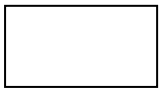
Chapter 3: Conditional Structures

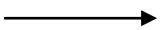
1. Introduction :

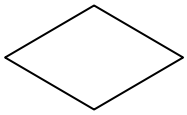
- Control structures also called structured instructions (actions).
  They make it possible to express the way of the sequence of execution of the instructions of an algorithm.
- There are three fundamental structures:
  1) Sequential actions,
  2) Conditional actions,
  3) Repeat actions.
- To describe these structures we use a textual notation (algorithm) and a graphic notation (flowchart).
- In a flowchart, the following symbols are used:


Represents the start and the end of the flowchart


Inputs / Outputs: Reading of data and writing of results.


Represents Actions (processing)


Represents the order of execution of operations (Sequence)

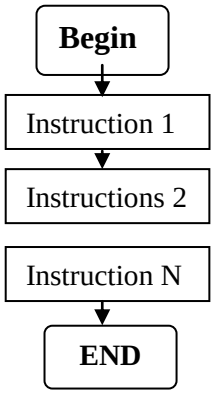

Represent conditions (Testing and decision)

2. Sequential actions:

- Sequential Actions come in the form of an ordered sequence of instructions grouped together in a block.

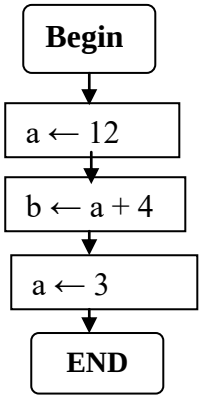
Syntax:

Begin
Instruction 1;
Instruction 2;
…… ……
Instruction N;
END.



Example :

Algorithm exp_sequ
a, b: integer;
Begin
a← 12; (Instruction 1)
b←a+4; (Instruction 2)
a ← 3; (Instruction 3)
END.



Execution trace:

	a	**b**
begin	**?**	**?**
a← 12;	**12**	**?**
b←a+4;	**12**	**16**
a ← 3;	**3**	**16**
END.	**3**	**16**

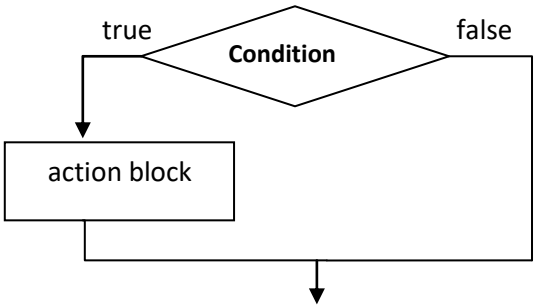
3. Conditional action:

3.1. Simple conditional action:

- It consists of two parts: condition and action.
  - ✓ The (condition) part describes a state which can be true or false (Boolean type expression).
  - ✓ The <Action Block> part represents a piece of an algorithm (one or more instructions).

Syntax:

…….
If (condition) then
< Block of actions (instructions)>
End if
……



Execution of the conditional action:

- If the condition is checked (**true**), the instructions of the < Block of actions > are executed then we continue the execution of the actions (instructions) located after the **End if**.
- If the condition is not verified (**false**), the part < Block of actions > inside the **If** is not executed and the execution of the algorithm is continued directly from the instruction which follows the **End if**.

Example: Write an algorithm that reads a real number identified by 'Nbr', then gives its absolute value.

Algorithm val_abs

Nbr: real;

Begin

Read (Nbr);

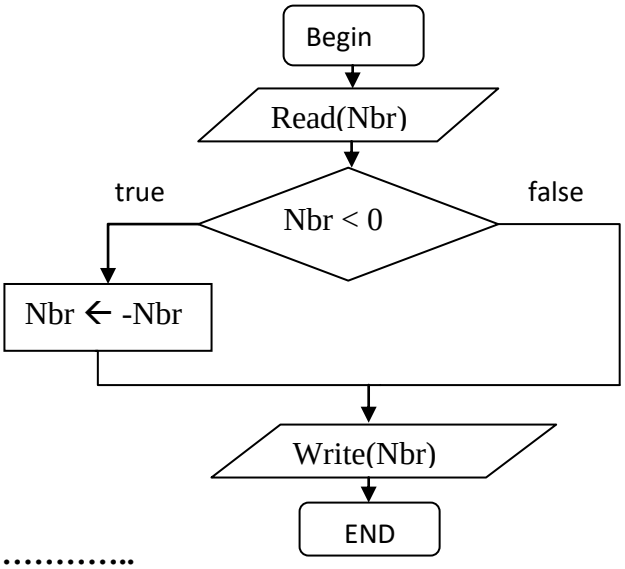
If (Nbr < 0) then

Nbr ← - Nbr ;

End if

Write (Nbr);

END.



Syntax in C++ language:

…….	……..
If (condition) **then**	**if** (condition) **{**
< Actions block (instructions)>	< Actions block (instructions)>
End if	**}**

Noticed :

- The condition is a Boolean type expression so it must include at least one comparison operator (<, >, =, ≠, etc.) or a Boolean variable.

3.2. Alternative action:

Syntax:

……

If (condition) **then**

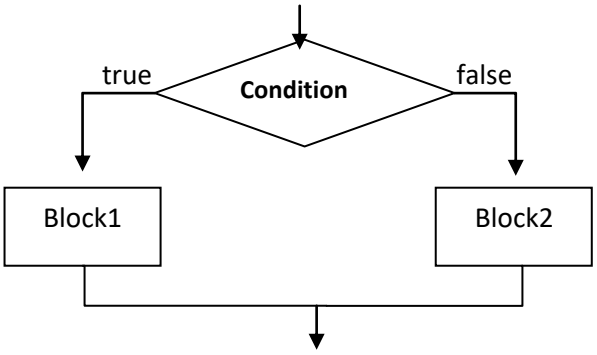
< Block of actions**1**(instructions) >

else

< Block of actions**2**(instructions) >

End if

……



Example: Write an algorithm that reads two real numbers and then determines the biggest of them?

Algorithm biggestNbr

x, y,Max: real;

Begin

read(x);

read(y);

If (x > y) then

Max←x;

else
Max←y;
End if
Write ('The biggest number is:', Max);
END.

Syntax in C++ language

……	**……**
If(condition) **then**	**if**(condition) {
< Actions block1 (instructions) >	< Actions block1 (instructions) >
	}
else	**else**
	{
< Actions block2 (instructions) >	< Actions block2 (instructions) >
End if	**}**
……	**…….**

Noticed:

- We can see an entire control structure as a single action (< Action block >) so there can be several **nested** control structures.

If (condition1) then
If (condition2) then
< Block of actions**1** (instructions) >
Else
< Block of actions**3**(instructions) >
End if
Else
< Block of actions**2**(instructions) >
End if

Example : Write an algorithm that reads two real numbers and then determines the biggest of them?

Algorithm biggest
x, y, Max: real;
Begin
Read (x, y);
If (x = y) then
Write (' Both are equal');
Else //x ≠ y
If (x > y) **then**
Max← x;
Write ('The biggest number is:', Max);

Else //x < y
Max←y;
Write ('The biggest number is:', Max);
End if
End if
END.

3.3. Multiple choice action:

- It makes it possible to distinguish several cases according to the values of an expression.
- The " **if** " allows to distinguish just two cases whereas the « **switch**» allows to distinguish a large number of cases.

Syntax:

Switch (expression**)**

Case 1: <action block 1>

Case 2: <action block 2>

……..

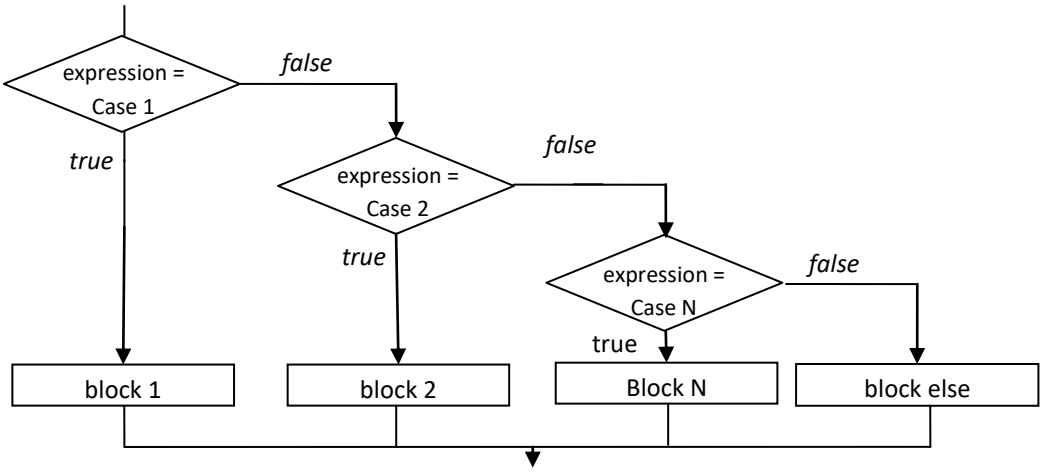
Case N: <action block N>

Else // else is optional

< else action block>

end switch

The organization chart of the " **switch** " structure is as follows:



Syntax in C++ language

Switch (expression**)**
Case 1:<action block 1>
Case 2:<action block 2>

Switch (expression**) {**
Case val1: { <action block 1> break; }
Case val2: { <action block 2> break;}

…….. ……..

Case N:<action block N> **Case** valN: { <action block N> break;}

Else // else is optional **default**

<else action block> **{**<default action block> break; }

end case **}**

Noticed :the " break; " is necessary to exit the structure depending on the case once the corresponding action block has been executed; so as not to test the other cases, but it is not an obligatory.

Example : Write the algorithm that reads a number (0, 1, 2, 3, 4) then gives its name (zero, one, two, three, four)?

Nested Alternate Action	**Multiple choice action**

Algorithm Number_name
n: integer;
Begin
write ('enter a number between 0 and 4:');
Read (n);
If (n=0) Then
write ('Zero');
else
If (n=1) Then
write ('A');
else
If (n=2) Then
write ('Two');
else
If (n=3) Then
write ('Three');
else
If (n=4) Then
write ('Four');
else
write ('error');
End if
End if
End if
End if
End if
END.
	Algorithm Number_name
	n: integer;
	Begin
	write ('enter a number between 0 and 4:');
	Read (n);
	switch (n)
	0: Write ('Zero');
	1: Write ('One');
	2: Write ('Two');
	3: Write ('Three');
	4: Write ('Four');
	else
	Write ('error');
	end switch
	END.