

Chapter 3 : Iterative Control Structures (Loops)

What should be done when an algorithm needs to repeat the same processing several times? How can we avoid writing the same instructions repeatedly? Can we ask an algorithm to repeat a calculation as long as a condition is satisfied or until it is no longer satisfied?

This chapter answers these questions.

3.1 Introduction

In previous chapters, we saw how instructions could be executed sequentially or conditionally, that is, by following a precise order or by choosing between several alternatives. However, in many situations, an algorithm must repeat a process several times, for example, to add several numbers, browse the grades of a class, or search for an element in a list. Manually repeating instructions would make the algorithm long, cumbersome, and difficult to modify.

To address this need, we introduce iterative (repetitive) structures, commonly known as loops.

3.2 Definition

A loop is a control structure that allows an algorithm to execute the same block of instructions several times, according to a given condition. It is used to automate repetitive processing without rewriting the same instructions [3].

3.3 Types of loops

In algorithmics and in the C language, three main types of loops are distinguished:

1. While
2. Repeat…until (do…while)
3. For

3.4 The“While” loop

The while loop repeats a block of instructions as long as a condition is true. The condition is tested before each iteration.

Syntax

Algorithmic :

```
While (condition) do
   Instruction(s) ;
End-While;
```

In C language :

```
while (condition) {
    // instructions ;
}
```

Example: Displaying the first ten positive integers

```
Algorithm Numbers;
Variables i: integer;

Begin
   i ← 1;
   While (i <= 10) do
      Write(i);
      i ← i + 1;
   End-While;
End.
```

```
#include<stdio.h>

int main(){
int i = 1;
while (i <= 10) {
    printf("%d\n", i);
    i++;
}
return 0 ;}
```

Notes :

1. If the condition is false from the beginning, the block is never executed.
2. Always ensure that the condition can become false; otherwise, an infinite loop may occur.

3.5 The “Repeat...until” loop (do...while)

Unlike the while loop, the repeat loop executes the block of instructions first, then tests the condition. Therefore, it executes at least once.

Syntax

Algorithmic :

```
Repeat
   Instruction(s) ;
Until (condition) ;
```

In C language :

```
do {
    // instructions ;
} while (condition) ;
```

Example : Let's revisit the previous example.

Algorithm Numbers**;**
Variables i : integer;

Début
i ← 1 ;
Repeat
Write(i);
i ← i+1 ;
Until (i > 10) **;**
Fin.

```c
#include<stdio.h>

int main(){
int i = 1;
do {
    printf("%d\n", i);
    i++;
  } while (i <= )10;
return 0 ;}
```

Notes :

1. The repeat loop is used when we want the processing to be executed at least once, regardless of the condition.

2. In C, while and do…while use the same condition, unlike in algorithmics.

3.6 The "For" loop

The for loop is used when the number of iterations is known in advance.

Syntax

In algorithmic :

For *id_ counter* ← *initial_val* ***to*** *final_val* ***do***
Instruction(s) ;
End-For ;

id_counter : Variable that will count the iterations
initial_val : The first value that the counter takes
final_val : The last value that the counter takes

In C language :

```c
for (initialisation; condition; Incrementing)
{

// instructions;
 }
```

Example : Let's revisit the previous example.
In algorithmic :

Algorithm Numbers ;
Variables i : integer;
Begin
For i ← 1 to 10 do
Write(i);
End-For;
End.

In C language :

```c
#include<stdio.h>

int main(){
for (int i = 1; i <= 10; i++)
{
   printf("%d\n", i);
}
    return 0 ;}
```

In this example, the process is static, meaning that the algorithm always displays the numbers from 1 to 10. To make it dynamic, we ask the user to enter a number, and the algorithm will display the numbers from 1 up to the entered number.

Example : Let's revisit the previous example.

Algorithm Numbers **;**

Variables i, n : integer;

Begin
Write("Please enter a strictly positive integer:");
Read(n) ;
For i ← 1 **to** n **do**
Write (i);
End-For;
End.

In C language :

```
#include<stdio.h>

int main(){

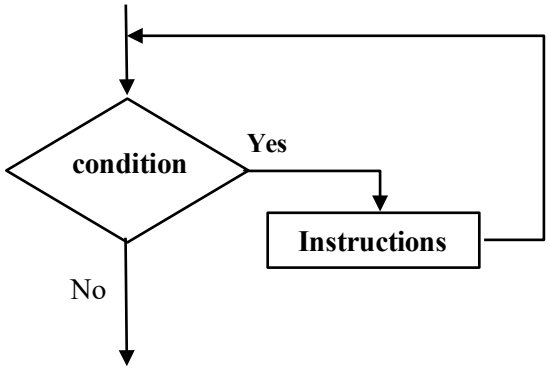
int n,i;

printf("Please enter a strictly positive integer");
scanf("%d",&n);
for (i = 1; i <= n; i++)
{
    printf("%d\n", i);
}
return 0 ;}
```

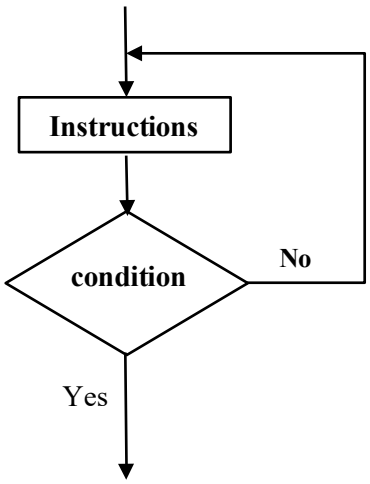
Note: In a for loop, the counter is automatically incremented by 1 at each iteration, unless another increment value is specified.

3.7 Iterative Structures in Flowcharts

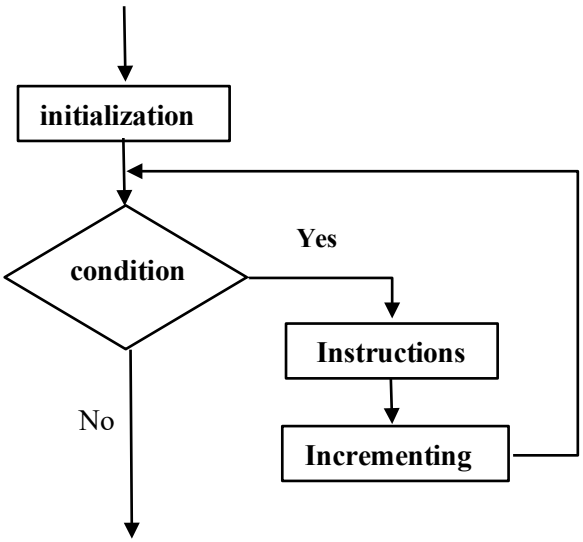
a. The "While" loop



b. The "Repeat" loop



c. The "For" loop



Notes :

1. The for loop can be transformed into a while or repeat loop, but the opposite is not always possible.

2. When transforming a for loop into a while loop, the initialization and incrementation of the counter must be handled manually.

3.8 Loops for Input Control (Validation)

When a program expects a specific value, the user may make a mistake by entering incorrect data. For example, when entering a person's age, the user may type a negative value or text instead of a number.

This type of error can cause:

- A runtime error, if the entered type does not match the expected type (e.g., the user types "twenty" instead of 20).

- A logical error, if the value is of the correct type but outside the allowed range (e.g., −5 or 250).

To avoid these situations, input validation must be implemented, that is, checking the data entered before continuing the program execution.

The principle consists of placing the read instruction inside a loop that repeats until the entered value is acceptable.

The Repeat loop is particularly well suited for this type of validation, although it can also be replaced by a While loop.

Let's revisit the previous example where we asked the user to enter a strictly positive number. We will now validate n to prevent any incorrect input.

Example : Let's revisit the previous example

```
Algorithm Numbers ;
Variables i, n : integer;

Begin
Repeat
 Write("Please enter a strictly positive integer ");
 Read(n) ;
Until (n>0 ) ;
For i ← 1 to n do
   Write(i);
End-For;
End.
```

```
#include<stdio.h>

int main(){

do
{printf("Please enter a strictly
positive integer \n");
scanf("%d", &n);}
while(n<=0);

for (int i = 1; i <= n; i++) {
    printf("%d\n", i);
}
return 0 ;
}
```

Conclusion

In this chapter, we studied iterative control structures, which are fundamental in algorithmics for processing data series, performing repetitive calculations, or traversing structures such as arrays.

Knowledge Assessment Test

MCQ:

1. Which loop guarantees at least one execution ?
a) while
b) for
c) do...while
d) none

2. Which loop is ideal if you know the number of iterations from the Begin?
a) for
b) while
c) do...while
d) switch

True or false:

3. The while loop always executes at least once.
4. A poorly designed loop can cause an infinite loop.
5. The for loop can be used with a decrement (e.g. i--).

Coding exercise:

6. Write a loop in C that displays the even numbers from 2 to 10 inclusive.

Quick answer key of the test:

- MCQ :
  1 → c) do...while
  2 → a) for
- True / False:
  3 → False
  4 → True
  5 → True