Chapter 3 :

Conditional structures

1. Introduction

At this stage of the course, algorithms are simply sequences of instructions. This makes it difficult to describe complex algorithms.

However, only three algorithmic control structures are needed to describe any sequence of actions:

- Sequence: This structure allows us to chain together two or more instructions.
- Condition: This structure allows us to choose between two or more actions based on a condition.
- Iteration: This structure allows us to repeat an action as long as a condition is true.

Control structures are used to change the order in which instructions are executed in a program. This can be used to repeat instructions, to execute instructions only if certain conditions are met, or to do both.

2. Simple Conditional Structure

A conditional statement is a control structure that allows us to execute an action only if a condition is true. It is written in the following form:

If ‹condition› then ‹Instruction_Block› ;

Where:

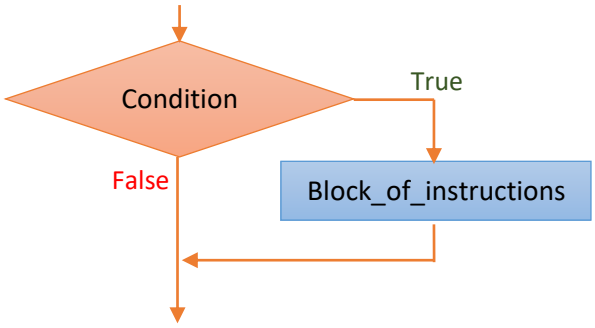
‹ *condition* › is a logical expression,

‹ *Instruction_Block* › is one or more instructions.

When a conditional statement is executed, the following actions are performed:

1. *The condition is evaluated.*
2. *If the evaluation returns an error, the statement is not executed and the program terminates. Otherwise, let E be the value of the condition.*
3. *If E is true, the block of instructions is executed.*
4. *If E is false, the block of instructions is not executed.*

Flow chart :



In C language :

```
if ( condition )
    {
        /* block_of_instructions */
     }
```

Note the absence of a semicolon (;) after the condition.

Exercise 1

Write an algorithm that prompts the user to enter an integer and prints "even" if the integer is even.

Exercise 2

Write an algorithm that inputs the grades for tutorial work, practical work, and final exam, calculates the average, and prints "Admitted" if the average is greater than or equal to 10.

Exercise 3

Write an algorithm that prompts the user to enter an integer and then calculates and prints its absolute value.

3. Alternating Conditional Structure

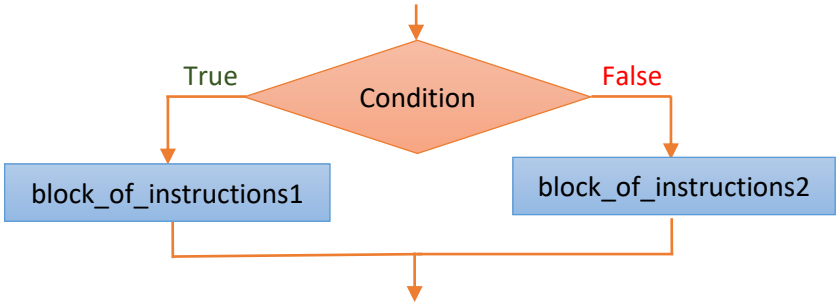
There is a type of conditional instruction called an alternating conditional instruction. This instruction executes a block of instructions if a condition is true, and another block of instructions if the condition is false. It is written in the following form:

If ‹ condition› then ‹ block_of_instructions1›
else ‹ block_of_instructions 2›;

The following actions are performed when an alternating conditional instruction statement is executed:

1. *The condition is evaluated.*
2. *If the evaluation results in an error, the statement is not executed and the program terminates. Otherwise, let E be the value of the condition.*
3. *If the value of E is true, block_of_instructions1 is executed.*
4. *If the value of E is false, block_of_instructions2 is executed.*

Flow chart:



In C language:

```
if ( condition )
     {
        /* bloc_d' instructions1*/
      }
else
     {
        /* bloc_d' instructions2*/
      }
```

Remarks

1. *If the block consists of only one statement, the braces are optional.*
2. *The condition can be a Boolean variable, which is a variable that can have one of two values: true or false.*

3. *It is possible to use multiple if statements in sequence to perform more complex tests. This is called nesting if statements. Nesting is the use of an if statement within the block of another if statement.*

Exercise 4

Write an algorithm that prompts the user to enter an integer and then prints "even" if the integer is even, and "odd" otherwise.

Exercise 5

Write an algorithm that inputs the grades for tutorial work, practical work, and final exam, calculates the average, and prints "Admitted" if the average is greater than or equal to 10, or displays "adjourned" if the average is less than 10.

Exercise 6

Write an algorithm to enter two real values and calculate and display the maximum.

Exercise 7

Write an algorithm that takes as input the temperature of water and outputs the state of the water, which can be gas, liquid, or solid.

Exercise 8

Write an algorithm that takes a given time (represented by hours and minutes) and calculates and displays the time after adding one minute.

For example, if the given time is 18:34, the output should be 18:35. If the given time is 18:59, the output should be 19:00.

4. Multiple Choice Conditional Structure

As we mentioned, a conditional statement is a statement that executes one or more other statements depending on the value of a Boolean expression. A conditional statement can be nested within another conditional statement, which means that the inner conditional statement is only executed if the outer conditional statement is true.

In some cases, we need to compare a variable to multiple different values. For example, we may want to perform a different processing depending on whether the value of a variable is 1, 2, 3, or more.

If we use the If statement, we would need to use nested conditional statements to achieve this.

For example, the following code would print the appropriate menu depending on the value of the variable x:


```
If x=1 then write ('fish menu')
Else  if x=2 then write ('vegetarian menu')
      Else  if x=3 then write ('traditional flat')
            Else  write ('error');
```

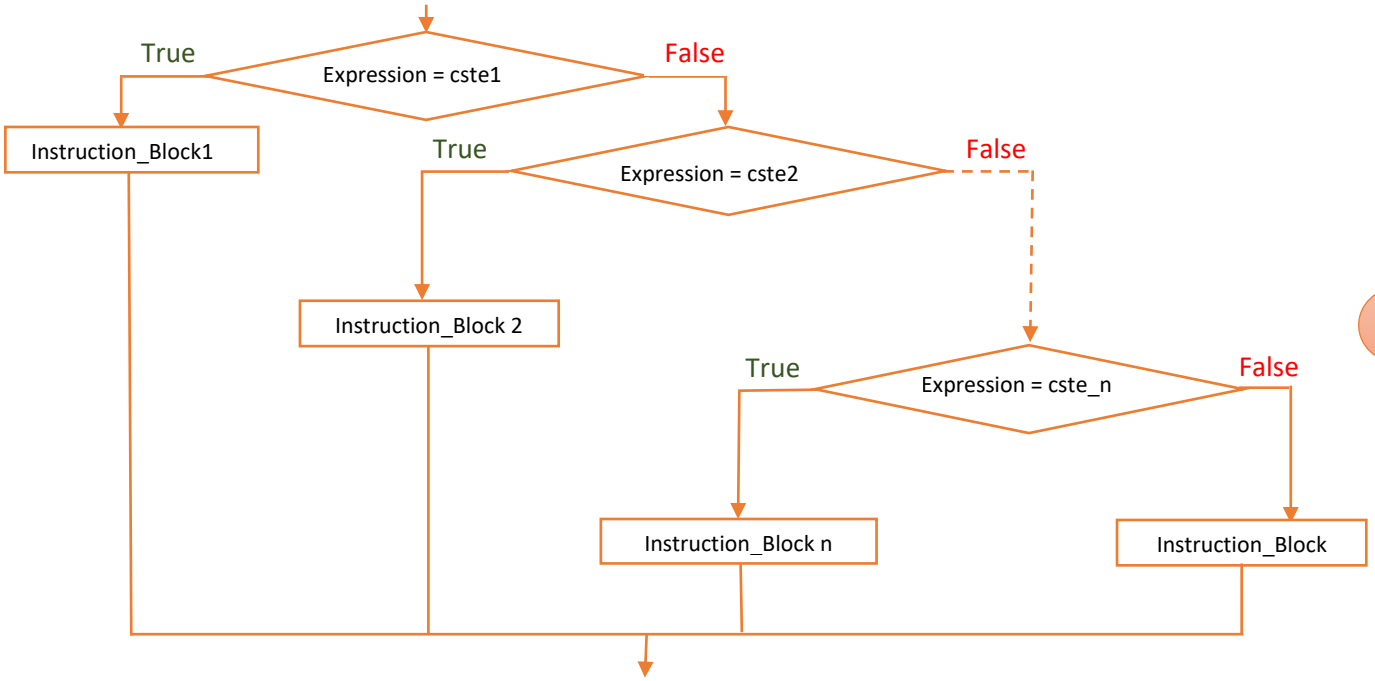
A more concise way to achieve this is to use the multiple-choice conditional statement, also known as the selection statement. This statement consists of an expression followed by a list of statements, each of which is preceded by a sequence of constants. It is written as follows:

```
Case ‹expression› of
‹constant1›: ‹Instruction_Block1›;
‹constant2›: ‹Instruction_Block2› ;
....
‹constant_n› : ‹ Instruction_Blockn› ;
[else: ‹ Instruction_Block›;]
End ;
```

The following steps are performed when a case statement is executed:

1. *The expression is evaluated.*
2. *f this evaluation returns Error, the instruction is not executed, the general execution terminates (it is necessary to avoid this kind of situation!). Otherwise, let E be the value of this condition..*
3. *If the value of E is equal to the first constant, the first block of code is executed.*
4. *If the value of E is equal to the second constant, the second block of code is executed.*
5. *This process continues until a constant is found that is equal to the value of E....*
6. *If the value of E is not equal to any of the constants, the Else block of code is executed, if it exists.*

Flow chart:



In C language:

```
switch ( expression ) {
case constante1 :
        /* bloc_d' instructions1*/ ;
         break ;
case constante2 :
        /* bloc_d' instructions2*/ ;
         break ;
…
case constant_i :
        /* bloc_d' instructions_i*/ ;
         break ;
…
[default :
        /* bloc_d' instructions */;]
}
```

Exercise 9

Write an algorithm that calculates the area of a geometric shape based on the user's choice. The shape can be a square, rectangle, or circle.

5. Branching instruction

The branching (or jump) statement allows you to jump to a statement in the program and continue execution from that statement.

To specify the destination of a branch, a label is used. Therefore, the branching instruction allows you to jump to a specific location in the program. This location is identified by a label.

A label is an identifier that is placed before a block of instructions. It is separated from the block by a colon character. Labels do not have to be declared explicitly; they are automatically recognized as labels when they are written before an instruction. Labels can be used anywhere within the block of instructions where they appear, but they are not recognized outside of the block.

It is written in the following form:

```
Label: instruction ;
```

And the branching instruction is written as follows:

```
Goto label ;
```

In algorithmics, branching is a powerful tool that can be used to control the flow of execution of a program. There are two types of branching instructions: unconditional and conditional.

Unconditional branch instructions are jumps that do not depend on any condition. They are used to skip over sections of code or to return to a previous point in the program.

Conditional branch instructions are jumps that depend on the value of a condition. They are used to execute different sections of code depending on the value of a variable or expression.

Unconditional branching statements are not very interesting in themselves. They are simply a way to jump to a new location in the program. However, conditional

branching statements are very powerful. They allow the programmer to control the flow of execution based on the value of a variable or other expression.

Branching statements can be used in three main ways:

- **Looping:** The branch instruction makes the program go back to the beginning to loop.
- **Jumping in the execution (label-end):** The branch instruction causes the machine to advance faster than expected in the execution of the program.
- **Returning to a previous piece of code:** Without necessarily having a loop, this would allow to execute a piece of code placed elsewhere in the program (this is the concept of procedure or function).

Remark

Branching instructions should be used with moderation in algorithms. They are often unnecessary and can lead to poor code quality.

In C language:

```
Label_name : Instruction Block ;
…
Goto Label_name;
```