REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université de M'sila
Faculté des Mathématiques et de l'Informatique
Département d'informatique

جامعة المسيلة
كلية الرياضيات والإعلام الآلي
قسم الإعلام الآلي

Chapter 3:
Conditional statements

Algorithms and data structure

Presented by : Dr. Benazi Makhlouf
Academic year : 2023/2024

Contents of chapter 02:

1. Introduction

The program is a set of instructions. Most of these instructions are executed in the order they appear.

Control structures are instructions that modify the sequential flow of the program, such as conditional structures, loops, function calls, and unconditional jump instructions.

Conditional structures:

We often encounter cases where we need to decide whether an instruction should be executed or not, depending on whether the condition is true or false.

There are three types of conditional structures:

- simple (if-then)
- compound (if-then-else)
- multiple-choice structure (switch).

2. The simple conditional structure "if-then"

It consists of two parts:

- **Condition**: a boolean expression whose value is either true or false.
- **Block of instructions**: executed if the condition is true, or ignored if the condition is false.

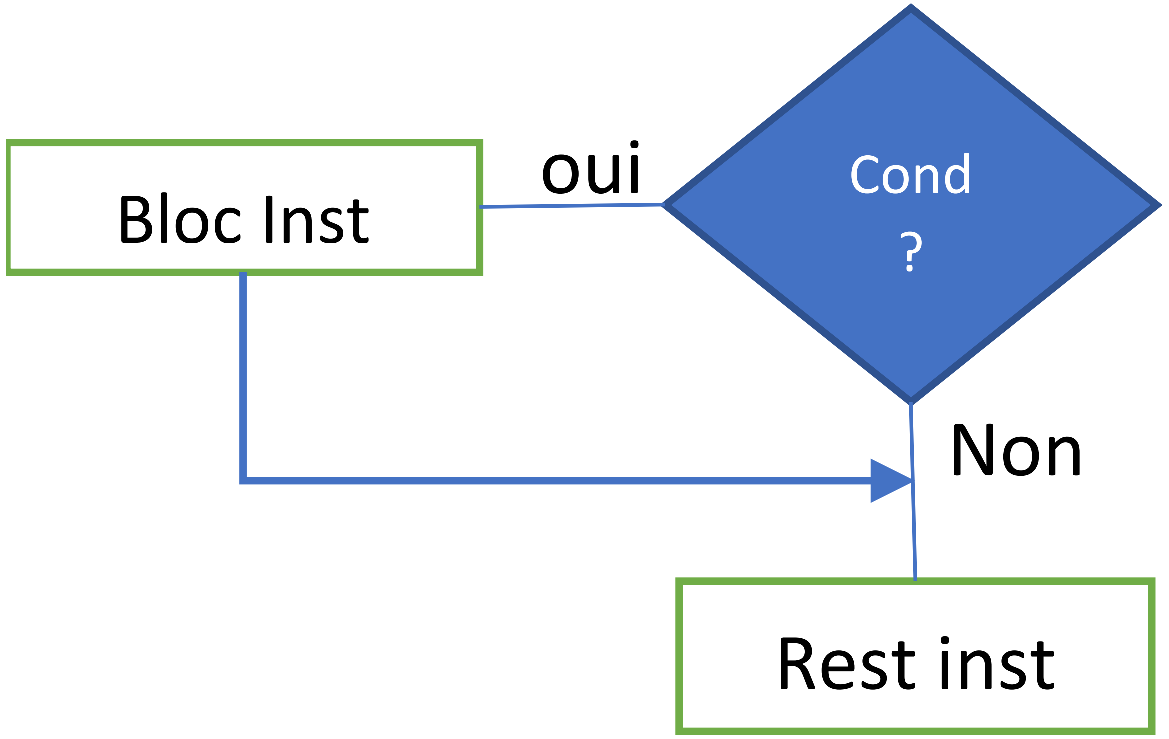
Syntax Algorithm

```
If Condition Then
 Block of instructions
EndIf
The rest of the instructions
```

Comments :

- `If` specifies what to do if the condition is true, but not what to do if it's false.
- **condition** is always between the words **if** and **then**
- To construct the condition, we use comparison operations (>, <, = (==), ≠ (!=), ...) and logical operations (and (&&), or(||), not(!), ...).

Algorigram



C language

Syntaxe

```
if (Condition)
{
   Block of instructions
}
The rest of the instructions
```

Comments :

- The **condition** is always enclosed in parentheses **()**.
- Instructions belonging to if in C are surrounded by two curly braces {}.
- curly braces {} can be omitted if they contain only one instruction.
- In C, the Boolean type is expressed as an int. Where false is expressed as 0 and true is any number other than 0. ($\neq 0$)
- There is no "; "after }.

Example

Write a program that reads an integer, then displays a warning if it's negative, then shows us its square.

Algorithm

```
algorithm root
var x :integer
Begin
 write ("enter a nbr")
 read (x)
 If x<0 then
  write ("nbr is negative")
 End If
 write ("the square is " , x*x)
End
```

C

```
#include <stdio.h>
int main()
{
 int x ;
 printf("enter a nbr \n") ;
 scanf("%d", &x) ;
 if ( x<0 )
 {// can be exempted
   printf("nbr is negative \n") ;
 }
 printf("the square is %d" , x*x) ;
}
```

The compound conditional structure "if then else"

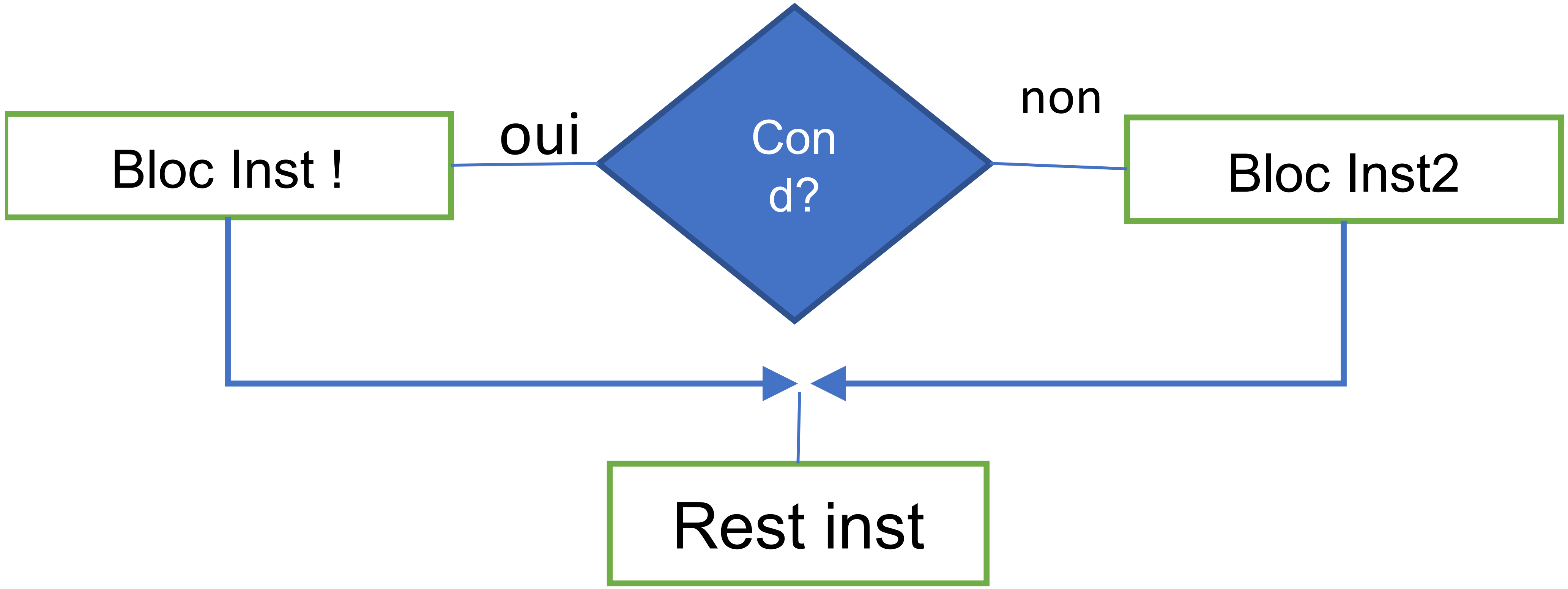
- The “if.. then... else...” structure is an extension of if simple. The compound conditional instruction “if else” has three parts:
- **Condition** : a Boolean expression whose value is **true** or **false**.
- **First block** of instructions : It is executed if the condition is true, or ignored if the condition is false.
- **Second block** of instructions : It is executed if the condition is false, or ignored if the condition is true condition true.

Syntax

Algorithm	C

If Condition **then**
  instruction block 1
else
  instruction block 2
End If
The rest of the instructions
	if (Condition**)**
	{
	  instruction block 1
	}
	else
	{
	  instruction block 2
	}
	The rest of the instructions

Algorigram



Example

Write a program that calculates the absolute value of an integer, then displays it on the screen.

Algorithm

```
algorithm absolute
var x, y :entier
begin
 write("enter a nbr")
 read(x)
 If x>=0 then
  y←x
 else
  y←-x
 End If
 write("|" , x ,"|=",y)
end
```

C

```
#include <stdio.h>
int main(){
 int x, y ;
 printf("enter a nbr \n") ;
 scanf("%d", &x) ;
 if ( x>=0 )
 { // can be deleted
   y=x ;
 }
 else
 { // can be deleted
   y=-x ;
 }
 printf(("|%d|=%d", x, y)) ;
}
```

```
if ( x>=0 )
  y=x ;
else
  y=-x ;
```

Conditional assignment in C

If we have a variable v takes one of the values v1 or v2 depending on condition b, i.e. :

```
if (b)
  v=v1 ;
else
  v=v2 ;
```

In this case, the " … ? … : … " can be used and its syntax is as follows:

```
            condition ? expression_if_true  : expression_if_false
```

- **condition** est une condition de type booléen
- **expression_if_true** Value returned if the condition is true.
- **expression_if_false** Value returned if the condition is false

Example

- `v=b ? v1 : v2 ;`
- `result= average >=10 ? "Admitted" : "Failed" ;`

If-else extension

If-else can be used to test multiple conditions and select the appropriate treatment for each case. For example, to determine if a student is admitted or not, there are several cases. Either the student is admitted without compensation, or admitted with compensation, or admitted with debts, or failed. To determine this, we need to look at the averages of the first and second semesters (s1 and s2), the annual average (MA), and the total credits earned (Crd).

```
If s1>=10 and s2>=10 then
 write("admitted without compensation")
 else
 If MA>=10 then
  write("admitted with compensation ")
 else
  If Crd>=45 then
   write("admitted with debts ")
  else
   write("adjourned ")
  End If
 End If
End If
```

```
if ( s1>=10 && s2>=10 )
  printf("admitted without compensation");
else if ( MA>=10 )
  printf("admitted with compensation ") ;
else if ( Crd>=45 )
  printf("admitted with debts ") ;
else
  printf("adjourned ") ;
```

The multiple choice conditional structure "switch"

The **switch** statement: is a specific case of nested if-else statements. It allows determining the block of code to be executed based on the value of a single expression. It is used when testing the same expression multiple times.

The switch statement consists of:

- The **expression** to be tested: usually of integer or character type, and it is typically a variable. For example: age.
- The **values** to be tested with their corresponding block of instructions for each value.
- An optional **default block** of instructions that will be executed if the current value does not match any of the values mentioned previously.

Syntax

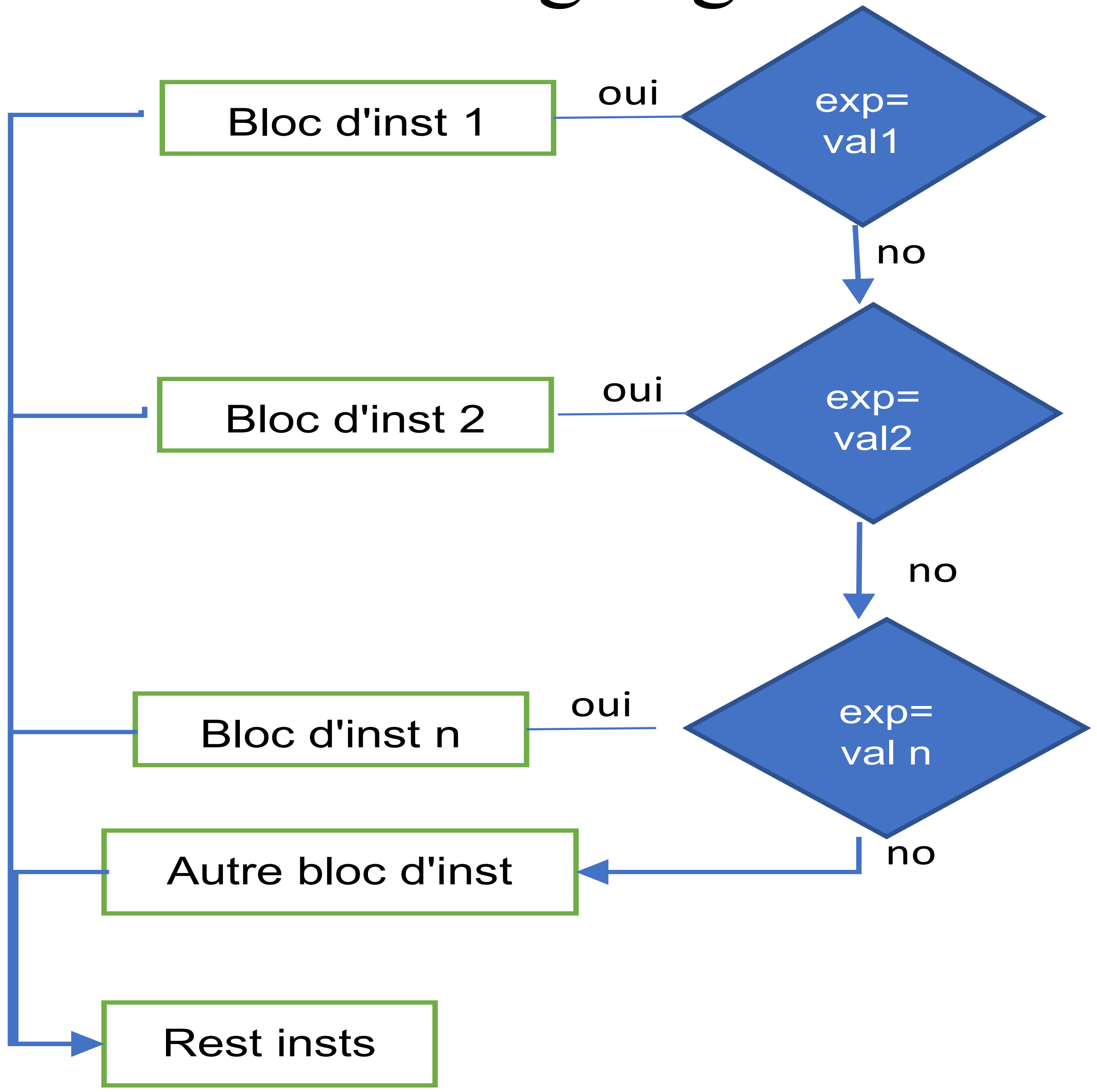
Algorithm	**C**
case expression **of**	**switch (** expression **)** {
val_1 : instruction block 1	**case** val_1 **:**
val_2 : instruction block 2	instruction block 1
…	**break ;**
val_n : instruction block n	…
else	**case** val_n **:**
another instruction block	instruction block n;
end case	**break ;**
The rest	**default:**
	another instruction block
	}
	The rest

–**expression** : calculated to obtain an integer or character value. Usually a variable.

–**val_1, ..., val_n** : a value or constant of the same type as the expression.

–**Instruction block** : one or more instructions that are executed if the value of the expression matches val_i.

Algorigram


Bloc d'inst 1
oui
exp= val1
no
Bloc d'inst 2
oui
exp= val2
no
Bloc d'inst n
oui
exp= val n
no
Autre bloc d'inst
Rest insts

The rules concerning the "switch" statement in C

The execution of the "**switch**" statement in C differs slightly from the “**case**" statement in algorithms. In C, after the block of "**case val_i**" has been executed, and if the execution does not encounter the "**break;**" statement, it will continue to execute the following block until it encounters the "**break;**" statement. The execution then proceeds to the rest of the instructions outside the "**switch**" block.

To make the "switch" statement in C equivalent to the “**case**" statement in algorithms, it is necessary to add the "**break;**" statement at the end of each block.

1. The curly braces {} and the parentheses () are required in C and cannot be removed.
2. The expression inside the "switch" must be of integer or character type.
3. Each "case" label inside the "switch" must be a constant expression.
4. Each val_i value must be different from the other. For example, it is illegal to write case 1 twice.
5. The case val_i can be placed in any order. However, it is recommended to arrange them in ascending order. This improves the readability of the program.
6. A block of statements can contain any number and type of instructions.
7. The break; statement is optional. It is used to exit a switch immediately, moving the program flow out of the switch.
8. The default block is optional. If none of the case val_i matches, the execution context will be moved to the default block. It should be the last case.

If two or more values have the same block of instructions, the algorithm can use a comma. In C, however, we use the first value without any instructions or **break;** before the next value with the shared block of instructions.

Example **The algorithm**	**C**

`7 ,9 : instruction block`	`case 7 :`
	`case 9 :`
	`  instruction block`
	`  break ;`

Example

Write a program that reads an integer less than 10, then displays the number in letters on the screen in English.

Algorithme	C
algorithm conversion	**#include** <stdio.h>
var nb :integer	**int** main(){
begin	int nb ;
write("enter a nbr")	printf("enter a nbr \n") ;
read(nb)	scanf("%d", &nb) ;
if nb=0 **than**	**if (** nb==0 **)**
écrire("zero")	printf("zero") ;
else if nb=1 **than**	**else if (** nb==1 **)**
write ("one")	printf("one") ;
else if nb=2 **than**	…
write("two")	**else if (** nb==9 **)**
…	printf("nine") ;
else	**else**
write ("not treated")	printf("not treated") ;
End if	**return 0** ;
…	}
End if	

Example

Write a program that reads an integer less than 10, then displays the number in letters on the screen in English.

Algorithm

```
algorithm conversion
var nb :integer
begin
 write("enter a nbr")
 read(nb)
 case nb of
   0 : write ("zero")
   1 : write ("one")
   2 : write ("two")
   …
   9 : write ("nine")
   else
     write ("not treated")
 end case
End
```

C

```c
#include <stdio.h>
int main(){
  int nb ;
  printf("enter a nbr \n") ;
  scanf("%d", &nb ) ;
  switch ( nb ) {
  case 0 : printf("zero") ;
           break ;
  case 1 : printf("one") ;
           break ;
  …
  case 9 : printf("nine") ;
         break ;
  default:
         printf("not treated") ;
  }
  return 0 ;
}
```

Branching statement

This is the process of jumping between program instructions executed by the processor, where it performs a "jump" to a specific address instead of continuing to execute instructions sequentially.

There are four instructions in C that can unconditionally change the flow of execution of a program:

break, **goto**, **continue**, and **return**.

break; statement

- The "**break;**" statement is used to exit the "**switch**" statement, moving the flow of execution to the first instruction after the "**switch**". In the case of a **nested** "**switch**" statement, it will only exit the innermost "**switch**" it is directly associated with.
- It is also used to exit **loops**. In this case, "**break**;" is usually inside an "**if**" statement.

Example

```
switch (grade){
  case ‘A’ :
  case ‘a’ : printf(" excellent \n") ;
             break ;
  case ‘b’ : printf("good\n") ;
  case ‘c’ : printf("you can do better\n") ;
             break ;
  default : printf("try again\n") ;
}
```

goto statement

The execution of the program can be redirected to a named instruction. Any instruction can be named with a valid ID ("label") followed by a colon ":" before it.

```
label : instruction;
```

To access this instruction from anywhere, we use the following syntax:

```
goto label ;
```

Example:

```
here: printf("zero");
```

To access the instruction "here" from anywhere, we use:

```
goto here;
```

continue; statement

It is used with loops and allows the flow to be moved to the end of the loop and to directly proceed to the next iteration without completing the loop.

return statement

It is used to exit functions (semester 2) and return the result.

Syntax

```
return expression ;
```

Example

```
return 0 ;
```

End Chapter 03