Основы программирования на Паскале

Основные элементы языка

Алфавит (разрешенный к использованию набор символов)) языка Паскаль составляют:

1. Прописные и строчные (воспринимаются одинаково) буквы латинского алфавита: A, B, …, Z, a, в, …, z.
2. Цифры от 0 до 9.
3. Символ подчеркивания " _ ".
4. Специальные символы:

Специальные символы:

- **+** плюс
- **-** минус
- ***** звездочка
- **/** дробная черта
- **>** больше
- **<** меньше
- **=** равно
- **;** точка с запятой
- **#** номер
- **`** апостроф
- **,** запятая
- **.** Точка
- **:** двоеточие
- **[ ]** квадратные скобки
- **{ }** фигурные скобки
- **$** знак денежной единицы
- **( )** круглые скобки
- **^** тильда (стрелка вверх)
- **@** коммерческое а
- пробел

Комбинации специальных символов могут образовывать составные символы:

- **: =** присваивание
- **< >** не равно
- **. .** диапазон значений
- **< =** меньше или равно
- **> =** больше или равно

Слова – это неделимые последовательности символов алфавита, отделенные друг от друга разделителями и несущие определенный смысл.


Слова
зарезервированные слова
идентификаторы
стандартные идентификаторы
идентификаторы пользователя

Зарезервированные слова являются составной частью языка, имеют фиксированное начертание и навсегда определенный смысл.

And	логическое И
array	массив
Begin	начало блока
Case	вариант
Const	константа
Div	деление нацело
Goto	переход на
Do	выполнять
Downto	уменьшить до
Else	иначе
End	конец блока
File	файл
For	для
Function	функция
If	если
Label	метка
Mod	остаток от деления
Not	логическое НЕ
or	логическое ИЛИ

- ***Идентификаторы (имена)*** используются для обозначения программ, переменных и постоянных величин, процедур, функций.

Общие правила написания идентификаторов

- 1. Идентификатор может состоять только из букв, цифр и символа подчеркивания.
- 2. Идентификатор начинается только с буквы или символа подчеркивания (исключение составляют метки, которые могут начинаться с цифры).
- 3. Максимальная длина идентификатора 127 символов, но значимы только первые 63.
- 4. Между двумя идентификаторами должен быть, по крайней мере, один пробел.

- ***Стандартные идентификаторы*** предназначены для обозначения стандартных, т.е. заранее определенных, объектов (констант, процедур и функций), например, **integer, sin, cos, write**.

- ***Идентификаторы пользователя*** применяются для обозначения объектов, определенных самим программистом.
- При их записи следует учитывать, что:
- 1) идентификаторы в программе должны быть уникальными;
- 2) нельзя использовать в качестве идентификаторов пользователя зарезервированные слова и стандартные имена;
- 3) имена для объектов программы надо выбирать так, чтобы они наилучшим образом отражали их значение.

Примеры записи идентификаторов пользователя:

- a, t1, r_756, summa – правильно;
- cos, while, c#, сумма – неправильно.

Константы и переменные

Константы – это элементы данных, значения которых не меняются в процессе выполнения программы. Константы задаются идентификаторами пользователя и описываются в разделе, который начинается зарезервированным словом **const**.

Пример описания констант:

```
const
t=13.4;  max=1000;  eps=0.15E –
5;  myname= ‘Петя Иванов’;
```

- ***Переменные*** – это величины, которые могут менять свои значения в процессе выполнения программы. Каждая переменная принадлежит к определенному типу данных. Имена переменных и их типы объявляются в разделе, который начинается зарезервированным словом **var**.

Пример объявления переменных.

```
var
  a, в: integer;
  summa: real;
```

Для лучшего понимания программы в ней может быть записан произвольный текст – ***комментарий***. Комментарий можно записать в любом месте программы, где разрешен пробел. Текст комментария ограничен символами { } или (* *) и может содержать любые комбинации латинских и русских букв, цифр и других символов алфавита языка Паскаль. Ограничений на длину комментария нет, он может занимать несколько строк.

Примеры:

(*Начало программы*)

{Пример комментария,
занимающего несколько строк}

Типы данных

Тип данных определяет:

1 формат представления данных в памяти компьютера;

2 множество допустимых значений, которые может принимать переменная или константа данного типа;

3 множество допустимых операций, применимых к этому типу.

Типы данных делятся:

стандартный, т.е. какому-либо заранее известному

пользовательский, т.е. определяемому программистом

Целочисленные типы данных

Целочисленные типы данных

Тип	Диапазон значений	Требуемая память (байт)
byte	0 ... 255	1
shortint	-128 ... 127	1
integer	-32768 ... 32767	2
word	0 ... 65535	2
longint	-2147483648 ... 2147483647	4

Над данными целого типа определены следующие операции:

- арифметические операции: + (сложение), - (вычитание), * (умножение), / (деление), div (деление нацело), mod (вычисление остатка от целочисленного деления), - которые вырабатывают результат целого типа, кроме операции деления, вырабатывающей результат вещественного типа;
- операции отношения: = (равно), < > (не равно), < (меньше), > (больше), < = (меньше или равно), > = (больше или равно), - которые вырабатывают результат логического типа.

Вещественные типы данных

Вещественные значения могут изображаться в форме с фиксированной точкой, например, 8.32, -546.271 или 0.017, а также в форме с плавающей точкой, т.е. парой чисел вида <мантисса>E<порядок>, например, 8.53 E+00 (8,53), 6.45721 E+02 (6,45721 · 102), 1.5 E-03 (1,5 · 10-3).

Вещественные типы данных

Тип	Диапазон значений	Мантисса	Требуемая память (байт)
real	$2{,}9 \cdot 10^{-39} \ldots 1{,}7 \cdot 10^{38}$	11 – 12	6
single	$1{,}5 \cdot 10^{-45} \ldots 3{,}4 \cdot 10^{38}$	7 – 8	4
double	$5{,}0 \cdot 10^{-324} \ldots 1{,}7 \cdot 10^{308}$	15 – 16	8
extended	$1{,}9 \cdot 10^{-4951} \ldots 1{,}1 \cdot 10^{4932}$	19 – 20	10
comp	$-2{,}0 \cdot 10^{63}+1 \ldots 2{,}0 \cdot 10^{63}-1$	10 – 20	8

Над данными вещественных типов определены арифметические операции +, -, *, /, а также операции отношения.

Булевский тип boolean

Данные этого типа представлены следующими значениями: ***true*** (истина) и ***false*** (ложь). Над ними определены логические операции ***and*** (логическое И), ***or*** (логическое ИЛИ), ***xor*** (исключающее ИЛИ), ***not*** (логическое НЕ). Для размещения в памяти переменной булевского типа требуется один байт.

Массивы

Регулярный тип, или ***массив***, есть упорядоченный набор данных одинакового типа.

Элементами массива могут быть данные любого типа.

Число элементов массива фиксируется при описании и в процессе выполнения программы не меняется.

Элементы, образующие массив, упорядочены таким образом, что каждому элементу соответствует совокупность ***индексов***, определяющей его местоположение в общей последовательности.

В качестве индексов обычно используют выражения целочисленных типов.

Массивы могут быть описаны в разделе ***var*** с использованием словосочетания ***array of*** (массив из), например:

```
var
 vectorx: array [1..50] of real;
 matrb: array [1..6, 1..6] of byte;
```

Если в такой форме описания массива задан один индекс, массив называется ***одномерным***, если два индекса – ***двумерным***, если n индексов – **n-мерным**.

```
var
  vectorx: array [1..50] of real;
  matrb: array [1..6, 1..6] of byte;
```

Для описания массива можно использовать предварительно определенные константы, например:

```
const
  n1=5; n2=8;
var
  masy: array [1..n1, 1..n2] of
   integer;
```

Массив может быть описан с помощью представления типа в разделе описания типа данных, например:

```
type
  mas = array [1..5, 1..6] of real;
var
  m: mas;
```

Арифметические выражения

Выражение – это конструкция языка, задающая порядок выполнения действий над элементами данных. Выражение состоит из **операндов** (констант и выражений, над которыми производится операция), круглых скобок и знаков операций.

В зависимости от типа результата различают **арифметические** и **логические** выражения.

Арифметическое выражение порождает целое или действительное (вещественное) значение.

В арифметических выражениях, кроме констант, переменных, скобок и знаков арифметических операций, могут использоваться **встроенные (стандартные) функции и процедуры**.:

- **abs (x: real / integer) real / integer** – вычисление |x|. Тип результата совпадает с типом параметра.
- **arctan (x: real): real** – вычисление arctg x (в радианах).
- **cos (x: real): real** – вычисление cos x. Параметр x задает значение угла в радианах.
- **exp (x: real): real** – вычисление ex.
- **frac (x: real): real** – вычисление дробной части x.
- **int (x: real): real** – вычисление целой части x как значение вещественного типа.
- **ln (x: real): real** – вычисление ln x.
- **pi: real** – возвращает значение числа π (3,14159265389793238 5).

- **sin (x: real): real** – вычисление sin x. Параметр x задает значение угла в радианах.
- **sqr (x)** – возведение в квадрат числа x. Тип результата совпадает с типом параметра x.
- **sqrt (x: real): real** – вычисление √x.
- **random: real** – генерирует значение случайного числа из диапазона 0..0,99.
- **random (n: word): word** – генерирует значение случайного числа из диапазона 0..n.
- **round (x: real): longint** – возвращает значение x, округленное до ближайшего целого числа.
- **trunc (x: real): longint** – возвращает ближайшее целое число, меньше или равное x, если x ≥0, и большее или равное x, если x<0.

Приоритет операций:

1. выполняются стандартные функции и процедуры,
2. умножение и деление,
3. сложение и вычитание.

При этом операции одинакового приоритета выполняются слева направо.

Оператор присваивания

Этот оператор обозначается **:=**. При его выполнении вычисляется выражение, стоящее в правой части, и значение выражения присваивается переменной, стоящей в левой части. Тип выражения должен соответствовать типу переменной.

Примеры операторов присваивания (переменные x, y – вещественного типа, m, n, k – целого типа):

```
y : = m*x –3/n;
n : = k*k*k;
```

Организация ввода и вывода

Для ввода данных с клавиатуры используются операторы **read** и **readln**, имеющие следующий формат:

```
read (x1, x2, ..., xn);
readln (x1, x2, ..., xn);
```

После выполнения оператора read курсор остается в этой же строке, а после выполнения оператора readln курсор автоматически переходит в начало следующей строки.

Для вывода используются операторы:

- **write** (y1, y2, .., yn);
- **writeln** (y1, y2, ..,yn);

Логические выражения

Логическое выражение вырабатывает результат логического типа: true и false. Эти выражения записываются с помощью операций отношений и логических операций.

Логические операции (L1 и L2 – логические выражения)

L_1	L_2	L_1 and L_2	L_1 or L_2	L_1 xor L_2	not L_1
true	true	true	true	false	false
true	false	false	true	true	
false	true	false	true	true	true
false	false	false	false	false	

Операции в порядке убывания их приоритета располагаются следующим образом:

Операция	Приоритет
not	первый (высший)
and	второй
or, xor	третий
=, < >, <, >, < =, > =	четвертый (низший)

Оператор условия if

Оператор **if** имеет вид :

if p then a1 else a2;

При выполнении этого оператора сначала вычисляется логическое выражение **p** (условие), в случае истинности которого выполняется оператор **a1**, а в случае ложности – оператор **a2**. Ключевые слова **if, then** и **else** имеют смысл **если**, **то** и **иначе**, соответственно.

Пример использования оператора **if**:

```
program sqroot; {Вычисление действительных  корней квадратного уравнения}
var
   a, b, c: real; {коэффициенты уравнения}
   x1, x2: real; {корни уравнения}
   d: real; {дискриминант}
begin
   writeln (’Введите коэффициенты уравнения:’);
   write (’a =’); readln (a);
   write (’b =’); readln (b);
   write (’c =’); readln (c);
   d: = b * b – 4 * a * c; {вычисление дискриминанта}
   if d > = 0
     then
       begin
          x1: = (-b + sqrt(d)) / (2 * a);
          x2: = (-b – sqrt(d)) / (2 * a);
          writeln (’Корни уравнения:’);
          writeln (’ x1= ’, x1 : 9 : 3, ’ x2= ’, x2 : 9 : 3);
       end
     else
       writeln (’Действительных корней нет.’);
end.
```

Оператор case

Оператор **case** используется для выбора одного из нескольких направлений дальнейшего хода программы. Этот оператор имеет вид:

```
case p of
    a: s1;
    b: s2;
    . .
    n: sn;
    else sn+1;
end;
```

При выполнении оператора **case** сначала вычисляется выражение **p**, называемое ***селектором выбора***. Выражение **p** должно принадлежать типу данных, имеющему конечное число значений (например: **integer**). Затем, в зависимости от полученного значения (если оно равно одной из констант a, b, …, n, которые называются ***константами выбора***), выполняется один из операторов s1, s2, …, sn, помеченный соответствующей константой.

Если значение выражения **p** не совпадает ни с одной из констант выбора, выполняется оператор sn+1, содержащийся после ключевого слова **else**, причем ветвь **else** в операторе **case** необязательна.

Зарезервированные слова **case**, **of**, **else** и **end** имеют смысл **вариант**, **из**, **иначе** и **конец**.

Пример использования оператора **case**:

```
program number; { Определение времени года по номеру
    месяца}
var
   month: integer; {номер месяца}
begin
    write (’Введите номер месяца:’);
    readln (month);
   writeln (‘Время года:’);
    case month of
        1, 2, 12: writeln (’зима’);
        3..5: writeln (’весна’);
        6..8: writeln (’лето’);
        9..11: writeln (’осень’);
        else writeln (’число должно быть от 1 до 12’);
    end;
end.
```

Лекция 3

Оператор перехода goto

Оператор перехода предназначен для передачи управления в другое место программы, т.е. для нарушения естественного порядка выполнения операторов.

Этот оператор имеет вид:

goto p;

Здесь **p** – метка, которой помечен некоторый другой оператор в программе. Зарезервированное слово **goto** имеет смысл **перейти**. Переход осуществляется к оператору, помеченному меткой **p**.

В качестве меток допускается использовать числа (от 1 до 9999) и идентификаторы. Все метки, используемые в программе, должны быть объявлены в разделе описания меток, начинающемся со слова **label**. Каждой меткой должен быть помечен один и только один из операторов в программе.

Пример использования оператора **goto**:

```
program jump;
label 1;
var n: integer;
begin
    read (n);
    if n > 1000 then goto 1 else n: = n +
   100;
    write (n);
1: end.
```

Организация программ циклической структуры

Для многократного повторения одних и тех же действий в Паскале предусмотрены три ***оператора цикла***. Если число повторений цикла известно, то применяется оператор **for**. Если число повторений заранее неизвестно, но известно условие завершения цикла, применяются операторы **repeat** и **while**.

При выполнении оператора **for** сначала вычисляется начальное значение **a**, которое присваивается переменной **i**, называемой ***параметром цикла***. Затем вычисляется конечное значение **b** и проверяется, имеет ли место равенство **i = b.** Если равенства нет, выполняется оператор **s**, который может быть составным, и переменная **i** увеличивается на единицу.

После этого проверка (не равен ли параметр конечному значению), выполнение оператора **s** и увеличение переменной **i** на единицу выполняется циклически до тех пор, пока не наступает равенство **i = b**. Параметр цикла **i**, начальное и конечное значения **a** и **b** могут принадлежать любому порядковому типу данных (например, **integer**). Если начальное значение превышает или равно конечному значению с самого начала, оператор **s** не выполняется ни разу.

Использованные здесь зарезервированные слова **for**, **to** и **do** имеют смысл **от**, **до** и **выполнить,** соответственно.

Возможна другая форма оператора цикла с параметром:

```
for i:= a down to b do s;
```

Здесь, чтобы выполнялся оператор **s,** начальное значение **a** должно ***превышать*** конечное значение **b.** Кроме того, в этом случае параметр **i** с каждым циклом ***уменьшается*** на единицу, пока не станет равным значению **b**.

Оператор цикла **for** имеет такие особенности:

- в теле цикла запрещается явно изменять значение параметра цикла;
- по завершении работы оператора **for** значение параметра цикла считается неопределенным.

Пример использования оператора цикла **for**:

```
program maxi; {программа нахождения наибольшего
    элемента одномерного массива}
var
   x: array [1..100] of real; {исходный массив}
   n: integer; {число элементов массива}
   k: integer; {параметр цикла}
   max: real; {наибольший элемент массива}
begin
   write (’n =’): readln (n);
   for k:= 1 to n do
     begin
       write (’a[’, k:3,’]=’): readln (a[k]);
     end;
   max: = a[1];
   for k:= 2 to n do
     if a[k] > max then max: = a[k];
   writeln (’наибольший элемент =’, max:10:4);
end.
```

Оператор цикла с предусловием while

При его выполнении сначала вычисляется логическое выражение **p** (условие), в случае **истинности** которого выполняется оператор **s** (являющийся, как правило, составным оператором). После этого вычисление условия, его проверка и выполнение оператора **s** повторяется до тех пор, пока выражение **p** не станет равным **false**. Тогда управление передается следующему (после **while**) оператору в программе. Если условие **p** равно **false** с самого начала, оператор **s**, который называется ***телом цикла***, не выполняется ни разу.

- Использованные здесь ключевые слова **while** и **do** имеют смысл **пока** и **выполнить**, соответственно

В качестве примера использования оператора **while** приведем программу вычисления числа **п** по формуле Грегори с точностью 0.5·10-7:

$$\frac{\pi}{4} = \sum_{n=1}^{\infty} (-1)^{n+1} \cdot \frac{1}{2n-1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \square$$

```
program calcpi;
const
   c = 0.5E – 7;
var
   a, sum: real;
   sign: integer;
   n: longint;
begin
   sign: = -1;
   sum: = 1.0;
   a: = 1.0;
   n: = 1;
```

```
    while abs(a) > c do
      begin
         a: = sign / (2 * n – 1);
         sum: = sum + a;
         sign: = - sign;
         n: = n + 1;
      end;
   sum: = 4 * sum;
   write (’pi =’, sum);
end.
```

Оператор цикла с постусловием repeat

Данный оператор имеет вид:

repeat s until p;

Сначала выполняется тело цикла (**s**), затем вычисляется логическое выражение **p** (условие), в случае **ложности** которого вновь выполняется тело цикла. Затем выполнение тела цикла, вычисление условия **p** и его проверка повторяются до тех пор, пока выражение **p** не станет равным **true**. Тогда управление передается следующему (за **repeat**) оператору в программе.

Использованные здесь зарезервированные слова **repeat** и **until** имеют смысл **повторять** и **пока не**, соответственно.

Пример использования оператора цикла **repeat**:

```
Program prost; {проверка, является ли введенное с клавиатуры
                натуральное число простым}
var
  n: integer; {введенное число}
  d: integer; {делитель}
  r: integer; {остаток от деления}
begin
  write ('Введите натуральное число:'); readln (n);
  d:= 2; {сначала будем делить на два}
  repeat
    r:= n mod d;
    if r < > 0 {n не разделилось нацело на d}
      then d:= d + 1;
  until r = 0; {пока не нашли число, на которое делится n}
  if d = n
    then writeln (n,' – простое число.')
    else writeln (n,' – не простое число.');
end.
```

Вложенные циклы

В программах на Паскале возможно использование вложенных циклов. Это подразумевает, что существует внешний цикл и один или несколько внутренних циклов. Каждое повторение внешнего цикла означает завершение всех внутренних циклов; при этом всем выражениям, которые управляют внутренними циклами, вновь присваиваются начальные значения.

Пример использования вложенных циклов:

```
Program summa; {программа вычисления суммы положительных
                                элементов матрицы A(5*8)}
var
   a: array [1..5, 1..8] of real; {исходная матрица}
   i, j: integer; {параметры циклов}
   sum: real; {сумма положительных элементов}
begin
   s:= 0;
   for i:= 1 to 5 do
      begin
         for j:= 1 to 8 do
            begin
               read (a[i, j]);
               if a[i, j] > 0 then s:= s + a[i, j];
            end;
         writeln;
      end;
   writeln ('сумма положительных элементов =', sum:10:6);
end.
```