M'hamed Bougara University - Boumerdes
Faculty of Sciences
Computer Science Department

Algorithms and Data Structures 1

L1 Computer Science

Dr RAHMOUNE Nabila

2023/2024

Chapter 2
Control Structures

Outline

- Introduction
- Conditional Statements
- Repeat Instructions
- Applications

Introduction

- We are going to introduce two extremely used instructions which make it possible to build an algorithm with nonlinear progress.
- **The conditional instruction** allows to execute or not a block of instructions.
- **The loop** makes it possible to go back in the algorithm, to reiterate a specific number of times the execution of a block.
- **Boolean** type variable (true or false), which conditions the rest of its progress.

Conditional Statement

The syntax

- The conditional instruction now allows us to design an algorithm that will not execute certain instruction blocks.

The conditional

The conditional statement determines whether the next block of statements is executed or not.

The condition is a boolean expression whose value determines the block of executed instructions .

Conditional Statement

The syntax

- The syntax of this statement is:

```
IF(condition) then
{
  Instruction block #1; //executed if condition equals True
}
ELSE
{
  Instruction block #2; //executed if condition equal False
}
```

- One of the two blocks is necessarily executed, the other will not be.
- Let's write the algorithm that reads two integers and outputs the larger of the two.

Conditional Statement

Max-of-two-integers algorithm

Var x,y, max: integer;	Variables have no known value at start
Begin	x=? y=? max=?
Read(x);	x= 5 y= ? max=?
read(y);	x= 5 y= 7 max= ?
if (x > y) then	The condition is evaluated: (5>7) (False)
{	This block is not executed
max ← x;	This block is not executed
}	This block is not executed
else	
{	This block is executed
max ← y;	x= 5 y= 7 max= 7
}	End of conditional block
write(" maximum: ", max);	Maximum display: 7
END	The variables no longer exist

Conditional Statement

Apps

- **The simple conditional**
- A simpler version is used if the alternative does not take place. The syntax of this instruction is then:

```
If (condition) then
{
instructions ;
 }
```

- Let's write an algorithm that reads an integer and displays its positive value.

Conditional Statement

Apps

- **The simple conditional**

Positive-value algorithm

```
Var value, positive: integer;
Begin
 read(value);
 positive ← value ;
if (positive<0) then
{
positive ← -1 x positive;
}
write(' the positive value is: ', positive);
END
```

Positive-value algorithm

```
Var   value: integer;
Begin
read(value);
if (value<0) then
{
value ← -1 x value;
}
write(' the positive value is: ', value);
END
```

Conditional Statement

The presentation

- The shifts in the writing of an algorithm (or of the program) are necessary for its good readability.
- To know how to present an algorithm is to show that we have understood its execution.

Max-of-Two-Integers Algorithm

```
Var  x,y, max: integer;
Begin
Read(x);
read(y);
if (x>y) then
{
max ← x ;
}
else
max ← y ;
write("the maximum is:", max);
END
```

Max-of-Two-Integers Algorithm

```
Var  x,y, max: integer;
Begin
Read(x);
read(y);
if (x>y) then
{
  max ← x ;
}
else
  max ← y ;
write("the maximum is:", max);
END
```

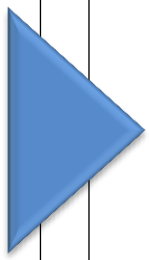
Conditional Statement

The presentation

- The shifts in the writing of an algorithm (or of the program) are necessary for its good readability.
- To know how to present an algorithm is to show that we have understood its execution.

Max-of-Two-Integers Algorithm

```
Var  x,y, max: integer;
Begin
read(x);
read(y);
if (x>y) then
{
max ← x ;
}
else
max ← y ;
write("the maximum is:", max);
END
```



Max-of-Two-Integers Algorithm

```
Var  x,y, max: integer;
Begin
read(x);
read(y);
if (x>y) then
{
 max ← x ;
}
else
  max ← y ;
write("the maximum is:", max);
END
```

Nestred Conditionnals

Usage

Example

- It is possible to nest program blocks one inside the other. Let's try to solve the problem of making the user read a note and displaying the comment associated with the note:
  - score from 0 to 8 inclusive: “Insufficient”;
  - score from 8 to 12 inclusive: “average”;
  - score from 12 to 16 inclusive: “ good ”;
  - rating from 16 to 20 inclusive: “very good”;

Nestred Conditionnals

The classic solution:

Comments-notes algorithm

```
Var
note: integer;
Begin
read(note);
 if (note ≤ 8) then
write("insufficient");
if (score > 8) AND (score ≤ 12) then
write(“medium”);
 if (score > 12) AND (score ≤ 16 ) then
write well ") ;
 whether (score>16) then
write("very good");
END.
```

Nestred Conditionnals

It is possible not to respect the presentation with tabs:

Comments-notes-best-bis algorithm
Var
note: integer;
Begin
read(note);
if (score ← 8) **then**
write("insufficient");
else if (score ← 12) **then**
write(“medium”);
else if (score ← 16) **then**
write well ") ;
Otherwise
write("very good");
END

Nestred Conditionnals

More elegant solution:

Comments-Ratings-Better Algorithm
Var
note: integer;
Begin
read(note);
if (score ≤ 8) **then**
write (“insufficient”);
else if (score ≤ 12) **then**
write (“medium”);
else if (score ≤ 16) **then**
write (“well”);
Otherwise
write (“very well”);
END

Nestred Conditionnals

Errors to evade

It is very common for beginners to forget the otherwise intermediate instructions.

```
Comments-notes-false algorithm
Var
note: integer;
Begin
read(note);
if (note ≤ 8) then
write("insufficient");
if (note ≤ 12) then
write(“medium”);
if (score ≤ 16) then
write well ") ;
Otherwise
write("very good");
END
```

Nestred Conditionnals

- It is essential to write an algorithm as readable and clear as possible.

```
If (condition) then
{
 Instruction block #1;
}
Otherwise
{
 If (condition) then
 {
Instruction block #2;
 }
 Otherwise
 {
 Instruction block #3;
 }
}
```

Repeat Instruction

The while loop

Syntax:
while *(prosecution_condition* **) do**
{
Instruction block
}

Definition

The loop

The repetition instruction, called a loop, allows the same block of instructions to be executed several times consecutively. The repetition is performed as long as the value of the Boolean expression is equal to True.

Repeat Instruction

- The following algorithm displays the integers from 1 to 5 on the screen.

```
display-of-the-first-5-integers algorithm
Variables: counter: integer;
Begin
counter ← 1; // initialization
 while (counter ≤ 5 ) do //continuation condition
{ //start of block
write(counter); //treatment
  counter ← counter + 1; //increment the counter
} //end of block
END
```

Repeat Instruction

- Let's represent the progress of the values of the variable and the boolean condition .

Counter	Condition: (counter ≤ 5)	Continuity condition
1	Initialization: before entering the loop	
1	1 ≤ 5: True	Enter the loop
2	2 ≤ 5: True	One more round
3	3 ≤ 5: True	One more round
4	4 ≤ 5: True	One more turn
5	5 ≤ 5: True	One more turn
6	6 ≤ 5: False	Get out of the loop

Repeat Instruction

Several equivalent algorithms

- The following conditions make it possible to exit the previous loop:
  - Stopping the loop when counter = 6, then the condition while(counter $\neq$ 6) make it work.
  - Stopping the loop when counter $\geq$ 6, then the condition while(counter < 6) does work.
  - Stopping the loop when counter > 5, then the condition while(counter $\leq$ 6) make it work.

Repeat Instruction

Several equivalent algorithms

counter ← 1;
while(counter ≤ 5) do {
write(counter); counter ← counter + 1;
}
	counter ← 1;
	while(counter ≠ 6) do { write
	(counter); counter ← counter + 1;
	}

counter ← 1;
while (counter < 6) do { write
(counter); counter ← counter + 1;
}
	counter ← 0;
	while (counter < 5) do {
	write(counter); counter ← counter + 1;
	}

Repeat Instruction

The stopping condition

- To write a loop, get in the habit:
  1. Look for the stopping condition;
  2. To write its negation using the table of correspondence of the stopping conditions which follows (to know).

Shutdown logic	$=$	$\neq$	$\geq$	$<$	$>$	$\leq$	**AND**	**OR**
Logic of continuity	$\neq$	$=$	$<$	$\geq$	$\leq$	$>$	**OR**	**AND**

Repeat Instruction

The stopping condition

- To write a loop, three steps are required:
  - The initialization of the variables of the counter, and in general of the block, before entering the loop.
  - The continuation condition. There are always different continuation conditions, all of which are fair (equivalent).
  - The modification of at least one value in the loop (the one that was initialized previously) so that the repetition expresses an evolution of the calculations.
- It is a serious mistake to neglect any of the previous points !! may not exit the loop (infinite loop).

Repeat Instruction

The syntax of other loops

The to-do loop

- The for-do loop is used very frequently in programming to repeat an execution a number of times known in advance.
- Let's see how to write the display of numbers from 1 to 5.

```
for ( counter ← 1 to 5
) do
{
write(counter);
}
```

Repeat Instruction

The syntax of other loops

The to-do loop

- Let's see through an example how to go from a to-do entry to a while-to-do entry :

Loop-while-doing algorithm

```
Var
 counter: integer;
counter ← 1;
while (counter ≤ 5 ) do
{                write (counter);
    counter ← counter + 1;
}
```

Loop-to-do algorithm

```
Var
 counter: integer;
for (counter ← 1 up to 5) do
{
write(counter);
}
```

Repeat Instruction

The syntax of other loops

The do-while loop

- The *do-while_while loop* performs the evaluation of the Boolean condition after performing the first round of the loop.

```
Counter ← 1 ; //initialization
do //continuation condition
{
write(counter); //treatment
counter ← counter + 1; //increment the //counter
} while(counter ≤ 4)  // pay attention to the condition
```

Repeat Instruction

The syntax of other loops

Application in programming

- To concretize the use of these loops, let's see how they are implemented in some common languages. Here is an example of use in C++ programming :

While Loop	Loop to-do	do-while loop

Int i=1;
while (i<= 5){
// the operation iterated 5 times
i=i+1 ;
}
	Int i;
	for (i=1;i<= 5;i=i+1){
	// the operation iterated 5 times
	}
		Int i=1;
		do {
		// the operation iterated 5 times
		i=i+1 ;
		} while (i<= 5);

Repeat Instruction

Apps

Loop and conditional

- The following algorithm has the user read five whole numbers and displays the largest at the end.

```
Greatest-of-5-integers algorithm
Var  counter, value, max: integer;
Begin
    read(value);
    max ← value ;
    counter ← 1;
    while (counter < 5 ) do
    {
          read(value);
          if (max< value) then
          { max ← value ;
          }
          counter ← counter + 1;
    }
    write(“''max equals '', max) ;
END
```

Repeat Instruction

Apps

Loop and conditional

value	max	counter	(counter<value)	Continuity condition
2	2	1	Variables before the while test	
2	2	1	1<5: True	True, first round (1)
8	8	2	2<5: True	True, one more turn (2)
1	8	3	3<5: True	True, one more turn (3)
4	8	4	4<5: True	True, one more turn (4)
7	8	5	5<5: False	False, exit from loop

Repeat Instruction

Usage

- There is only one block of statements to repeat during a loop. But the block can itself be composed of one or more loops. These are called nested loops.
- Let's take note entry as an example, to extract the best of all. Let's add as an additional constraint that a note must be between 0 and 20. If this is not the case, the algorithm must warn the user so that he starts typing again.

```
Algorithm enter-notes-between-0-and-20
Var
note: integer;
Begin
   write( ' 'enter a note:'');
   read(note);
   while ((score < 0 ) AND (score > 20) ) do
   { write( ''you made a mistake, try again :'');
         read(note );
   }
END
```

Nested Loops

Usage

Let's integrate this block into the 5-note input algorithm described above.

```
Greatest-of-5-integers algorithm
Var
 counter, note, max: integer;
Begin
    write(“''enter a note:'');
    read(note);
    while (score < 0 ) AND (score >20 ) do
    { write(''you made a mistake, try again:'');
            read(note);
    }
max ← score;
counter ← 1;
```

```
while (counter < 5 ) do
{
    write(“''enter a note:'');
    read(note);
    while (score < 0 ) AND (score >20 ) do
    { write(''you made a mistake, try again:'');
            read(note);
}
if (max< score) then
{ max ← score ;
}
 counter ← counter + 1;
}
write(“the highest note is”, max);
END
```