**Операторы Turbo Pascal 7**

**Содержание**

1. Операторы ввода и вывода информации.

2. Операторы языка ТР.7.

2.1 Составной оператор Begin ... end.

2.2 Условный оператор If.

2.3 Оператор безусловного перехода Goto.

2.4 Оператор выбора Case.

3. Операторы циклических процессов.

3.1 Оператор цикла с параметром FOR.

3.2 Оператор цикла с предварительным условием While.

3.3 Оператор цикла с последующим условием Repeat.

**1. Операторы ввода и вывода информации**

Для того, чтобы ввести какие-либо данные в память машины, используют операторы ввода **READ** или **READLN.**

При использовании **READ** после ввода информации курсор останется мигать на той же строке, что и вводимый параметр. При использовании **READLN** курсор перейдет на новую строку.

*Read (x);*

*Readln (x);*

В некоторых случаях с помощью первого оператора вводят несколько переменных:

*Read (x, y, z)*

Чтобы вывести на экран дисплея результат расчета или какую-либо другую информацию (комментарий, текст…), используют операторы вывода **WRITE** или **WRITELN:**

*Write (x);*

*Writeln (y);*

*Writeln (`при х = `, x, `\_ \_ y = `, y);*

Удобно сочетать как текстовую информацию, так и подсчитанную цифровую. При этом текстовая информация заключается в апострофы, а выводимые элементы разделяются запятыми.

Если с помощь **WRITE** или **WRITELN** на экран выводятся дробные числа, то машина представляет их в неудобном для восприятия экспоненциальном виде:

*16,23 => 1.6230000000Е + 01*

Чтобы выводить на экран числа в привычном виде, в операторе вывода у выводимых дробных параметров записывают их формат, т.е. количество позиций, отводимых под дробную часть выводимого числа. Эти параметры отделяются друг от друга двоеточием:

*Y = 16,23*

*Writeln (y: 5: 2)*

*(5 – общее количество позиций под число, 2 – количество знаков после запятой)*

Формат можно указывать и при выводе на экран целых чисел:

*Writeln (x : 4)*

Если выводимое целое число занимает меньшее количество позиций, чем под него отведено, то машина сдвинет это число от левой части экрана до количества незаполненных позиций.

**2. Операторы языка ТР.7**

В ТР.7 в программах используют следующие операторы:

**2.1 Составной оператор Begin ... end**

Применяется в программе в тех случаях, когда те или иные конструкции программы предусматривают выполнение в них только одного оператора, а нам в данной программе нужно выполнить несколько операторов. В этом случае эти операторы заключают в «операторные скобки» **BEGIN…END**, и программа думает, что она имеет дело с одним составным оператором.

*Begin*

*Оператор 1*

*Оператор 2*

*- - - - - - - - -*

*оператор N*

*end;*

Исполнительная часть программы представляет собой большой составной оператор, т.к. также заключается в «операторные скобки» **BEGIN…END.**

После слова **BEGIN** и перед словом **END** точку в запятой ставить не обязательно, т.к. они являются своеобразными знаками препинания.

**2.2 Условный оператор If .**

Условный оператор If реализует в программе алгоритмическую конструкцию «развилка».

***Схема линейного алгоритма***

Исходные данные

Вычисление **y = f (x)**

Вывод результатов расчета

***Схема алгоритма «развилка»***

Исходные данные

Проверка условия

Вычисление **y1**

Вычисление **y2**

Вывод результатов расчета

Различают 2 варианта оператора **IF:**

* 1. ***«Полная развилка»*** в общем виде выглядит так:

|  |
| --- |
| IF <условие> THEN <оператор 1>  ELSE < оператор 2> ; |

**Пример:**

*If (х >= 0) then у: = sqrt (х)*

*else у: = abs (х);*

* 1. ***«Короткая развилка»***

|  |
| --- |
| IF <условие> THEN <оператор 1> ; |

*If (х >= 0) then у: = sqrt (х);*

В этих конструкциях условие – проверяемое логическое условие.

В каждой ветви оператора **IF** по умолчанию может выполняться только один оператор. Если же нам необходимо в каждой ветви выполнить по нескольку операторов, то их заключают в «операторные скобки» **BEGIN…END**:

|  |
| --- |
| IF <условие> THEN <оператор 1>  ELSE BEGIN  < оператор 2>;  < оператор 3>;  < оператор 4>;  END; |

Часто в программах приходится вычислять по нескольку функций в зависимости от истинности или ложности проверяемых логических условий. В этих случаях в программах применяют несколько операторов **IF –** либо несколько «полных развилок», либо комбинацию из «коротких» и «полных развилок»:

|  |
| --- |
| IF <Y 1> THEN <оператор 1>  ELSE < оператор 2> ;  IF <Y 2> THEN <оператор 3>  ELSE < оператор 4> ; |

|  |
| --- |
| IF <Y 1> THEN <оператор 1>  IF <Y 2> THEN <оператор 2>  IF <Y 3> THEN <оператор 3>  IF <Y 4> THEN <оператор 4> |

С увеличением количества проверяемых условий программа становится более громоздкой и в ее структуре могут возникать логические несоответствия.

Для того, чтобы сократить количество проверяемых логических условий и правильнее записывать алгоритм программы, необходимо применять так называемые «вложенные» конструкции оператора **IF.**

В общем виде:

|  |
| --- |
| If <Y 1> THEN <оператор 1>  ELSE IF < Y 2> then < оператор 2>  ELSE IF < Y 3> THEN < оператор 3>  ELSE IF < Y 4> THEN < оператор 4> ; |

**Пример:**

***Составить программу для вычисления функции Y с помощью коротких развилок и с помощью вложенных операторов:***

|  |  |  |
| --- | --- | --- |
|  | { |  |
|  | х |
| Y = | х + 5 |
|  | х 2 |
|  |  |
|  |  |  |
| I. | | | |
| Program Y\_\_X;  Uses Crt;  Var X, Y: Real;  Begin  Clrscr;  Write (`Введите Х: \_`);  Readln (x);  If (X < 0) then Y:= sqrt (abs (x));  If (X < 0) then Y:= х + 5;  If (X < 0) then Y: = х \* х;  Writeln (`При х =`, x:4:2, `\_ \_ \_ Y= `), y:4:3;  Readkey;  End. | | | |

|  |
| --- |
| II. |
| Program Y\_\_X;  Uses Crt;  Var X, Y: Real;  Begin  Clrscr;  Write (`Введите Х: \_`);  Readln (x);  If (X < 0) then Y:= sqrt (abs (x))  else if (Х = 0) then y:= x + 5  else y: = x \* x;  Writeln (`При х =`, x:4:2, `\_ \_ \_ Y= `), y:4:3;  Readkey;  End. |

При использовании в программах **IF** двойных условий применяют для связи этих условий между собой логические операторы **AND, NOT, ОR.**

Чтобы логическое условие записывалось правильно, необходимо каждую половину двойного условия заключать в скобки:

*0 х 5*

*if (x >=0) and (x <= 5) then*

**2.3 Оператор безусловного перехода Goto**

Оператор безусловного перехода Gotoпозволяет изменить ход выполнения программы, начиная с какого-либо оператора, который помечен в программе специальной меткой.

Все метки в программе должны быть объявлены заранее в разделе меток **LABEL.**

В разделе **LABEL** метки перечисляются через запятую:

*Label 1, а, W, c1…;*

В качестве меток можно использовать:

* обычные имена, задаваемые программистом
* числа от 0 до 9999.

Оператор, который помечается меткой, отделяется от нее двоеточием:

*1: Writeln (x);*

*- - - - - -*

*goto 1;*

Ссылка на помеченный оператор осуществляется оператором **GOTO** с указаниемимени метки.

|  |
| --- |
| Пример: Нахождение частного от деления X и Y, которое бы осуществило страховку программы от деления на 0.  Program Delenie;  Uses Crt;  Label 1;  Var X, Y, Z: Real;  Begin  Clrscr;  Write (`Ввод Х: \_`);  Readln (x);  1: Write (`Ввод Y: \_`);  Readln (y);  If (Y = 0) then begin  Writeln (`Делить на 0 нельзя!`);  Goto 1  End  else Z: = x / y;  Writeln (`Z =`, `Z:4:`1 `);  Readkey;  End. |

**2.4 Оператор выбора Case**

Для реализации в программах помимо условного оператора **IF** используется оператор **СASE.**

В общем виде:

|  |
| --- |
| СASE <селектор> OF  <Вариант 1> : <оператор 1>;  <Вариант 2> : <оператор 2>;  - - - - - - -  <Вариант N - 1> : <оператор N - 1>;  ELSE <оператор N >;  END; |

*где селектор – выражение порядкового типа (например,* ***INTEGER, CHAR*** *), значение которого вычисляется в программе заранее, вариант 1, 2, …, N – возможные значения селектора.*

В каждой ветви оператора **CASE** на месте любого варианта может стоять либо одно значение селектора, либо несколько таких значений через запятую, либо значения селектора в виде отрезка.

В каждой ветви оператора **CASE** по умолчанию может выполняться только 1 оператор. Если же нам необходимо выполнить несколько операторов, то их заключают в «операторные скобки» **BEGIN…END.**

В любом случае в операторе **CASE** должны быть перечислены все возможные значения селектора либо их указания в явном виде в вариантах от 1 до N – 1, либо отнесением к пункту **ЕLSE** в неявном виде.

*Case X of*

*1: Y: = X + 1;*

*2, 3, 4: Y : X + 2;*

*5 .. 10: Y: = X + 3;*

*else Y: = 0;*

*end;*

|  |
| --- |
| Пример:  Проверка чисел от 1 до 10 на четность  Program Chet;  Uses Crt;  Var X: Integer;  Begin  Clrscr;  Write (`Введите Х от 1 до 10`);  Readln (x);  Case X of  1, 3, 7, 9: Writeln (`Это нечетное число`);  2, 4, 6, 8, 10: Writeln (`Это четное число`);  else Writeln (`Ввод неверен!`);  end;  Readkey;  End. |

**3. Операторы циклических процессов**

Во многих программах приходится многократно повторять некоторые их участки. Такие повторы реализуются с помощью циклических процессов (циклов)

**3.1 Оператор цикла с параметром FOR**

Позволяет многократно выполнять в программе какой-либо оператор заранее известное число раз.

Существует 2 разновидности цикла **FOR**:

1. **Цикл FOR с увеличивающимся параметром:**

|  |
| --- |
| FOR <i>: = <st> TO <F> DO <оператор> |

1. **Цикл FOR с уменьшающимся параметром:**

|  |
| --- |
| FOR <i>: = <F> DOWNTOWN <St> DO <оператор> |

*I – индекс или параметр цикла, представляющий собой выражение порядкового цикла (Integer, Char).*

*St и F – соответственно, начальное и конечное значение параметра цикла.*

*For i: = 1 to Y: = x + i*

*For i: = 5 down to 0 do writeln (`ЗВФ`);*

Если в цикле **FOR** необходимо выполнить несколько операторов, то их заключают в «операторные скобки» **BEGIN…END.**

В цикле **FOR** нельзя задавать шаг, отличный от 1 *(т.к. программа по умолчанию в этом цикле изменяет индекс цикла на 1)*

|  |
| --- |
| Пример:  Составить программу для нахождения среднего арифметического размера у деталей в партии.  Program Sr\_\_rasmer;  Uses Crt;  Var i, N: Integer;  S, r, Sr: Real;  Begin  Clrscr;  Write (`Количество деталей в партии: \_ \_ `);  Readln (N);  S: = 0;  For i: = 1 to N do  Begin  Write (`Введите размер `, i, `-той детали: \_ \_ `);  Readln (r);  S: = S + r;  End;  Sr: = S/N;  Writeln (`Sr = `, Sr:5:3);  Readkey;End. |

**3.2 Оператор цикла с предварительным условием While**

Позволяет многократно выполнять в программе какой-либо один оператор заранее неизвестное число раз.

Структура цикла **WHILE:**

|  |
| --- |
| WHILE< условие> DO <оператор>; |

Условие – логическое условие, истинность которого проверяется перед каждым заходом на цикл. Если проверяемое условие верно, то цикл **WHILE** выполняется вновь. Как только условие становится ложным, мы выходим из цикла. Если условие никогда не становится ложным, то программа зацикливается.

|  |
| --- |
| Для выхода из зацикливания необходимо нажать комбинацию клавиш:  Ctrl + Break |

Если в цикле **WHILE** необходимо выполнить несколько операторов, то их заключают в «операторные скобки» **BEGIN…END.**

|  |
| --- |
| Пример:  Модернизировать программу по расчету среднего арифметического размера деталей в партии (см.выше) с помощью цикла WHILE.  Program Sr\_\_rasmer1;  Uses Crt;  Var i, N: Integer;  S, r, Sr: Real;  Begin  Clrscr;  Write (`Количество деталей в партии: \_ \_ `);  Readln (N);  S: = 0; i: = 1;  While (I <= N) do  Begin  Write (`Введите размер `, i, `-той детали: \_ \_ `);  Readln (r);  S: = S + r;  I: = I + 1;  End;  Sr: = S/N;  Writeln (`Sr = `, Sr:5:3);  Readkey;  End. |

**3.3 Оператор цикла с последующим условием Repeat**

Позволяет многократно выполнятьлюбое количество операторов заранее неизвестное число раз.

Структура:

|  |
| --- |
| REPEAT <оператор 1>;  <оператор 2> ;  <оператор 3> ;  - - - - - - -  <оператор N> ;  UNТIL <условие>; |

В цикле **REPEAT** условие проверяется на выходе из цикла, поэтомуэтот цикл выполняется хотя бы 1 раз в любом случае.

Выход из цикла **REPEAT** осуществляется в том случае, когда проверяемое условие становится истинным.

Если оно не становится истинным никогда, то программа зацикливается.

|  |
| --- |
| Для выхода из зацикливания необходимо нажать комбинацию клавиш:Ctrl + Break |

|  |
| --- |
| Пример:  Модернизировать программу по расчету среднего арифметического размера с помощью цикла REPEAT.  Program Sr\_\_rasmer2;  Uses Crt;  Var i, N: Integer;  S, r, Sr: Real;  Begin  Clrscr;  Write (`Количество деталей в партии: \_ \_ `);  Readln (N);  S: = 0; i: = 1;  Repeat  Write (`Введите размер `, i, `-той детали: \_ \_ `);  Readln (r);  S: = S + r;  I: = I + 1;  Until (I > N);  Sr: = S/N;  Writeln (`Sr = `, Sr:4:2);  Readkey;End. |