Chapter 3: Control structures

1. Introduction

The control structures allow an algorithm (program) to not be purely sequential (linear chain of instructions).

In certain cases, the algorithm may:

- only execute specific instructions under certain conditions.
- repeat the same instructions several times.

In general, there are two types of control structures that allow this.

- The conditional structures: only execute certain instructions under certain conditions.
- The repetitive structures (loops): repeat the same instructions a certain number of times (under certain conditions)

2. Conditional structures

There are three types of conditional structures:

- Simple conditional structure: if. . .endif
- Compound conditional structure: if. . . . else . . . . endif.
- Multiple choice structure: AccordingTo. . else. . endAccordingTo.

2.1. Simple conditional structure

This structure allows to execute one or more instructions if a condition (a condition is an expression whose evaluation result can be TRUE or FALSE) is true and to exit the control structure if the condition is false.

The expression can be simple (single condition) or composed (several composed conditions with logical operators AND, OR, NOT).

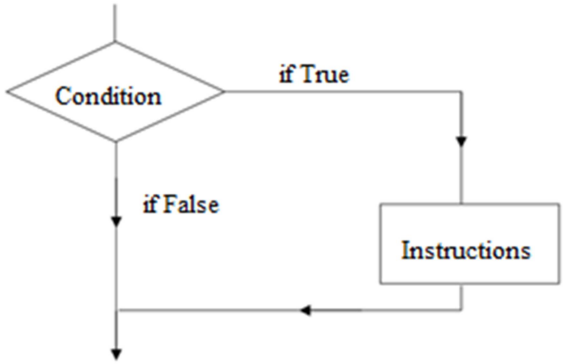
Syntax:

if (condition) **then**

Instructions ;

Endif ;

The simple conditional structure can be represented in a flowchart as follows:



A. Examples :

Exp 1 :

```
Algorithm positive;
Var
  X : integer ;
Begin
 read(X) ;
  if (X ≥ 0) then
    write(X) ;
  Endif ;
End.
```

Semantics: *We display the value of X if it is only greater than or equal to 0.*

Exp 2 :

```
Algorithm positive ;
Var
  X : integer ;
Begin
   Read(X) ;
   if (X < 0) then
     X ← X * ( -1);
   Endif ;
 Write(X) ;
End.
```

Semantics: *If the value of X is negative (less than 0) we multiply it by -1, the displayed result is always a positive value.*

Exp 3 :

```
Algorithm range ;
Var
  X : integer ;
Begin
  read(X) ;
  if ((X > 10) AND (X < 15)) then
    Write(X) ;
  Endif ;
End.
```

Semantics: *We display the value of X if it is between 10 and 15.*

Exp 4 :

```
Algorithm range ;
Var
  X : integer ;
Begin
 Read(X) ;
 if (X > 10) then
    if (X < 15) then
         write(X) ;
    endif ;
  endif;
end.
```

Semantics: *This example is equivalent to the previous example, except that this time we use nested tests.*

B. Simple conditional structure using C language:

The simple conditional structure is represented in C language as follows:

Algorithm	Program in C

if (condition) **then**
Instructions ;
endif ;
	if (condition)
	{
	Instructions ;
	}

Remark : If the bloc of instructions is composed of only one operation the brace symbol "{ }" become not obligatory.

Example :

Algorithm	**Program 1**	**Program 2**

Algorithme condition ;
Var
X : integer ;
Begin
read(X) ;
if (X > 10) then
write(X) ;
endif ;
end.
	int main ()
	{
	Int X;
	Scanf(''%d'' , &X);
	if (X > 10)
	{
	printf(X) ;
	}
	
	return 0 ;
	}
		Int main ()
		{
		Int X;
		Scanf(''%d'' , &X);
		if (X > 10)
		printf(X) ;
		
		return 0 ;
		}

Programs 1 and 2 are both correct.

Exercises :

1) Represent the examples 3 and 4 by a flowchart.
2) Study the flowing algorithms.

Exp a :	***Exp b :***

Begin
Read(X) ;
if (X < 10) then
if (X > 15) then
write(X) ;
endif ;
endif ;
end.
	Begin
	read(X) ;
	if (X < 10) OR (X > 15) then
	write(X) ;
	endif ;
	end.

Semantics :	***Semantics : ... .***

2.2. Compound conditional structure

Allows choosing between two blocks of instructions which ones to execute, according to the condition, if it is true, the first block of instructions is executed, otherwise, if it is false, the second block of instructions is executed.

Syntax:

if (condition) **then**

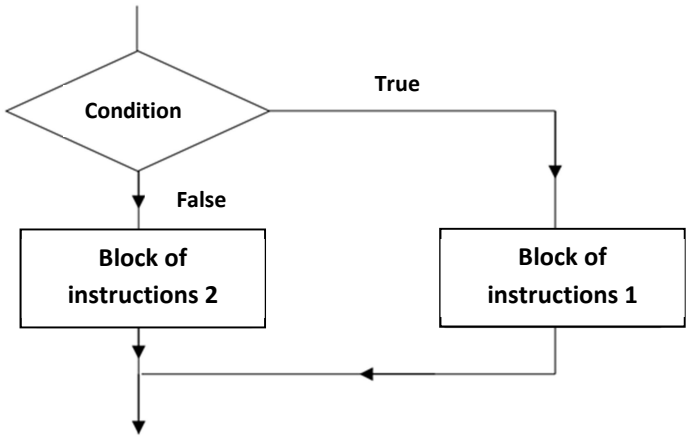
Block of instructions 1;

else

Block of instructions 2;

endif ;

The simple conditional structure can be represented in a flowchart as follows:



A. Translation to C programming language

The compound control structure is represented in C language as follows:

Algorithm	Program in C

if (condition) then
Block of instructions 1;
else
Block of instructions 2;
endif ;
	If (condition)
	{
	Block of instructions 1;
	}
	else
	{
	Block of instructions 2;
	}

Example:

Consider the following algorithm which displays whether a number is positive or negative.

```
Algorithm pos_neg;
var
    A: integer;
Begin
write ("Enter the value :");
read (A);
  if (A ≥ 0) then
    write (“positive”);
  else
    write (“negative”);
  endif ;
End.
```

Question: Translate the algorithm into C program.

Exercises:

1- Write an algorithm that displays the price of a printers order. The price of single printer (Canon 3110) is 50000 DA. From the fifth printer any other purchased will be counted 45000 DA. Translate the algorithm into C program.

Solution:

```
Algorithm order ;
  Var
    Nb, price : integer ;
 Begin
   read (nb);
   if (nb > 5) then
      price ←  (50000*5)+(45000 x (nb-5 )) ;
  else
      price ←  5000 x nb ;
  endif ;
  write(''the total price = '', price) ;
Fin.
```

2- Write an algorithm that displays the set of solutions on R of the first degree equation ax + b = 0

Solution:

```
Algorithm  first_degree;
Var
      a,b:integer;
Begin
   Read(a,b);
   If (a≠0) Then
        Write("The solution of the equation = ",-b/a);
   else
     if (b=0) Then
          Write("The solution of the equation is indeterminate");
     else
          Write("The solution to the equation is impossible");
     endif ;
   endif ;
end.
```

2.3. Multiple choice conditional structures

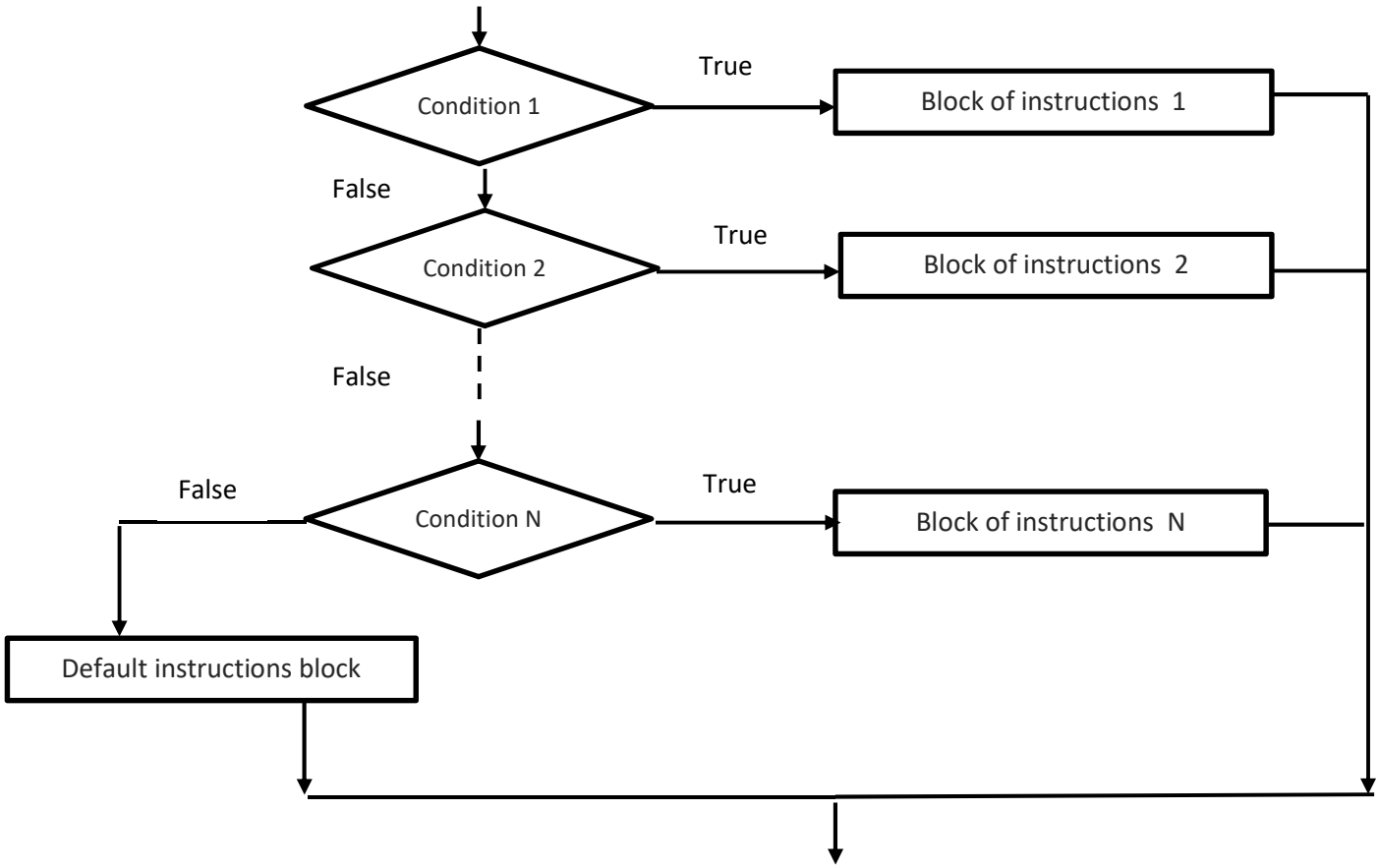
The multiple choice structure allows comparing an object (variable or expression) to a set of values, and executes one of several blocks of instructions, depending on the actual value of the object. A default statement block may be provided in the case where the object is not equal to any of the listed values.

This structure avoids the need for nested conditional structures and offers better comprehension of the solution.

Syntax :

Algorithm	Program in C
AccordingTo (variable ou expression) do Value 1 : block of instructions 1 ; Value 2 : block of instructions 2 ; …… Value N : block of instructions N ; **Else** default instructions block; **endAccordingTo;**	**switch(x)** **{** case value_1 : block of instructions 1 ; Break; case value_2 : block of instructions 2 ; Break; …….. case value_N : block of instructions N ; Break; **default:** default instructions block; **}**

The multiple choice conditional structure can be represented in a flowchart as follows:



Remarks:

- The default line is optional
- In the multiple choice instruction, the order of presentation changes nothing.
- The expression and the cases values must be in the same type.

Example:

In the following example, the value of X is displayed in letters, if it is equal to 1, 2 or 3 otherwise it displays an error message.

```
Algorithm display_letter;
  Var
    X: integer;
  Begin
    Read(X);
    AccordingTo (X) do
      1: write (“One”);
      2: write (“Two”);
      3: write (“Three”);
      Else  write(‘’The value entered is: > 3 or < 1’’);
     endAccordingTo;
 end.
```

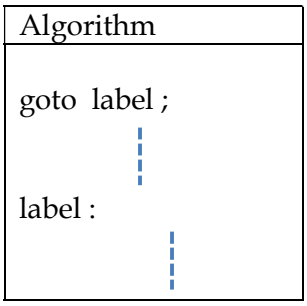
Program in C :

```
#include <stdio.h>
Int main ()
{
  Int x;
  Scanf(‘’%d’’,&x);
  switch(x)
  {
    case 1 : printf(“one”);
             Break;
    case 2 : printf(“two”);
             Break;
    case 3 : printf(“Three”);
             Break;
   default:
             printf(“the value entered is : > 3 or < 1”);
  }
```

2.4. Branching operator

It is possible to jump directly to an instruction using a branch operation of the form **goto identifier**. The identifiers are labels or addresses of branching.

Syntax:



Example :

```
Algorithm branching;
  Var
        a: integer;
 begin
        read (a);
        goto 1;
        a ← a+1 ;
        1: a ← a+2;
        write (a) ;
 End.
```

If the value of a = 5 the displayed result will be 7.

Note :

The use of the go to branch should be avoided, because it reduces the readability of the codes and often leads to semantic errors.

Exercise: Consider the following algorithm:

```
Algorithm Branching;
  Var
     a : integer;
 begin
    label1: write (" Enter an integer number :");
    read (a);
    if (a mod 2 = 0) then
       goto label2; // conditional branching
     endif ;
    write("the number entred is odd !");
    goto label1; // unconditional branching
    label2: write ("the number is even");
 end.
```

Write the algorithm in C language ?

Solution :

```
#include<stdio.h>
void main()
{
int a,b;
label1: printf (" Enter an integer number :"):
scanf("%d",&a);
if (a % 2 == 0)
 {
   goto label2 ;
 }
  printf("\n the number entred is odd  ! \n ");
  goto label1 ;
  label2 : printf("\n the number is even \n ");
}
```

Exercise 1:

Using the branch instruction, write an algorithm that calculates the sum of two real numbers entered on the keyboard at the request of the user, it means, repeat the steps by asking the user at the end of each calculation whether he wishes to repeat the sum calculation operation (Yes/No) or not.

- Translate into a C program.

Solution in C language :

```
#include<stdio.h>
void main()
{
        int a,b,s,c;

        et1:
           scanf("%d",&a);
           scanf("%d",&b);
           s=a+b;
           printf("la somme = %d \n",s);
           printf("do you want to repeat the calculation operation ?  1 : Yes, 2 : No");
           scanf("%d", &c);
           if(c==1)
                 goto et1;

           printf("\n Thank you \n");
}
```

Exercise 2 :

Write an algorithm that asks the user for two integers and, without calculating the product, it displays whether the product is negative, positive or equal to zero.

- Translate into a C program.