Chapter 3: Loops

Presentation of the concept of loop

- Sometimes, we repeat a specific code instruction multiple times to solve a problem until a specific condition is met. This is known as iterations, which allows us to write code once and execute it multiple times.
- Loops provide code reusability and simplifie the steps of problem-solving. We can execute the same code a finite number of times
- There are mainly three types of loops :**repeat**, **For**And **While**.

Presentation of the concept of loop

- Usually, Loops have three main elements which are:
  - **Initialization**: we assign an initial value to a variable used when evaluating the loop condition.
  - **The loop condition**: is a Boolean expression often evaluated before the execution of the loop body as in the case of while and for loops, and after the execution of the loop body as in the case of repeat and do - while. In any case, the body of the loop is running until the condition will be evaluated as false.
  - **Variation of the loop condition**: this is usually done at the end of the loop body as an increment or decrement of a counter used to evaluate the loop condition as in the case of while, repeat and do- while. Forgetting or incorrectly performing this step causes the loop body to execute infinitely, or give unexpected results.

The loop while

- **While** loop is used to execute the body of the loop until a specific condition is false. We apply this loop when we don't know how many times it will run. The loop while consists of a loop condition, a code block as loop body and a loop update expression. First, the loop condition is evaluated, and if true, the code in the body of the loop will be executed. This process repeats until the loop condition becomes false.

The loop while

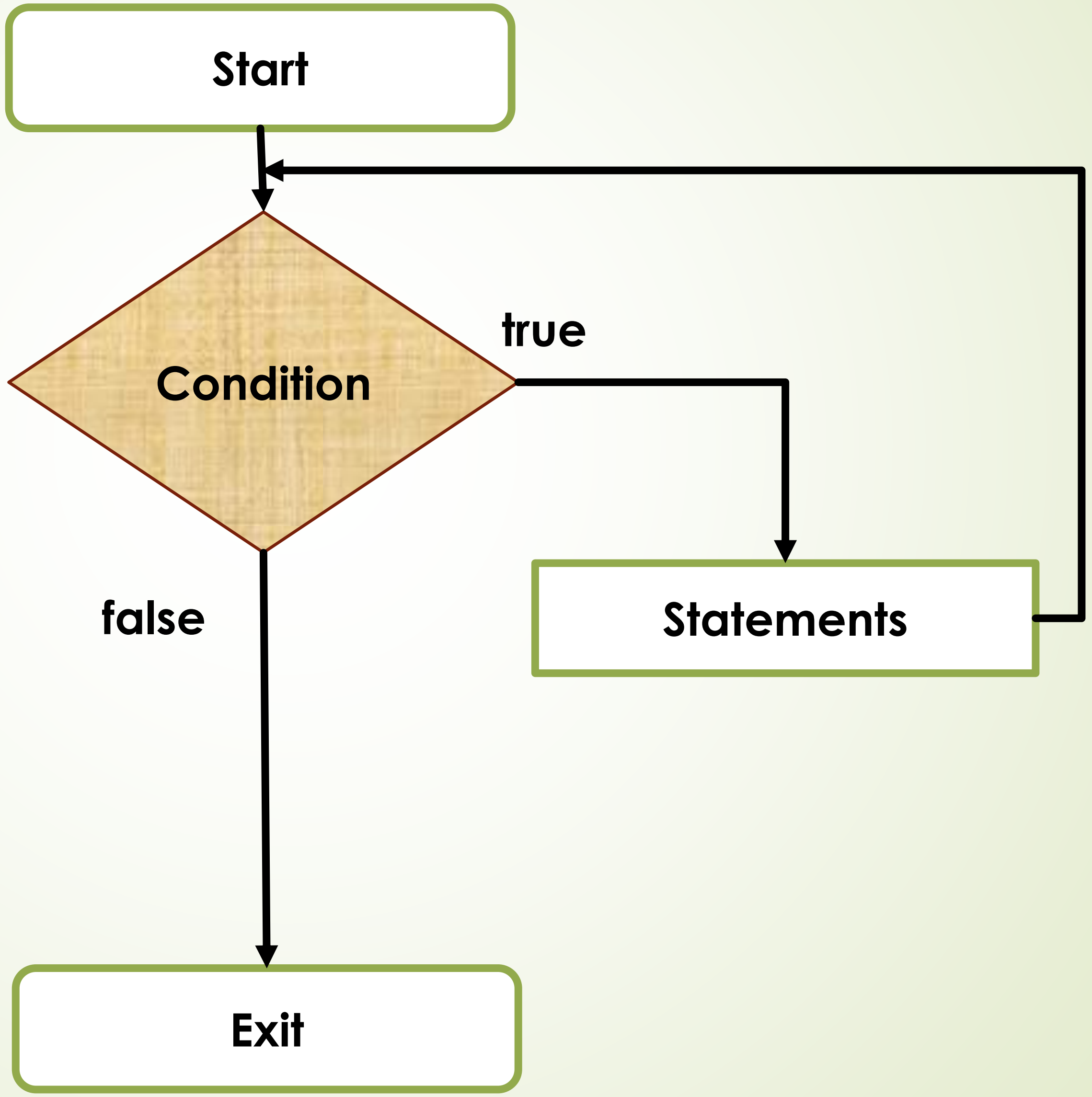
```
Initialization of the loop
While (condition of the loop) do
begin
    Loop's body
    Condition update
end;
```

Syntax of while loop in algorithmics

```
Initialization of the loop
while(condition of the loop)
{
    Loop's body
    Condition update
}
```

Syntax of while loop in C

The flowchart of the While loop



Example of using the While loop

- Count from 10 to 20

```
algorithm count;
var i:integer;
begin
 i←10; //initialization
 While(i<=20) DO//the number 20 is included
 begin
 write(i);
 i←i+1; //we increment the counter i (variation of the
//condition and preparation of the next //iteration)
END;
END.
```

The do-while loop

- The do-while loop is a control flow instruction that executes a block of code at least once, then repeatedly executes the block, or not, based on a Boolean condition given at the end of the block.
- In the control structure do-while, there are three main elements which are:
  - Action
  - Updating loop condition
  - Loop Condition Assessment

The do-while loop

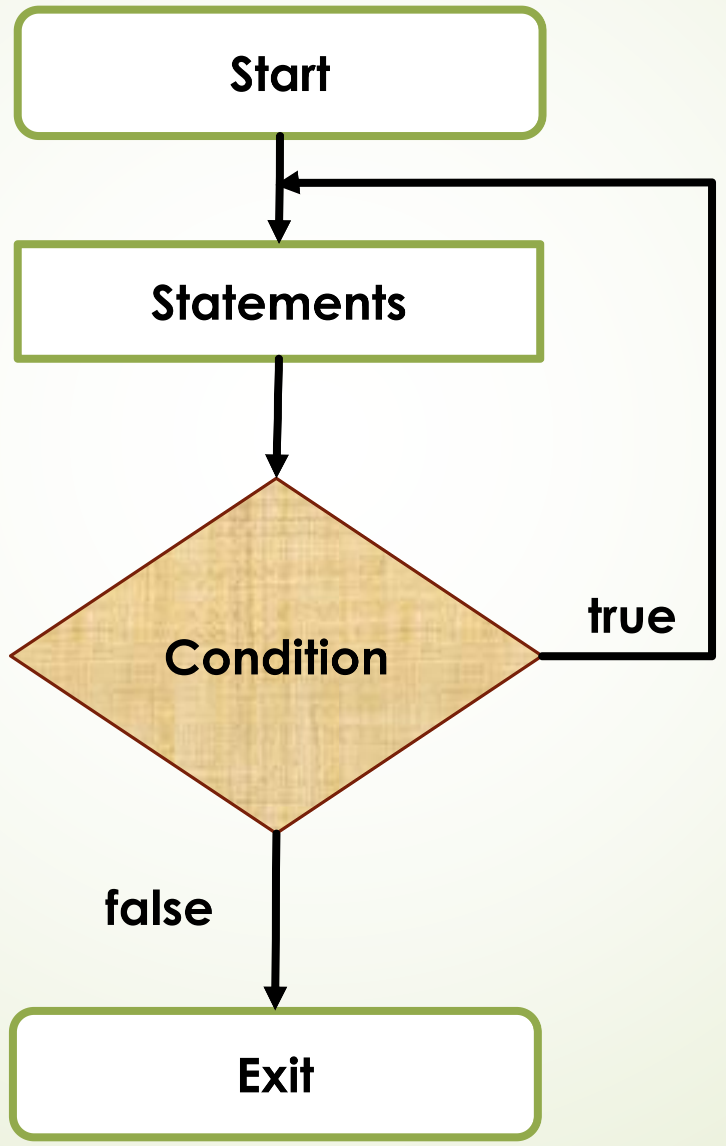
```
Initialization of the loop
DO
begin
Loop's body
 Condition update
END;
While(condition)
```

Syntax of the do – while loop in algorithmics

```
Initialization of the loop
do
{
Loop's body
 Condition update
}
while(condition loop);
```

Syntax of the do-while loop in C

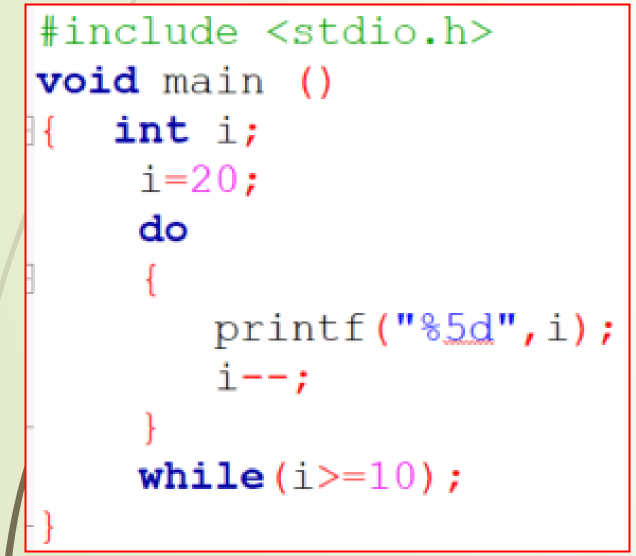
The flowchart of do – While loop



Example of using the While-do loop

- Count from 20 to 10

In C language



```c
#include <stdio.h>
void main ()
{   int i;
    i=20;
    do
    {
        printf("%5d",i);
        i--;
    }
    while(i>=10);
}
```

In Algorithmics

```
algorithm countv2;
var i:integer;
begin
 i←20;//initialization
 do
 begin
       write(i);
       i←i-1;// variation of the condition
 end;
 While (i>=10) //the number 10 is included
end.
```

The loop for

- The FOR loop is preferable to previous loops when the number of times a block of instructions is to be executed is known in advance. The main use of the loop **FOR** is to manage a counter (integer type)
  The for loop is preferred when:
  - The variation interval is known
  - The increment step is fixed
  - Loop exit is related to the loop index
- On the other hand, we use the 'while' loop when the condition of the loop is:
  - Composed of complex logical expression,
  - Evaluates additional variables other than the loop counter,

Loop syntax for

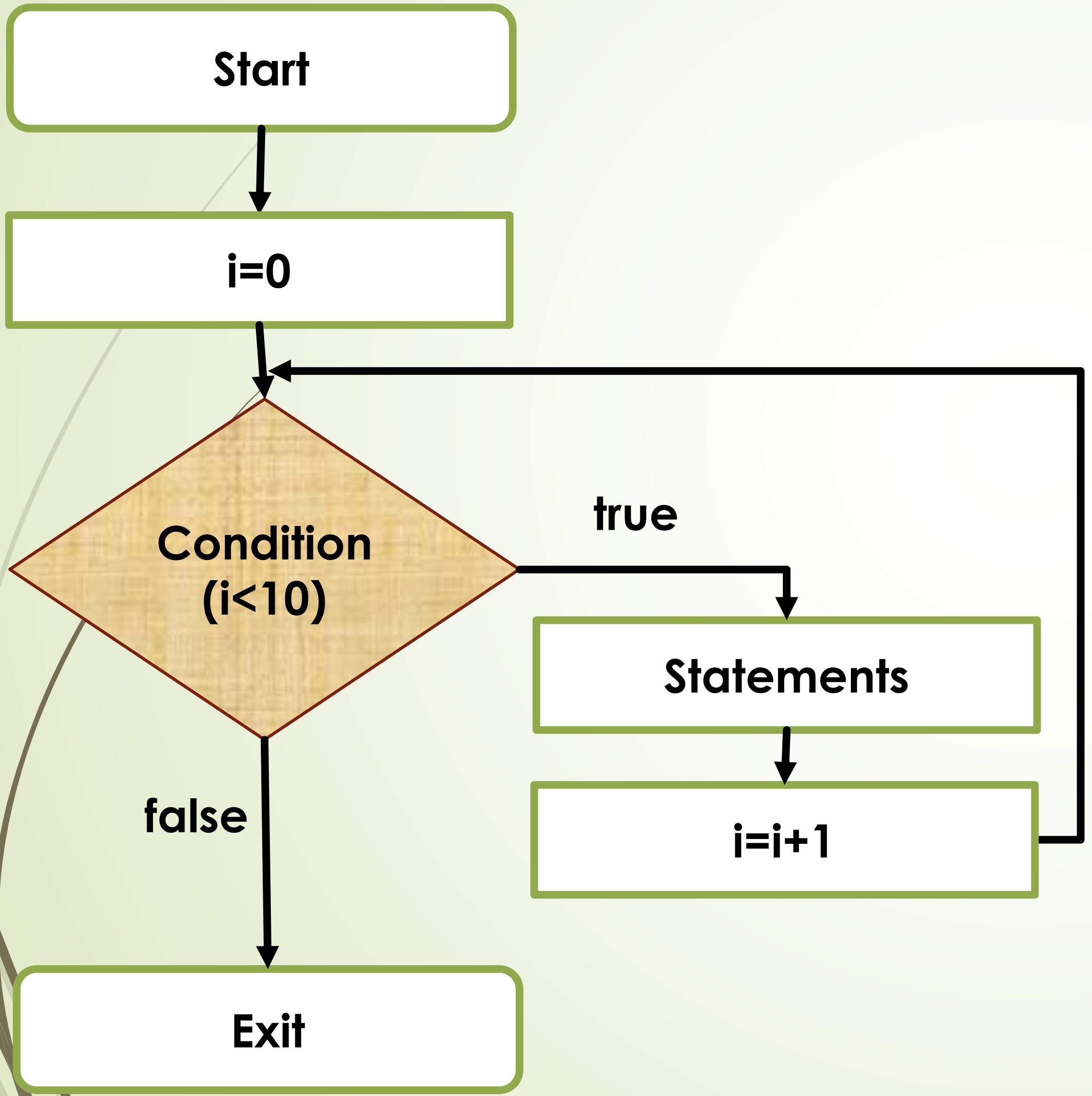
```
for i← x to y do
begin
Loop's body
end;
```

Syntax of the for loop in algorithmics

```
for (i=initial value; condition; variation of i)
{
Loop's body
}
```

Syntax of the for loop in C

Example of The flowchart of 'for' loop

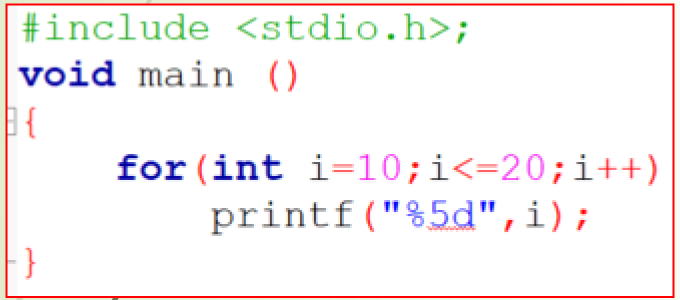


```
for(int i = 0;i<10;i++)
{
        statements;
}
```

Example of using the loop for

- Count from 10 to 20

In C language



```c
#include <stdio.h>;
void main ()
{
    for(int i=10;i<=20;i++)
        printf("%5d",i);
}
```

In Algorithmics

```
algorithm countv2;
var i:integer;
begin
 for i← 10 to 20 do
 begin
     write(i);
 end;
end.
```

Initial counter value

Condition

The main instruction and automatic increment of i

The 'repeat' loop

- The loop **repeat-until** executes a block of instructions repeatedly, until a given condition becomes true. The condition will be re-evaluated at the end of each iteration of the loop.
  Because the condition is evaluated at the end of each iteration, a repeat/until loop will always execute at least once.
- The 'repeat' loop is similar in structure to the 'do-while' loop.
- In C language, we use the 'do-while' loop.

Repeat loop syntax

```
Initialization of the loop
repeat
begin
Loop's body
 Condition update
end;
Until(condition)
```

Syntax of the repeat loop in algorithmic

Example of using the loop repeat

- Count from 20 to 10

In Algorithmics

```
algorithm countv3;
var i:integer;
begin
 i←20;//initialization
 repeat
 begin
       write(i);
       i←i-1;// variation of the condition
 end;
 until (i<10)//the number 10 is included
end.
```

Initial counter value

The main instruction and decrement of i

Stop Condition

Operations with loops

- **Navigate a range in ascending order**
  - Initial value of the counter = the start of the rank;
  - The stopping condition: the counter must be less than or equal to the upper limit of the rank;
  - The step (the variation of the condition): it is carried out with an increment of 1 in the loop counter.

Navigate a range in ascending order

Final counter value = 20 (upper limit)

The loop for

```
algorithm count;
var i:integer;
begin
 for i← 10 to 20 do
 begin
      write(i);
 end;
end.
```

The loop while

```
algorithm count;
var i:integer;
begin
 i← 10;
 while (i<=20) do begin
      write(i);
      i←i+1;
 end;
end.
```

The do-while loop

```
algorithm count;
var i:integer;
begin
 i← 10;
do
 begin
      write(i);
      i←i+1;
 end;
 while(i <=20)
end.
```

Initial counter value = 10 (lower limit)

Step = 1

Navigate a range in descending order

- Initial value of the counter = upper limit of the rank;
- The stopping condition: the counter must be greater than or equal to the lower limit of the rank;
- The step (the variation of the condition): it is carried out with a decrement of 1 in the loop counter.

Navigate a range in descending order

The loop for

```
algorithm count;
var i:integer;
begin
 for i← 20 to 10 do
 begin
      write(i);
 end;
end.
```

The loop while

```
algorithm count;
var i:integer;
begin
 i←20;
 while(i>=10) do begin
       write(i);
       i←i-1;
 end;
end.
```

The do-while loop

```
algorithm count;
var i:integer;
begin
 i←20;
 do
 begin
       write(i);
       i←i-1;
 end;
 while(i>=10)
end.
```

Loop application examples

- Sum of first 10 numbers

```
algorithm count;
var i,sum:integer;
begin
sum←0;
 i← 1;
 while (i<=10) do
 begin
 sum←sum+i;
        i←i+1;
 end;
write(sum);
end.
```

Initial value =0; the 0 does not disturb the addition operation

Need to traverse the rank [1,10]

The table detailing the execution flow of loops

- The loop execution flow table is necessary to see the evolution of the variables manipulated inside the loops as well as the correct checking of the condition and the initialization of the value of the loop counter. This table is composed of n lines for n iterations executed by the loop, plus the initial state of the variables before launching the loop. In columns, we find the iteration number, the variables manipulated inside the loop, and the output if applicable.

Examples of using the loop execution table

- Sum of the first 5 numbers

```
algorithm count;
var i,sum:integer;
begin
sum←0;
 i← 1;
 while(i<=5)do
 begin
 sum←sum+i;
 write(sum+ " ");
        i←i+1;
 end;
end.
```

iteration	i	sum	output
initialization	0	0	-
1	1	1	1
2	2	3	1 3
3	3	6	1 3 6
4	4	10	1 3 6 10
5	5	15	1 3 6 10 15

Infinite loops

- The problem of infinite loops occurs when there is an error in the condition of the loop, or in the variation of the condition, below are the possible causes of an infinite loop:
  - The counter is not varied (incremented or decremented), this is not the case of the loop **for** because the variation of the counter is done automatically.
  - Vary the loop counter so as to never check the stopping condition, in this case, it is necessary to clearly specify the range of the counter, its limits, and the movement of the counter (increment or decrement).
  - Vary the loop counter inside a conditional statement.
  - Wrong condition.

Infinite loops

- To avoid the infinite loop problem, you must take the following precautions:
  - Carefully study the range of variation of the counter.
  - Specify the loop condition carefully by taking into account the extreme values of the counter.
  - Involve the counter in the loop condition and don't forget to vary it.

Examples of infinite loops

```
sum algorithm;
vari,sum:integer;
Begin
sum←0;
 i← 1;
 while (i<=10) do
 begin
 sum←sum+i;
      end;
write(sum);
end.
```

```
sum algorithm;
vari,sum:integer;
Begin
sum←0;
 i← 1;
 while(i<=10) do
 begin
 sum←sum+i;
       i← i - 1;
 end;
write(sum);
end.
```

```
sum algorithm;
vari,sum:integer;
Begin
sum←0;
 i← 1;
 while(i>=1) do
 begin
 sum←sum+i;
       i← i + 1;
 end;
write(sum);
end.
```

```
algorithmsumNumPairs;
vari,sum:integer;
Begin
sum←0;
 i← 1;
 while(i<=10) do
 begin
 if(i mode2 = 0) then
 begin
 sum←sum+i;
       i← i + 1;
         end;
 end;
write(sum);
end.
```

Example of mathematical applications of loops

The factorial of a number

```
 fact=1;
for (inti=2;i<=x;i++)
 fact=fact*i;
```

The power function (x to the power n)

```
power=1;
for (int i=n;i>=1;i--)
power = power *x;
```

Series offibonacci

```
 int a=0,b=1,c,
i=3,n=8;
printf("%d %d ",a,b);
while (i<=n)
{
        c=a+b;
        a=b;
        b=c;
        printf("%d ",c);
        i++;
}
```

Nested loops

- A nested loop means a loop inside another loop. We can have any number of loops inside another loop.

Example:

```
For i←0 to 10 do
for j← 0 to 5 do
statements;
```

References

- *Introduction to algorithms:135 corrected exercises.*Authors: Chantal Richard, Patrice Richard.
- *"For" loops*. (nd). Accessed November 1, 2022, athttps://www.irit.fr/~Julien.Pinquier/Docs/MABS/co/09%20-%20Boucle%20Pour.html?mode=html
- *Fundamentals of Loop andIterationinProgramming*. (nd). Accessed October 22, 2022, athttps://www.enjoyalgorithms.com/blog/fundamentals-of-loop-in-programming-and-algorithms
- Nested Loopsin Cwith Examples. (2019, November 25).*GeeksforGeeks*.https://www.geeksforgeeks.org/nested-loops-in-c-with-examples/