Chapter 4:
Loops

Algorithms and data structure 1

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

Contents of chapter 04:

1. Introduction

- a **loop** is a control structure that aims to execute a set of instructions repeatedly.

- There are **three** types of loops:
    1. Conditional loop with a **pre-condition "While"** : The condition is checked before the first iteration..
    2. Conditional loop with a **post-condition "Do...While"** : The condition is checked after the first iteration.
    3. **Iterative** loop "**for**": A **counter** is used to determine the number of iterations.

- **Infinite** loop: A loop that never stops.

2. «while» loop

The "**while**" loop is a pre-condition loop in which a set of instructions is executed repeatedly based on a Boolean condition. The "**while**" loop can be seen as a repetition of the "**if**" statement. It is used when we have a set of instructions that repeat with the possibility that they may not be executed at all (0 times or more).

It consists of two parts:

- **Condition**: This is an expression of boolean type, which has a value of either true or false.

- **Block of instructions**: It is executed as long as the condition is true.

syntax

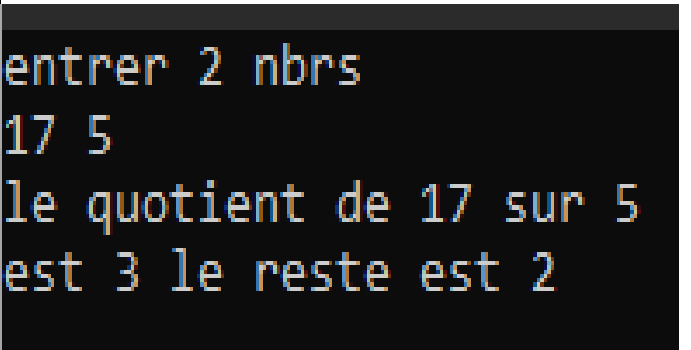| Algorithm	C

while Condition **do**
  Block of instructions
end while
The rest of the program
	while (Condition)
	{
	  Block of instructions
	}
	The rest of the program

in the algorithm, the condition is always between the words **while** and **do**, whereas in C it's always between parentheses ().

Algorigram
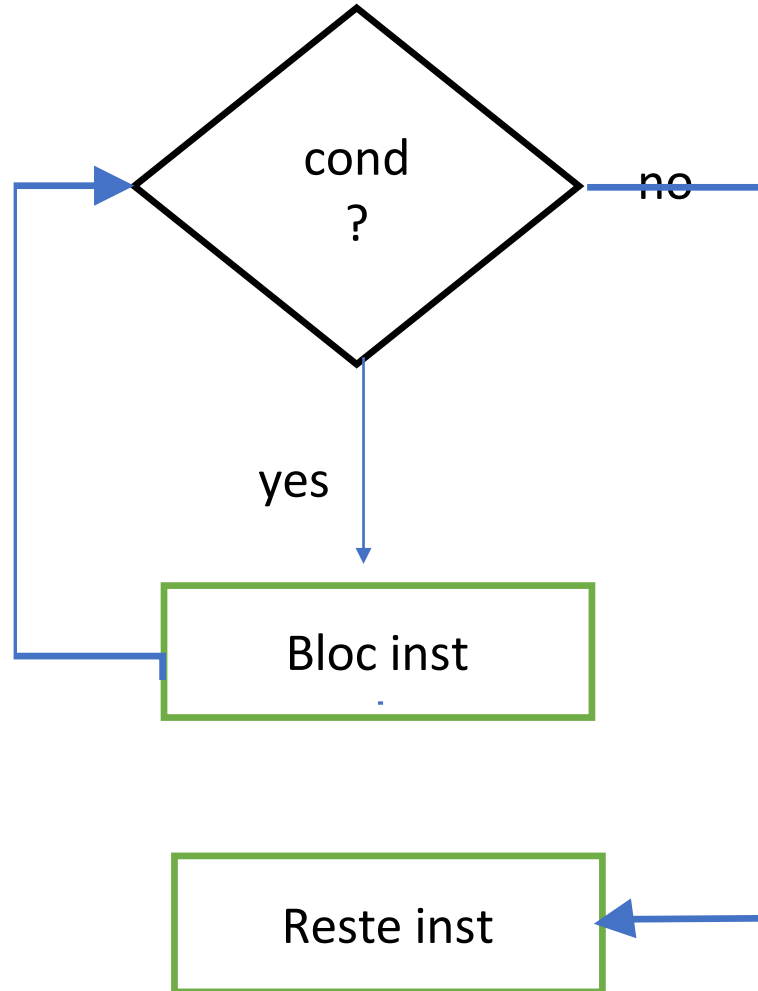
Example

Write a program that reads two integers and then displays the quotient of the first number divided by the second, without using the division operator (/).

Algorithm	C	screen

algorithm quotient
var x, y, q, r :integer
/*x first nbr, y second, q quotient, r remainder*/
begin
  write ("enter 2 nbrs")
  read(x, y)
  q←0
  r←x
  while r>=y **do**
    r←r-y
    q←q+1
  end while
  write ("The quotient of", x, "over", y, "is", q, "the remainder is", r)
end
	``c
	#include <stdio.h>
	int main()
	{
	 int x, y, q, r ;
	 printf("enter 2 nbrs\n") ;
	 scanf("%d%d", &x, &y) ;
	 q=0 ;
	 r=x ;
	while (r>=y)
	 {
	  r-=y ;
	  q++ ;
	 }
	printf("the quotient of %d over %d is %d the remainder is %d\n", x, y, q, r) ;
	}
	``
		entrer 2 nbrs
		17 5
		le quotient de 17 sur 5 est 3 le reste est 2

3.   « do… while» loop

The "**Do...While**" loop is a post-condition loop in which a set of instructions is repeatedly executed. It is used when we have a set of instructions that need to be repeated and executed at least once, regardless of the condition (1 or more).

The loop consists of two parts:

- **Instruction block**: It is executed as long as the condition is true, except the first time it is executed, regardless of the condition.
- **Condition**: An expression of Boolean type, with a value of either true or false.

syntax

Algorithm	C

do
   `Instruction block`
while `Condition`
`The rest of the instructions`
	`do {`
	   `Instruction block`
	`}`
	while `(Condition) ;`
	`The rest of the instructions`

- The "do...while" loop always ends with a semicolon ";"
- The "do...while" loop can be expressed as "**Repeat...Until**" (until the condition is satisfied). In this case, the condition becomes a termination condition rather than a continuation condition, which is the negation of the "while" loop condition.
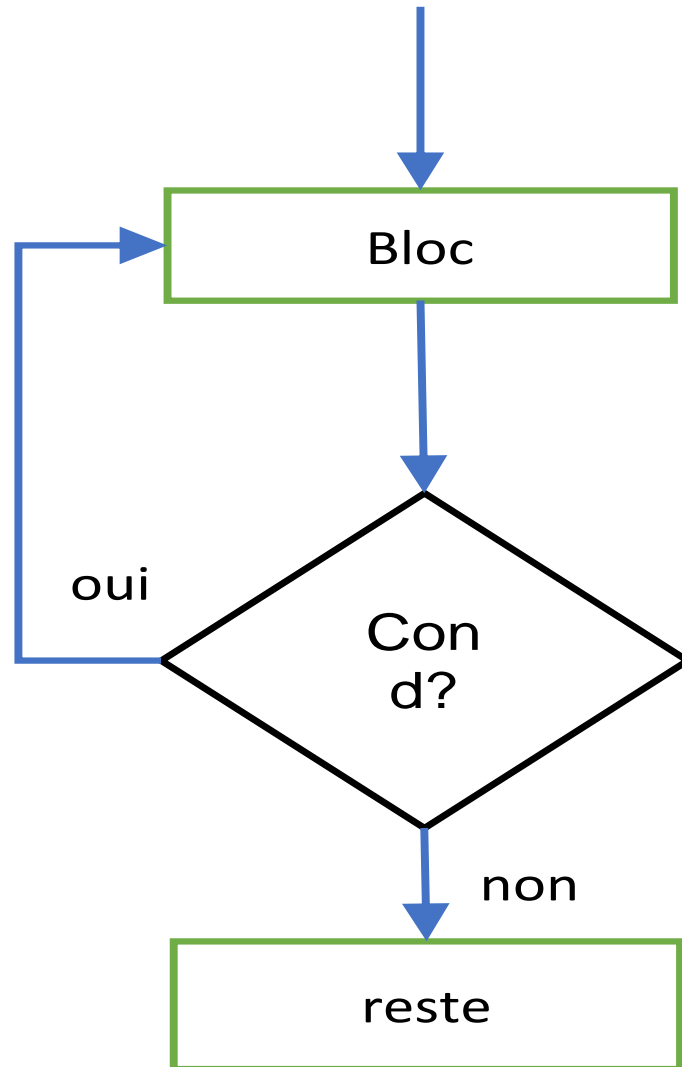
Repeat
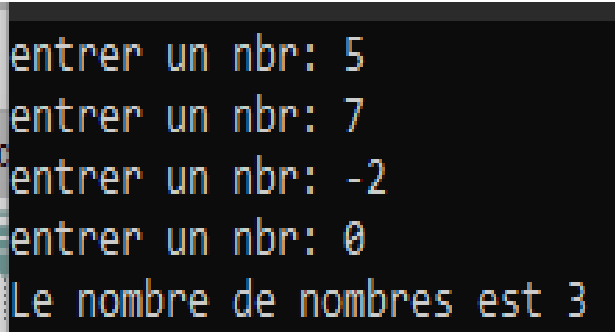
  `Instruction block`

Until not `Condition`

`rest of the instruction`

3.2. Algorigram

Example

Write a program that reads a set of integers using a single variable, stops when it reads the first 0, and then displays the number of integers entered.

Algorithm	C	Ecran
```algorithm readNbrs var x, nb :integer /*x to read nbrs, nb to count nbrs*/ begin nb←0 do write ("enter a nbr") read (x) nb←nb+1 while x≠0 write("The number of numbers is", nb-1) end```	```#include <stdio.h> int main() { int x, nb ; nb=0 ; do { printf("enter a nbr") ; scanf("%d", &x) ; nb++ ; } while (x!=0) ; printf("The number of numbers is %d", nb-1) ; }```	entrer un nbr: 5 entrer un nbr: 7 entrer un nbr: -2 entrer un nbr: 0 Le nombre de nombres est 3

4. « for » loop

The "**for**" loop is an unconditional iterative loop in which a set of instructions is executed iteratively a **predetermined** number of times.

The loop consists of two parts:

- **Counter**: To count the number of iterations, it is a variable of type integer or character. It defines its initial value, final value, and the method to increment or decrement it.
- **Block of instructions**: to be executed in each iteration.

« **for** » syntax in algorithms

Algorithm

For counter ←initial_value **To** final_value **step** step **Do**
  Instruction block
end For
The rest of the instructions

- counter : A name for an integer or character variable.
- initial_value : This is the initial value taken by the counter variable.
- final_value : This is the final value the counter variable can take.
- step : It is the value to which the "counter" variable evolves at the end of each iteration, where " counter ← counter + step." Usually, "step" is equal to **1**.

Comments

1. The value of "`counter`" cannot be modified inside the loop.

2. The "`initial_value`" and "`final_value`" are calculated only once before the loop execution.

3. The part "(**step**)" is optional, and if absent, it means that "`step`" is +1.

4. If the "`step`" is positive, it is added to "`counter`" until "`counter` ≥ `final_value`".

5. In the case of a negative "`step`", it is decremented until "`counter` ≤ `final_value`".

6. "`counter` = `final_value`" is executed.

7. If "`initial_value`" is greater than "`final_value`" and the "`step`" is positive, the "**for**" loop is not executed.

8. if "`initial_value`" is less than "`final_value`" and the "`step`" is negative, the "**for**" loop is not executed.

« **for** » syntax in C

The "**for**" loop in C is more general than the "for" loop in the algorithm. It's closer to the conditional "**while**" loop than to the "for" loop.

The general form	Equivalent of algorithmic syntax

```for (initialization ; test ; iteration)```
```{```
```   Instruction block```
```}```
```The rest of the instructions```
	```for (c=init_v ; c<=final_v ;```
	```c++)```
	```{```
	``` Instruction block```
	```}```
	```The rest of the instructions```

The first line of the "for" loop consists of three parts enclosed in parentheses (), all optional, separated by a semicolon ";".

- **Initialization**: This part is executed once before the loop starts. It is usually used to assign an initial value to the counter. For example: i=0.

- **Condition**: An expression of type Boolean. Its value must be true for the loop to execute. If the condition is false, the loop is exited. It is evaluated at the beginning of each iteration of the loop. Usually, the counter is tested. For example: i<10.

- **Iteration**: It is executed at the end of each iteration. It is usually used to increment or decrement the counter. For example: i++ or i--.

Remarks

- The variable (the counter) can be declared in the initialization part, in which case its scope is limited to inside the "for" loop, not outside of it.
- The value of the counter in the iteration part can be incremented, decremented, or modified in any other way.
- All parts of "**for**" (initialization, condition, iteration) are optional; they can be omitted and left empty. However, the semicolon "**;**" is mandatory and cannot be omitted. The following script is valid**: for (; ;)**
- The initialization part and the iteration part can contain multiple instructions separated by commas '**,**'.
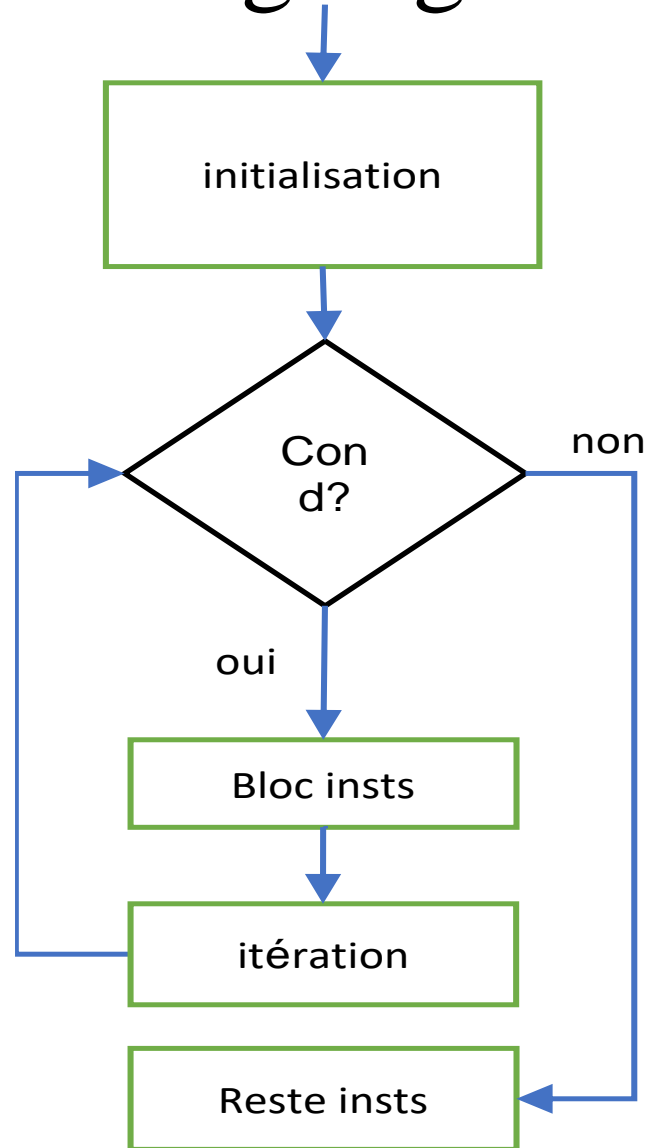- The semicolon "**;**" is the empty instruction.

example

The following example code is equivalent

```
int i=0;
Int j=10;
for ( ; ; ){
 if (!(i<j)) break;
 i++;
 j--;
}


for (int i=0, int j=10 ;i<j ; i++,j--);
```
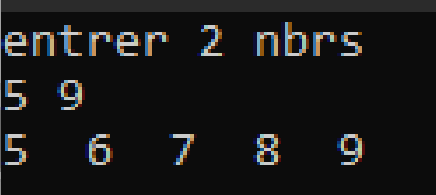
Algorigram

Example

Write a program that reads two integers and then displays all the integers in between.
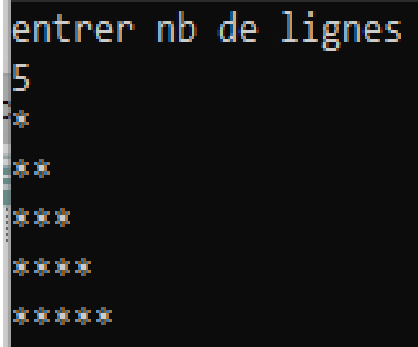
Algorithm	C	screen

algorithm numbers
 var x, y, i :integer
/*i is the counter*/
begin
  write ("enter 2 nbrs")
  read (x, y)
  for i←x **to** y **Do**
    write (i)
  end for
end
	`#include <stdio.h>`
	`int main()`
	`{`
	`  int x, y, i ;`
	`  printf("enter 2 nbrs\n") ;`
	`  scanf("%d%d", &x, &y) ;`
	`  for ( i=x ;i<=y ;i++ )`
	`    printf("%d\t", i) ;`
	`  return 0 ;`
	`}`
		entrer 2 nbrs
		5 9
		5 6 7 8 9

5. Nested loops

one loop in another.

Example

Write a program that reads the number of lines n, and then displays on the screen in the first line *, in the second line **, in the third line ***, and so on until it displays n * in the last line.

Algorithm	C	screen

algorithm stars
var n, i, j :integer
/*i, j counters*/
begin
  write ("enter nbr. of lines")
  read (n)
  for i←1 to n Do
    for j←1 to i Do
      write("*")
    end for
  end for
end
	`#include <stdio.h>`
	`int main()`
	`{`
	`  int n, i, j ;`
	`  printf("enter nbr. of lines\n") ;`
	`  scanf("%d", &n) ;`
	`  for ( i=1 ;i<=n ;i++ ) {`
	`   for ( j=1 ;j<=i ;j++ )`
	`     printf("*") ;`
	`   printf("\n") ;`
	`  }`
	`  return 0 ;}`
		entrer nb de lignes
		5
		*
		**
		***
		****
		*****

6. Loop equivalence

In general:

- Any "While" loop can be expressed by the "Do" loop, by adding a condition before the "Do" loop.

- Any "Do" loop can be expressed by the "While" loop, by adding the block of instructions before the "While" loop.

- Any "For" loop can be expressed by the "While" loop, by assigning the initial value of the counter before the "While" loop, using the final value as the termination condition, and adding the instruction that modifies the value of the counter at the end of the loop.

- However, it is not always possible to express the "While" loop or the "Do" loop with the "**For**" loop in **algorithm**, unless there is a counter.

- In **C**, "While" or "Do...While" loops can be expressed with "For", and all loops can be expressed with "Goto" and "If" statement.

While examples

while	do…while

<pre>…
r=x ;
while (r>=y) {
  r-=y ;
  q++ ;
}
printf(…) ;
}</pre>
	<pre>…
	r=x ;
	if (r>=y)
	 do {
	  r-=y ;
	  q++ ;
	 } while (r>=y)
	printf(…) ;
	}</pre>

for	goto +if

<pre>…
r=x ;
for (;r>=y;) {
  r-=y ;
  q++ ;
}
printf(…) ;
}</pre>
	<pre>…
	r=x ;
	again :
	if (r>=y) {
	  r-=y ;
	  q++ ;
	  goto again ;
	}
	printf(…) ;
	}</pre>

do…while

do…while	while	for	goto +if
… nb=0 ; **do {** printf("enter a nbr") ; scanf("%d", &x) ; nb++ ; **} while (x!=0) ;** printf(…) ; }	… nb=0 ; printf("enter a nbr") ; scanf("%d", &x) ; nb++ ; **while (x!=0) {** printf("entrer un nbr") ; scanf("%d", &x) ; nb++ ; **}** printf(…) ; }	… nb=0 ; printf("enter a nbr") ; scanf("%d", &x) ; nb++ ; **for (;x!=0 ;) {** printf("enter a nbr") ; scanf("%d", &x) ; nb++ ; **}** printf(…) ; }	… r=x ; again : printf("enter a nbr") ; scanf("%d", &x) ; nb++ ; **if (r>y)** **goto** again ; printf(…) ; }

for

for	while	do…while	goto +if
… **for (i=x;i<=y;i++)** printf("%d\t",i); …	… **i=x ;** **while (i<=y){** printf("%d\t",i); **i++ ;** **}** …	… **i=x ;** **if (i<=y)** **do {** printf("%d\t",i); **i++ ;** **} while (i<=y)** …	… **i=x ;** again : **if (i<=y) {** printf("%d\t",i); **i++ ;** **goto** again ; **}** …

7. Loop termination commands

- These commands are used within the loop to perform an early exit from the loop, generally when checking a condition.

- Any "**for**," "**while**," or "**do**...**while**" loop can be terminated by executing any of the jump instructions such as: *break*, *return*, or *goto* (to a label outside the loop).

- The **continue** statement only terminates the current iteration, jumps to the end of the loop, and starts the next iteration.

- These instructions are used within an "**if**" statement.

- In the case of **nested loops**, *break* and *continue* only exit the inner loop.

Example

`for (int i=1 ;i<10 ;i++){`
  `if(i%3==0) continue ;`
  `printf("%d\t", i) ;`
`}`
	`for (int i=1 ;i<10 ;i++){`
	  `if(i%3==0) break ;`
	  `printf("%d\t", i) ;`
	`}`

All numbers will appear, going beyond the multiple of 3 1 2 4 5 7 8	The loop stops at the first multiple of 3 1 2

End of Chapter 04