Chapter 4: Repeat Structures (Loops)

1. Introduction :

- The repetition structure is one of the three fundamental structures which are:
  1) Sequential actions,
  2) Conditional actions,
  3) Repeat actions.

2. Repeat (iterative) actions (loops) :

- Iterations (loops) allow the same series of instructions to be repeated several times.
- There are mainly two types of loops:
  - The loop which makes it possible to repeat instructions until a condition of stop, it is about the loop **while** And **repeat**.
  - The loop which allows instructions to be repeated a certain number of times, this is the loop **For**

2.1. The "While" loop:

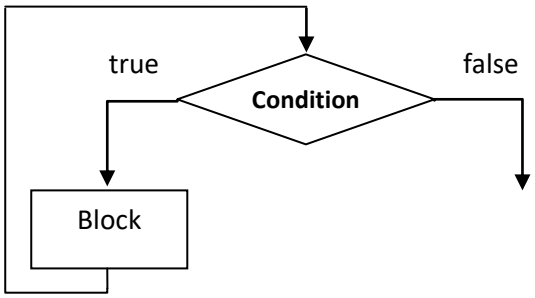
- It allows the repetition of one or more actions as long as the condition remains verified.

Syntax:

While (condition) **do**

< Action block >

end while



- This involves repeating the execution of the instructions of the < Action block > as long as the condition is verified and stopping their execution as soon as the condition becomes unverified.

Example :

Write an algorithm that calculates the sum of the first n positive integers?

Algorithm sum-ent1

n, i, Som: integer;

Begin

Read(n); /* to know where we will stop*/

Som←0; /* the sum is initialized to 0*/

i←0; /* we must specify the initial value of i*/

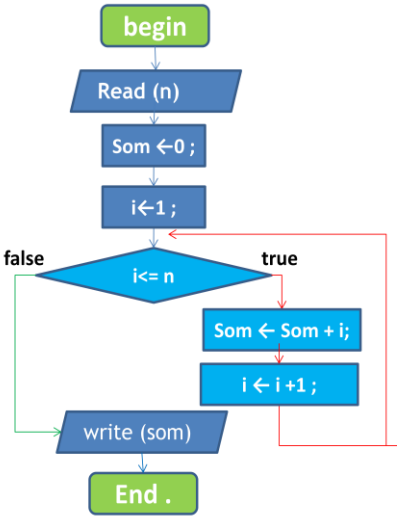
While(i ≤ n) Do

```
Som←Som + i;        /* the new value of Som receives the old value + the current value
                       of i (in this iteration)*/
i←i+1;              /*  increment of i by 1 at each iteration*/
```

End while
Write (Som);
END.

Noticed :

- The While repetitive action is used when the number of repetitions is not known a priori and when it is possible not to execute the actions of the block at all; in the previous example, if the user enters a negative value of n (-3 for example) the block inside the loop While never executes.
- The flowchart of the previous example is as follows:



Properties

- We know that when exiting the loop, the loop condition is false.
- We do not know the number of iterations to perform, but at each iteration, we check if the condition is true or false.

2.1.1 Exercise: (Coffee vending machine)

You are asked to write an algorithm that asks the user the following question: “Do you want a coffee? he must respond on the keyboard with “Y” (i.e. Yes), or “N” (No).

- If the answer is yes, the computer displays "here is your coffee"
- Otherwise it displays "see you soon".

The first idea consists in making a conditional test **If** on a character variable which represents the answer of the user on the question, if he answers with 'Y' i.e. Yes, if not it means No, but the question here, does in the answers differing from 'Y' mean no, it may be that it's a typo, it is better that we take that in consideration. By displaying an error message, and giving the user another chance to re-enter their answer until a 'Y' or 'N' answer will be the best solution which is as follows:

```
algorithm Distributor
rep: character;
begin
Writing ('would you like a coffee?');
Read (rep);
while((rep <>'Y') and (rep <>'N')) do
 Write ('erroneous entry');
Read (rep);
End while
if (rep = 'Y') then
Write ('here's your coffee');
Else //rep='N'
   Write('see you soon');
end if
End .
```

2.2. The "For" loop :

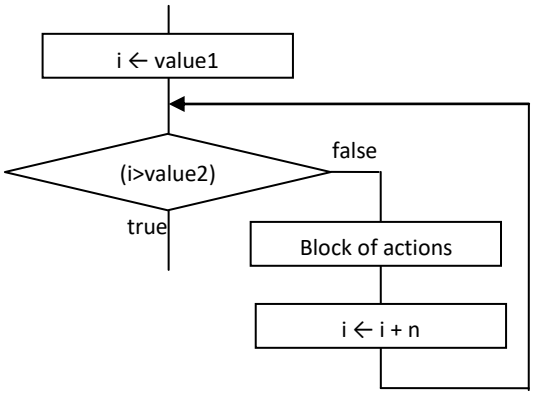
- The For loop repeats an instruction or a block of instructions a given number of times.
- It uses a variable called control variable, or iteration counter to control the number of iterations.

Syntax:

For i = Value1 **to** Value2 (**step=** n) **do**

<Action block>

End For



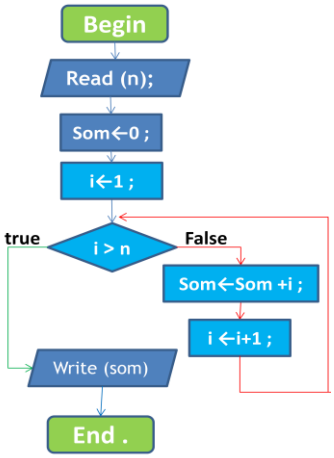
Remarks :

- The underlined words are keywords
- The diagram opposite represents The flowchart of the For loop

Example :

We return to the previous example; the sum of the first n positive integers; now using the For loop.

Algorithm som_int
n, i, som: integer;
Begin
Read(n);
som ← 0;
For i = 1 **to** n (**step** = 1) **do**
som ← som + i;
endfor
Write (som);
END.



Remarks:

- This algorithm will stop as soon as i reaches the value of n. so at the exit of the loop the value of i = n + 1.
- The For loop is used when the number of times known in advance. For this, we must specify the initial value and the final value and possibly the step.
- The counter is incremented automatically. The step, when not specified, is equal to 1.

2.2.2 Exercise: (Phone device password)

You are asked to write an algorithm that displays the following message to the user: "enter the password". The user enters the password, if it is correct it displays the message: "device unlocked" otherwise, it displays: "error, you have three chances".

We can solve it by using the for loop, because we know in advance the maximum number of repetitions in case of error password entry (which is 3 times). The solution will be as follows:

- we will display the message that asks the user to enter the password.
- Read the password by a variable of String type (pass).
- We are going to loop 3 times through the loop to (value1=3, value2=1, step= -1), and each time We do a test on the variable pass, if it is Different from the value stored in device ('12345'), an error message is displayed and we give the user the possibility to re-enter the password,
- at the exit of the for loop, we do a test If-else on pass value.

```
algorithm Password
pass: string;
begin
Write ('enter password');
Read (pass);
For i = 3 to 1 (step = -1)do
  If (pass <> '12345' )then
  Write ('error, you still have',i,' chance');
  Read (pass);
  End if
End for
if (pass='12345') then
Write ('device unlocked');
Else //pass ≠'12345'
Write('you can not use the device');
end if
End .
```

Issue : in case the user entered the correct password in the first time, the computer will do 3 iterations in the loop without doing anything!!! Since the condition of **if** is always false.

To solve this problem, we must use another loop structure which is the **Repeat** loop that we will study in the next title.

2.3. The "Repeat" loop:

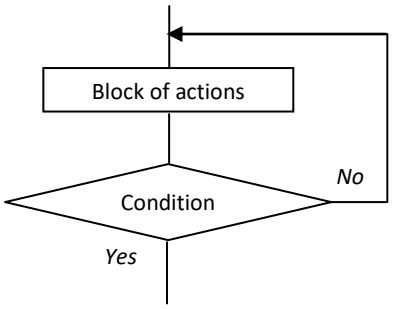
- This action expresses the repetition of one or more actions until the condition becomes verified.

Syntax:

Repeat

<Action block>

Until(condition)

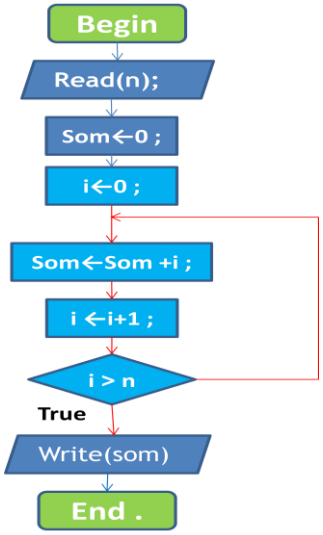


- This involves repeating the execution of the instructions of the block <Action block> as long as the condition is not verified and stopping their execution as soon as the condition becomes verified.

Example :

Repeat the algorithm that calculates the sum of the first N positive integers using the Repeat action.

```
Algorithm sum-ent4
        n , i ,som:integer;
Begin
Read(n);
som← 0;        // the sum is initialized to 0
i← 0;          / /the initialization of i
Repeat
        Som← Som + i;
        i←i + 1;                //increment of i
Until(i>n)
Write(som);
END.
```



Noticed :

- The "Repeat" action is used when the number of repetitions is not known a priori and when the actions of the block must be executed at least once (because the evaluation of the condition is at the end).

2.3.1 Exercise: (Phone device password)

We will resume the exercise of unlocking a mobile phone, we will solve it using the loop repeating the solution will be as follows:

- We put the instruction that allows to display the message 'enter the password', and initializes the variable i by the maximum number of chances that we will give to the user to enter the password pass ; i←3;
- Since the principle of the repeat loop is to execute the block of instructions before having the condition assessed, we can put the instruction read (pass), once inside the loop, then we do an If-end test on the value of pass
- The condition that allows us to get out of the repeat loop is: (pass='12345') or (i < 1);

```
algorithm Password
pass: string;
begin
Write ('enter password');
Read (pass);
I← 3;
Repeat
Read (pass);
If (pass <>'12345') then
Write ('error, you have ',i,' chance');
End if
i← i-1;
Until ((pass='12345') or (i<1))
if (pass='12345') then
Write ('device unlocked');
Else //pass ≠'12345'
Write(' you cannot use the device');
end if
End .
```

i.e. either the password entered is correct
or the number of chances is exhausted
(in our case is 3) .
so the algorithm is:

2.4. Nested loop structures

Two or more loops can be used, one inside the other, they can be For or While or Repeat, or even a mix between them. The structure will be as follows

While (condition 1) do	For i = V1 to V2 not =n do	Repeat
While (condition 2) do	For j = V3 to V4 pas= m do	While (condition1) do
<instructions block>	< instructions block>	< instructions block>
end while	End for	end while
end while	End for	Until (condition 2)

Example:

Write an algorithm that reads a natural number N, then calculates and displays the sum:

$S=(1+2+3+\cdots+N)+(2+3+\cdots+N)+\cdots+(N-1+N)+N$.

The solution:

Algorithm sum
N,i,j,S,S1: integer;
Begin
Read (N);
S←0; /* S is the total sum initialized to 0*/
For i = 1 to N **Do** // i starts from 1 to N
S1←0; /*S1 represents the sum between brackets initialized to 0 */
For j = i to N **Do** // j starts from the current value of i to N (in brackets)
S1←S1+i;
End For
S←S+S1; // at the end of the second loop, we add the current value of S1 to the total sum s
End For
Write (S); // displays just the total sum S.
END.