

Chapter 2 : Alternative Control Structures (Conditions)

What happens when an algorithm has to choose between several possible actions? How can an algorithm decide to execute one instruction rather than another? Can we make an algorithm behave differently depending on the data entered by the user?

This chapter will answer these questions.

2.1 Introduction

In the previous algorithms, instructions were executed sequentially, that is, always in the same order, one after another. However, in most real-life situations, an algorithm must be able to make decisions based on certain conditions. These conditions are essential to adapt the behavior of an algorithm to different cases.

2.2 Purpose

To understand the usefulness of conditional structures, let us return to Chapter 1, in which we wrote an algorithm that calculates the average of a module based on tutorial session and exam grades. However, if we now want to determine whether the module is passed or not, we face a new situation: the result to be displayed depends on the value of the average.

With basic instructions (assignment, read, write), we cannot express a choice between two possible actions. This is why we need a new type of instruction: conditional structures, which allow the algorithm to make a decision depending on whether a condition is true or false, and thus guide the execution of the algorithm accordingly [3].

2.3 Condition Concept

A condition is a logical expression that allows the algorithm to test a situation and make a decision based on the result of this test.

Note : In mathematics, we often write $10 \leq x \leq 20$ to indicate that x is comprised between 10 and 20.

However, in algorithmics (and in the C language), this notation is not valid as such, because an expression can only compare two values at a time. Therefore, this condition must be broken down into two comparisons linked by a logical operator: $(10 \leq x)$ and $(x \leq 20)$.

2.4 Simple Alternative Test (If.. then)

The simple alternative test allows an instruction to be executed only if a condition is true. If the condition is false, the instruction is simply ignored.

Syntax :

Algorithmic :

If (condition) **then**
    Instructions ;
End-If ;

In C language :

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

Example:

Algorithm Majority;
Variables age : integer;
Begin
Write("Please enter your age:") ;
Read(age) ;
If (age >= 18) **then**
  Write("You are an adult.");
End-If ;
End.

2.5 Double Alternative Test (If .. Else)

The double alternative test allows a choice between two blocks of instructions: one is executed if the condition is true, and the other if it is false.

Syntax:

Algorithmic :

If (condition) **then**
  Instructions;
Else
  Instructions;
End-If;

In C language :

```
if (condition) {
    // instructions ; }
else {
    // instructions ;
        }
```

Example : Let's improve the previous algorithm

```
Algorithm Majority;
Variables age: integer;
Begin
   Write("Please enter your age:");
   Read(age);
   If (age >= 18) then
      Write("You are an adult.");
   Else
      Write("You are a minor.");
   End-If;
End.
```

2.6 Multiple Choice (Nested Conditional Structure or Switch)

Multiple choice allows testing several possible cases of the same variable or expression. It can be done in two ways:

- Using several *if/else* instructions (statements).
- Using the switch structure for discrete cases in C language.

Syntax (with nested IF)

Algorithmic :

```
If (condition1) then
   Instructions1;
Else
   If (condition2) then
      Instructions2;
   Else
      Instructions;
   End-If;
End-If;
```

In C language :

```
if (condition1)
        {// instructions1;}
else
        if (condition2)
            {// instructions2;}
         else
            {// instructions ;}
```

Syntax (with Case..of)

Algorithmic :

```
Case (id_variable or Expression) of

  value1:  instructions for value1 ;
  value2:    instructions for value2 ;
        …
  valueN:  instructions for valueN ;
  else:  instructions if no value matches;
End-Case ;
```

In C language :

```
switch (id_variable or Expression) {
   case value1:
        // instructions for value1
        break;
   case value2:
        // instructions for value2
        break;
   case valueN:
        // instructions for valueN
        break;

   default:    /* Instructions if no value
matches*/
        break; }
```

Example in C :

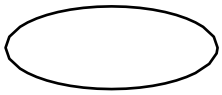
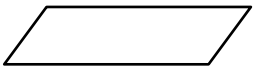
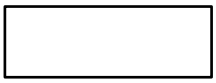
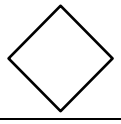
```
#include <stdio.h>

int main() {
  int day;
    /* Days are represented as integers because switch does not accept
    strings */
  printf("Please enter a day\n");
  scanf("%d", &day);

  switch (day) {
   case 1: printf("Sunday\n"); break;
   case 2: printf("Monday\n"); break;
   case 3: printf("Tuesday\n"); break;
   case 4: printf("Wednesday\n"); break;
   case 5: printf("Thursday\n"); break;
   case 6: printf("Friday\n"); break;
   case 7: printf("Saturday\n"); break;
   default: printf("Unknown day\n");
  }
  return 0;
  }
```

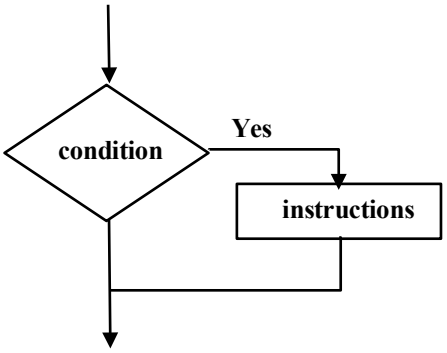
2.7 Flowchart Representation (Algorigram)

A flowchart is a graphical representation of the solution to a problem. It is similar to an algorithm. Each type of instruction in an algorithm has a corresponding representation in a flowchart. The symbols used in flowcharts are:

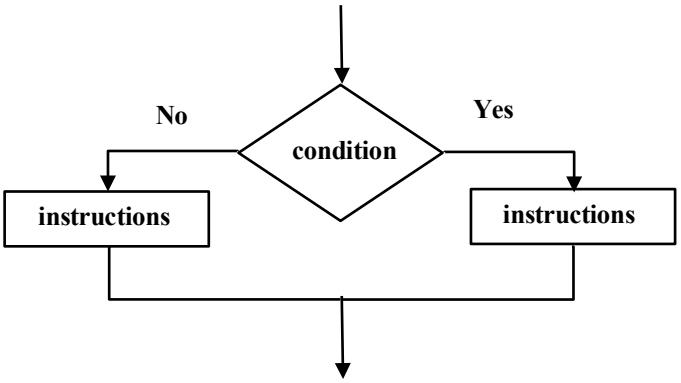
	Represents the begin and end of the flowchart.
	Input/Output: reading data and displaying results
	Calculations and processing
	Tests et decisions.
	Order of execution (flow of operations).
	Connector.

2.8 Conditional Structures in Flowcharts

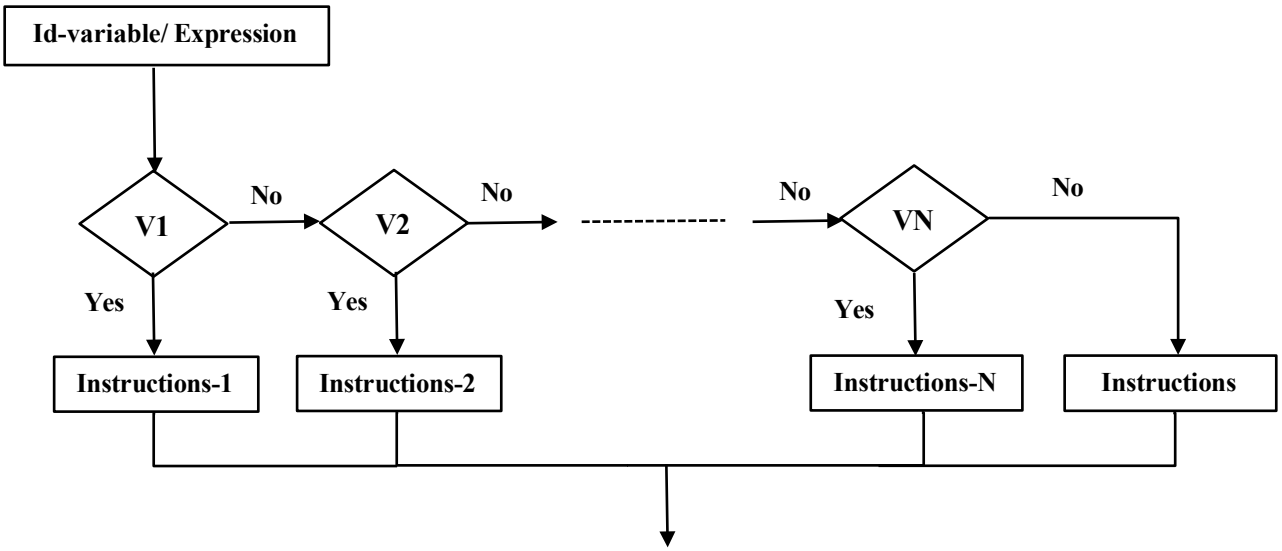
a. Simple alternative test



b. Double alternative test



C . Multiple choice (Case..of)



Conclusion

In this chapter, we studied conditional structures, which are essential elements that allow an algorithm to make decisions and adapt its behavior to different situations. Thanks to these structures, a program can now react intelligently to input data and choose the appropriate execution path.

However, many problems require not only making decisions, but also repeating certain actions several times. That is why, in the next chapter, we will study loop (repetition) structures**.**

Knowledge Assessment Test

1. Multiple Choice Questions (MCQs):

1.1 Which statement in C language allows a block to be executed only if a condition is true?
a) `while`
b) `if`
c) `for`
d) `switch`

1.2 What is the logical operator in C for "AND"?
a) `||`
b) `!`
c) `==`
d) `&&`

1.3 What structure allows us to deal with several cases based on the value of a variable?
a) `if`
b) `while`
c) `switch`
d) `do...while`

2. True or false:

2.1 The `else` statement is required after an `if`.
2.2 The `!=` operator means "equal to".
2.3 The `switch` structure is useful only for integer types and characters.

3. Open-ended questions / coding:

Using `switch`, write a program in C that displays the names of the wilayas of Algeria corresponding to their codes (Ex 6 Béjaïa).

Correction:

MCQ :
1.1 → b) `if`
1.2 → d) `&&`
1.3 → c) `switch`

True/False:
2.1 → False
2.2 → False
2.3 → True