Условный и сложный операторы . Операторы цикла. Массивы.

Операторы ветвления

К операторам, позволяющим из нескольких возможных вариантов выполнения программы (ветвей) выбрать только один, относятся if и case.

Условный оператор if

Оператор if выбирает между двумя вариантами развития событий:

if <условие> then

<операторы>

[else

<операторы>]

Условный оператор if работает следующим образом:

1. Сначала вычисляется значение <условия> - это может быть любое выражение, возвращающее значение типа **boolean.**
2. Затем, если в результате получена "истина" (true), то выполняется оператор, стоящий после ключевого слова then, а если "ложь" (false) - без дополнительных проверок выполняется оператор, стоящий после ключевого слова else. Если же else-ветвь отсутствует, то не выполняется ничего.

Вложенный оператор If

```
if i>0 then
if s>2 then
s:=1
else
else
 s:=-1;
```

```
if i>0 then
Begin
 if s>2   then
s:=1;
end
else
s:=-1;
```

Оператор выбора case

Оператор case позволяет сделать выбор между несколькими вариантами:

case <переключатель> of
<список_констант> : <один_оператор>;
[<список_констант> : <один_оператор>;]
[<список_констант> : <один_оператор>;]
[else <один_оператор>;]end;

```
Var
operation:Char;
x,y,z:real;
stop:Boolean;
…….
Case operatoin of
‘+’: z:=x+y;
‘-’: z:=x-y;
‘*’: z:=x*y;
‘/’: z:=x/y;
Else
Stop:=true;
End;
```

дополнительные правила, относящиеся к структуре оператора **Case** :

1. Переключатель должен относиться только к порядковому типу данных, но не к типу **longint.**
2. Переключатель может быть переменной или выражением.
3. Список констант может задаваться как явным перечислением, так и интервалом или их объединением.
4. Повторение констант не допускается.

Тип переключателя и типы всех констант должны быть совместимыми.

Пример оператора выбора:

```
case symbol  of
  'a'..'z', 'A'..'Z'   : writeln('Это латинская буква');  'а'..'я',
   'А'..'Я' : writeln('Это русская буква');
                  '0'..'9' : writeln('Это цифра');
  ' ',#10,#13,#26 : writeln('Это пробельный символ');
   else
 writeln('Это служебный символ');
  end;
```

Массивы

Массивы представляют собой ограниченную упорядоченную совокупность однотипных величин. Каждая отдельная величина называется **компонентой массива.**

Тип компонент может быть любым, принятым в языке ПАСКАЛЬ, кроме файлового типа.

Тип компонент называется базовым типом.

Вся совокупность компонент определяется одним именем. Для обозначения отдельных компонент используется конструкция, называемая **переменной с индексом или с индексами:**

A[5] S[k+1] B[3,5].

Описание массива

Для того чтобы задать массив, необходимо в разделе описания переменных **(var)** указать его размеры и тип его компонент.

Общий вид описания (одномерного) массива:

array[<левая_граница>..<правая_граница>] of <тип_компонент>;

Например, одномерный (линейный) массив, состоящий не более чем из 10 целых чисел, можно описать следующим образом:

var a1: array [1..10] of integer;

Нумерация компонент массива

Не обязана начинаться с 1 или с 0 - можно описывать массив, пронумерованный любыми целыми числами. Необходимо лишь, чтобы номер последней компоненты был больше, чем номер первой:

var a1: array [-5..4] of integer;

Нумеровать компоненты массива можно не только целыми числами. Любой порядковый тип данных (перечислимый, интервальный, символьный, логический, а также произвольный тип, созданный на их основе) имеет право выступать в роли нумератора.

допустимы следующие описания массивов:

type

charrr = 'a','c'..'z'; (- отсутствует символ "b")

a2: array [charrr] of integer; - 25 целых компонент

a3: array [shortint] of real; - 256 вещественных компонент

Общий размер массива не должен превосходить 65 520 байт.

Нельзя задать массив

a4:array[integer]of byte;

поскольку тип **integer** покрывает 65 535 различных элементов.

Тем более нельзя использовать тип **longint.**

Тип компонент массива может быть любым:

```
var
a4: array[10..20] of real; - массив из компонент простого типа
a5: array[0..100] of record1; - массив из записей
a6: array[-10..10] of ^string; - массив из указателей на строки
a7: array[-1..1] of file; - массив из имен файловых переменных
a8: array[1..100] of array[1..100] of char; - двумерный массив (массив векторов)
```

Для краткости и удобства многомерные массивы можно описывать и более простым способом:

var

a9: array[1..10,1..20] of real; - двумерный массив 10 x 20

a10: array[boolean, -1..1,char, -10..10] of word; - четырехмерный массив 2 x 3 x 256 x 21

Общее ограничение на размер массива - не более 65 520 байт - сохраняется и для многомерных массивов.

Количество компонент многомерного массива вычисляется как произведение всех его "измерений".

Таким образом, в массиве a9

a9: array[1..10,1..20] of real; - двумерный массив 10 x 20

содержится 200 компонент, а в массиве a10 - 32 256 компонент.

a10: array[boolean, -1..1,char, -10..10] of word; - четырехмерный массив 2 x 3 x 256 x 21

Для ввода или вывода массива в список ввода или вывода помещается переменная с индексом, а операторы ввода или вывода выполняются в цикле. Первый индекс определяет номер строки, второй - номер столбца. Двумерные массивы хранятся в памяти ЭВМ по строкам. Инициализация массивов (присвоение начальных значений всем компонентам массивов) осуществляется двумя способами.

Первый способ - с использованием типизированных констант, например:

```
type
Dim10= Array[1..10] of Real;
 const
M10: Dim10 = ( 0, 2.1, 4, 5.65, 6.1, 6.7, 7.2, 8,
 8.7, 9.3 );
```

При инициализации двумерных массивов значения компонент каждого из входящих в него одномерных массивов записывается в скобках:

```
type
 Dim3x2= Array[1..3,1..2] of Integer;
const
 M3x2: Dim3x2= ( (1, 2), (3, 4), (5, 6) );
```

Второй способ инициализации - использование разновидности процедуры **FillChar:**

```
FillChar( var V; NBytes: Word; B: Byte );
```

Эта процедура заполняет участок памяти однобайтовым значением. Например, для обнуления массива

A[1..10] of Real можно записать:

```
FillChar(A, 40, 0);   или   FillChar(A, SizeOf(A), 0);
```

Обращение к компонентам массива

Массивы относятся к структурам прямого доступа. Это означает, что возможно напрямую (не перебирая предварительно все предшествующие компоненты) обратиться к любой интересующей нас компоненте массива.

Доступ к компонентам линейного массива осуществляется так:

<имя_массива>[<индекс_компоненты>] а многомерного - так:

<имя_массива>[<индекс>,_,<индекс>]

Примеры использования компонент массива:

```
a2['z']:= a2['z']+1;
a3[-10]:= 2.5;
a3[i+j]:= a9[i,j];
```

Примеры задания массивов типизированными константами:

```
type
mass = array[1..3,1..2] of byte;
const
a: array[-1..1] of byte = (0,0,0); {линейный}
b: mass = ((1,2),(3,4),(5,6)); {двумерный}
s: array[0..9] of char = '0123456789';
```

Операторы циклов

Для того чтобы обработать несколько однотипных элементов, совершить несколько одинаковых действий и т.п., разумно воспользоваться оператором цикла - любым из четырех, который наилучшим образом подходит к поставленной задаче.

Оператор цикла повторяет некоторую последовательность операторов заданное число раз, которое может быть определено и динамически - уже во время работы программы.

Замечание: Алгоритмы, построенные только с использованием циклов, называются итеративными - от слова итерация, которое обозначает повторяемую последовательность действий.

for-to и for-downto

В случае когда количество однотипных действий заранее известно (например, необходимо обработать все компоненты массива), стоит отдать предпочтение циклу с параметром (for).

Инкрементный цикл с параметром

for i:= first to last do <оператор>;

Цикл for-to работает следующим образом:

1. вычисляется значение верхней границы **last;**
2. переменной **i** присваивается значение нижней границы **first;**
3. производится проверка того, что **i<=last;**
4. если это так, то выполняется <оператор>;
5. значение переменной **i** увеличивается на единицу;
6. пункты 3-5, составляющие одну итерацию цикла, выполняются до тех пор, пока **i** не станет строго больше, чем **last**; как только это произошло, выполнение цикла прекращается, а управление передается следующему за ним оператору.

какое количество раз отработает цикл for-to в каждом из трех случаев:

- first < last: цикл будет работать last-first+1 раз;
- first = last: цикл отработает ровно один раз;
- first > last: цикл вообще не будет работать.

Декрементный цикл с параметром

аналогичный вариант цикла **for**, который позволяет производить обработку не от меньшего к большему, а в противоположном направлении:

for i:= first downto last do <оператор>;

Цикл **for-downto** работает следующим образом:

1. переменной **i** присваивается значение **first;**
2. производится проверка того, что **i>=last;**
3. если это так, то выполняется <оператор>;
4. значение переменной **i** уменьшается на единицу;
5. пункты 1-3 выполняются до тех пор, пока **i** не станет меньше, чем **last;** как только это произошло, выполнение цикла прекращается, а управление передается следующему за ним оператору.

Если при этом

- first < last, то цикл вообще не будет работать;
- first = last, то цикл отработает один раз;
- first > last, то цикл будет работать first-last+1 раз.

while и repeat-until

Если заранее неизвестно, сколько раз необходимо выполнить тело цикла, то удобнее всего пользоваться циклом с предусловием (while) или циклом с постусловием (repeat-until).

Общий вид этих операторов:

while <условие_1> do <оператор>;

repeat <операторы> until <условие_2>;

Условие окончания цикла может быть выражено переменной, константой или выражением, имеющим логический тип.

Замечание: на каждой итерации циклы for и while выполняют только по одному оператору (либо группу операторов, заключенную в операторные скобки begin-end и потому воспринимаемую как единый составной оператор). В отличие от них, цикл repeat-until позволяет выполнить сразу несколько операторов: ключевые слова repeat и until сами служат операторными скобками.

Так же, как циклы for-to и for-downto, циклы while и repeat-until можно назвать в некотором смысле противоположными друг другу.

Последовательности действий при выполнении этих циклов таковы:

Для while:

1. Проверяется истино ли <условие_1>.
2. Если это так, то выполняется <оператор>.
3. Пункты 1 и 2 выполняются до тех пор, пока <условие_1> не станет ложным.

Для repeat-until:

1. Выполняются <операторы>.
2. Проверяется, ложно ли <условие_2>
3. Пункты 1 и 2 выполняются до тех пор, пока <условие_2> не станет истинным.

Таким образом, если <условие_1> изначально ложно, то цикл **while** не выполнится ни разу. Если же <условие_2> изначально истинно, то цикл **repeat-until** выполнится один раз.

break и continue

Существует возможность прервать выполнение цикла (или одной его итерации), не дождавшись конца его (или ее) работы.

break прерывает работу всего цикла и передает управление на следующий за ним оператор.

continue прерывает работу текущей итерации цикла и передает управление следующей итерации (цикл **repeat-until**) или на предшествующую ей проверку (циклы **for-to, for-downto, while).**

Замечание: При прерывании работы циклов for-to и for-downto с помощью функции break переменная цикла (счетчик) сохраняет свое текущее значение, не "портится".

Оператор безусловного перехода goto

при всей его нежелательности все-таки существует ситуация, когда предпочтительно использовать именно его. Эта ситуация - необходимость передачи управления изнутри нескольких вложенных циклов на самый верхний уровень.

Дело в том, что процедуры break и continue прерывают только один цикл - тот, в теле которого они содержатся. Поэтому в упомянутой выше ситуации пришлось бы заметно усложнить текст программы, вводя много дополнительных прерываний. А один оператор goto способен заменить их все.

Вывод массива, удобный для пользователя

Задача. Распечатать двумерный массив размерности MxN удобным для пользователя способом. (Известно, что массив содержит только целые числа из промежутка [0..100].)

Алгоритм. нужно вывести массив построчно, отражая его структуру.

Реализация

```
for i:= 1 to n do
begin
for j:= 1 to m do
write(a[i,j]:4);
 writeln;
end;
```

Как с помощью цикла можно ввести значения элементов с клавиатуры:

```
For I:=1 to n do
Begin
        writeln('Введите значение элемента ',I' );
        readln(a[i]);
End;
```

Как присвоить значения элементам массива или как заполнить массив с помощью датчика случайных чисел:

```
For I:=1 to n do
Begin
        a[i]:=random(21)-10;
End;
```

Массив будет заполнен целыми числами от –10 до 10

Как подсчитать сумму значений его элементов:

```
S:=0
For I:=1 to n do
begin
S:=s+a[i];
Writeln('сумма элементов массива =  ',s);
Readln;
end;
```