КЕМЕРОВСКИЙ ИНСТИТУТ (филиал)
РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ТОРГОВО-ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ
КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Информатика и программирование

Лебедева Т.Ф.

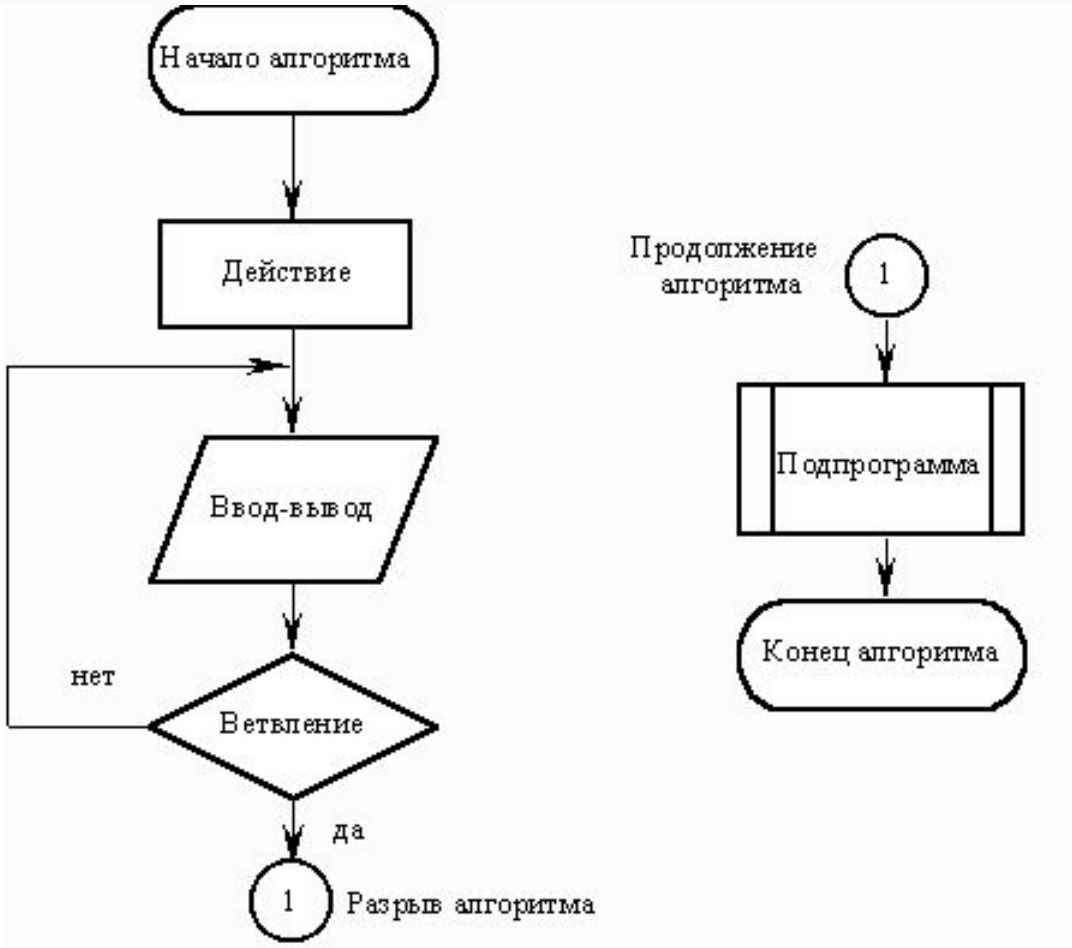
Блок-схема отличается следующим:

- каждому действию соответствует определенный вид фигуры (овал, прямоугольник, параллелограмм, ромб, шестиугольник);
- внутри фигур записываются формулы или краткая инструкция;
- фигуры соединяются линиями со стрелками, которые называются *линиями потока* и указывают направления перехода от одной операции к другой. Причем, если выбирается направление вниз или вправо, то стрелки можно не ставить;
- фигуры или блоки в блок-схемах могут иметь номера, проставляемые слева в разрыве верхней линии;
- линии потока не должны пересекаться, поэтому при необходимости используются *соединители* – элементы с буквой или цифрой внутри.

Конфигурация элементов схем определена

ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем»

пример блок-схемы



В 1977 году математики Бем и Якопини доказали, что алгоритмы сколь угодно сложной структуры могут быть реализованы с использованием всего 3-х управляющих структур:

- Последовательное выполнение операций - ***следование***;
- Ветвление алгоритма на группы операций в зависимости от выполнения некоторых условий **-** ***ветвление***;
- Циклическое многократное выполнение группы операций до выполнения некоторого условия, формируемого в процессе вычислений **-** ***цикл***.

Соответствующие операторы в записи алгоритмов называются *условными операторами* и *операторами циклов* (следование не имеет специального оператора и выражается просто последовательной записью инструкций вычисления, ввода, вывода).

Условные операторы в наших примерах звучат как:

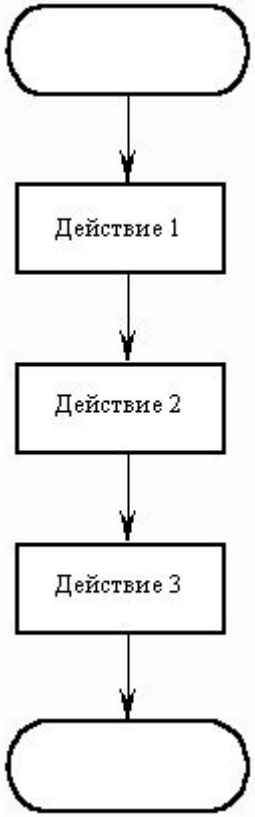
- *если* <условие выполнено> *то* последовательность операций *иначе* другая последовательность операций.

Операторы циклов в описаниях на естественном языке мы формулируем словами:

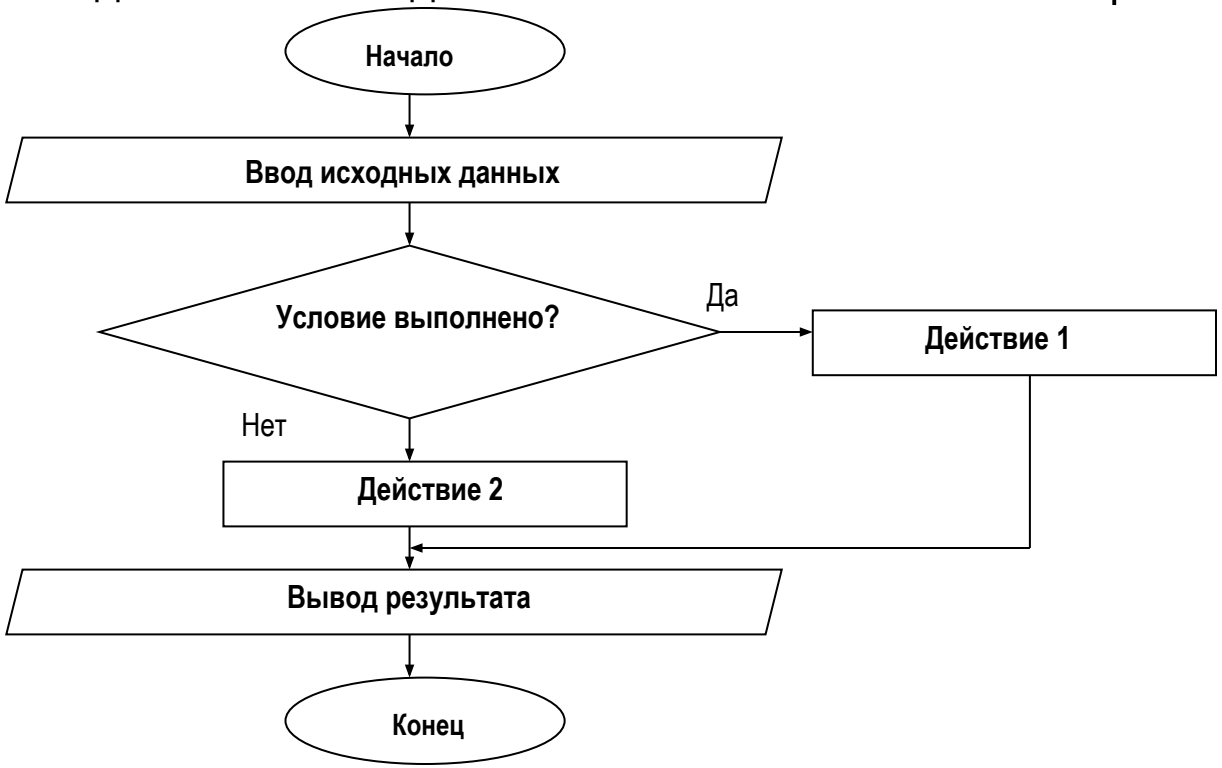
- 1. "Пока истинно некоторое условие - повторять заданные действия" (цикл с предусловием);
- 2. "Повторять заданные действия пока ложно некоторое условие" (цикл с постусловием);
- 3. "Повторять заданные действия N раз" (цикл со счетчиком).

В зависимости от последовательности выполнения действий в алгоритме выделяют алгоритмы линейной, разветвленной и циклической структуры.

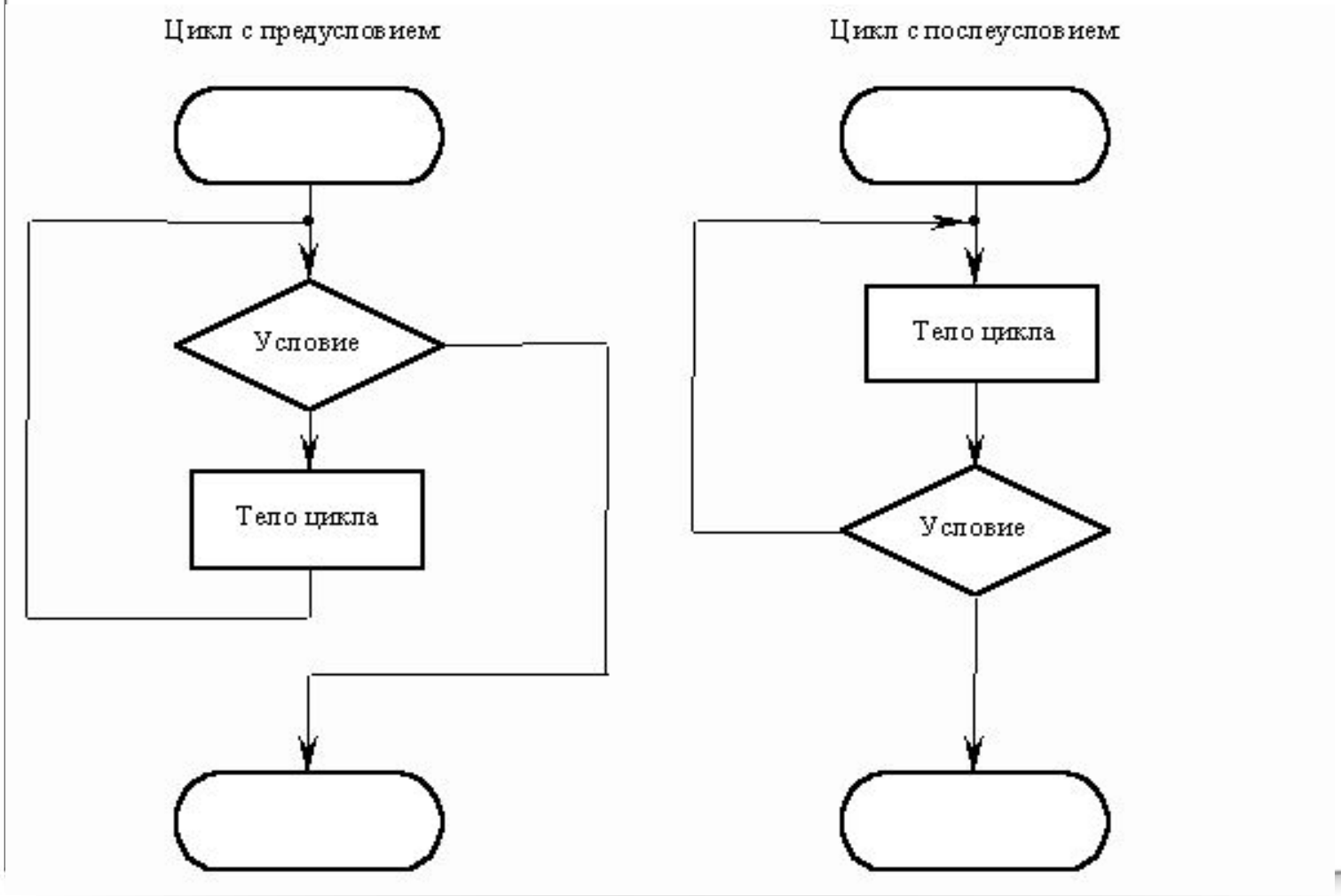
В алгоритмах **линейной структуры** действия выполняются последовательно одно за другим:



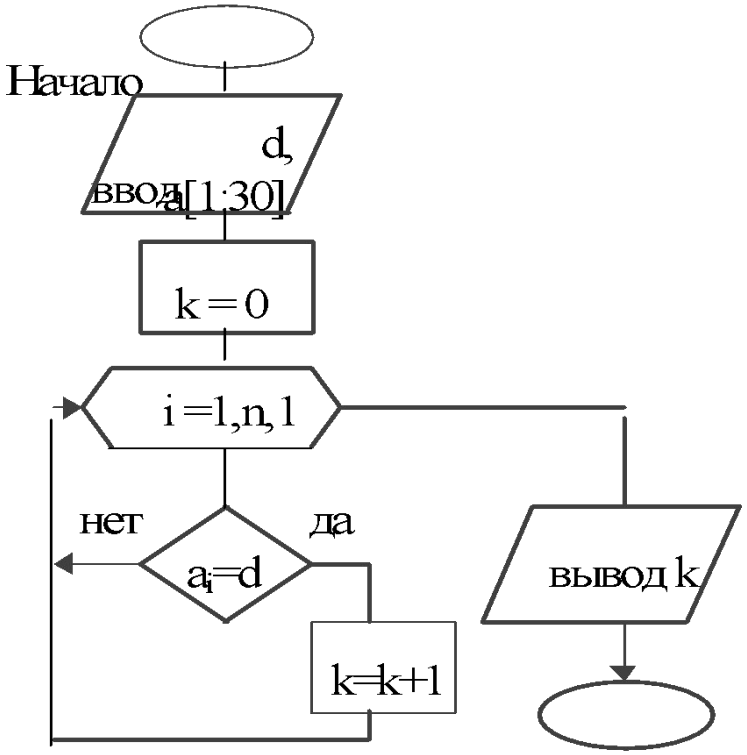
В алгоритмах **разветвленной структуры** в зависимости от выполнения или невыполнения какого-либо условия производятся различные последовательности действий. Каждая такая последовательность действий называется ветвью алгоритма.



В алгоритмах **циклической структуры** в зависимости от выполнения или невыполнения какого-либо условия выполняется повторяющаяся последовательность действий, называющаяся телом цикла.



Цикл со счетчиком



2.4 Понятие алгоритмического языка

Языки программирования (алгоритмические языки) – специально разработанные искусственные языки, предназначенные исключительно для записи алгоритмов, исполнение которых поручается ЭВМ.

Наиболее общей классификацией языков программирования является по степени зависимости от машинного языка:

- **Машинно-зависимые –** машинные, машинно-ориентированные – низкий уровень
- **Машинно-независимые** – процедурные, проблемные – высокий уровень

Машинно-зависимые языки – это языки, наборы операторов и изобразительные средства которых существенно зависят от особенностей ЭВМ (внутреннего языка, адресности, структуры памяти и т.д.).

Машинный язык – это система команд компьютера. Программы, написанные на машинном языке не требуют компиляции**.** Пример фрагмента программы для трехадресной ЭВМ:

код операции	адрес 1-го операнда	адрес 2-го операнда	адрес результата
001	0 0 1 1	0 1 1 0	0 1 0 1

Машинно-ориентированные языки - языки символического кодирования: (Автокод, Ассемблер). Операторы этого языка – это те же машинные команды, но записанные мнемоническими кодами, а в качестве операндов используются не конкретные адреса, а символические имена, например:

```
ADD  1,  B
```

Языки программирования **высокого уровня** машинно-независимы, т.к. они ориентированы не на систему команд той или иной ЭВМ, а на систему операндов, характерных для записи определенного класса алгоритмов. Однако программы, написанные на языках высокого уровня, занимают больше памяти и медленнее выполняются, чем программы на машинных языках.

В **процедурных** языках программа явно описывает действия, которые необходимо выполнить, а результат задается только способом получения его при помощи некоторой процедуры, которая представляет собой определенную последовательность действий. Среди процедурных языков выделяют в свою очередь **структурные** и **операционные языки**.
В структурных языках одним оператором записываются целые алгоритмические структуры: ветвления, циклы и т.д. В операционных языках для этого используются несколько операций. Широко распространены следующие структурные языки: Паскаль, Си, Ада, ПЛ/1. Среди операционных известны Фортран, Бейсик, Фокал.

Проблемно-ориентированные языки предназначены для пользователей-непрограммистов и направлены для решения узкого круга задач. На них определяется *что делать*, а не *как это делать*.

К непроцедурному программированию относятся **функциональные** и **логические** языки

Еще один вид классификации языков программирования определяется существующей методологией программирования:

1) **методология императивного программирования**

языки программирования: FORTRAN (1954), ALGOL (1960) , PASCAL (1972) , C (1974)

2) **методология объектно-ориентированного программирования**

языки программирования: Simula (1962), Smalltalk (1972), C++ (1983), Object Pascal (1984), Java (1995), C# (2000)

3) **методология функционального программирования**

языки программирования: LISP (1958), Refal (1968), Miranda (1985), F# (2008)

4) **методология логического программирования**

языки программирования: PROLOG (1971)

5) **методология программирования в ограничениях**

языки программирования: УТОПИСТ (1980), IDEAL (1981), OPS5 (1987)

Ядра методологий определяются способом описания алгоритма.

2.5 Основные понятия императивного языка программирования

Обычный разговорный язык состоит из четырех основных элементов: символов, слов, словосочетаний и предложений. Алгоритмический язык содержит подобные элементы, только слова называют элементарными конструкциями, словосочетания - выражениями, предложения - операторами. Алгоритмический язык (как и любой другой язык), образуют три его составляющие: алфавит, синтаксис и семантика.

Алфавит – фиксированный для данного языка набор символов (букв, цифр, специальных знаков и т.д.), которые могут быть использованы при написании программы.

Синтаксис - правила построения из символов алфавита специальных конструкций, с помощью которых составляется алгоритм.

Семантика - система правил толкования конструкций языка.

Таким образом, программа составляется с помощью соединения символов алфавита в соответствии с синтаксическими правилами и с учетом правил семантики.

Методология императивного программирования – это исторически первая поддерживаемая аппаратно методология программирования. Она ориентирована на модель фон Неймана. Императивное программирование пригодно для решения задач, в которых последовательное исполнение каких-либо команд является естественным.

Основным синтаксическим понятием является оператор. Первая группа – атомарные или простые операторы, у которых никакая их часть не является самостоятельным оператором, например оператор присваивания, оператор перехода, оператор вызова процедуры.

Вторая группа – структурные (составные) операторы, объединяющие другие операторы в новый, более крупный оператор, например, составной оператор, оператор цикла, операторы выбора.

Для описания синтаксиса языка программирования будет использована расширенная **система обозначений Бэкуса-Наура**. В ней одни понятия определяются через другие последовательно. Основные понятия заключаются в угловые скобки.

Символ **::=** означает «определяется как»

Символ **|** означает «или»

Символ ***** означает «произвольное количество повторений (в том числе ноль раз) того символа, за которым он указан»

Символы, указанные в квадратных скобках, являются необязательными.

<оператор>**::=** <простой оператор> | <структурный оператор>

<простой оператор> **::=** <оператор присваивания>|

<оператор вызова >

<структурный оператор> **::=** <оператор последовательного исполнения>| <оператор ветвления>|<оператор цикла>

<оператор присваивания> **::=** <переменная>:=<выражение>

<оператор вызова > ::= <имя подпрограммы>[(<параметр> *)]

<оператор последовательного исполнения> **::= begin** <оператор> * **end**

<оператор ветвления> **::= if** <логическое выражение> **then** <оператор>* **[else** <оператор> *]

<оператор цикла> **::= while** <логическое выражение> **do** <оператор>

Язык программирования Паскаль. Знакомство со средой программирования Турбо Паскаль.

Паскаль – язык профессионального программирования, который назван в честь французского математика и философа Блеза Паскаля (1623–1662) и разработан в 1968–1971 гг. Никлаусом Виртом. Первоначально был разработан для обучения, но вскоре стал использоваться для разработки программных средств в профессиональном программировании.

Турбо Паскаль – это система программирования, созданная для повышения качества и скорости разработки программ (80-е гг.). Слово Турбо в названии системы программирования – это отражение торговой марки фирмы-разработчика Borland International (США).

Систему программирования Турбо Паскаль называют *интегрированной* (integration – объединение отдельных элементов в единое целое) средой программирования, т.к. она включает в себя редактор, компилятор, отладчик, имеет сервисные возможности.

Основные файлы Турбо Паскаля:

Turbo.exe – исполняемый файл интегрированной среды программирования;

Turbo.hlp – файл, содержащий данные для помощи;

Turbo.tp – файл конфигурации системы;

Turbo.tpl – библиотека стандартных модулей, в которых содержатся встроенные процедуры и функции (SYSTEM, CRT, DOS, PRINTER, GRAPH, TURBO3, GRAPH3).

Структура программы

Программа на Borland Pascal состоит из трех частей:

- заголовка,
- раздела описаний,
- раздела операторов.

Заголовок программы не является обязательным, он состоит из служебного слова program и идентификатора - имени программы.

Р а з д е л о п и с а н и й содержит описания всех используемых *программой ресурсов* (полей данных, подпрограмм и т.д.).

Р а з д е л о п е р а т о р о в заключается в так называемые *операторные скобки* **begin ...end** и заканчивается точкой.

Между операторными скобками записывают управляющие операторы программы, которые разделяют специальным знаком - точкой с запятой «;». Если точка с запятой стоит перед end, то считается, что после точки с запятой стоит «пустой» оператор.

В тексте программы возможны комментарии, которые помещают в фигурные скобки.

Посмотрим, как выглядит Pascal программа, которая реализует алгоритм Евклида для определения наибольшего общего делителя двух чисел a и b:

```
Program example; {заголовок программы}
{раздел описаний}
Var   a,b : integer; {объявление переменных}
{раздел операторов}
Begin
    Write ('Введите два натуральных числа:') ;
 {запрашиваем ввод данных}
    Readln(a,b); {вводим значения}
    while a<>b do   {цикл-пока a ≠b}
      if a>b then a:=a - b {если a>b, тогда a:=a-b} else b:= b - a;
{иначе b:=b-a}
    Writeln('Наибольший общий делитель равен ',a);
 {выводим результат}
End.    {конец программы}
```

- **Программы на языке Паскаль имеют блочную структуру:**

1. Блок типа PROGRAM – имеет имя, состоящее только из латинских букв и цифр. Его присутствие не обязательно, но рекомендуется записывать для быстрого распознавания нужной программы среди других листингов.
2. Блок описаний состоит в общем случае из 7 разделов:
   - раздел описания модулей (**uses**);
   - раздел описания меток (**label**);
   - раздел описания констант (**const**);
   - раздел описания типов данных (t**ype**);
   - раздел описания переменных (**var**);
   - раздел описания процедур и функций;
3. Операторный блок, который содержит раздел описания операторов.

Общая структура программы на языке Паскаль следующая:

```
Program ИМЯ..;      {заголовок программы}
Uses ...;                   {раздел описания модулей}
Var ..;          {раздел объявления переменных}
  ...
Begin            {начало исполнительной части программы}
  ...            {последовательность
  ...             операторов}
End.             {конец программы}
```

Алфавит языка программирования TURBO Pascal 7.0 включает:

1. строчные и прописные буквы латинского алфавита (a..z, A..Z) и знак подчеркивания (_), который также во многих случаях считается буквой;
2. цифры (0...9);
3. специальные знаки, состоящие из одного и более символов:

- знаки арифметических операций : **+ - * / mod div**
- служебные знаки: **= : < > [ ]** $ # **; { } ( ) ‘ @ := (* *) . ,**
- знаки операций отношения: **<= >= = > < <>**
- знаки логических операций: **not and or xor**

- служебные слова (эти сочетания считаются единым целым и их нельзя использовать в программе в другом качестве):

absolute	end	insline	procedure	to
and	external	interface	program	type
array	file	interrupt	public	unit
begin	for	label	record	until
case	forward	mod	repeat	uses
const	fidnction	nil	set	var
div	goto	not	shl	while
do	if	of	shr	with
downto	implementation	or	string	xor
else	in	private	then	

Константы и переменные. Типы переменных

Все данные, с которыми оперирует программа на Pascal, должны быть описаны.

Все объекты программы – *переменные, константы, функции, процедуры, типы, модули* имеют имена, которые называются **идентификаторами**.

Идентификаторы образуются из букв и цифр (к буквам относится и знак подчеркивания). Первый символ – обязательно буква.

Запоминаются первые 63 символа. Регистр символа не имеет значения.

Поэтому *SUMMA, summa, Summa* – это одно и тоже имя.

Данные в программе могут присутствовать в виде констант и переменных.

Константы.

Константы определяются один раз и не изменяются во время выполнения программы. Используют следующие типы констант :

- целые и вещественные десятичные числа, например, 25, 6.12, 0.125e10 (см. примечание);
- шестнадцатеричные числа - должны начинаться со знака «$»,
- например, $64;
- логические константы - **true** (истина) и **false** (ложь);
- символьные константы - записываются либо в апострофах,

Например, 'A', либо в виде соответствующих кодов по таблице ASCII, причем в последнем случае перед кодом ставится знак «#», например #65 (этот код соответствует символу А латинское);

- строки символов - записываются в апострофах, например 'ABCD';
- конструкторы множеств [3,6,8] ;
- «нулевой» адрес - nil

Примечания. 1. В программировании принято при записи вещественных чисел вместо запятой для разделения целой и дробной частей числа использовать точку.

2. Обычно при записи в программе или выполнении операций ввода-вывода вещественные числа записывают в так называемом формате с фиксированной точкой, указывая в начале целую часть числа, а затем, после точки, дробную, например: 0.5, -3.85 . Но иногда бывает удобно задавать числа в формате с плавающей точкой, т.е. в виде мантиссы и порядка. При этом мантиссу записывают перед порядком и отделяют от него строчной или прописной латинской буквой «е»,

например: запись 1.5e-10 соответствует значению 1,5 x 10 ^ (-10), а запись 0.5E23 соответствует значению 0,5x10^23.

Константы используются в трех формах: как литералы, как поименованные константы и как типизированные константы.

- *Литерал* представляет собой значение константы, записанное непосредственно в программе (например, в выражении

2+5.1*x использованы два литерала «2» и «5.1»).

- *Поименованные константы* объявляются в инструкции раздела описаний const. Обращение к ним осуществляется по имени (идентификатору).

Например;

- *Const min=23; max=45;* {десятичные константы}
- *a16=$A0;* {шестнадцатеричная константа}
- *Ch1 = #94; ch2 = ‘a’;* {символьные константы}
- *stroka= ‘end’ ;* {строковая константа}
- *V=[3,6,8..9];* {конструктор множества}

- *типизированные константы* представляют собой переменные, объявленные в разделе Const , которым присвоены начальные значения, например:

Const k : integer = 10;

a : array[1..4] of real=(2.5, -7.96, 7.54, 32.89);

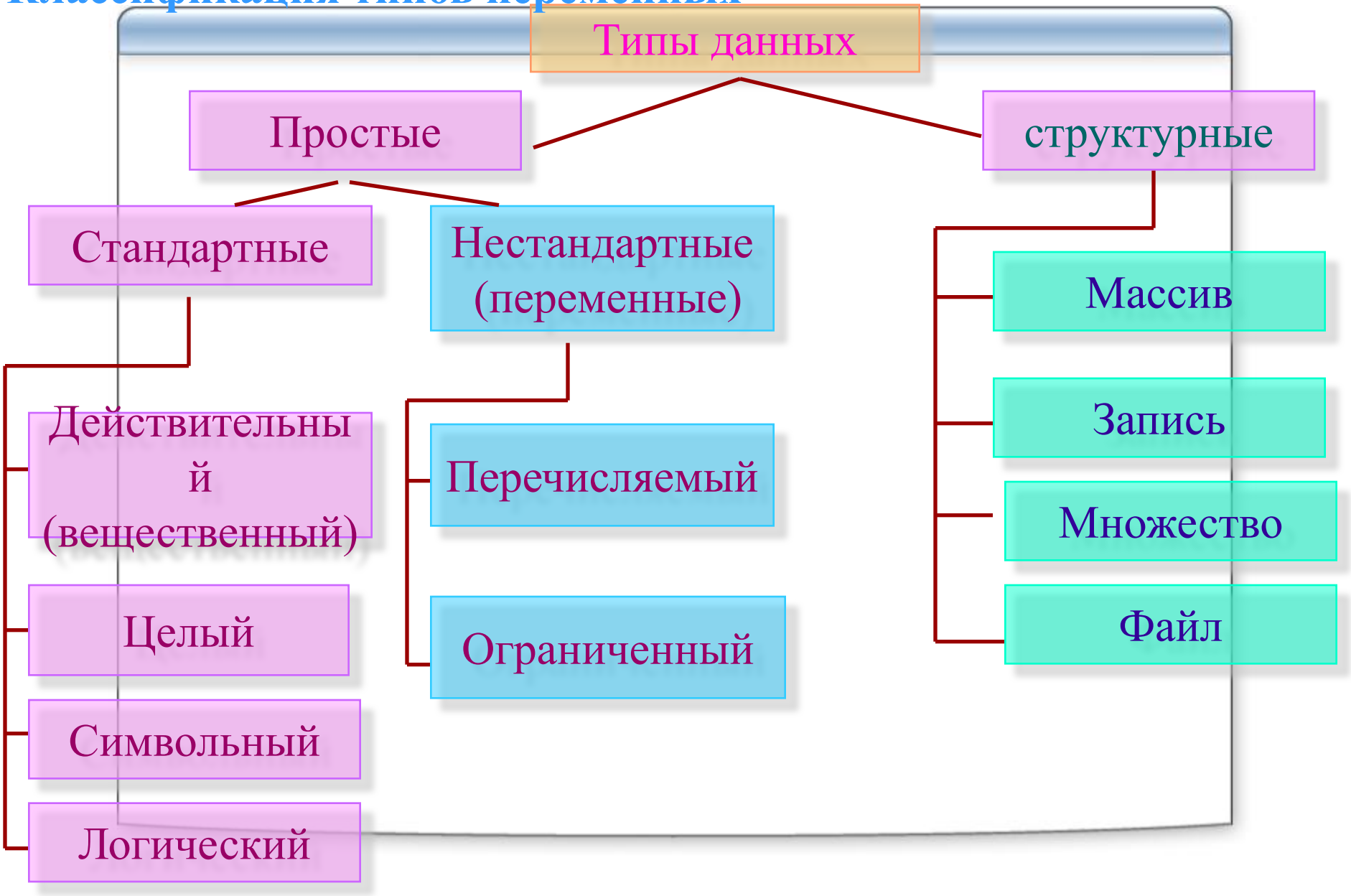
Переменные. Переменные - поименованные объекты программы, которые могут изменяться в процессе выполнения программы.

Их объявление также выполняют в разделе описаний программы, причем при этом указывается не только идентификатор переменной, но и ее тип. Обращение к переменным также осуществляют по идентификатору.

Тип переменной определяет возможный набор значений данной переменной, размер ее внутреннего представления и множество операций, которые могут выполняться над переменной.

На рисунке показана классификация типов

Классификация типов переменных



Простые (скалярные) типы описывают упорядоченные наборы значений. Они делятся на порядковые и вещественные.

Группа **порядковых типов** объединяет типы переменных, набор значений которых конечен, группа *вещественных* типов - типы с условно бесконечным набором значений.

Порядковые типы переменных делятся на стандартные, перечисляемые и диапазоны. *Стандартно* заданы следующие типы:

целые типы (представлены в табл. 1);

булевский тип **Boolean** включает только два значения - false (0) и true (1), но в памяти значения данного типа занимают целый байт. Логический тип данных часто называют булевым по имени английского математика Д. Буля, создателя математической логики. В языке Паскаль имеются две логические константы TRUE и FALSE. Логическая переменная принимает одно из этих значений и имеет тип Boolean. Для сравнения данных предусмотрены следующие операции отношений: <, <=, =, <>, >, >=. А также существуют специфичные для этого типа логические операции OR - или; AND - и; NOT - не. При проверке некоторых условий результат операции может быть истинным или ложным. Например, 3>5 ложь.

символьный тип **Char** Значениями данного типа является множество всех символов по таблице ASCII : русская или латинская большая или маленькая буква, цифра, знак препинания, специальный знак (например, "+", "-", "*", "/", "", "=" и др.) или пробел " ". Каждый из символов имеет уникальный номер от 0 до 255, т. е. внутренний код, который возвращает функция ORD. Символьная константа или символьная переменная - любой символ языка, заключённый в апострофы. Например,

Var Simvol : char;

Нестандартные порядковые типы необходимо описывать при объявлении переменных: *перечисляемый и диапазон*.

Таблица 1 – Целые типы данных

Название	обозначение	Диапазон значений	Длина внутреннего представления, байт
целое	integer	-32768..32767	2 (со знаком)
Короткое целое	shortint	-128..127	1 (со знаком)
Длинное целое	longint	– 2 147 483 648 .. +2 147 483 648	4 (со знаком)
байт	byte	0..255	1 (без знака)
слово	word	0..65535	2 (без знака)

Перечисляемый тип формируется из значений, определенных программистом при объявлении типа. Перечень значений задают, через запятую в круглых скобках, например:

Var D : (Mon, The, Wed, Thu, Fr, Set, Sun); ...{переменная D может принимать только указанные значения}

Примечание. Во внутреннем представлении значения перечисляемого типа кодируются целыми числами, начиная с нуля. Так, идентификатору Mon будет соответствовать O, The – 1 и т. д.

Объявляя переменную перечисляемого типа, можно сначала определить новый тип, а затем уже переменную этого типа, например:

Type Day=(Mon, The, Wed, Thu, Fr, Set, Sun); {объявление нового типа}

Var D: Day;.., {объявление переменной данного типа}

Тип переменной *диапазон* определяется как диапазон значений некоторого уже определенного ранее или стандартного *порядкового* типа. При его описании также можно использовать конструкцию объявления типа, например:

Type Data=1..31; {диапазон одного из целых типов}

Var DataN : Data;..,

или, не описывая тип отдельно, ту же переменную можно объявить следующим образом:

Var DataN :1..31; letter : 'A' .. 'K' ;

Вещественные типы используют для представления чисел, содержащих дробную часть. Во внутреннем представлении мантисса и порядок вещественных чисел хранятся раздельно, причем количество разрядов под мантиссу и порядок регламентируются типом числа. Соответственно *обработка вещественных чисел в компьютерах выполняется с некоторой конечной точностью,* которая зависит от количества двоичных разрядов, отведенных для размещения мантиссы.

Количество разрядов для записи порядка числа определяет диапазон чисел, для представления которых можно использовать разрядную сетку данного типа. В табл. 2 приведены характеристики вещественных типов Turbo Pascal.

Простые типы (стандартные)

Таблица 2 - Вещественные типы данных Real, Single, Double

тип	диапазон значений	требования к памяти в байтах
Real	$\pm 2.9*10^{-39}$ – $\pm 1.7*10^{38}$	6
Single	$\pm 1.5*10^{-45}$ – $\pm 3.4*10^{38}$	4
Double	$\pm 5.0*10^{-324}$ – $\pm 1.7*10^{308}$	8

Компьютер, по умолчанию, представляет действительные числа в виде чисел с плавающей точкой. Такое представление чисел не очень нравится пользователям. Поэтому мы будем “заставлять” компьютер выдавать действительные числа в более привычном варианте следующим образом:

R:m:n, где R – действительное число, m – количество позиций, отводимых для всего числа, n – количество позиций, отводимых для дробной части.

Например, если мы хотим вывести на экран число Chislo с фиксированной точкой, причем знаем, что для вывода этого числа достаточно 7 мест, а вывод дробной части ограничим сотыми, то мы запишем вывод так:

Write (Chislo:7:2);

Строковый тип (string)

Значением строковой величины является строка переменной длины до 255 символов (быть может пустая). Строковая константа или строковая переменная представляет собой произвольную последовательность символов, заключенную в апострофы. Например,

```
Var     Stroka : string[20];
```

Пример программы с описаниями данных различных типов:

```
Program TipDann;
        Uses    Crt
        Var
          Chislo1 : Integer;
          Chislo2 : Real;
          Simvol  : Char;
          Stroka  : String;
          Logika  : Boolean;
        Begin
          ClrScr;
          Chislo1 : =12;
          Chislo2 : =Chislo1*2;
          Chislo2 : =Chislo2/5;
          Simvol : ='d';
          Stroka : ='Строчка';
          Logika : = Chislo1> Chislo2;
          WriteLn ('Вывод значений:');
          WriteLn ('Значение переменной Chislo1 : ',Chislo1);
          WriteLn ('Значение переменной Chislo2 : ',Chislo2:5:2);
          WriteLn ('Значение переменной Simvol : ',Simvol);
          WriteLn ('Значение переменной Stroka  : ',Stroka);
          WriteLn ('Значение переменной Logika  : ',Logika);
        End.
```

Выражения

Все вычисления и другие преобразования данных в программе записываются в виде *выражений.* Обычно выражение включает несколько операций, которые выполняются в порядке их приоритетности. Различают:

- *арифметические* операции: + (сложение), - (вычитание), * (умножение), / (деление вещественное) - эти операции применяют к вещественным и целым числам, результат - также число;

div (деление целочисленное), **mod** (остаток целочисленного деления) - эти операции применяют к целым числам, результат - также целое число;

- операции *отношения:* > (больше), < (меньше), = (равно),<> (не равно), >= (не меньше), <= (не больше) - эти операции применяют к числам, символам, символьным строкам и некоторым другим типам данных, результат - значение логического типа;

- *логические* операции: **and** (и), or (или), **xor** (исключающее или), **not** (не) - эти операции выполняют с логическими переменными и константами, результат - значение логического типа;
- *поразрядные* операции: **and** (и), or (или), **xor** (исключающее или), **not**
- **shr** (сдвиг вправо), **shl** (сдвиг влево) - эти операции выполняют с целыми числами, результат - целое число;
- *строковая* операция: + (сцепление строк) - выполняется над символами и строками, результат - строка;
- операции *над множествами:* + (объединение), - (дополнение), * (пересечение), результат - множество; in (определение принадлежности элемента множеству), результат ~ значение логического типа ;
- операция *над указателями:* @ (определение адреса программного объекта), результат - адрес

Таблица 3 – Приоритеты операций

Операции	Приоритет
@, not	1
*, /, div, mod, and, shr, shl	2
+, -, or, xor	3
>, <, o, =, <=, >=, in	4

Для изменения порядка выполнения операций в выражении используют круглые скобки. В выражениях также допускается использование стандартных и определенных программистом функций. Им присваивается высший приоритет.

Арифметические выражения

Арифметические выражения формируются из констант, переменных, функций, знаков операций и круглых скобок.

Запись выражений, содержащих арифметические операции, выполняется «в строку», порядок выполнения операций определяется скобками. Особенно внимательно следует программировать выражения, включающие операции различных приоритетов. Например:

1) запись ***a+b/c*** предполагает, что вначале выполняется операция деления,

а затем сложения;

2) запись **(a+b)/c*d** предполагает, что сумма a+b делится на c, а затем умножается (!) на d.

При программировании арифметических выражений также следует учитывать

правила выполнения операций, перечисленные ниже.

1. Операции «целочисленное деление» и «определение остатка от деления» применимы только к операндам целых типов, например: **6 div 4 =1**, а

6 mod 4 = 2. Если в операции участвуют переменные, то они должны быть

объявлены как целые.

Для получения при делении целых значений результата с точностью до дробной части необходимо использовать операцию вещественного деления: **6/4=1.5.**

2. При выполнении арифметических операций над числами различных типов выполняется *неявное преобразование типов:*

а) если один операнд целого типа, а другой - вещественного, то переменная целого типа преобразуется к вещественному типу; результат операции - значение вещественного типа;

б) если в качестве операндов использованы вещественные или целые переменные различных типов, то их значения *преобразуются к типу с наибольшей разрядной сеткой*; результат операции того же типа. Так, если в выражении есть переменные **integer** и **longint,** то значения будут преобразованы в тип **longint** и того же типа будет полученный результат.

Порядок выполнения операций при вычислении арифметических выражений можно регулировать при помощи скобок по обычным правилам. Там, где скобки отсутствуют, ЭВМ выполняет операции в следующем порядке:

1) вычисляет значение всех функций, определенных программистом и стандартных функций;
2) выполняет слева направо все операции умножения и деления;
3) выполняет слева направо все операции сложения и вычитания.

I Арифметические функции

1) **Abs(x),** где аргумент и результат являются переменными целого или вещественного типа – вычисляет модуль (абсолютную величину) числа **x;**
2) **cos(x),** где аргумент и результат являются переменными вещественного типа – вычисляет косинус **x;**
3) **Sin(x),** где аргумент и результат являются переменными вещественного типа – вычисляет синус **x;**
4) **Frac(x),** где аргумент и результат являются переменными вещественного типа – выделяет дробную часть числа **x;**
5) **Int(x),** где аргумент и результат являются переменными вещественного типа – выделяет целую часть числа **x;**
6) **Pi,** где результат является переменной вещественного типа – вычисляет значение π**= 3.4157….**
7) **Random(x),** где аргумент и результат являются переменными целого типа – генерирует случайное число в пределах от 0 до x включительно. Если параметр x не задан, то формируется вещественное число от 0 до 1. Перед использованием данной функции нужно инициализировать генератор случайных чисел при помощи процедуры **Randomize (см. ниже);**
8) **Sqr(x),** где аргумент и результат являются переменными целого или вещественного типа – вычисляет $\mathbf{x^2}$**;**
9) **Sqrt(x),** где аргумент и результат являются переменными целого или вещественного типа – вычисляет $\sqrt{}$**x.**
10) **ln(x),** где где аргумент и результат являются переменными вещественного типа – вычисляет логарифм натуральный **x;**
11) **exp(x),** где аргумент и результат являются переменными вещественного типа – вычисляет экспоненту **x** ($\mathbf{e^x}$) ;

II Функции преобразования типов

1) **Chr(x),** где аргумент типа Byte, а результат типа Char– возвращает символ, у которого код в таблице ASCII равен x;
2) **Ord(x),** где аргумент может быть любого порядкового типа, а результат типа LongInt – возвращает порядковый номер значения x при начале нумерации с нуля;
3) **Round(x),** где аргумент вещественного типа, результат типа Longint – округляет число x до ближайшего целого;
4) **Trunc(x),** где аргумент вещественного типа, результат типа Longint – выделяет целую часть числа x.

III Функции для порядковых типов

1) **Odd(x),** где аргумент типа Longint, а результат логического типа – определяет, является ли число четным (результат false) или нечетным (результат true);
2) **Pred(x),** где аргумент и результат любого порядкового типа – получает предшествующее значение;
3) **Succ(x),** где аргумент и результат любого порядкового типа – получает последующее значение;
4) **Upcase(x**), где аргумент и результат типа Char – преобразует букву латинского алфавита в соответствующую ей заглавную (буква x может быть как строчной, так и заглавной).

IV Процедуры для порядковых типов

1) **Dec(x),** где аргумент любого порядкового типа – уменьшает значение переменной x на 1;
2) **Dec(x,n),** где x любого порядкового типа, а n типа LongInt – уменьшает значение переменной x на n;
3) **Inc(x),** где аргумент любого порядкового типа – увеличивает значение переменной x на 1;
4) **Inc(x,n),** где x любого порядкового типа, а n типа LongInt – увеличивает значение переменной x на n;
5) **Randomize** – инициализирует генератор случайных чисел.

Правила применения функций:

чтобы воспользоваться функцией, нужно указать ее в правой части оператора присваивания;

- при обращении к функции необходимо в круглых скобках указать ее аргументы;
- в разделе описания переменных правильно указывайте типы переменных, которые хотите употребить в качестве аргумента или результата функции;
- в одном выражении можно обратиться к нескольким функциям.

Правила применения процедур:

- для выполнения процедуры ее надо вызвать в программе в виде оператора;
- в разделе описания переменных правильно указывайте тип переменной, которую хотите употребить в качестве аргумента процедуры.

Примеры записи формул

Математическая запись	Запись выражения на Pascal
1) $\sin^2 x - \cos\sqrt{x}$	sqr(sin(x)) – cos(sqrt (y))
2) $\operatorname{tg} x - \ln \lvert a^3 - b^{-3} \rvert$	sin(x)/cos(x) – ln(abs(a*a*a – 1/(b*b*b)))
3) $2{,}35 \operatorname{ctg}(\pi+1) - y^4$	2.35*cos (pi+1) / sin (pi+1) – sqr(sqr(y))
4) $e^{x/2} - \operatorname{tg} \frac{n}{x+4}$	exp(x/2) – sin(n/(x+4)) / cos(n/(x+4))
5) $abc - x^a$	a*b*c – exp(a * ln(x))
6) $\sqrt[3]{\frac{x+y}{x}}$	exp(1/3 * ln((x+y)/x))
7) $\frac{\frac{a}{x}+\frac{2b}{5y}}{1-z}$	(a/x + 2*b/(5*y)) / (1-z)
8) $\operatorname{tg}\cos\sqrt[5]{\frac{x}{y}}$	sin(cos(exp(1/5*ln(x/y))))/cos(cos(exp(1/5*ln(x/y))))
9) $(a+b)^{-3x}$	exp((-3*x) * ln(a+b))

Комментарии к примерам

1) Из-за отсутствия операции возведения в степень при записи выражений со степенями рекомендуется:

a) Возведение в целую положительную степень заменяется умножением (пример 2)

b) Возведение в целую отрицательную степень заменяется делением на произведение сомнохителей (пример 2)

c) Для четных положительных степеней использовать функцию sqr (примеры 1,3)

d) Во всех других случаях ($\mathbf{x^a}$) возведение в степень вычисляется как экспонента от показателя степени, умноженного на натуральный логарифм основания (**exp(a * ln(x))** (примеры 5,6,8)

2) Отсутствие функций tg, ctg приводит к усложнению выражений (примеры 2,3,4,8). Если аргумент у функции tg или ctg достаточно сложное выражение (примеры 4,8), рекомендуется ввести новую переменную для обозначения аргумента, например:

```
……..
t := cos(exp(1/5*ln(x/y)));
z := sin(t) /cos(t);
……..
```

Оператор присваивания

a := b;

С помощью оператора присваивания в программе записываются действия, связанные с изменением значений переменных. При выполнении этого оператора вычисляется выражение **b**, приведенное в правой части, и его результат заносится в переменную **a**, имя которой указано слева. Если оператор присваивания записывается в последовательности операторов, то после него ставится точка с запятой.

Например:

а) *Var a,b,c:real;*

Begin ...

*c:=(a*a - sin(b))/(a+25.1);* ...

б) *Var v: boolean; a: integer; b:real;*

Begin a:=8;b:=5.1;

v: =(a>5) and (b> =8); {v получит значение false}...

Для корректного выполнения операции присваивания результат выражения и переменная, записанная в правой части оператора присваивания, должны иметь одинаковые или совместимые типы.

Совместимыми считаются:

- все целые типы;
- все вещественные типы;
- диапазон некоторого базового типа и базовый тип;
- два диапазона одного базового типа;
- символ и строка.

Если тип переменной и выражения несовместимы, то возникает синтаксическая ошибка:

Type mismatch

При несовпадении типов правой и левой частей оператора присваивания для совместимых типов происходит неявное преобразование результата выражения к типу переменной, указанной в правой части. Например:

```
Var
s1, s2 :string; {переменные типа string}
k : char {переменные типа char}
I,j : integer {переменные типа integer}
x,y:real; {переменные типа real}
Begin
.....
X:=sqrt(y);
x:=sqrt(i);
Y:=sin(j);
i:= cos(x);
X:= trunc(y);
i:= I / j;
S1:=s2 + k;
S2:= s1 + x;
I: = j * round(x);
…….
```

Какой из этих операторов присваивания вызовет ошибку несоответствия типов?

Операторы Write и WriteLn

Мы уже использовали операторы Write и WriteLn, но нам необходимо подробнее остановиться на правилах применения этих операторов.

Write (англ. писать) – оператор, который используется для вывода информации на экран. Оператор WriteLn выполняет то же самое действие, но так как у него есть еще окончание Ln (line - англ. линия, строка), то после вывода на экран нужного сообщения, он дополнительно переводит курсор на следующую строчку.

Общий вид:

Write (список выражений);

WriteLn (список выражений);

Процедуры Write и WriteLn используются не только для вывода результата, но и для вывода различных сообщений или запросов. Это позволяет вести диалог с пользователем, сообщать ему, когда ему нужно ввести значения , когда он получает результат, когда он ошибся и др. Оператор WriteLn можно применить и без параметров. В этом случае напечатается строка, состоящая из пробелов, и курсор будет переведен на другую строку. Это иногда нам нужно для лучшего восприятия ввода данных.

Операторы Read и ReadLn

Вспомним, что основное назначение ЭВМ – сэкономить человеческий труд. Поэтому необходимо обеспечить возможность, однажды написав программу, многократно ее использовать, вводя каждый раз другие данные. Такая гибкость в языке обеспечивается операторами Read и ReadLn. Этими операторами вводится информация с клавиатуры.

Общий вид:

Read(переменная, переменная...)

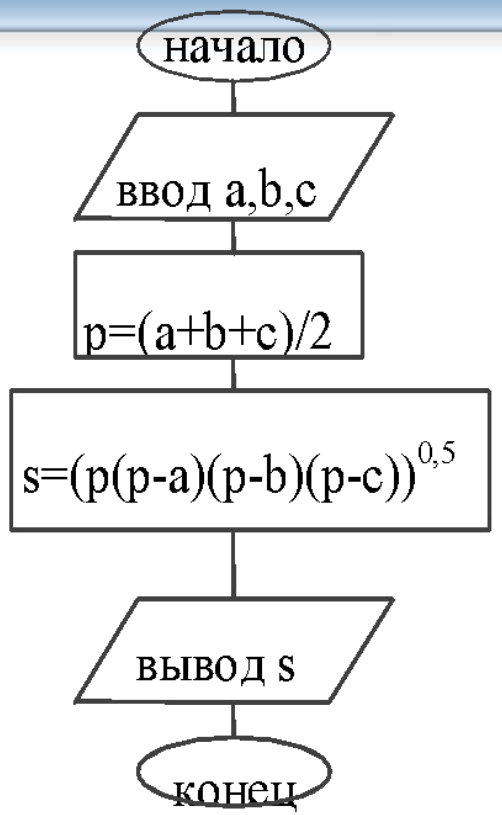
ReadLn(переменная, переменная...)

При выполнении процедуры Read ожидается ввод перечисленных в скобках значений. Вводимые данные нужно отделить друг от друга пробелами. Присваивание значений идет по очереди.

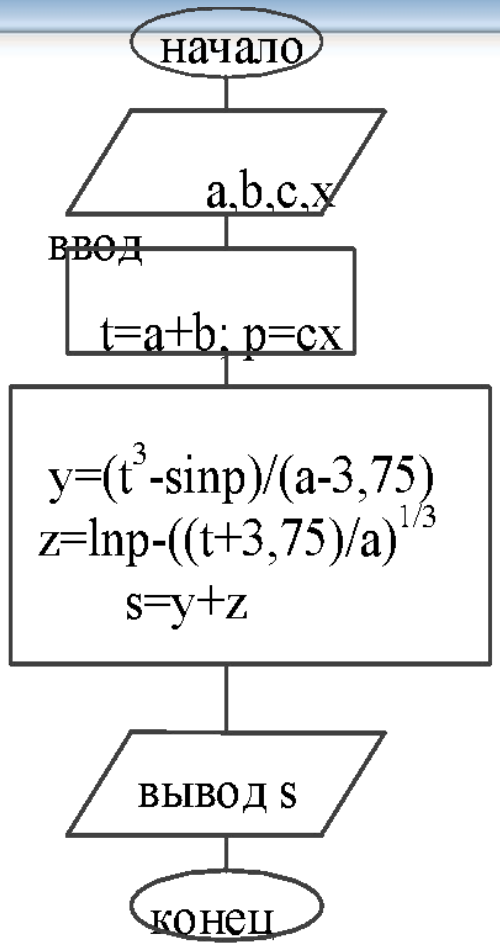
Например, если вводятся значения 53 и X, то при выполнении оператора Read(a, b) переменной a будет присвоено число 53, а переменной X – буква X. Причем, отметим, чтобы не было аварийной ситуации, нужно правильно определить тип данных в разделе Var; в нашем случае a:integer, a b:char.

Особых различий при чтении и записи в использовании операторов Read и ReadLn нет. После выполнения оператора ReadLкурсор переходит на новую строку и следующий оператор чтения считывает данные с новой строки.

Часто процедуру ReadLn без параметров применяют в конце программы для задержки: до нажатия на клавишу <Enter> результат выполнения программы остается на экране. Это очень полезно делать для анализа результатов.



Блок-схема к примеру 1



Блок-схема к примеру 2

```
Программа для примера 1
Program primer_1;
        Uses    Crt;
        Var
             a, b, c, p , s : Real;
 Begin
  ClrScr; {процедура очистки экрана, входящая в модуль CRT}
  write('введите через пробел значения сторон треугольника
        a,b,c,');
  readln(a,b,c );
            p : =(a + b + c) / 2;
              S : = sqrt ((p * (p – a)*(p – b)*(p – c));
             WriteLn ('Значение площади s = ',s :8:2, '  кв.см.');
              readln; {задержка}
        End.
```

```
Программа для примера 2
Program primer_2;
        Uses    Crt;
             Const k=3.75;
        Var
             a,b,c,x,t,p,y,z,s : Real;
          Begin
            ClrScr; {процедура очистки экрана, входящая в модуль CRT}
            write('введите через пробел значения a, b, c, x');
            readln (a, b, c, x );
           t : =a+b;  p : = c * x;
           y : =(t * t* t  - sin (p) / (a – k);
           z: =ln (p) – exp (1/3 * ln ((t + k) / a);
           S : = y + z;
            WriteLn ('Значение переменной y = ',y :8:2);
            readln; {задержка}
        End.
```

Вопросы?