести предупреждающее сообщение.

**Тесты:**

1. в массиве не больше трех нулей;
2. максимальный элемент расположен на первом месте в массиве;
3. перед максимальным элементом нет положительных элементов;

4. есть положительные элементы перед максимальным элементом;

5. все элементы перед максимальным элементом положительные.

**Фрагмент процедуры:**

```
kol:=0;
for i:=1 to n do
   if x[i]=0 then
       kol:= kol +1;
If kol = 0 Then
    Label4.Caption:=' в массиве не больше трех нулей'
 Else
    begin
      max:=x[1];   n_max:=1
      for i:=2 to n do
         if x[i] > max then
          begin
            max:=x[i];
            n_max:=i
          end;
       if n_max=1
         then label4.Caption:=’максимальный элемент - первый’
         else
           begin
             k:=0;
             for i:=1 to n_max-1 do
               if x[i] > 0 then
                  begin
                    k:= k +1; y[k]:=x[i]
                  end;
             if  k = 0
               then  label4.Caption:='нет положительных элементов перед
                                          максимальным'
              else
                 <вывод массива y>
          end
    end;
```

## Тема 8. Обработка двумерных массивов

### 8.1. Общие сведения

Двумерный массив или матрица - это упорядоченная последовательность величин одного типа, имеющих одно имя, но различающихся индексами.

Каждой величине или элементу массива соответствуют два целых числа (индекса), определяющие положение элемента в массиве (номер строки и номер столбца).

Размер матрицы определяется двумя целыми числами – количеством строк и количеством столбцов.

Матрица x

1 2 -1 4 n=2 (количество строк)
5 6 3 -3 m=4 (количество строк)

### 8.2. Описание массива

1 способ (раздел описания переменных):

```
var
   <идентификатор>: array [N1нач..N1кон, N2нач..N2кон] of
                          < тип элементов>;
```

Обычно N1нач и N1кон =1, тогда N1кон и N2кон – это максимально возможное число строк и столбцов в массиве.

Пример:

```
  x: array [1..10,1..5] of real;
```

2 способ:

```
type
  <имя типа> = array [N1нач..N1кон, N2нач..N2кон] of
                    < тип элементов>;
var
   <идентификатор>:<имя типа>;
```

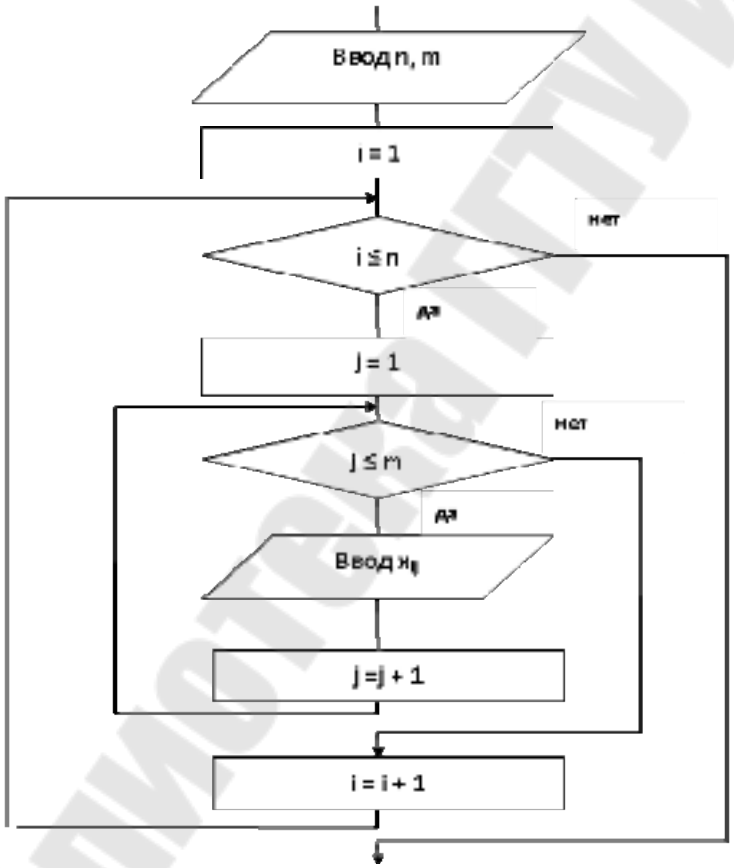
Пример:
```
type
   matr=array [1..10,1..5] of  real;
var
   x:matr;
```

**Обращение к элементу массива:**

<идентификатор массива> [№ строки, № столбца]

Пример: x[1,4]  x[i,j]

## 8.3. Ввод массива




n – число строк
m – число столбцов
i – номер строки
j – номер столбца

*Рис 8.1.* Графическая схема алгоритма ввода матрицы

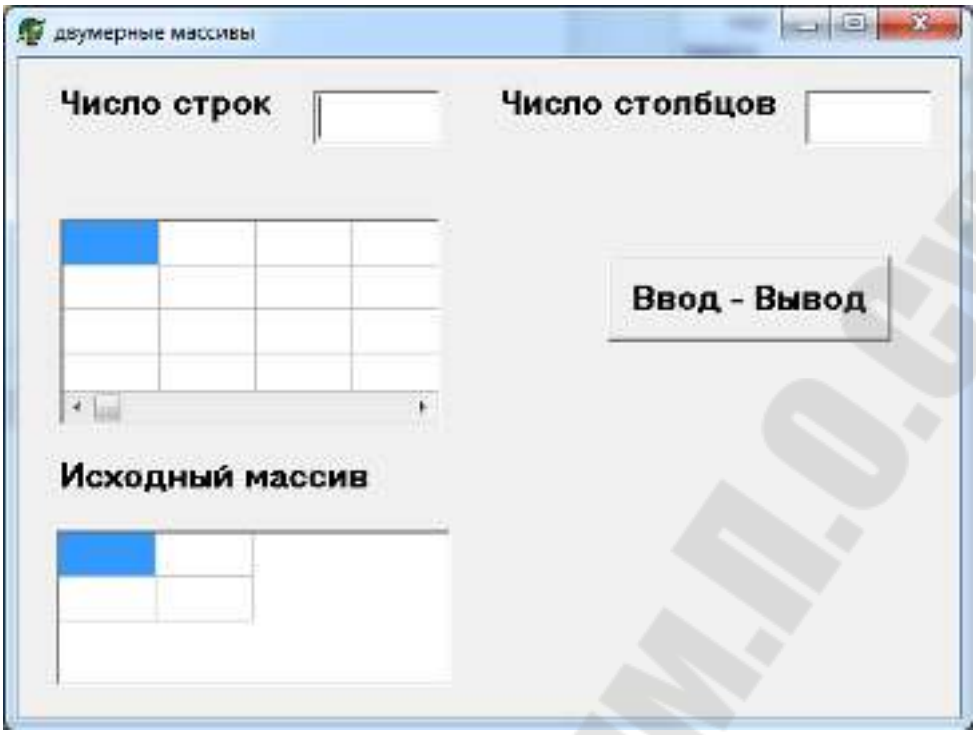



*Рис 8.2.* Вид окна проекта

Для ввода элементов массива используется ВК StringGrid.

Свойства ВК StringGrid, которые устанавливаются в ИО:

- RowCount = максимально возможному числу строк и
- ColCount = столбцов в описании массива
- FixedRows = 0
- FixedCols = 0
- Options.GoEditing = True
- Options.GoTabs = True

**Процедура ввода**

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

    var
      n,m,j,i:byte;
      x:array [1..10,1..10] of real;
    begin
```

```
  n:=StrToInt(Edit1.Text);
  m:=StrToInt(Edit2.Text);
  for i:=1 to n do
    for j:=1 to m do
      x[i,j]:=StrToFloat(StringGrid1.Cells[j-1,i-1]);
   StringGrid1.RowCount:=n;
   StringGrid1.ColCount:=m;
 end;
```

Можно использовать другие методы ввода массива, аналогичные вводу одномерного массива:

- использование двух кнопок и установка размера ВК StringGrid после ввода числа строк и столбцов матрицы;
- использование процедуры обработки события onChange для ВК, созданных для ввода числа строк и столбцов матрицы.

Можно использовать для ввода числа строк и столбцов матрицы (как и размера одномерного массива) ВК **SpinEdit** (вкладка **Samples**). Он предназначен для ввода и отображения чисел. Свойство Value содержит текущее значение числовой величины. Оно имеет тип Integer, поэтому не нужно использовать функции преобразования типа, не нужно проверять правильность ввода с помощью защищенного блока.

Например: m:= SpinEdit1.Value

Свойства **MinValue** и **MaxValue** содержат минимальное и максимальное значение диапазона возможных значений. Для массива их нужно установить в соответствии с описанием массива (**MinValue** =1, **MaxValue** = максимальному числу элементов в описании массива).

## 8.4. Вывод массива

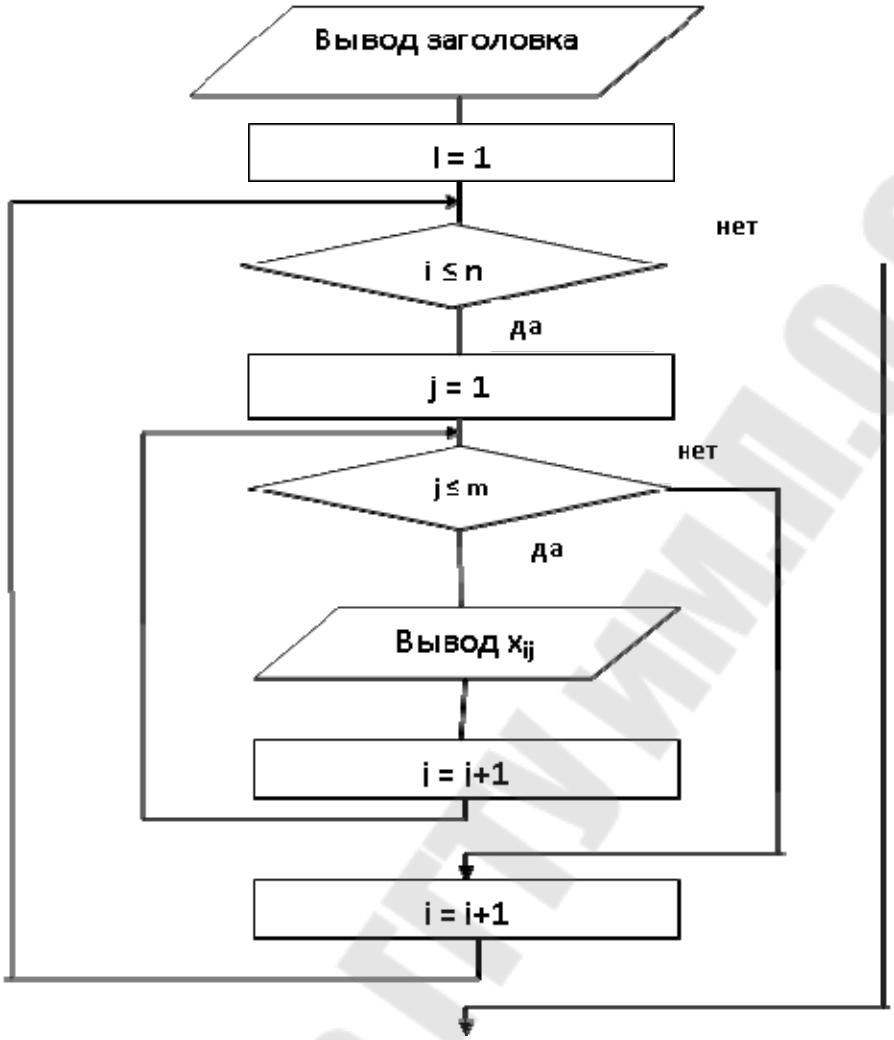



*Рис 8.3.* Графическая схема алгоритма вывода матрицы

Для вывода элементов массива используется ВК StringGrid.

Свойства ВК StringGrid, которые устанавливаются в ИО:

- RowCount = любое, будет установлено в процедуре
- ColCount = после ввода размерности массива
- FixedRows = 0 (=1, если с заголовками столбцов)
- FixedCols = 0 (=1, если с заголовками строк)
- Visible = False, если ввод и вывод на одной форме

**Текст процедуры вывода**

```
With StringGrid2 do
 begin
     RowCount:=n;
     ColCount:=m;
     Visible:=True;
     for i:=1 to n do
       for j:=1 to m do
          Cells[j-1, i-1]:=FloatToStr(x[i, j]);
 end;
```

## 8.5. Типовые алгоритмы обработки двумерных массивов

### 8.5.1. Вычисление сумм, количеств, произведений элементов матрицы, удовлетворяющих заданному условию

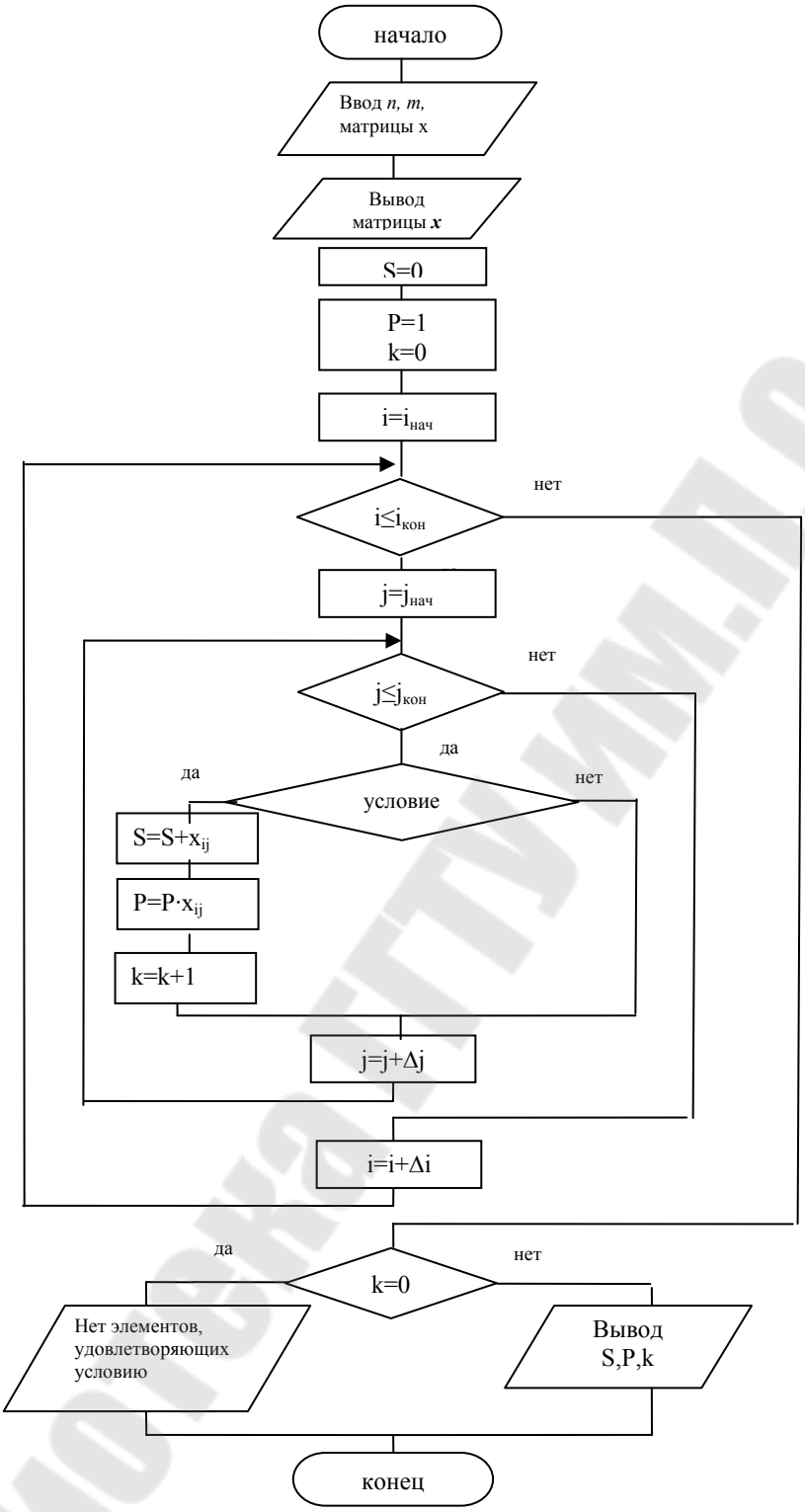



*Рис 8.4.* Общая графическая схема типового алгоритма

**Пример 8.1**. Определить количество положительных элементов, расположенных в строках с номерами, кратными трем.

```
k:=0;
i:=3;
While j<=n do
     begin
       for j:=1 to m do
          if x[i,j]>0 then k:=k+1;
       i:=i+3
     end;
 if k = 0
   Then
      Label4.Caption:= 'нет полож эл-в в строках кр. трем'
   Else
     Label4.Caption:='Количество полож эл-в в строках кр. трем ='
                          + FloatToStr(s);
```

**Пример 8.2**. Вычислить произведение элементов, находящихся в последнем столбце матрицы и больших заданного числа a.

```
k:=0;
P:=1;
for i:=1 to n do
     if x[i,m]>a then
        begin
           P:=P*x[i,m];
           k:=1;
         end;
if k = 0
   Then
      Label4.Caption:= 'нет эл-в > a в последнем столбце'
   Else
      Label4.Caption:=' Произведение эл-в > a в последнем
                          столбце=' + FloatToStr(P);
```

### 8.5.2. Поиск максимальных и минимальных элементов матрицы и определение их местоположения в матрице

max – максимальный элемент матрицы

nom1 – номер строки, в которой находится максимальный элемент

nom2 – номер столбца, в котором находится максимальный элемент

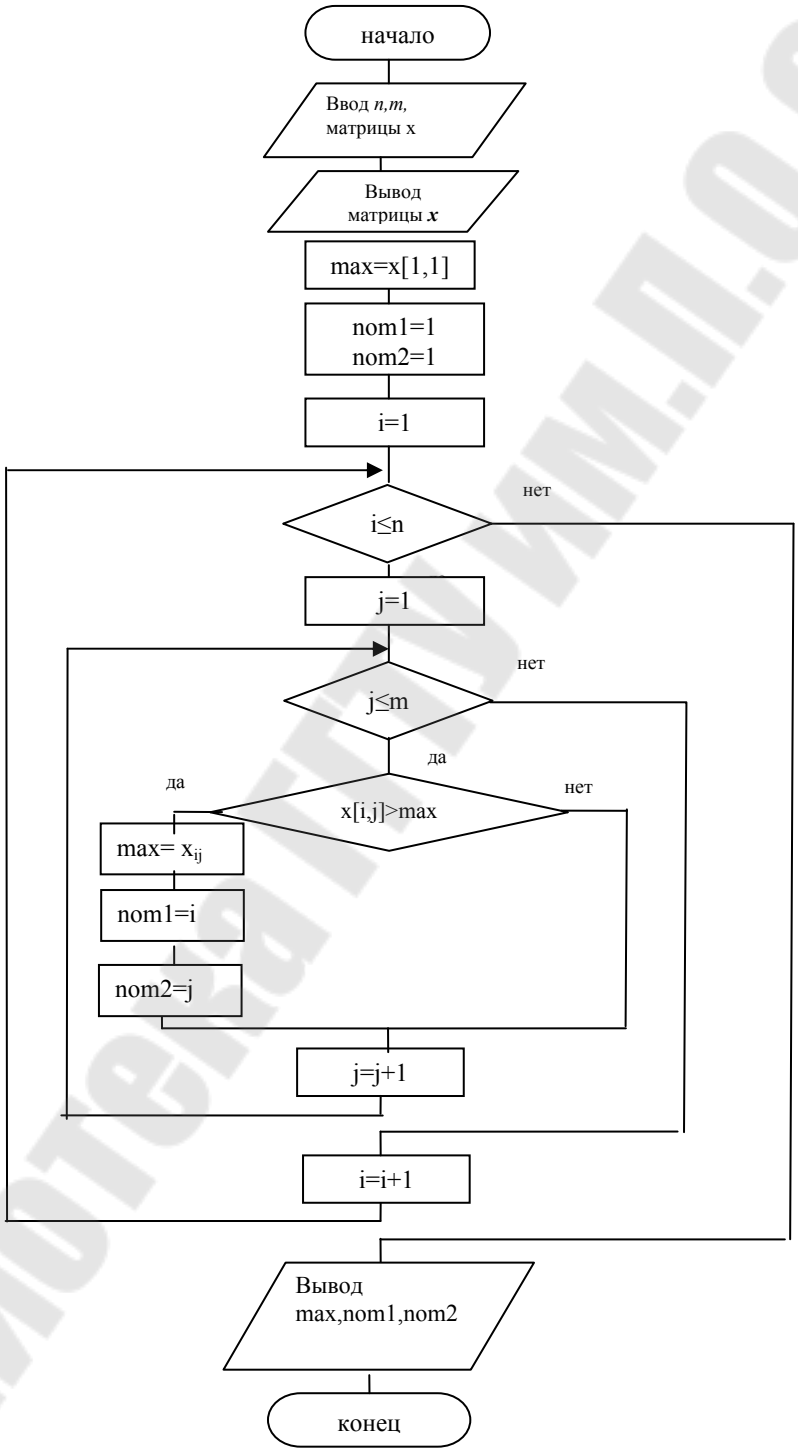



*Рис 8.4.* Общая графическая схема типового алгоритма

**Пример 8.3**. Найти минимальный элемент среди элементов, расположенных в столбцах с четными номерами и заменить его суммой элементов первой строки.

**Фрагмент процедуры**

```
min := x[1, 2];
nom1 := 1;
nom2 := 2;
for i := 1 to n do
    begin
      j := 2;
      while j <= m do
        begin
          if x[i, j] < min then
              begin
                min := x[i, j];
                  nom1 := i;  nom2 := j
              end;
          j := j + 2
        end;
   end;
S := 0;
for j := 1 to m do
  S := S + x[1, j];
x [nom1, nom2] := S;
< вывод x >
```

**Пример 8.4.** Найти максимальный элемент пятого столбца и номер строки, в которой он находится.

```
max := x [1, 5];
nom_str := 1;
for i := 1 to n do
    if x[i, 5] > max then
       begin
          max := x[i, 5];
          nom_str := i
       end;
<вывод max, nom_str>
```

**Пример 8.5.** Найти максимальный элемент в каждой строке матрицы и вычислить сумму этих максимальных элементов.

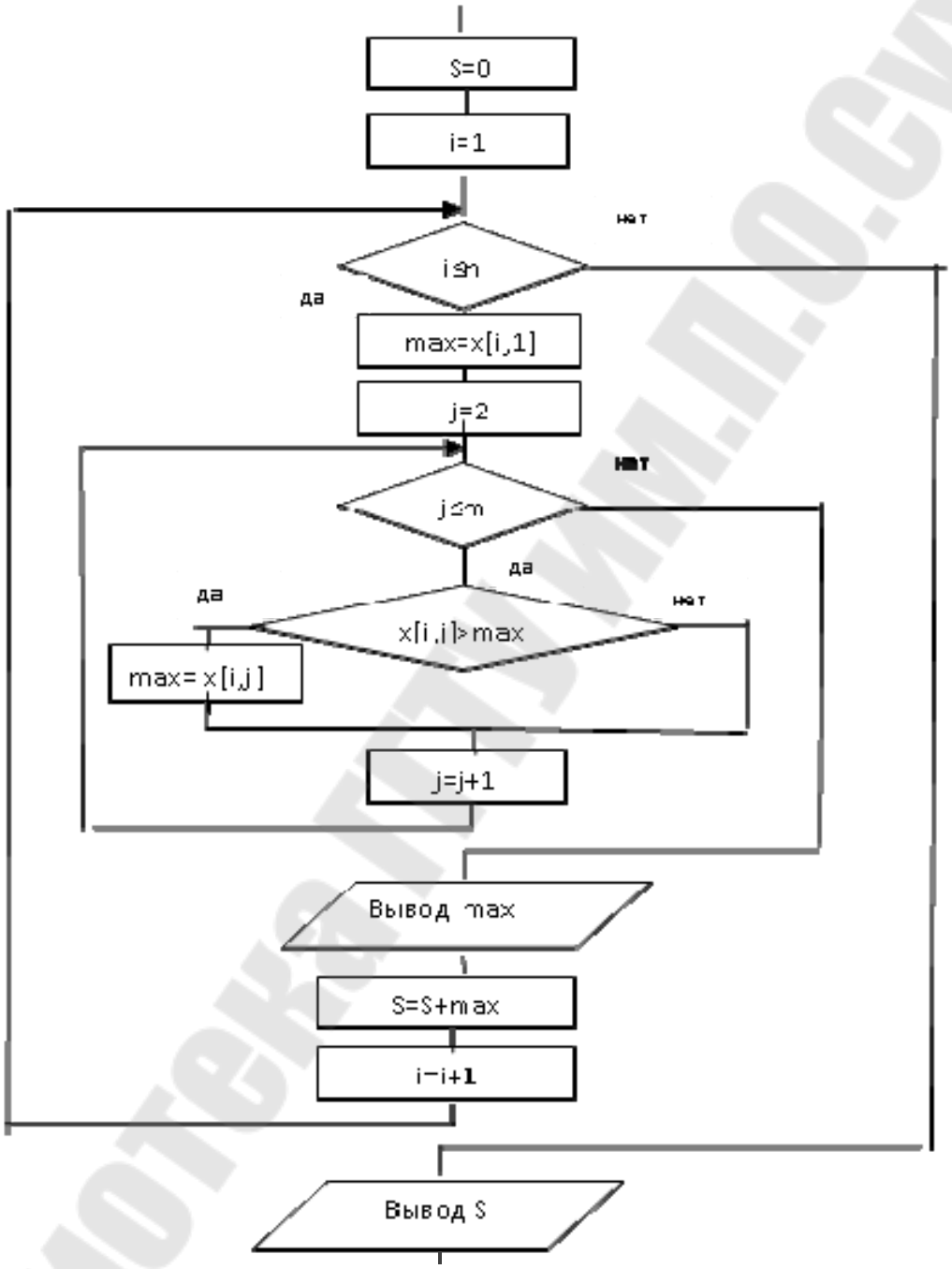



*Рис 8.5.* Фрагмент графической схема алгоритма примера 8.5

**Пример 8.6.** В каждом столбце матрицы найти сумму квадратов элементов, а затем вычислить произведение этих сумм.

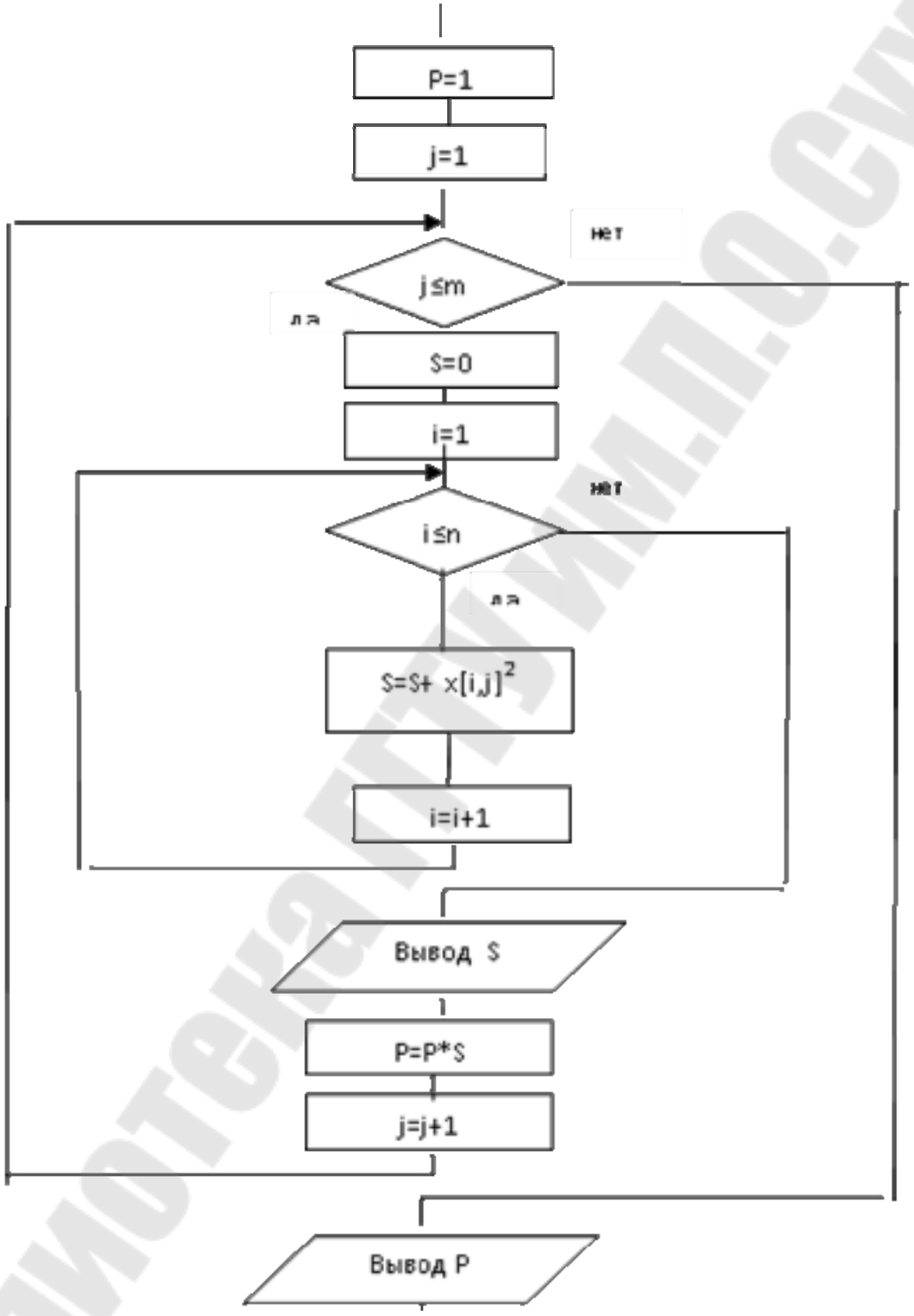



*Рис 8.6.* Фрагмент графической схемы алгоритма примера 8.6

**Замечания к алгоритмам обработки массивов:**

1. Если вычисляется одно значение переменной (сумма, произведение, максимум) для всей матрицы, то ее начальное значение устанавливается перед внешним циклом, а результат выводится после завершения внешнего цикла.
2. Если вычисляется значение переменной для каждой строки или каждого столбца, то начальное значение устанавливается внутри внешнего цикла перед внутренним циклом, а результат выводится или используется после завершения внутреннего цикла внутри внешнего.
3. Если нужно вычислить значение переменной для каждой строки, то внешний цикл организуется по номеру строки, если для каждого столбца, то по номеру столбца.
4. Если значение переменной вычисляется для одной строки или одного столбца, то вложенные циклы не нужны. Организуется один цикл по номеру столбца или номеру строки соответственно.

## 8.6. Программирование циклов с досрочным завершением

Задачи обработки массивов, использующие циклы с досрочным завершением, часто имеют следующие формулировки:

1. Определить есть ли в массиве элемент или группа элементов, удовлетворяющих заданному условию.
2. Определить место первого элемента или группы элементов, удовлетворяющих заданному условию.
3. Определить, все ли элементы или группы элементов удовлетворяют заданному условию.

При решении таких задач могут возникнуть ситуации, определенные исходным массивом, при которых нет необходимости просматривать весь массив полностью, т.к. в первом случае, найден элемент или группа элементов, удовлетворяющие заданному условию, а во втором случае найден элемент или группа элементов, не удовлетворяющие заданному условию.

Если в массиве не нужно просматривать все элементы, строки или столбцы, следовательно, нужно завершить цикл просмотра элементов при выполнении некоторого дополнительного условия.

Дополнительное условие включается в условие цикла, оно содержит второй параметр цикла.

Для программирования таких алгоритмов всегда используется оператор цикла While, т.к. число повторений цикла нельзя определить до начала цикла, оно зависит от элементов массива.

**Пример 8.7.** Определить, есть ли в одномерном массиве положительные числа.

**Фрагмент процедуры**

```
k:=0;
i:=1;
While (i<=n) and (k=0) do
    begin
      if x[i]>0 then k:=1;
      i:=i+1
    end;
if k=0 then
     ……
```

**Пример 8.8.** Определить, все ли элементы массива больше заданного числа z.

**Фрагмент процедуры**

```
k := 0;
i := 1;
While (i <= n) and (k = 0) do
   begin
     if x[i] <= z  then k := 1;
     i:=i+1
   end;
if k = 0 then
    <все элементы больше z >
    else
     <не все элементы больше z >;
```

**Использование логических переменных в циклах с двумя параметрами.**

Логические переменные могут принимать только одно из двух значений true (истина) или false (ложь). Для их описания используется тип Boolean. В памяти эти переменные занимают 1 байт.

Использование переменных целого и логического типов, соответственно:

```
var                                  var
  k: byte;                             l: boolean;

if k=0 then ……                       if l = true then ……   (if l….)
if k=1 then ……                       if l = false then ……  (if not l….)

While (i<=n) and (k=0) do            While (i<=n) and l do
   ……..                                 …….. (l=true)
```

**Пример 8.9.** Определить, является ли одномерный массив упорядоченным по не убыванию.

Массив упорядочен, если для всех i = 1 .. n-1 выполняется соотношение между элементами массива:

$x_i < x_{i+1}$ – по возрастанию
$x_i <= x_{i+1}$ – по не убыванию
$x_i > x_{i+1}$ – по убыванию
$x_i >= x_{i+1}$ – по не возрастанию

**Фрагмент процедуры:**

```
l=true;
i:=1;
While (i <= n-1) and l do
  if x[i] <=  x[i+1]
    then i:=i+1
    else  l:=false
if  l then
  ……
```

**Пример 8.10.** Определить, есть ли в матрице столбец, в котором все элементы положительные.

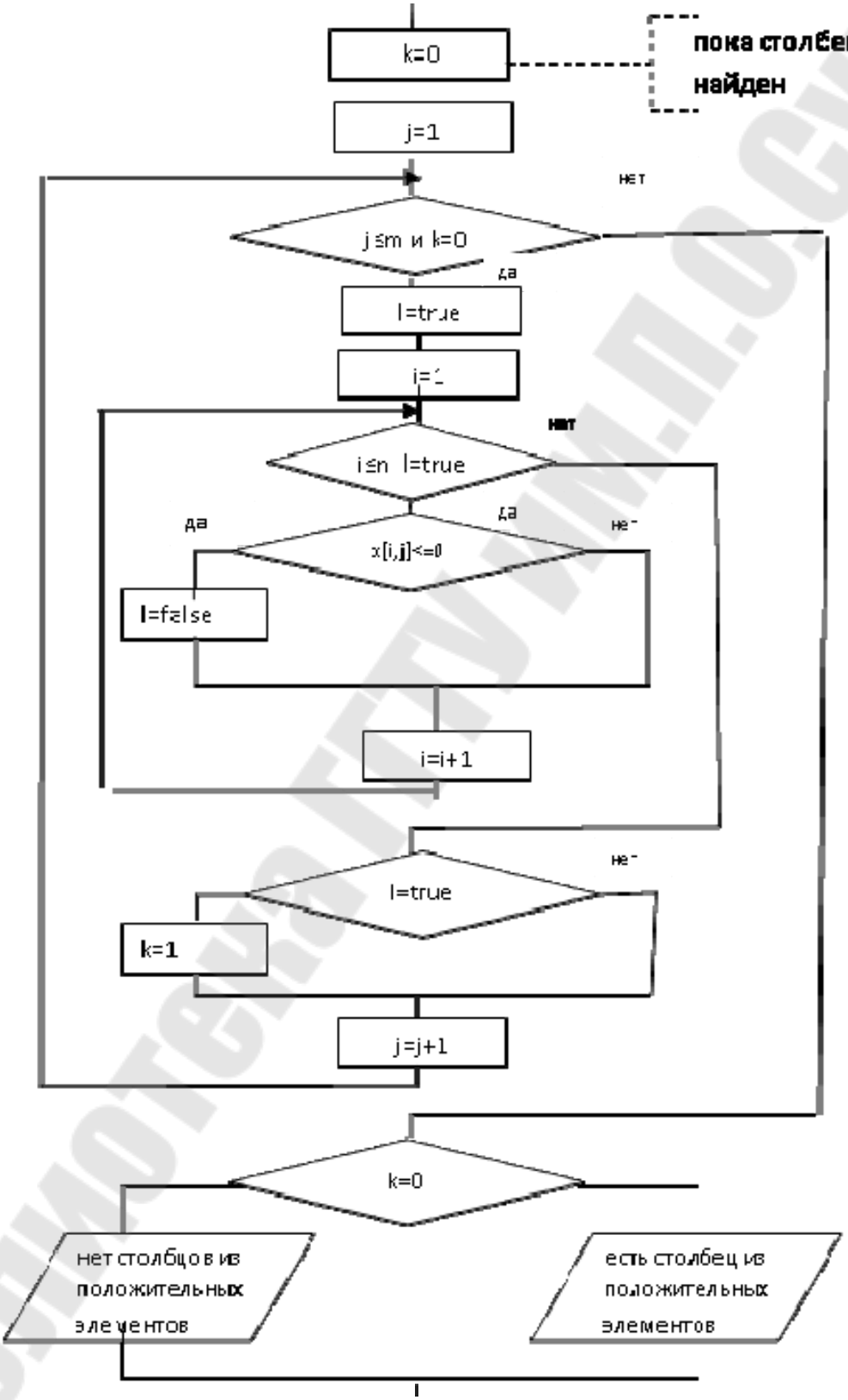



*Рис 8.7.* Фрагмент графической схемы алгоритма примера 8.10

## 8.7. Работа с динамическими массивами

1. Описание динамического массива:
   x:array of real; - одномерного
   x:array of array of real; - двумерного

1. Выделение памяти перед использованием
   setlength (x,n);
   setlength (x,n,m); n – размер по первому индексу, m – размер по 2 индексу

2. Нижняя граница индекса у динамических массивов всегда 0.
   Первый индекс изменяется от 0 до n-1, второй от 0 до m-1

3. Завершение работы с массивом, освобождение памяти:
   x:=NIL; или Finalize (x);

Пример обработки двумерного массива:

```
var
    x:array of array of real;
begin
  n:=StrToInt(Edit1.Text);
  m:=StrToInt(Edit2.Text);
  setlength(x,n,m);
  for i:=0 to n-1 do
    for j:=0 to m-1 do
      x[i,j]:=StrToFloat(StringGrid1.Cells[j,i]);
    for i:=0 to n-1 do
       for j:=0 to m-1 do
        StringGrid2.Cells[j,i]:=FloatToStr(x[i,j]);
<работа с массивом>
  Finalize (x);
end;
```

## Тема 9. Обработка символьной информации

### 9.1. Представление символьной информации в памяти ПК

Набор кодов всех символов представлен в кодовой таблице. Способ кодировки может быть различным. В ОС DOS использовался код ASCII (American Standard for Information Interchange), в ОС Windows однобайтный код ANSI (по имени American National Standard Institute, предложившего этот код), двухбайтный код Unicode. Первая часть таблицы с кодами 0-127 постоянна для всех таблиц. Во второй части располагаются символы национальных алфавитов, причем эти символы могут находиться в таблицах на разных местах.

В стандартных шрифтах Windows Arial, Times New Roman русские буквы от А до Я имеют соответственно коды 192..223, строчные буквы от а до я коды 224..255, выделяются символы Ё (код 168) и ё (код 184). Поэтому, если предполагается сортировка информации, содержащей русские буквы, символы Ё и ё лучше не использовать. Символы цифр от 0 до 9 имеют коды соответственно 48..57. Символы латинских букв расположены подряд в кодовой таблице, сначала заглавные, затем строчные. Символы с кодами от 0 до 31 относятся к служебным символам. Если они используются в программе, то считаются пробелами.

### 9.2. Описание переменных для хранения символьной информации

В Delphi для хранения символьной информации используются переменные двух типов: символьного и строкового.

Переменная символьного типа может содержать только один символ, в памяти занимает один байт, для ее описания используется тип char. Символьная константа – это один символ, заключенный в апострофы.

Пример:

```
var   c1,c2:char;
begin
   c1:= 'z' ;
…….
end;
```

Переменная строкового типа может содержать последовательность символов. Строковая константа – это последовательность символов, заключенная в апострофы.

Для описания строк в Delphi используются четыре типа. Для описания короткой строки используется типы String [N], где N<=255 или ShortString (String [255]), для длинной - тип String.

Широкие строки типа WideString по своим свойствам идентичны строкам типа String, но для представления каждого символа используются два байта.

Общим для всех типов является то, что строка рассматривается как одномерный массив символов, нумерация хранимых символов начинается с 1.

Для короткой строки String [N] (при компиляции, статически) в памяти выделяется N+1 байт, ее длина меняется от 0 до N. Максимальная длина такой строки 255 символов. Нулевой символ используется для хранения длины строки.

Длинная строка может содержать от 0 до 2 Гб символов. Память для строки выделяется динамически во время выполнения программы и ограничена только размером доступной программе памяти.

## 9.3. Операции над строковыми и символьными переменными

**Операция (оператор) присваивания.**

Если длина строки-приемника меньше длины строки-источника, то лишние символы усекаются.

Пример:

```
var
   s1:string[100];
   s2:string;   s3: string[2];
begin
   s1:= 'abcd'
   s2:=s1;
   s3:=s1;   s3→ 'ab'
…….
end;
```

**Операция сцепления** (+) применяется для объединения нескольких строк в одну.
Пример:

```
var
   s,s1:string[12];
   s2,s3:string[2];
begin
   s1:= 'Группа';
   s2:= 'ЗИС';
   s3:= '12';
   s:=s1 + ' ' + s2+'-' + s3;    s→ 'Группа ЗИС-12'
end;
```

**Операции отношения ( < , <= , > , >= , = , <> )**

Символы можно сравнивать, при этом сравниваются их коды в кодовой таблице.

'a' > 'A', т.к. код 224>192

В соответствии с расположением символов в кодовой таблице

'0' <…< '9' <…<'$A_{лат}$' <…< '$Z_{лат}$' <…< 'a'<….< 'z'<… < '$A_{рус}$' < …<'Я' <…< '$a_{рус}$' <…< 'я'

будет упорядочена символьная информация.

Строки сравниваются посимвольно слева направо. Недостающие символы более короткой строки заменяются значением #0. Две строки считаются равными, если они имеют одинаковую длину и на соответствующих позициях находятся одинаковые символы (их коды совпадают). Та строка считается большей, у которой первый несовпадающий символ имеет больший код или строка имеет большую длину.

Пример: 'ЗИС-11' < 'ЗИС-12' 'ИПК и ПК' > 'ИПК'

**Обращение к отдельным символам строки**

```
var
   s:string[10];      array [1..10] of char;
```

Обращение к символу строки имеет вид

**<идентификатор строки> [номер символа]**

s [5] – пятый символ строки.

s:= 'ЗИС-11' s[5]:=2 s → 'ЗИС-12'

s[0] – для короткой строки определяет длину строки. В нулевом байте хранится символ, код которого равен длине строки. Для того, чтобы получить числовое значение длины строки, нужно использовать функции ord(s[0]) для короткой строки или length(s) для любой строки.

## 9.4. Стандартные функции для работы с символами

Объявим переменную

```
var  Ch:char;
```

- **Chr (B**:**Byte):Char** – возвращает символ, код которого равен B.

Пример: Chr (65) → 'А$_{лат}$' Chr (49) → '0'

Вместо функции можно использовать знак #.

Для вывода текста в ВК Label в несколько строк можно использовать символ перехода на новую строку (CR) с кодом 13.

Label1.Caption:=’первая часть строки’ + #13 + ‘вторая часть строки’

- **Ord (Ch):Byte** – возвращает числовое значение, соответствующее коду символа в кодовой таблице.

Пример: Ord ('А$_{лат}$') → 65 Ord ('0') → 49

- **Succ (Ch), Pred (Ch),** – возвращают, соответственно, следующий или предыдущий символ в кодовой таблице, т.е. символы, для которых

Ord (Succ (Ch))= Ord (Ch)+1  Ord (.Pred (Ch))= Ord (Ch)-1

Пример: Ch:='B' ;  Pred (Ch) → 'A'  Succ (Ch) → 'C'

- **UpCase (Ch)–** возвращает прописную букву для строчной латинской буквы, для остальных символов возвращает сам символ.

Пример: Заменить в строке все строчные латинские буквы прописными.

```
var
  s,s1:string;
    i:byte;
begin
   s :=Edit3.Text;
   For i:=1 to length(s) do
      s[i]:=upcase(s[i]);
end;
```

## 9.5. Стандартные функции для работы со строками

```
var
   S,S1:string;
   m,n:Integer;
   X:real (integer);
```

- **Length (S)** – определение текущей длины строки.

Пример:

```
n:=Length (S)
S:= 'Гомель'     n=6
S:= ' '          n=1
S:= ''           n=0
```

- **Copy (S,m,n)** – выделение части строки S длиной n символов, начиная с позиции m. Строка S не изменяется.

S:= 'факультет ЭФ ';
S1:= Copy (S,1,9)  S1→ 'факультет'
S1:= Copy (S,11,2)  S1→ ‘ ЭФ‘

- **Pos (S1,S)** – определение номера позиции строки S, начиная с которого строка S1 первый раз входит в строку S. Если S не содержит S1, то функция возвращает ноль.

Примеры:
S:= 'абракадабра';  S1:= 'аб '

k:= Pos (S1,S)  k=1
k:= Pos ('ра',S)  k=3
k:= Pos ('та',S)  k=0

- **Concat (S1,S2,….SN)** – сцепление строк S1,S2,….SN в одну строку в указанном порядке. Если результат помещается в короткую строку, то его длина должна быть не больше 255.

Примеры:
S1:= 'груп'  S2:= 'па'  S3:= 'ЗИС-12'
S:= Concat (S1,S2, '  ',S3)  или S:= S1+S2+' '+S3
S→ 'группа  ЗИС-12'

- **UpperCase (S), LowerCase (S)** – возвращают исходную строку, в которой, соответственно, все строчные латинские буквы заменены прописными или прописные строчными .
- **AnsiUpperCase (S), AnsiLowerCase (S) -** возвращают исходную строку, в которой, соответственно, все строчные буквы кириллицы заменены прописными или прописные строчными

s1:= UpperCase (s)

- **StringOfChar (Ch,n:Integer)** – создает строку из n раз повторенного символа Ch

s:= StringOfChar ('*',20)

### 9.6. Стандартные процедуры для работы со строками

- **Delete (S, m, n)** – удаление из строки S n символов, начиная с позиции m. Результат остается в строке S, ее длина уменьшается на n символов.

Пример:

```
S:= ' ИПК и ПК'
delete (S, 1, 6)    S→  ' ПК'
delete (S, 4, 5)    S→  ' ИПК'
```

- **Insert (S1, S,m)** – вставка строки S1 в строку S, начиная с позиции m.

Пример:

```
S:= ' факультет ЭФ'    S1:= 'ГГТУ '
Insert (S1, S, 1)      ' ГГТУ факультет ЭФ'
Insert (' им. П.О.Сухого, ',S, 6)
 S→ ' ГГТУ им. П.О.Сухого, факультет ЭФ'
```

- **Str (X[:p:q],S)** – преобразование числового значения X в строку S. X может иметь формат вывода, в соответствии с которым он преобразуется в S, в котором **p** - это общее количество символов в представлении числа в строке, **q** – число знаков в дробной части числа.

Примеры:

```
X:=1500 (целое)          Str (X:6 , S) ;    --1500
X:=15.7  (веществ)       Str (X:6:2 , S) ;    -15,70
```

- **Val (S, X, Code)** – преобразование значения в строке S в число X целого или вещественного типа. Code равен нулю, если преобразование прошло без ошибок, противном случае Code равен номеру первого ошибочного символа, X в этом случае не определено. Разделителем целой и дробной части вещественного числа в строке может быть только **.** (точка).

Примеры:

```
S:= '1450'   Val (S, X, Code)  X=1450  (целое)     Code = 0
S:= '145.7'  Val (S, X, Code)  X=145.7 (веществ) Code = 0
S:= '145Я'   Val (S, X, Code)  X не определено     Code = 4
```

**Пример 9.1.** В заданной строке символов изменить год 2011 на 2012.

**Способ 1.**

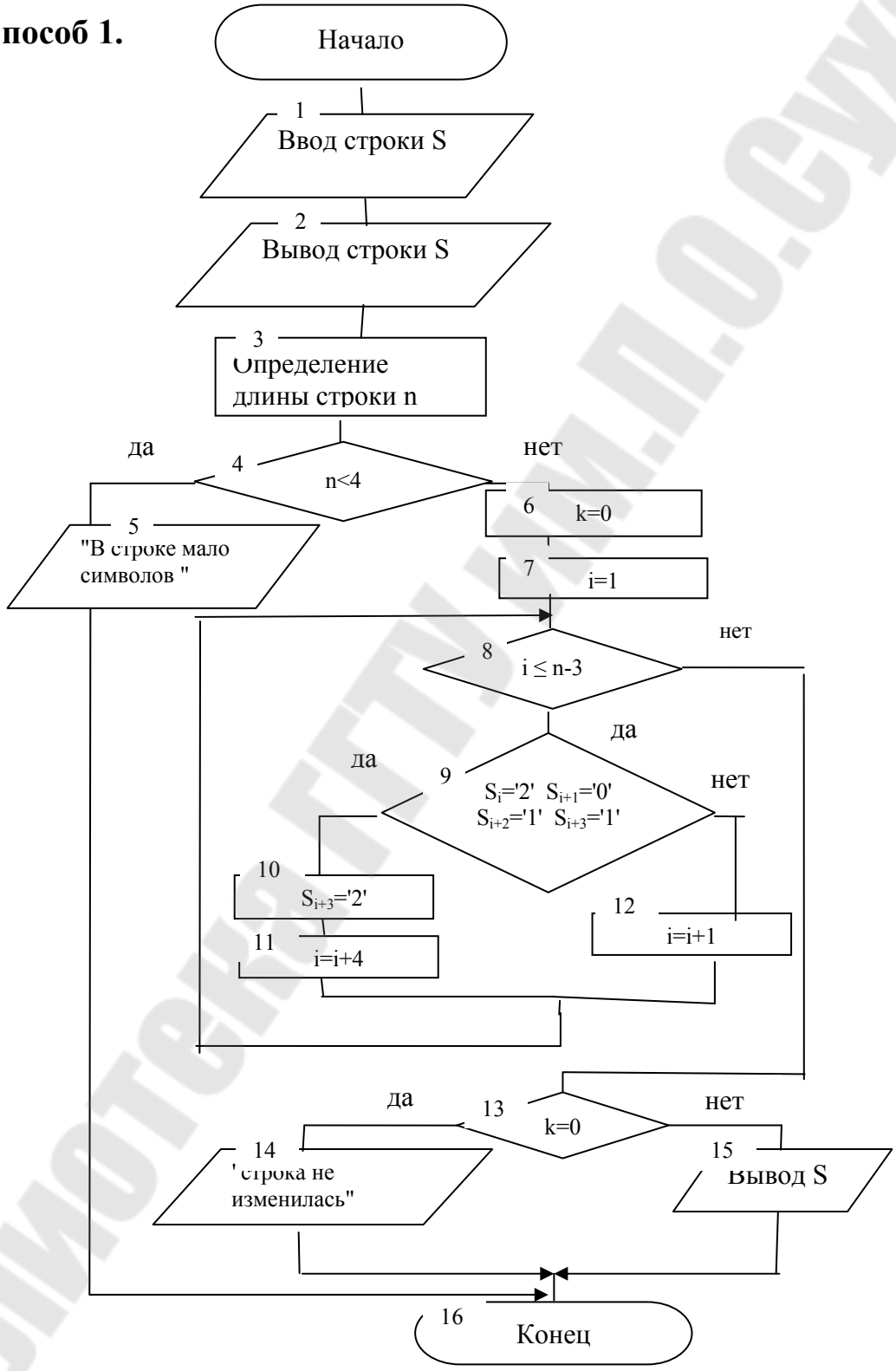



*Рис. 9.1.* Графическая схема алгоритма примера 9.1 (способ 1)

**Фрагмент процедуры:**

```
n := length(S);
if n < 4
then ShowMessage ( 'Мало символов' )
else
begin
  i := 1; k := 0;
  While i< = n - 3 do
    if (s[i]='2') and (s[i+1]= '0')  and (s[i+2]='1')  and (s[i+3]='1')
    then
      begin                              s[i]+…+s[i+3]='2011'
        s[i+3] :='2'; k:=1;              copy(s,i,4)= '2011'
        i:=i+4                           concat(s[i],…s[i+3])= '2011'
      end
   else
     i:=i+1;
if k=1
   then ShowMessage ( 'Строка не изменилась‘ )
   else  <вывод S>
end
```

**Способ 2.**

**Фрагмент процедуры:**

```
k:=pos('2011', s);
if k=0
   then ShowMessage ('Нет 2011 в строке')
else
begin
   w:=true;
   While w do      (k<>0)
      begin
        delete (s, k+3,1);
        insert ('2', s, k+3);
        k:=pos('2011', s);
```

```
      if k=0
        then w:=false
    end;
<вывод s>
end;
```

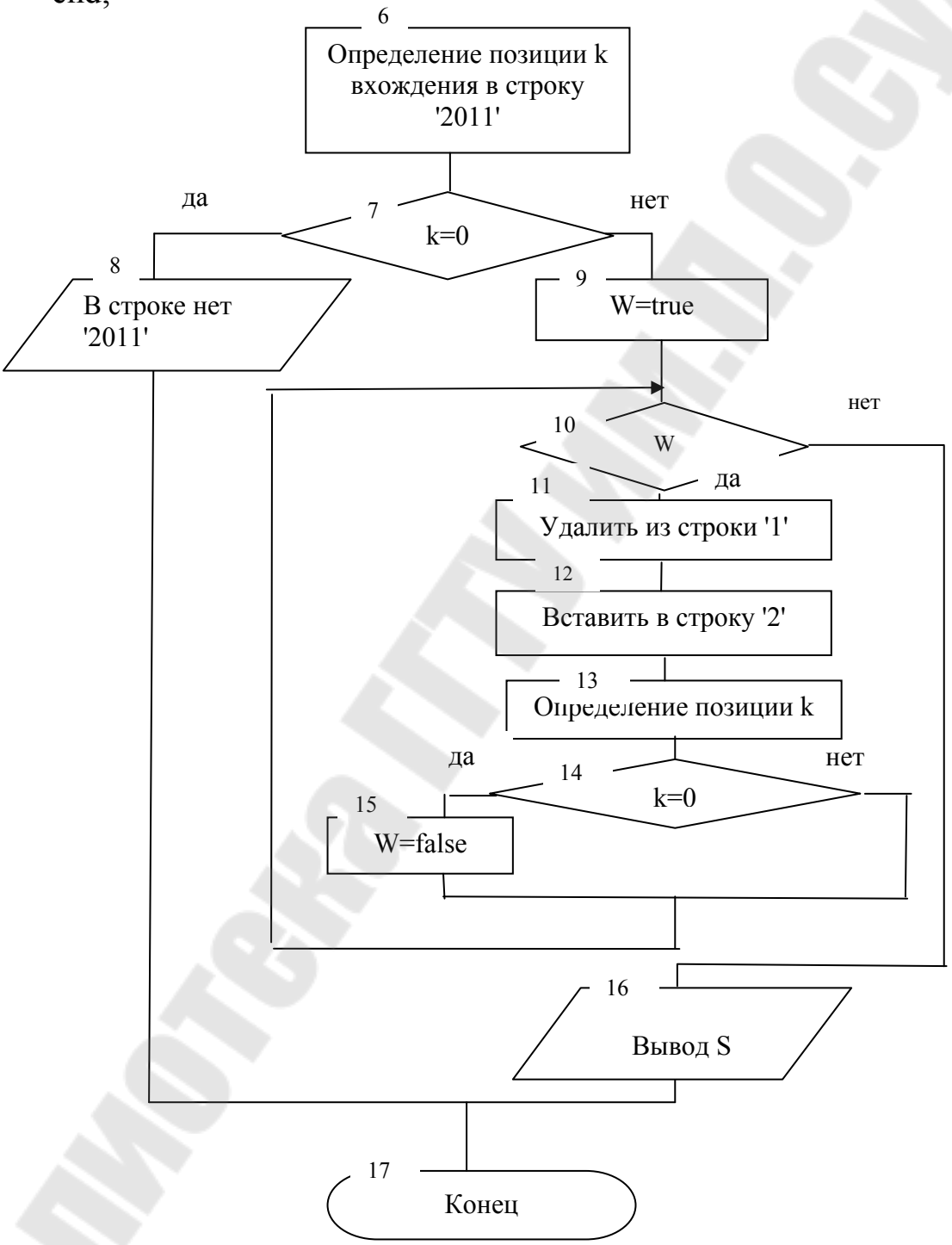



*Рис. 9.2.* Графическая схема алгоритма примера 9.1 (способ 2)

**Пример 9.2.** Удалить из строки все символы ‘*’.

```
k:=pos( '*',s);
if k=0
  then ShowMessage (' Нет “* “ в строке')
 else
   begin
     While k<>0 do
       begin
         delete (s,k,1);
          k:=pos( '*',s);
       end;
      <вывод s>
   end;
```

**Пример 9. 3.** Удалить из строки все ведущие пробелы.

```
    w:=true;
    While w and (length(s)<>0) do
        if s[1] = ' '
            then delete (s,1,1)
            else w:=false;
   <вывод s>
```

**Пример 9.4.** После каждой цифры вставить такую же цифру.

```
    i:=1;
    While i <= n (length(s)) do
        if (s[1] >='0 ' ) and (s[i]<= '9')
            then
               begin
                insert (s[i],s,i+1);
                n:=n+1;
                i:=i+2
             end
           else
             i:=i+1;
           <вывод s>;
```

**Пример 9.5.** Задана строка текста, состоящая из нескольких слов. Слова состоят из любых символов и разделяются одним или несколькими пробелами. Необходимо выполнить следующие действия:

- подсчитать количество букв в каждом слове и общее количество слов;
- после каждой буквы второго слова вставить "/";
- удалить пятое слово;
- вывести слова, длина которых меньше 4;
- подсчитать число слов, начинающихся на "я".

**Фрагмент процедуры:**

```
if  length(s) =0
  then <вывод 'строка пустая'>
  else
   begin
    k:=0;  m:=0; i:=1;
    While i <= length(s) do
     begin
       if (s[i] < > ' ')
            then
              begin
               k:=k+1; p:=0;  l:=i;     //начало слова
               if s[l]='я' then m:=m+1;
                While i <= length(s) and (s[i] <> ' ')  do
                 begin                          //движение внутри слова
                   p:=p+1;
                   if k=2  then
                     begin
                       insert ('/' , s, i+1);
                      i:=i+2
                     end
                   else
                     i:=i+1;
                end
               <вывод p,k>;  //количество букв в каждом слове
               if k=5
                 then
```

```
                    begin
                      delete(s, l, p);   //удалить пятое слово
                       i=i-p
                    end;
                 if p<4
                   //вывести слова с длиной < 4
                   then  < вывод слова (copy(s, l, p))>;
            end
        i:=i+1
     end;
    if k=0 then  < строка пробелов>
           else
            begin
              if k=1 then
                  < 'в строке одно слово, она не изменилась'>
                  else <вывод k, s>;
              if m=0 then
                   <нет слов, начинающихся на "я"'>
                   else <вывод m>
            end
  end;
```

## Тема 10. Использование подпрограмм. Процедуры и функции

### 10.1. Общие сведения. Область действия подпрограмм

Подпрограмма – это логически законченная и специальным образом оформленная часть программы, имеющая собственное имя.

Подпрограммы могут быть стандартными (sin(x), power(x,y)) или подпрограммами пользователя. Все подпрограммы пользователя должны быть описаны в разделе описаний. Подпрограммы пользователя могут быть результатом обработки событий.

В Delphi используются два вида подпрограмм:

- процедуры;
- функции.

Любая подпрограмма может быть реализована в виде процедуры. Функции используются в том случае, если результатом

работы подпрограммы является одно значение (сумма, максимум и т.д.).

Подпрограмма может содержать в своем разделе описаний описания других подпрограмм. В разделе операторов подпрограммы можно обратиться только к той подпрограмме, которая была описана ранее или к подпрограмме, которая описана в разделе описаний данной подпрограммы.


*Рис.10.1.* Область действия подпрограмм

Подпрограммы пользователя можно разместить в собственном модуле. В этом случае они будут доступны любой подпрограмме

модуля, к которому подключается данный модуль с помощью предложения uses.

**Для создания собственного модуля необходимо:**

- в ИСР выполнить команду *File – New – Unit (for win 32 в TurboDelphi)*;
- определить имя модуля в заголовке модуля, например
  Unit my_module;
- разместить в интерфейсной части модуля (**interface**) заголовки подпрограмм, а их описания в исполняемой части модуля (**implementation**); в интерфейсной части можно описать глобальные типы, константы и переменные, в частности типы, используемые в заголовке подпрограмм, в исполняемой части локальные для модуля объекты;
- сохранить модуль под тем же именем **Save As** my_module.pas;
- можно выполнить компиляцию модуля *Project – Compile* или он буден скомпилирован при первом выполнении программы, к которой будет подключен;
- в модуле формы подключить свой модуль:

```
implementation
    uses my_module;
```

## 10.2. Описание процедуры. Формальные параметры

```
Procedure  <имя> ([список формальных параметров]);
//   заголовок процедуры

 <раздел описаний>;
begin
   <раздел операторов>
end;
```

тело процедуры

Пример заголовка процедуры:

```
Procedure Proizv (x,y: integer; z: real);
```

Формальные параметры – это переменные, которые необходимы для связи процедуры с вызывающей подпрограммой. Они используются в операторах тела процедуры и являются для процедуры исходными данными или результатом ее работы.

Формальные параметры перечисляются в списке с указанием их типов, разделяются символом “**;**”, однотипные параметры можно объединять, а их имена разделять запятыми. Типы формальных параметров должны быть простыми (real, byte, integer, string) или должны быть указаны в виде определенного ранее идентификатора типа для массивов или для короткой строки типа String [N].

```
type
  mas= array [1..20] of real;
Procedure  PR (x:mas);
```

В качестве формальных параметров могут использоваться визуальные компоненты.

**Пример 10.1.** Описание процедуры вывода одномерного массива.

```
type
  mas= array [1..20] of real;
Procedure  VIVOD (x: mas; n: byte; st: TStringGrid);
  var
    i: byte;
  begin
    For i:=1 to n do
        St. Cells[i-1,0]:=FloatToStr(x[i]);
  end;
```

Delphi поддерживает открытые одномерные массивы – формальный параметр подпрограммы, описывающий тип элементов массива, но не определяющий его размерность и границы индекса.

```
array  of  <тип элементов>;
```

Внутри подпрограммы такой параметр рассматривается как одномерный массив с нижней границей индекса равной нулю.

Верхняя граница индекса открытого массива определяется стандартной функцией High**.**

**Пример 10.2.** Подпрограмма вывода, которая может использоваться для вывода массивов вещественных чисел разной размерности:

```
Procedure  VIVOD (x: array of real; st: TStringGrid);
  var
    i: byte;
  begin
    For i:=0 to High(x) do
        St. Cells[i,0]:=FloatToStr(x[i]);
  end;
```

Для передачи подпрограмме массивов переменной длины и размерности можно использовать вариантные массивы.

### 10.3. Обращение к процедуре (вызов процедуры). Фактические параметры

Для обращения к процедуре используется оператор:

<имя процедуры> ([список фактических параметров]);

Список фактических параметров может содержать переменные, выражения, константы. Они разделяются запятыми.

**Пример 10.3.** Обращение к процедуре вывода одномерного массива:

```
VIVOD (x, n, StringGrid1);
VIVOD (y, m, StringGrid2);
VIVOD (z, 10, StringGrid3);
```

с открытым одномерным массивом

```
VIVOD (x,  StringGrid1);
VIVOD (y, StringGrid2);
VIVOD (z,  StringGrid3);
```

Список фактических параметров при обращении к процедуре должен соответствовать списку формальных параметров в описании этой процедуры:

- количество фактических параметров должно совпадать с количеством формальных параметров;
- порядок следования фактических параметров должен совпадать с порядком следования формальных параметров.

Delphi проверяет совпадение типов соответствующих параметров, за смысловым соответствием должен следить программист. Никакого соответствия имен нет, они могут совпадать или различаться для соответствующих параметров.

**Пример 10.4.** Процедура вывода двумерного массива:

```
type
      matr = array [1..10,1..5] of real;
```

Заголовок процедуры:

```
Procedure VIVOD (x:matr; n,m:byte; …..);
```

Обращение к процедуре для матрицы $A_{k*l}$ :

VIVOD (A, k, l, ….); - верно

VIVOD (k, l, A, ….); - неверно, будет обнаружено Delphi (несоответствие типов)

VIVOD (A, l, k, ….); - неверно, не будет обнаружено Delphi, ошибка обнаруживается на этапе тестирования.

## 10.4. Классы формальных параметров. Способы передачи параметров в подпрограммы

Формальные параметры разделяются на входные и выходные. Входные параметры являются для процедуры исходными данными, выходные – результатом ее работы.

Любой формальный параметр может быть параметром-значением, параметром-переменной или параметром-константой. Если параметр определен как параметр значение, то в подпрограмму

передается адрес копии соответствующего фактического параметра и подпрограмма работает только с копией. Поэтому изменение формального параметра в подпрограмме не приводит к изменению фактического. Так передаются входные параметры. Достоинство способа это защита от изменений и передача выражений, недостаток – увеличение объема памяти.

Если параметр определен как параметр-переменная, то в подпрограмму передается адрес соответствующего фактического параметра. Поэтому любое изменение формального параметра приведет к изменению фактического. Так передаются параметры, которые являются результатом работы подпрограммы или в ней изменяются (выходные, но они же могут быть и входными). Перед такими параметрами необходимо указывать служебное слово **var**.

Можно использовать параметры-переменные с зарезервированным словом **out**. Такие параметры не могут использоваться как входные, они предназначены только для передачи результата.

При использовании параметра-константы в подпрограмму передается адрес соответствующего фактического параметра, но изменить значение параметра в подпрограмме невозможно. Для описания таких параметров используется служебное слово **const**. Их можно использовать при передаче в подпрограмму входных массивов.

## 10.5. Область действия идентификаторов. Локализация имен

Все переменные, описанные в теле подпрограммы, а так же ее формальные параметры являются **локальными** переменными этой подпрограммы. Они могут использоваться только в той подпрограмме, в которой они описаны. Память, выделяемая для локальных переменных, будет освобождена после того, как подпрограмма закончит работу.

Кроме локальных переменных в подпрограмме можно использовать ранее объявленные переменные. Они являются **глобальными** по отношению к данной подпрограмме. Если они описаны до первой подпрограммы модуля, то могут использоваться в любой подпрограмме модуля. Если они описаны в одной из подпрограмм, то могут использоваться в разделе операторов этой

подпрограммы и в любой из подпрограмм, описанных далее в этой подпрограмме. Все это относится также к любым идентификаторам.

Один и тот же идентификатор может быть описан и как локальный и как глобальный. Тогда локальное описание закрывает глобальное и делает его недоступным. Внутри подпрограммы будет использоваться локальный идентификатор, а вне подпрограммы – глобальный, это две разные переменные.

**Пример.10.5.** Вычислить сумму элементов массива x, состоящего из m элементов и массива y из 10 элементов.

Описание процедуры для вычисления суммы элементов одномерного массива.

```
type mas= array [1..10] of real;
Procedure sum (x:mas; n:byte; var s:real);
  var
    i:integer;
  begin
    s:=0;
    for i:=1 to n do
      s:=s+x[i];
  end;
```

Обращение к процедуре:

```
sum (x, m, s1);
sum (y,10, s2);
s:=s1+s2;
Label4.Caption:='Сумма элементов двух массивов=' +
                    FloatToStr(s);
```

### 10.6. Использование функций

Функция – это частный случай подпрограммы, когда подпрограмма имеет единственный выходной параметр простого типа (не массив и не короткая строка).

**Описание функции. Формальные параметры**

Function <имя> ([список формальных параметров]): <тип>;
<тело функции>

Имя функции и список формальных параметров формируется по тем же правилам, что и у процедуры. Список формальных параметров не содержит, как правило, выходных параметров.

тип – это тип результата, обязательно простой.

В теле функции должен быть хотя бы один оператор, обычно выполняемый последним, который присваивает найденное значение имени функции или внутренней переменной result.

result := <выражение>;
или
<имя>:= <выражение>;

**Пример.10.6.** Описание функции для вычисления суммы элементов одномерного массива.

```
function sum (x: mas; n:byte): real;
   var
    i:integer; s:real;
   begin
    s:=0;
    for i:=1 to n do
       s:=s+x[i];
    result:=s  (или sum:=s)
   end;
```

**Обращение к функции. Фактические параметры**

Вызов функции происходит с помощью имени функции, которое может использоваться в выражениях аналогично стандартным функциям. После имени функции в круглых скобках должен быть указан список фактических параметров. Он формируется по тем же правилам синтаксиса и соответствия формальным параметрам, что и у процедуры.

**Пример.10.7.** Вычислить сумму элементов массива x, состоящего из m элементов и массива y из 10 элементов, используя приведенное в примере 10.6 описание функции вычисления суммы элементов массива.

Обращение к функции:

```
s1:= sum (x, m);
s2:=sum (y,10);
s:=s1+s2
Label4.Caption:='Сумма элементов двух массивов=' +
                    FloatToStr(s);
```

или

```
Label4.Caption:='Сумма элементов двух массивов=' +
                  FloatToStr(sum (x, m)+ sum (y,10));
```

## 10.7. Примеры решения задач с использованием подпрограмм

**Пример.10.8.** Вычислить значение y в зависимости от заданного вида функции (одной из трех) $y = \frac{arctg\ bx}{1+\sin^2 x}$,…..

```
Function f 1(b,x:real): real;
begim
  f1:=arctan(b*x)/(1+sqr(sin(x));
 end;
…….
Function f (b,x:real; nom:byte): real;
begim
 case nom f
      1: f:=f1(b,x);
      2: ……..
  end;
 end;
```

<основная программа>

```
begim
  <ввод b,x >
  <определение номера функции n>
  y:= f(b,x,n);
  <вывод y >
end;
```

**Пример.10.9.** Вычислить общее количество положительных элементов в массивах **x, y, z**.

Выделим логически законченные действия и оформим их в виде подпрограмм.

1. Ввод массива – процедура
   Входные параметры:
   Ed (типа TEdit) – ВК для ввода числа элементов
   St (типа TStringGrid) – ВК для ввода элементов массива

   Выходные параметры:
   n – число элементов
   x – массив

Заголовок процедуры:
Procedure VVOD(var x:mas; var n:byte; Ed:TEdit; St:TStringGrid);

2. Вывод массива – процедура
   Входные параметры:
   St (типа TStringGrid) – ВК для вывода элементов массива
   n – число элементов
   x – массив
   Выходные параметры:
   нет

Заголовок процедуры:
Procedure VIVOD( x:mas; n:byte; St:TStringGrid);

3. Вычисление количества положительных элементов массива – функция
   Входные параметры:

n – число элементов
x – массив
Выходные параметры:
k – количество положительных элементов

Заголовок функции:
Function KOL (x: mas; n: byte): real;

Схема алгоритма подпрограммы оформляется аналогично схеме алгоритма основной программы, только вместо блока «Начало» используется блок с именем подпрограммы и списком формальных параметров, а вместо блока «Конец» используется блок «Выход».

В схеме алгоритма блок, в котором используется обращение к подпрограмме, имеет вид:

Предопределенный процесс 

**Основная программа:**

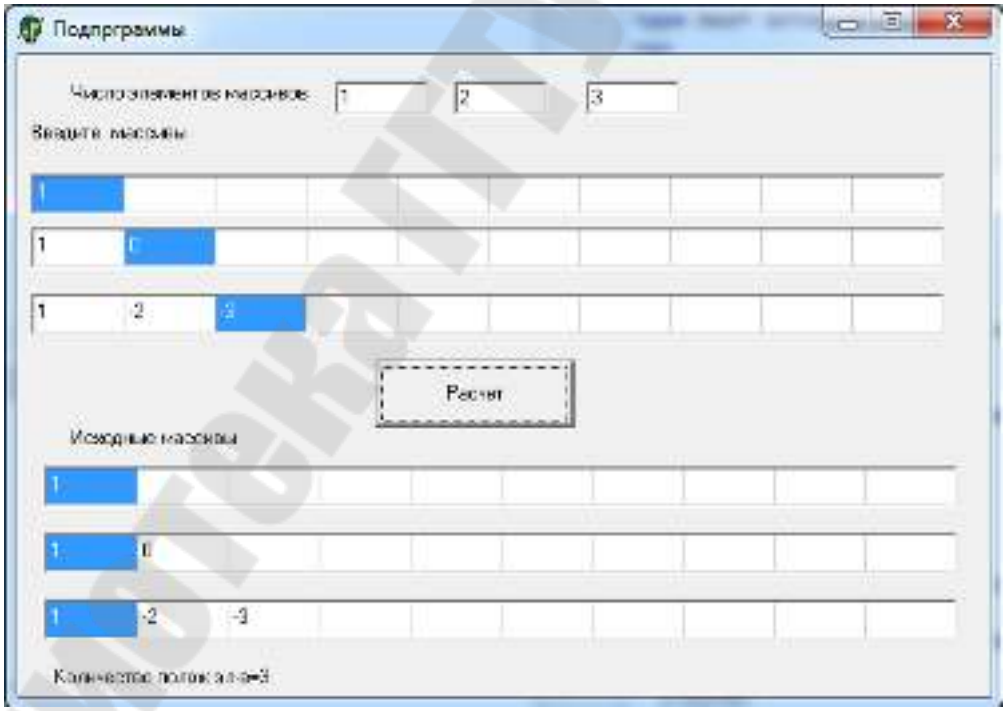



*Рис.10.2.* Вид окна проекта

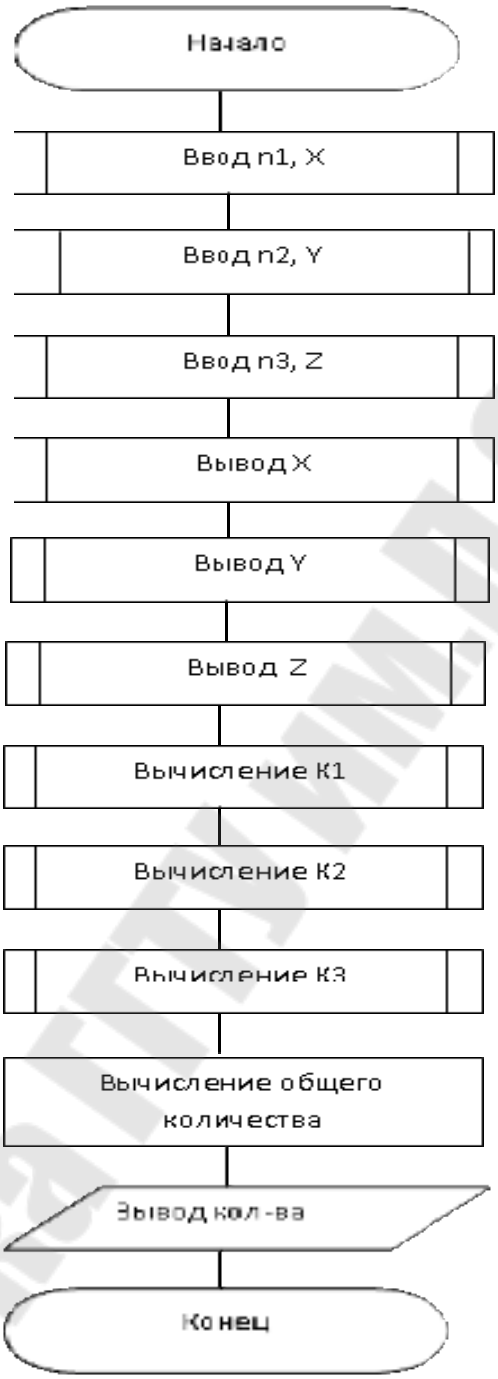



*Рис.10.3.* Графическая схема алгоритма основной программы

Фрагмент процедуры:

```
procedure TForm1.Button1Click(Sender: TObject);
type mas= array [1..10] of real;
var
 …
```

```
<описания подпрограмм>…
begin
  VVOD (x,n1,Edit1,StringGrid1);
  VVOD (y,n2,Edit2,StringGrid2);
  VVOD (z,n3,Edit3,StringGrid3);

  VIVOD (x,n1,StringGrid4);
  VIVOD (y,n2,StringGrid5);
  VIVOD (z,n3,StringGrid6);

   k:= KOL(x,n1)+KOL(y,n2)+KOL(z,n3);
   <Вывод k >
end;
```

**Пример. 10.10.** Поменять местами первый столбец, в котором все элементы положительные, со столбцом, в котором находится максимальный элемент пятой строки

Выделим логически законченные действия и оформим их в виде подпрограмм.

1. Ввод матрицы – процедура
   Выходные параметры:
   n, m – число строк и столбцов матрицы
   a – матрица

```
Procedure VVOD (var a:matr; var n,m:byte);
```

2. Вывод матрицы – процедура
   Входные параметры:
   n, m – число строк и столбцов матрицы
   a – матрица
   Выходные параметры: нет

```
Procedure VIVOD (const a:matr; n, m:byte);
```

3. Обмен местами двух столбцов матрицы – процедура
   Входные параметры:
   n – число строк матрицы
   a – матрица,

k , l – номера столбцов, которые меняются местами
Выходные параметры:
a – матрица,

Procedure OBMEN (var a:matr; n, k, l:byte);

4. Определение: все ли элементы столбца положительные – функция
Входные параметры:
n – число строк матрицы
a – матрица
k – номер определяемого столбца

Выходные параметры:
w : Boolean = true (все элементы положительные)
= false (не все элементы положительные)

Function POL (const a:matr; n, k:byte):boolean;

5. Поиск номера столбца максимального элемента заданной строки – функция
Входные параметры:
m– число столбцов матрицы
a – матрица
k – номер заданной строки
Выходные параметры:
l – номер столбца с максим. элементом

Function NOM (const a:matr; m, k:byte):byte;

**Основная программа:**

```
procedure TForm1.Button1Click(Sender: TObject);
type
 matr=array[1..10,1..10] of real;
var
   ……
begin
  VVOD(a, n, m);
```

```
  VIVOD(a, n, m);
  w:=false; j:=1;
  while  (j<=m) and not w do
    if POL(a, n, j)
       then  begin k:=j w:=true end
       else j:=j+1;
  if  not w then
        < 'нет столбцов из положительных элементов'>
    else
     begin
        l:=NOM(a ,m ,5);
        OBMEN (a, n ,k, l);
         VIVOD(a, n, m)
      end;
end;
```

## 10.8. Использование модальных окон для ввода нескольких массивов

Модальные окна обычно требуют от пользователя принятия каких-либо решений и выполнения соответствующих действий (нажатия определенных кнопок, например). В момент закрытия окна свойству ModalResult формы присваивается значение, соответствующее действию пользователя, которое можно анализировать после закрытия окна.

Стандартные кнопки BitBtn типа OK, Yes, Cancel автоматически закрывают модальное окно и помещают в ModalResult результат в виде именованных констант mrXXX, имеющих определенное числовое значение. Для кнопки OK – mrOK, для кнопки Yes – mrYes, для кнопки No – mrNo. Их можно анализировать с помощью операторов:

```
Form2.ShowModal;
If  Form2ModalResult = mrXXX …
```

Оператор, следующий за ShowModal, начнет выполняться только после закрытия модального окна.

Нельзя писать процедуру обработки события для кнопки BitBtn и помещать в нее процедуру закрытия окна Close, т.к. в этом случае значение свойства ModalResult не будет соответствовать нажатой кнопке.

Для того, чтобы очистить компоненты для ввода, установить фокус ввода, установить размеры StringGrid можно написать процедуру обработки события onShow для этой формы и процедуру onChange для компонентов SpinEdit.

Процедура ввода матрицы в модальном окне:

```
Procedure VVOD(var a:matr; var n,m:byte);
 var
  i,j:byte;
 begin
  with Form2 do
   begin
     SpinEdit1.Value:=1;
     SpinEdit2.Value:=1;
     for i:=1 to 100 do
      for j:=1 to 100 do
         StringGrid1.Cells[i-1,j-1]:='';
     ShowModal;
     if ModalResult=mrOK then
      begin
       n:=SpinEdit1.Value;
       m:= SpinEdit2.Value;
       for i:=1 to n do
        for j:=1 to m do
         a[i,j]:=StrToFloat(StringGrid1.Cells[j-1,i-1]);
      end;
   end;
 end;
```

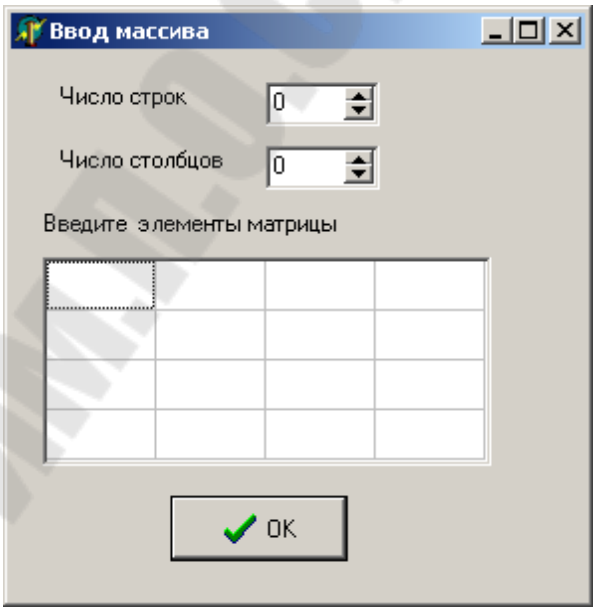



*Рис.10.4.* Вид окна

## Литература

1. Delphi. Программирование на языке высокого уровня: Учебник для вузов / В.В. Фаронов . – СПб.: Питер, 2004. – 640 с.

2. Delphi7: Учебный курс / С.И. Бобровский. - СПб.: Питер, 2004.-735с.

3. Основы программирования в Turbo Delphi / Н.Культин. – СПб.:BHV, 2007 – 384с.

4. Водополова Н.В., Мисюткин В.И., Чабуркина С.А. Основы алгоритмизации. Практическое пособие к лабораторным и контрольным работам по курсам «Информатика» и «Основы информатики и вычислительной техники». – Гомель, 2004, № 2963.

5. Коробейникова Е.В., Токочаков В.И. Работа в интегрированной среде DELPHI. Практическое пособие для студентов всех специальностей дневного и заочного отделений. – Гомель, 2004, № 2910.

6. Коробейникова Е.В., Токочаков В.И. Программирование в среде DELPHI. Практическое пособие по курсу «Информатика» для студентов всех специальностей. – Гомель, 2005, № 2986.

7. Токочаков В.И., Коробейникова Е.В. Практическое пособие по курсу «Информатика» для студентов всех специальностей «Программирование в среде Delphi». – Гомель, 2005, № 3534.

## Содержание

**Чабуркина** София Абелевна
**Емельянченко** Наталья Сергеевна

# ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ НА ЯЗЫКАХ ВЫСОКОГО УРОВНЯ

**Курс лекций**
**по одноименной дисциплине**
**для слушателей специальности 1-40 01 73**
**«Программное обеспечение информационных систем»**
**заочной формы обучения**

Подписано в печать 17.05.13.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 7,67. Уч.-изд. л. 7,23.
Изд. № 9.
http://www.gstu.by

Отпечатано на цифровом дуплекаторе с макета оригинала авторского.
Учреждение образования «Гомельский государственный
технический университет имени П. О. Сухого».
246746, г. Гомель, пр. Октября, 48