Chapirte 4 : Repetitive structures « loops »

1. Introduction

In algorithms, some instructions may be repeated and it would not be very wise to rewrite them. Repetitive structures can be used to control repetition.

A repetitive structure (or iterative structure) repeats the execution of instructions in a determined or indeterminate number of times. An iterative structure is also called a loop.

Mainly, there are three forms of loops, the use of one or the other is related to the repetition test and algorithm context.

2. The loop "While"

The loop While allows you to repeat the same block of instructions several times as long as a condition remains true.

Syntax:

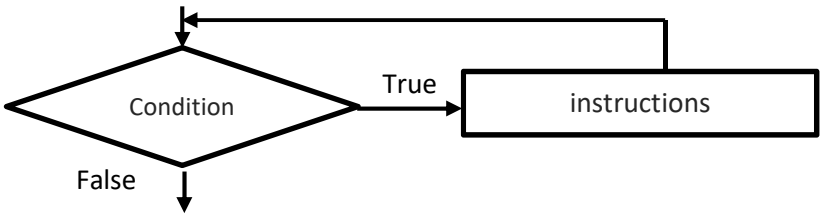
while (Condition**) do**
  Instruction 1 ;
  Instruction 2 ;
  ……
  Instruction 3 ;
endwhile;

// Instructions to repeat;

Semantic :

If the condition is true the group of instructions is executed then **endwhile** returns execution to the condition test, and so on until the condition will be evaluates to false.

Flowchart representation:



The While loop in C language:

```
While (Condition)
 {
    instructions ;
 }
```

Remarks :

- The block of instructions being repeated until the condition becomes false, means that the instructions to be repeated must modify the parameters involved in the condition in order to exit the loop. If the condition always remains true we can get an infinite loop problem.

Example: The following algorithm calculates the sum of integers in the interval [1, 30]

```
Algorithm Sum;
  Var
          i, s :  integer;
  Begin
   s ← 0 ;
   i ← 1 ;
   While (i ≤ 30) Do
     s ← s + i;
     i ← i+1;
    EndWhile;
     Write (''the sum is :  '', s);
  end.
```

Application exercises:

Exercise 1: Write an algorithm that asks the user for an integer, and then calculates its factorial.

Exercise 2: Write an algorithm that displays whether an integer A is perfect or not.

Exercise 3: Write an algorithm that calculates the division of a real number by an integer according to the formula: ($z = x/y$) ($y \neq 0$).

Exercise 4: write an algorithm that calculates the number of integer values entered to arrive at a sum not exceeding 500.

3. The loop '' Repeat''

The Repeat loop has the same function of the While loop, except that the condition is at the end of the loop and not at the beginning.

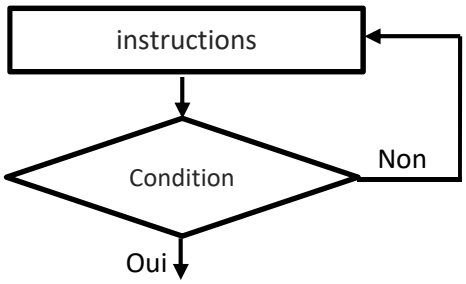
The difference between the two forms is that a loop with While may never be executed because the condition may be false initially while in the loop Repeat the instructions to repeat are executed at least once.

Syntax :

Repeat
  Instruction 1;
  Instruction 2;
  …
  Instruction n ;
Until (Condition);

Note :
The Repeat condition is an end condition where the repetition is finished when the condition is true, whereas, the While condition is a repeat condition where the repetition is finished when the condition evaluates to false.

Flowchart representation:



Example:
Write an algorithm that calculates the division of a real number by an integer according to the formula: (z = x/y) (y ≠ 0).

Solution :

```
Algorithm Division;
 var
            x, z : real;
            y : integer;
 begin
            read (x);
            repeat
              read (y) ;
            until (y ≠ 0);
            z ← x / y;
            write (z);
 end.
```

Analysis :
When executing this algorithm, the transition to the z calculation instruction is only made when the entry of the value of y is different from zero.

Solution using while loop:

```
Algorithm Division;
 var
            x, z : real;
            y : integer;
 begin
            read (x, y);
            while (y = 0) do
              read (y) ;
            endwhile;
            z ← x / y;
            write (z);
 Fin.
```

Analysis:
The read instruction is then written twice: the first to test the condition and the second to repeat the reading until a non-zero value is entered.

Syntax in C language:

```
Do
 {
  Statement 1;
  Instruction 2;
  …
  Statement n;
 }
While( ! Condition);
```

Exercise :
Write an algorithm that assists the user with messages asking for a number between 1 and 10 until the answer matches.

Solution :

```
Algorithm  interval_match;
 var
   n: integer;
 Begin
   Repeat
      Write("Give an integer between 1 and 10");
      Read(n);
      If (n<1 or n>10) Then
         Write("Wrong entry. Please start again!!") ;
      endif;
    until (n>=1 and n<=10);

   Write("The value entered is correct, Thank you!!") ;
end.
```

4. The loop "For"

The loop "For" uses a counter to control the number of iterations (number of repetitions) of the sequence of instructions.

Syntax:
For (counter ← Initial_value **To** Final_value **Step** = n) **Do**
  Instruction block;
EndFor ;

Semantics:
The ***counter*** of iterations is initialized by ***initial_value*** which will automatically increments according to the value the ***Step n*** after each execution (iteration) until reaching the ***final_value***.

Remarks :

- The type of the counter is integer
- The step is optional. It represents the increment step when it is positive and decrement step when it is negative.
- By default the increment step is equal to 1, in this case we do not have to mention it.

Syntax in C language:

```
For(counter=initial_value; Stop condition; Step)
 {
  Instructions block;
}
```

Note:
Stop condition: condition to exit the loop, in general (counter < final_value)
Step: increment or decrement instruction (exp: i=i +1 or i++ / i = i-1 or i--)

```
For (i ← 1 To 10) Do
 write( i ) ;
endfor ;
```

Example1:
To display the numbers from 1 to 10 on the screen, you can use the For loop as follows:

Algorithm	Program in C

For (i ← 1 To 10) **Do**
  Write(i);
EndFor ;
	For(i=1 ; i<10 ; i++)
	{
	  Printf(''%d'', &i) ;
	}

Example2:
To display even numbers from 1 to 10 on the screen, we can use the following For loop:

For (i ← 2 **To** 10 **Step** = 2) **Do**
  Write(i);
EndFor ;

Conclusion :
There are no precise rules to justify the choice of a type of loop (While, Repeat, For) but generally it depends on the desired processing:

- The **while** loop is the simplest and can be used in all situations.
- If in the algorithm/program we have a counter which increments/decrements by a precise step until reaching a final value, in this case, the **For** loop is best suited
- If the desired iteration requires the execution of a block at least once, in this case the **Repeat** loop is recommended;

5. Nested loops

Two loops are nested if one is contained within the instruction block of the other.
Some algorithms require nesting of loops, for example, we wish to display all the multiplication tables from 1 to 9.

- displaying the multiplication table of a number, for example the multiplication table of 7, requires a loop;
- in addition, we must display the multiplication table for each number between 1 and 9, which requires a second loop containing the first.

Example :
Consider the following algorithm which displays the multiplication tables from 1 to 9.

Analyse :

i x j = multiplication result
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63

Solution:

```
Algorithm multiplication_table;
 var
    i, j, r : integer;
 Begin
    For (i ←1 To 9) Do
        For (j ←1 To 9) Do
           r ← i * j ;
           Write (i, " x ", j, " = ", r);
        EndFor;
   EndFor;
end.
```

Exercises :

Write algorithms that calculates

- the sum S1 of odd numbers less than N : S1 = 1 + 3 + 4 + …..
- the sum S2 : S2 = 1 + 2 + 3 + … + N.
- the sum S3 : S3 = $1 + 2^2 + 3^2 + \ldots + N^2$.
- the power N of X : $X^N = X * X * \ldots * X$ (N fois)
- the factorial F of a number N : $F = N! = N * (N - 1) * (N - 2) * \ldots * 3 * 2 * 1$
- the sum S4 : S4 = 1 ! + 2 ! + 3 ! + … + N !