Chapter3

Loops (in algorithmic language and in C)

What's a loop for?

Consider the following example:

Algorithm Exemple1
Begin

```
write(1)
write(2)
write(3)
write(4)
write(5)
```

End

And let's take the example: Write an algorithm / program that displays the number 2 1000 times?
it is neither practical nor logical to repeat write(2) 1000 times or to write write(2, 2, ...2)2 1000 times , so what is the solution?
A loop is used to repeat an instruction (or a list of instructions) several times.
There are three main types of loop:

- Loops for repeating an instruction a certain number of times, the **For loop**
- Loops for repeating an instruction until a stop condition is reached, these are the **While** and **Repeat loops**.

Passing through a loop is called iteration

1. The For loop

The for loop instruction is used to repeat the execution of a block of instructions. It uses an iteration counter and an initial and final counter value. The counter is automatically incremented by 1. The main feature of this instruction is that the number of iterations to be performed is known in advance.

The syntax of the loop for instruction is :

Algorithm	**C code**
1) **For**cpt from vi to vf **do** <instruction_1> **End For**	**1)** **For** (cpt=vi ; vi<=vf ; vpt++) <instruction_1> ;
2) **For**cpt from vi to vf **do** <instruction_1> <instruction_2> …….. <instruction_n> **End For**	**2)** **For** (cpt=vi ; vi<=vf ; vi++) **{**<instruction_1> ; <instruction_2> ; …….. ; <instruction_n> ; **}**

Note: cpt: counter, vi: initial value, vf: final value

In the for loop, at each point in time, we know the number of iterations already performed and also the number of iterations remaining.

2. The whileloop

While loops are used to perform iterations as long as a certain condition is verified. Since we can't know the number of iterations to be performed, at each iteration we check whether the condition is true or false. As soon as the condition is false, we exit the loop.

The syntax of the While loop instruction is:

Algorithme	**C code**
1) **While** <condition>**do** <instruction_1> **End while**	**1)** **while**<condition> <instruction_1> ;
2) **While** <condition>**do** <instruction_1> <instruction_2> …….. <instruction_n> **End while**	**2)** **while**<condition> **{**<instruction_1> ; <instruction_2> ; …….. ; <instruction_n> ; **}**

Note: the number of instruction repetitions in the while loop is [0..k].

3. The Repea tloop

The Repeat loop instruction is used to repeat the execution of a block of instructions. At each iteration, a Boolean expression (condition) is re-evaluated:

- If the expression is TRUE: the loop is stopped and the instruction following Repeat is executed;
- If the expression is FALSE: continue the loop by executing the next iteration.

The syntax of the Repeat loop instruction is:

Algorithm	**CCode**
1) **Repeat** <instruction_1> **Until**<condition> **2)** **Repeat** <instruction_1> <instruction_2> …….. <instruction_n> **Until**<condition>	**1)** **Do** <instruction_1> ; **while**<condition> **2)** **Do** **{**<instruction_1> ; <instruction_2> ; …….. ; <instruction_n> ; **}** **while**<condition> ;

Notes :

- The number of instruction repetitions in the repeat loop is [1...k].
- In general, if the first iteration is performed unconditionally, the Repeat instruction can be used instead of the While instruction.
- The Repeat instruction is equivalent to While instruction.
- The For instruction is a special case of the While instruction.

Remarks on iterative instructions

An instruction can be a single instruction or a block. The various loops may be nested, so a certain layout is required to make the program easier to read.

4. Sequence breaks

When you want to exit a loop while the passing condition is still valid, you use operations called sequence breaks. Sequence breaks are used when several conditions (multiple conditions) condition the execution of a set of instructions.

In C, instruction breaks are achieved by four instructions that correspond to their work level:

- **continue**
- **break**
- **goto**
- **return**

Two library function calls can also be used to change the order in which the program is executed:

void exit (intstatus) : calling this function terminates execution of the program.

voidlongjmp(jmp_bufenv, int val) : which jumps to a restart point set in the programme.

3.1. continue

This instruction jumps to the next iteration in a loop at the end of the block.

3.2. break

It is used to exit a block associated with a repetitive or alternative instruction controlled by the if, for, while or do while statements. It can only be used to exit a nesting level.

3.3. return

It terminates the execution of the function in which it is located and returns to the calling function.

3.4. goto

Allows you to go anywhere within a function. It is associated with a label, which is a string of characters followed by a colon ":"

The goto branch command and the locate by label instruction must be in the same function.

Exemple :

Algorithm	**C Code**

Algorithm example
Variable x,i : integer
Begin
x←0, i←1
loop : x←x+i
i←i+1
if (i<100) Then
go to loop
Endif
write(x)
End.
	#include<stdio.h>
	main()
	{
	int i,x;
	i=1; x=0;
	loop : x=x+i;i++;
	if (i<100)
	goto loop;
	printf("%f",x);
	}

Exercices à domicile

Exercise 1

Write an algorithm that asks for a number between 10 and 20, until the answer is correct. If the answer is greater than 20, a message will appear: "Smaller", and conversely, "Larger" if the number is less than 10.

Exercise 2

Write an algorithm that asks for a starting number, and then displays the next ten numbers. For example, if the user enters the number 17, the program will display the numbers from 18 to 27.

Exercise 3

Write an algorithm that asks for a starting number, and then writes the multiplication table for that number, presented as follows (where the user enters the number 7):
Table of 7 :
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
…
7 x 10 = 70

Exercise 4

Write an algorithm that asks for a starting number and calculates the sum of the integers up to this number. For example, if you enter 5, the program should calculate: 1 + 2 + 3 + 4 + 5 = 15
NB: you only want to display the result, not the breakdown of the calculation.

Exercise 5

Write an algorithm that successively asks the user for 20 numbers, and then tells the user which of these 20 numbers is the largest:

Enter number 1: 12

Enter number 2: 14

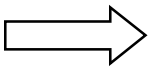
...

Enter number 20: 6

The largest of these numbers is: 14

Exercise 6

Write an algorithm that successively asks the user for 10 numbers, and at the end displays the largest of these 10 numbers and its rank in the list entered!
Example:
Enter number 1: 13
Enter number 2: 17
.....
Enter number 10: 5



The largest of these numbers is: 17
This was the 2nd number entered

Exercise 7

Write the algorithm that checks whether a positive number N is prime or not. A number is prime if it is only divisible by 1 and itself.
RQ: display an error message if N<=0

Exercise 8

Write an algorithm that first reads an integer value Then it reads 20 random integers Then it determines how many times the first value has been entered (not counting the first entry).

Exercise 9

Write an algorithm implementing the following game between two players: The first user enters an integer that the second must guess. To do this, he can make as many attempts as he likes. Each time he fails, the program tells him whether the integer is greater or smaller than his guess.
A score is displayed when the integer is found.

Exercise 10

Write an algorithm to display all pairs (x,y) , where x is an integer between 1 and p and y is an integer between 1 and q .
p and q are strictly positive integers read from the keyboard.

Example : p = 3 q= 4
The pairs displayed are :
(1,1),(1,2),(1,3), (1,4)
(2,1),(2,2),(2,3),(2,4)
(3,1) , (3,2) , (3,3) , (3,4)