

Lectures' Topics

APROG - Algoritmia e Programação

Algorithmics

by Maria da Conceição Neves

Contents

1. Software development concepts
2. Procedural Programming: Algorithms and Data Structures
3. Structured Programming Methodology
4. Control flow structures
5. Data, data types, operators and expressions
6. Logic basics
7. Analysis and description of algorithms.
8. Manual Tracing of the algorithms.

Maria da Conceição Neves



1. Software development

- Software development is an activity very important in current society - Information and Knowledge Society.
- Using computers in almost all areas of knowledge requires software solutions more and more sophisticated and complex.
- As the complexity of the problem increases is necessary to use an approach based on **engineering principles**.

Maria da Conceição Neves



1. Software Crisis

- The growth of the software cost and its poor reliability become a great problem.
- In the 1970s there is the recognition of a set of problematic situations that are identified as "**Software Crisis**".
- It was necessary to approach the software development as an activity strictly comparable to that used in the various fields of engineering. This raises a new discipline "**Software Engineering**".

Maria da Conceição Neves

1. Software Engineering

- Software engineering aims producing software efficiently in terms of cost and reliability.
- Software engineering is the practical application of scientific knowledge in the stages of software development:
 - Requirements specification,
 - Analysis,
 - Design,
 - Implementation and testing
 - Maintenance.

Maria da Conceição Neves

1 Producing software is not only Programming

- Since **Requirements Specification** to the creation of a **software product** must be done much more than programming.
- There are several paradigms to address the software development such as the procedural paradigm or object-oriented. In anyone of these paradigms the **Analysis and Conception phases** are very relevant.

1. APROG Purposes

- One of the APROG's goals is to introduce the students in the software development.
- Start by developing logical thinking skills through developing **algorithms and data structures**.
- But **software developing is not only programming ...**

Maria da Conceição Neves

1 Software Process used in this course

- In simple problem solving using **procedural paradigm**, whose requirements are specified already, (usually in a text) we start immediately for:
 - analysis (what to do)
 - design solution (how to do) through an algorithm and data structure
 - coding them in a programming language.
 - program validation using an appropriate test plan.

2.Procedural Programming

Action oriented programming

PROGRAM =
Algorithm
+
Data Structure

Maria da Conceição Neves

2. What is programming?

Program is

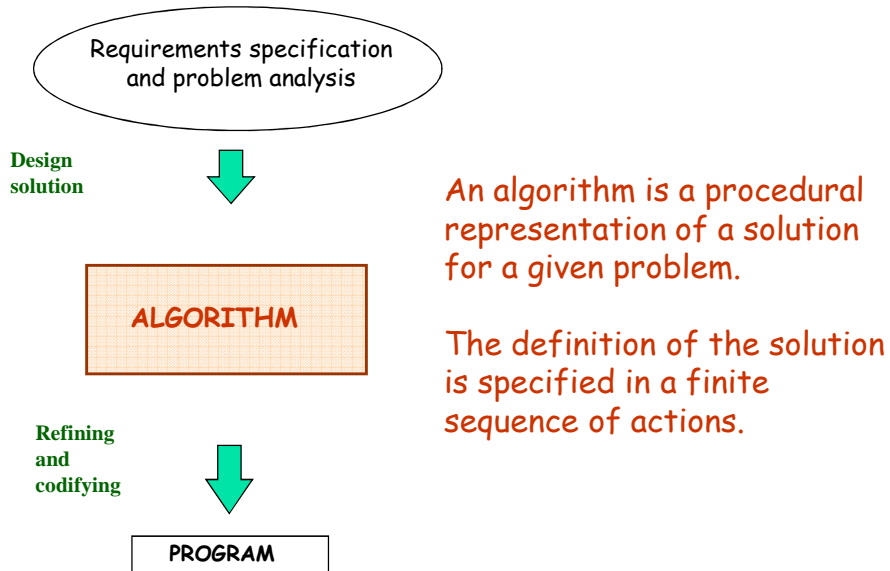
design algorithms

and

describe them in a programming
language

Maria da Conceição Neves

2. Algorithmic modeling



Maria da Conceição Neves

2. Algorithm

An algorithm is a finite and well-defined (unambiguous) set of instructions describing the logical steps for performing a task.

A correct algorithm is one that face a valid input must produce a unique and correct output.

An algorithm must be effective in solving the problem and efficient in order to solve the problem with the best performance.

Maria da Conceição Neves

2. ALGORITHM

- **Algorithm** - sequence of logical steps to perform a task
- **Algorithm's Properties**
 - Should allow communication with outside - **Data input and results output**
 - Must be finite - **Achieving the solution in finite time**
 - Must be well defined - **No ambiguity**

Maria da Conceição Neves

2. Design and Description of Algorithms

- In Design phase are conceived algorithmic solutions to problems.
- The algorithms will be described in a programming language in order to be executed by a computer.
- Because programming languages are complex, at an early stage we describe algorithms using a pseudo language or flowcharts to allow the programmer to be focused on the algorithmic solution.

Maria da Conceição Neves

2. Algorithm's verification

- There are several methods to algorithm's verification.
- In this course we will use:
 - Simulate the execution step by step (manual tracing)
 - Implementation (in a programming language) and Testing

Maria da Conceição Neves

2. Algorithms manipulate data

- Data types:

integers and real numbers,
characters, strings,
logic values, ...

Each data type has an associated set of operations

- Data structure defines how data are organized and how they are accessed and changed.

Examples:

simple variables,
arrays mono- and multi-dimensional,
lists, queues, trees, graphs, ...
files (data structures in secondary memory).

Maria da Conceição Neves



2. Data in main memory and secondary

- When a program is running your data is in memory. When the program ends, or you turn off the computer, the data is lost.
- To save data persistently it's necessary to save it to a file.
- Files are usually stored on disk.
- The files are usually organized into directories.
- Each file is identified by a unique name or name combined with the directory name.

Maria da Conceição Neves



2. Working with files

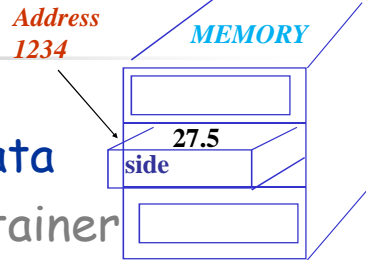
- Working with text files is like working with books, the algorithm has to describe operations
 1. Open the file for reading or writing
 2. Go to the appropriate position
 3. Read and write on the file
 4. Close the file
- When trying to read data from a file that does not exist **an exception occurs this is an error in execution.**

OBS: This issue will be addressed in detail later in this course.

Maria da Conceição Neves

2. Variable

- Variable is where the program stores the data
- Variable is a data container
- Variable is a memory location with
 - Unique name (e.g. side) that should reflect its use
 - Address: (e.g. 1234)
 - Value or content (e.g. 27.5)



Maria da Conceição Neves

2. Variable

- Variable has an associated data type that defines:
 - The set of values that the variable can store and
 - The type of operations in which the variables can occur
- The size of the memory location depends on the type of value that will store.

Maria da Conceição Neves

2. Constantes

- Constants are special variables whose value is fixed, does not change throughout the execution of a given program.
- A program in which there are fixed values they should be defined as constants
- Examples:
 - π value should be defined as
`const real pi=3.1415`
 - VAT tax for luxury goods
`const real tax=0.23`
 - Majority in Portugal
`const int majority=18`

Maria da Conceição Neves

2. Data Structure notion

The program's data structure refers to

- Organization of data in variables
- Assign values to variables
- Accessing the value of the variables
- and other associated operations

Maria da Conceição Neves

2. Algorithms description- Our Pseudo-Code

Data structure description **ED** // variables and data types

Process description **ALG** (opcional)

Start / Inicio

...

End / Fim

Input and output instructions **read()** / ler()

write() / escrever()

Attribution instruction **a ← b + c**

Flow control instructions

Sequence, Decision and Repetition

Maria da Conceição Neves

2. Algorithms description - Flowchart

■ Algorithms graphic representation

Symbols (simplified version)



Start / End



Input ⇔ read



Output ⇔ write



Actions ⇔ attributions



Decision



Flow line

Maria da Conceição Neves

3. Structured Programming Methodology

- In the 70s, motivated by the existing software crisis, Dijkstra developed a methodology that aims to develop programs reliable and easy to maintain.
- This methodology was called structured programming and defines a set of rules for development programs.
- The Pascal language was developed to support this new methodology.

Maria da Conceição Neves

3. Fundamental Theorem of Structured Programming

Refers to the instruction execution order

It's possible to write any program using only the three basic control flow structures:

- **Sequence** - ordered statements executed in sequence
- **Decision** - select alternately one or other set of statements depending on a evaluation of a condition
- **Iteration** - a block of statements is executed in loop until the program reaches a certain state .

Maria da Conceição Neves

4. Program's flow control

- A program is a sequence of instructions.
- Programs can allow multiple branches, ie several sequences of alternative and cycles instructions.
- To direct the flow of execution among the branches there are flow control statements.

Single Branch

Several Branches

Several Branches(with loops)



- The flow control statements are the basic units of a program.
- The flow control statements may be used in combination.
- Each flow control statement has a single entry point and a single exit point.

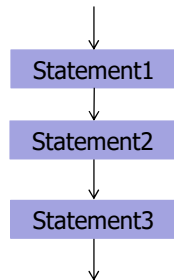
Maria da Conceição Neves

4. Algorithm or Program Flow Control

- Program flow control refers to the order in which instructions are executed.
- Basic flow control structures:
 - Sequence (Sequential statements)
 - Decision (Conditional statement)
 - If (...) then
...
else ...
 - Iteration (loops)
 - While (...) ...
 - Repeat while (...)
 - For(...) ...

Maria da Conceição Neves

4. Control Structure - Sequence

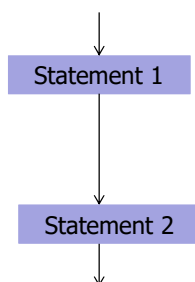


- **Sequence is the simplest control structure**
- Performs a statement followed by another, in the order they appear in the program
- The normal execution flow of the program is sequential unless directed otherwise by other control structure.

Maria da Conceição Neves

4. Control Structure - Sequence

Flowchart



Pseudo- code

Statement 1
Statement 2

OBS:

If necessary to separate
with semicolon (;)

Maria da Conceição Neves

4. Example

- Calculate the area of a square

DS

var real side, area

ALG

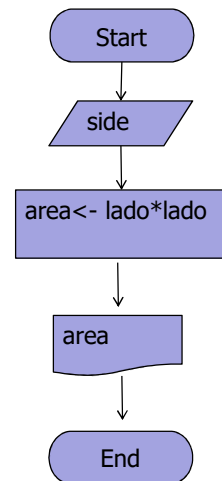
Start

read(side)

area \leftarrow side * side

Write("Square area=", area)

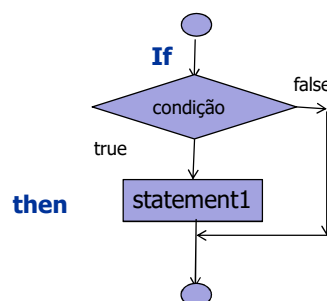
END



Maria da Conceição Neves

4. Control Structure - Decision

- If (condition) then statement1
 - Test the condition and
 - If true (then) perform statement1
 - If false (else) none action performed.

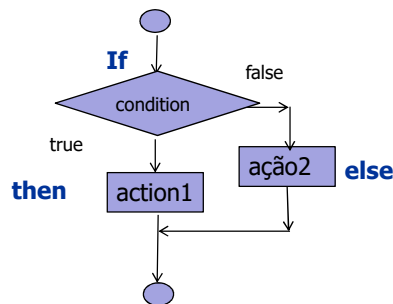


Maria da Conceição Neves

4. Control Structure - Decision

■ If (condition) then action1 else action2

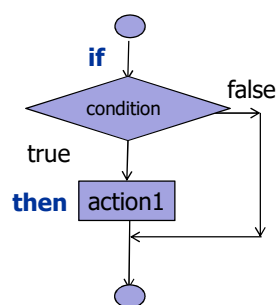
- Test condition and
 - If true (**then**) action1 performed
 - If false (**else**) action2 performed



Maria da Conceição Neves

4. Control Structure - Decision

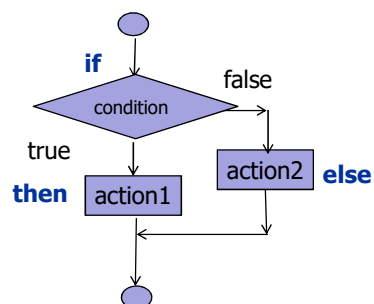
Flowchart



Pseudo-code

```
If (condition) then  
    action1  
EndIf
```

Flowchart



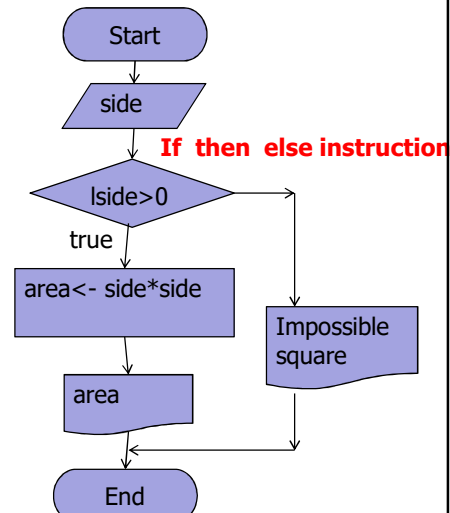
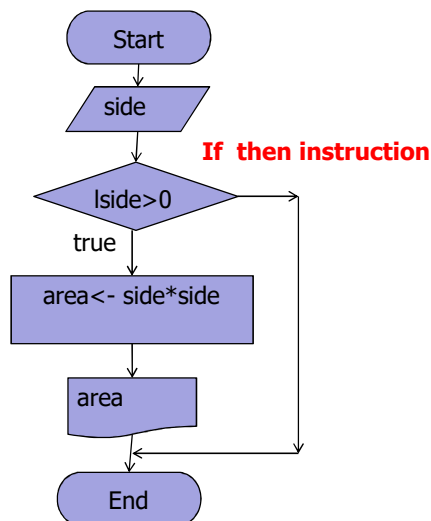
Pseudo-code

```
If (condition) then  
    action1  
else  
    action2  
EndIf
```

Maria da Conceição Neves

4. Example

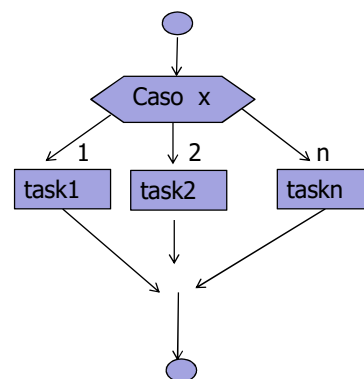
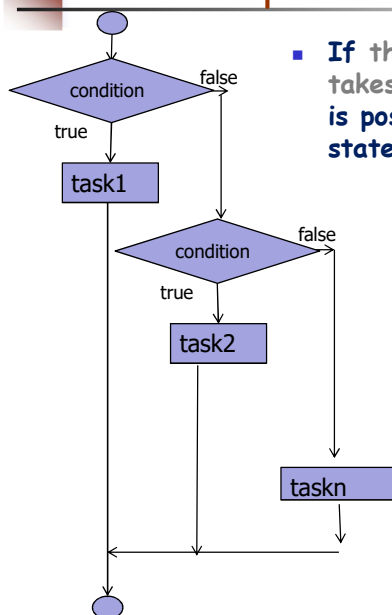
- Calculate the area of the square after testing the square possibility



Maria da Conceição Neves

4. Multiple Decision

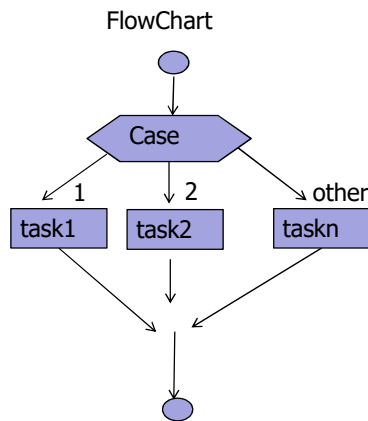
- If the test condition is about a variable that takes one value of a discrete set of values it is possible to replace the statement If by the statement Case (multiple alternatives)



Maria da Conceição Neves

4. Multiple decision

■ Case statement



Pseudo-Code

Switch (variable)

case value1:
tasks1

case value2:
tasks2

default:
tasksn

EndSwitch

Maria da Conceição Neves

4. Multiple Decision(Example)

Pseudo- code

switch month case

1, 3, 5, 7, 8, 10, 12 :
days ← 31;

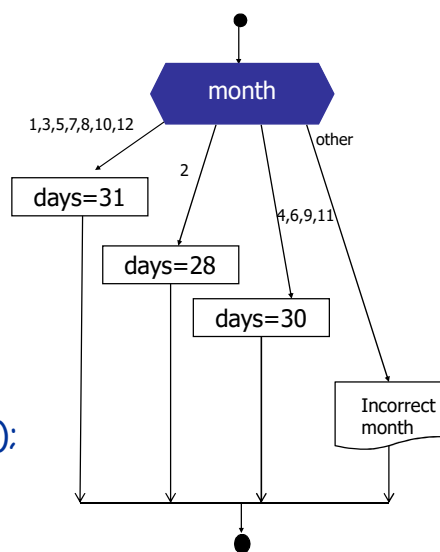
2:
days ← 28;

4, 6, 9, 11:
days ← 30;

default:
write("Incorrect month");

endswitch

Flowchart



And if the year is leap?

Maria da Conceição Neves

4. Control Structure - Repetition or Iteration

- This structure is also called **Loop**
- It is used when a set of instructions in a program has to be run again and again.
- Always involves the verification of a given condition to determine when to end the loop.

Maria da Conceição Neves

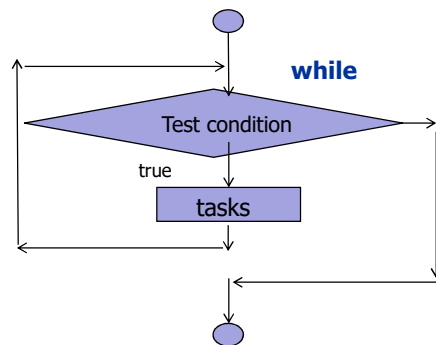
4. Several types of Loops

- If the **test condition** is done **at the loop entry** we will have the structure
 - **While ... do ...**
- If the **test condition** is done **at the loop exit** we will have the structure
 - **Do ... while ...**
- If known the start and end values of the loop and its progression may specify the thresholds (limits) and the step.
A counter is used to indicate the progress through the cycle. The structure will be
 - **For (i=...; ... ;)**

Maria da Conceição Neves

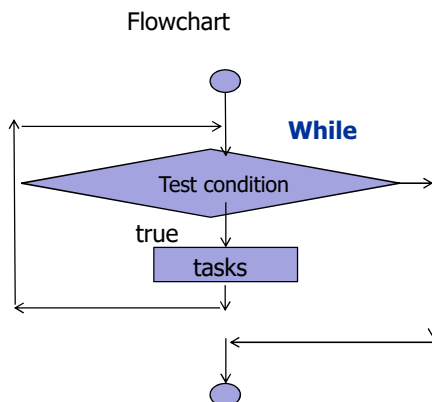
4. Loop - While (...) ...

- the **test condition** is done **at the loop entry**
- If the test condition is true the loop tasks are performed and control back again to the test condition.
- If the test condition is false the loop is finished



Maria da Conceição Neves

4. Loop - While (...) ...



Pseudo-code

While (condition)

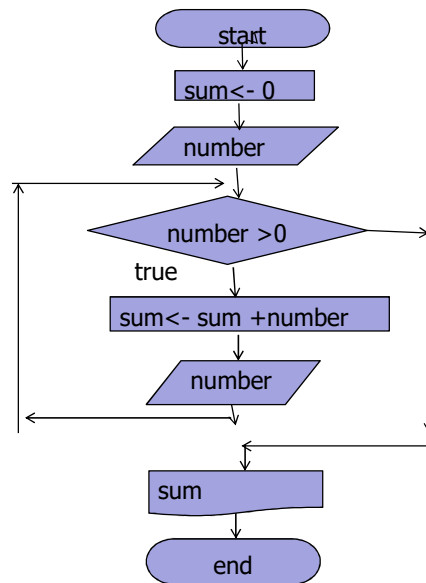
tasks

endWhile

Maria da Conceição Neves

4. Example

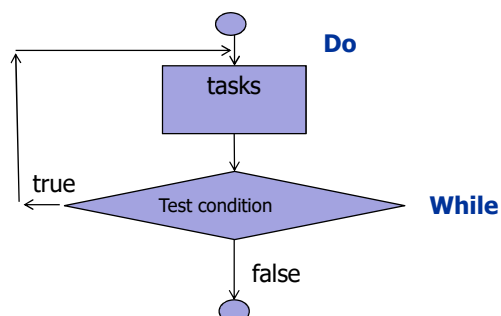
- Calculate the sum of a sequence of positive numbers



Maria da Conceição Neves

4. Loop- Repeat ... while (...)

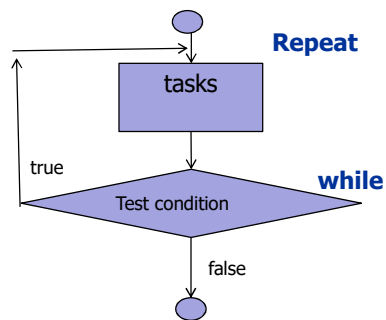
- Start the loop performing the iteration tasks and at the end of each iteration tests the condition to verify if the loops end or not.
- If the test is true it is done a new iteration.
- If the test is false the loop ends.



Maria da Conceição Neves

4. Repeat... while (...)

Flowchart



Pseudo Code

Repeat

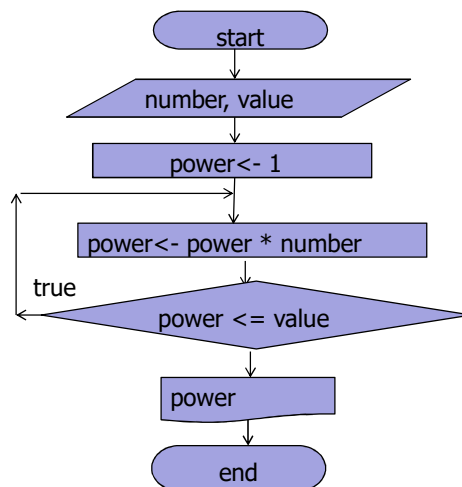
tasks

while (condition)

Maria da Conceição Neves

4. Example

- Calculates the first power of a number that is greater than a certain value.



Maria da Conceição Neves

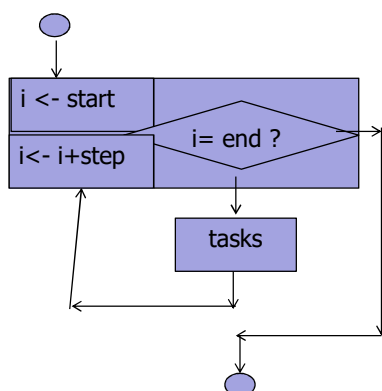
4. Loop - For(... ; ... ; ...)

- This structure is used when we know the start and end values of the cycle and its progression.
- This structure requires that specify the limits of the cycle.
- A counter is used to indicate the progress through the cycle.
- It is necessary to indicate the step of the cycle (or as the counter is updated).

Maria da Conceição Neves

4. Iterative structure- For (...)

Flowchart



Pseudo-code

For (i <- start; i < end; i + step)

tasks

EndFor

Maria da Conceição Neves

4. Iterative structures (loops)

- What kind of loop use?

- **While (...)**

- This is used when the number of iterations is not known. The test of a condition to decide to enter the cycle must be made at the beginning, before performing each iteration.

- **Repeat ... while (...)**

- This is used when the number of iterations is not known. The test of a condition to decide to stop the cycle must be made at the end of the cycle after performing each iteration.

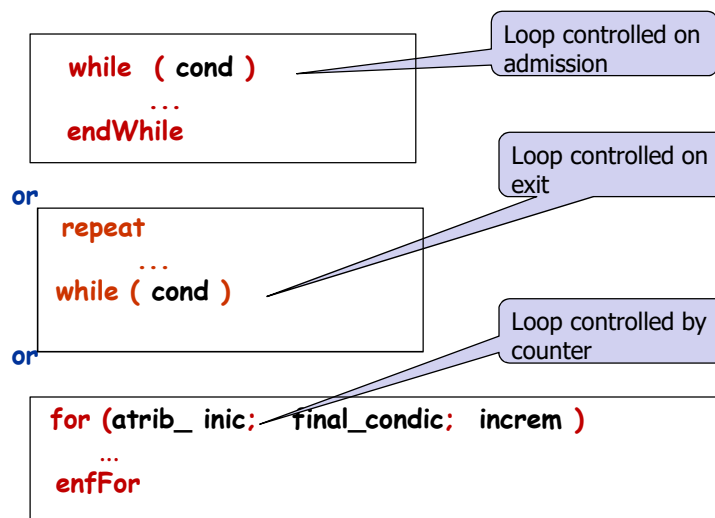
- **For(... ; ... ; ...)**

- This is used when we know the number of iterations to perform.

Maria da Conceição Neves

4. The three kind of Loops

In summary:



Maria da Conceição Neves

5. Data types used in Pseudo-code

- Programming languages provide the ability to represent and manipulate various data types such as: integer, real, logical values (True or False), alphanumeric values (characters and character sequences).
- Each data type has an associated set of operations or operators to manipulate.
- For example:

Data Type	Arithmetic operators
INTEGER	+ addition - subtraction * multiplication DIV integer division (quotient) MOD rest of integer division
REAL	+ addition - subtraction * multiplication / division (quotient)

Maria da Conceição Neves

5. EXPRESSIONS

- With operands and operators are constructed expressions.
- An expression is a sequence of operands and operators.
- We will organize the operators:
 - ASSIGNMENT OPERATOR
 - ARITHMETIC OPERATORS
 - RELATIONAL OPERATORS
 - LOGICAL OPERATORS

Maria da Conceição Neves

5. OPERATORS

■ ASSIGNMENT OPERATOR

← Exemplo: valor ← 18*32

■ ARITHMETIC OPERATOR

+	-	*	/
addition	subtraction	multiplication	divisão
DIV		MOD	
integer division quotient		rest of integer division	

■ RELATIONAL OPERATORS

<	>	<=	>=	=	!=
less than	greater than	less or equal than	...	equal	not equal

■ LOGICAL or BOOLEAN OPERATORS

AND	OR	NOT
conjunction	disjunction	negation

Maria da Conceição Neves

5. Arithmetics Operators

■ Operands:

- They are numbers or numeric expressions

■ Operators:

- Adding +
- Subtraction -
- Multiplication *
- Division /
- Rest of Integer Division MOD
- Integer quotient DIV

■ Tables of operations

■ Properties of operations

5. Floating point numbers

- Floating point numbers are represented in the hardware of a computer as binary fractions.
 - The decimal 0.125 is the value $1/10 + 2/100 + 5/1000$
 - The binary fraction 0.001 has the value $0/0 + 2/4 + 1/8$
 - They have the same value but the first is written in base 2 and the other at the base 10
- Most of the decimal fractions can not be represented exactly as binary fractions.
- Programming languages that support floating point arithmetic deal with approximate values.

Maria da Conceição Neves

6. Fundamentals of Logic

- In a language an expression with meaning is a designation or a statement that can be affirmative, exclamative,
 - **Designation** refers to or denotes an object or entity
 - 2 7-3 a
 - **Affirmative Statement** refers to an affirmation
 - $2 > 7$ $5+1=6$
- In Logic only be considered affirmative statements that are either true or false.
These statements are called **propositions**.
- Every proposition has one and only one logic value that is **True** (1) or **False** (0).

Maria da Conceição Neves

6. Propositions

■ Simple Propositions

- $2+3 = 5$ logic value True
- 4 is an odd number logic value False

■ Compound Propositions - combines two or more propositions using logic operators (connectives)

- $(2+1 > 2)$ AND $(3 < 5)$ logic value True
- $(7 <= 2)$ OR $(3 > 2)$ logic value True
- $(3+1 = 2)$ AND $(3 > 2)$ logic value False
- NOT $(5 > 9)$ logic value True

logic value True is represented by 1
logic value False is represented by 0

Maria da Conceição Neves

6. Proposicional calculus (Basics)

- Propositional calculus is the study of logical operations on propositions.
- Operands are logical values of propositions
- Logical or Boolean operations:
 - Conjunction AND
 - Disjunction OR
 - Negation NOT (operador unário)
 - Implicação
 - Equivalência

6. Negation Operation

■ Operator NOT (operator unary)

■ Truth tables:

Consider p being a proposition

		NOT p
p	NOT	
0		1
1		0

If p for falso

If p for verdadeiro

NOT p é verdadeiro

NOT p é falso

■ Properties

$$\text{NOT} (\text{NOT } p) \Leftrightarrow p$$

Maria da Conceição Neves

6. Conjunction Operation (AND)

Consider p , q and r as propositions

■ Truth Table

		$p \text{ AND } q$	
p	q	AND	
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

$p \text{ AND } q$ is true if and only if
 p is true and q is true.

■ Properties of Conjunction

Commutative

$$p \text{ AND } q \Leftrightarrow q \text{ AND } p$$

Associative

$$(p \text{ AND } q) \text{ AND } r \Leftrightarrow p \text{ AND } (q \text{ AND } r)$$

Neutral Element 1 $p \text{ AND } 1 \Leftrightarrow 1 \text{ AND } p \Leftrightarrow p$

Absorber Element 0 $p \text{ AND } 0 \Leftrightarrow 0 \text{ AND } p \Leftrightarrow 0$

6. Disjunction operation (OR)

Consider p, q and r as propositions

■ Truth Table

$p \text{ OR } q$

	q	
	0	1
p	0	1
	1	1

$p \text{ OR } q$ is true
if at least one of the propositions was true

■ Properties of Disjunction

Commutative

$$p \text{ OR } q \Leftrightarrow q \text{ OR } p$$

Associative

$$(p \text{ OR } q) \text{ OR } r \Leftrightarrow p \text{ OR } (q \text{ OR } r)$$

Neutral Element 0

$$p \text{ OR } 0 \Leftrightarrow 0 \text{ OR } p \Leftrightarrow p$$

Absorber Element 1

$$p \text{ OR } 1 \Leftrightarrow 1 \text{ OR } p \Leftrightarrow 1$$

6. Morgan's Laws

Consider p, q and r propositions

1. The negation of a conjunction is the disjunction of the negations.

$$\text{NOT } (p \text{ AND } q) \Leftrightarrow (\text{NOT } p) \text{ OR } (\text{NOT } q)$$

2. The negation of a disjunction is the conjunction of the negations.

$$\text{NOT } (p \text{ OR } q) \Leftrightarrow (\text{NOT } p) \text{ AND } (\text{NOT } q)$$

6. Conditions

Condition or propositional expression - any expression with variables that turns into a proposition (true or false) whenever you assign values (belonging to the respective domains) to variables

Logics Conditions					
	$N > 0$	$N < 100$	$\text{NOT}(N > 0)$	$(N > 0) \text{ AND } (N < 100)$	$(N > 0) \text{ OR } (N < 100)$
If $n=45$	1	1	0	1	1
If $n=-5$	0	1	1	0	1
If $n=105$	1	0	0	0	1

Examples:

Condition

$N > 0$

$(N > 0) \text{ AND } (N < 100)$

$(N >= 0) \text{ OR } (N < 100)$

Negated condition

$N <= 0$

$(N <= 0) \text{ OR } (N > 100)$

$(N < 0) \text{ AND } (N >= 100)$

Maria da Conceição Neves

7. Algorithm to calculate the area of a triangle

ED

var real base, altura, area

ALG

INICIO

ler(base, altura)

area \leftarrow base*altura /2

escrever("Area do Triângulo=", area)

FIM

The input values validation is essencial.

What to do in order to ensure that the input data is valid?

ler(base, altura)

repetir

ler(base, altura)

enquanto (base \leq 0 OR altura \leq 0)

The algorithms presented in this paper, for simplicity, we assume that then the input will always be valid. However algorithms could be responsible for validating the data proceeding analogously as shown.

Maria da Conceição Neves

Algorithm to calculate the area of a rectangle

ED

```
var real lado1, lado2, area
```

ALG

INICIO

```
ler(lado1,lado2) ;
```

```
area ← lado1 * lado2 ;
```

```
se (lado1=lado2)
```

```
então escrever("Area Quadrado=", area);
```

```
senão escrever("Area Rectângulo",area);
```

```
fimse
```

FIM

Plano de Testes

Traçagem

Maria da Conceição Neves

7. Decision structures embedded

Example: Make a program that receive 3 integers and determine the largest and the smallest of them.? OBS: (Consider that if the numbers are 5,5,5 the answer The LARGEST is 5 and The SMALLEST is 5 is valid)

E.D. var int n1,n2,n3

INICIO

```
ler(n1,n2,n3);
```

```
SE (n1>n2)
```

```
ENTÃO
```

```
SE (n1>n3)
```

```
ENTÃO
```

```
/* n1>n2 and n1>n3*/
```

```
escrever ("The LARGEST is" , n1);
```

```
SE (n2>n3)
```

```
ENTÃO /*n1>n2 and n1>n3 and n2>n3*/
```

```
escrever ("The SMALLEST is" , n3);
```

```
SENÃO /*n1>n2 and n1>n3 and n2<=n3*/
```

```
escrever ("O MENOR é" , n2);
```

```
FIMSE
```

```
SENÃO /* n1>n2 and n1<=n3*/
```

```
escrever ("The LARGEST is" , n3);
```

```
escrever ("The SMALLEST is" , n2);
```

```
FSE
```

```
SENÃO
```

```
/* n1<=n2 */
```

```
SE (n2>n3)
```

```
ENTÃO
```

```
/* n1<=n2 and n2>n3 and n1>n3*/
```

```
escrever ("The LARGEST is" , n2);
```

```
SE (n1>n3)
```

```
ENTÃO /*n1<=n2 and n2>n3 and n1>n3*/
```

```
escrever ("The SMALLEST is" , n3);
```

```
SENÃO /*n1<=n2 and n2>n3 and n1<=n3*/
```

```
escrever ("The SMALLEST is" , n1);
```

```
FIMSE
```

```
SENÃO /* n1<=n2 and n2<=n3*/
```

```
escrever ("The LARGEST is" , n3);
```

```
escrever ("The SMALLEST is" , n1);
```

```
FIMSE
```

FIM

Maria da Conceição Neves

7. Loop controlled at the entrance

Determine the average of a sequence of numbers ended by "sentinel value"

E.D. real num, sum, media int count

ALG

INICIO

count ← 0 start counter

sum ← 0 start sum

read (num)

WHILE (num <> sentinel)

count ← count +1 increment the counter

sum ← sum + num update count

ler (nota)

ENDWHILE

media ← sum / count ATTENTION: Divide by Zero is impossible

write(media)

FIM

Maria da Conceição Neves

7. Example

Make a program that receive a set of values representing the number of km traveled by a car and the spent fuel determine the average fuel consumption by 100km. The input values end when introduced km equal 0.
(Pseudo code in portuguese)

ED var real km, litros, somakm, somalitros, media

ALG

INICIO

somakm ← 0; somalitros ← 0

ler(km);

ENQUANTO (km > 0) FAZER

ler(litros)

somakm ← somakm + km

somalitros ← somalitros + litros

ler(km)

FIMENQUANTO;

media ← somalitros / somakm * 100

escrever(media)

FIM

Maria da Conceição Neves

7. Loops with test condition at the end of each iteration

Elaborate an algorithm to determine the first power of a given number that is greater than another given number N

```

E.D. var int    number, N, power
ALG.
START
  read(number)
  read(N);
  power ← 1      start variable potencia
  DO
    power ← power * number
  WHILE (potwer <= N)
  write(power)
END
  
```

Attention
You must change the algorithm
in order to deal with exceptional cases
powers of 0 and 1

Maria da Conceição Neves

7. Example Loop Do ... while()

Make a program that allows writing in sequence the digit of units, tens, ... of an integer. Consider that have the div and mod operators respectively calculate the quotient and rest of the integer division

ED var int n,d

```

ALG
START
  read(n)
  DO
    d ← n mod 10
    write (d)
    n ← n div 10
  WHILE (n!=0);
END
  
```

TRAÇAGEM

start		
n=453		
d=453 mod 10=3 3	d=45 mod 10=5 5	d=4 mod 10=4 4
n=453 div 10 =45	n=45 div 10 =4	n=4 div 10 =0
45 ≠ 0? Sim	4 ≠ 0? Sim	0 ≠ 0? Não

end

Maria da Conceição Neves

7. Example

Write a program that given an integer builds a new integer constituted by the same digits, but in reverse order.

Example: Input: 453 Output: 354 (Pseudo code in portuguese)

ED var int n,novo,y

ALG

INICIO

ler(n)

novo ← 0

REPETIR

y ← n mod 10

novo ← novo*10+y

n ← n div 10

ENQUANTO (n!=0);

escrever("Novo=", novo)

FIM

TRAÇAGEM

início

n=453

novo=0

y=453 mod 10=3

novo=0*10+3=3

n=453 div 10 =45

45 ≠ 0? Sim

y=45 mod 10=5

novo=3*10+5=35

n=45 div 10 =4

4 ≠ 0? Sim

y=4 mod 10=4

novo=35*10+4=354

n=4 div 10 =0

0 ≠ 0? Não

Novo=354

fim

Maria da Conceição Neves

7. Loop controlled by counter

Determine the average of ingress note of nalunos students in a class

E.D. var real nota, soma, media var int contador, nalunos

INICIO

(Pseudo code in portuguese)

ler(nalunos)

contador ← 0

iniciar contador

soma ← 0

iniciar acumulador

enquanto (contador < nalunos) fazer

ler (nota)

contador ← contador +1

incrementar contador

soma ← soma +nota

actualizar acumulador

fimenquanto;

media ← soma /contador

escrever(media)

FIM

Maria da Conceição Neves

7. Loop controlled by counter

(number of iterations defined)

Determine the average of ingress note of nalunos students in a class

E.D. var real nota, soma, media var int i, nalunos

START

read(nalunos)

soma \leftarrow 0

iniciar acumulador

for (i \leftarrow 1 até nalunos passo 1)

read (nota)

soma \leftarrow soma +nota

actualizar acumulador

endFor

media \leftarrow soma /nalunos

write(media)

END

Maria da Conceição Neves

7.Example

Make a program that given a sequence of numbers, not empty, terminated 0 determine which is the biggest one

ED var num, major

(Pseudo code in portuguese)

ALG

INICIO

ler(num)

major \leftarrow num

ENQUANTO (num \neq 0) FAZER

SE (num > major)

ENTAO

major \leftarrow num

FIMSE

ler(num)

FIMENQUANTO

escrever(major)

FIM

Change the algorithm in order to calculate the lowest number

Maria da Conceição Neves

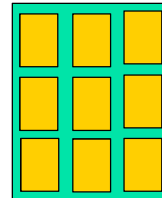
7. Other examples:

1. Make a program that allows write the multiplication table of a given number

```
ED var int num, i
ALG
INICIO
  ler(num)
  PARA( i=1 ATE 10 PASSO 1)
    escrever(num,"x",i,"=", num*i)
  FIMPARA
FIM
```

2. Make a program to write on A4 sheet the multiplication tables from 1 through 9.

Obs. Follow the usual format



Maria da Conceição Neves

7. What does the following program written in pseudo-code?

E.D. var real Lado1, Area

ALG

Início

ler (Lado1) // Instrução de entrada de dados

Area ← Lado1* Lado1 //Instrução de atribuição

/* Instrução de saída de resultados */

escrever("Area do quadrado =", Area);

Fim

1. What would happen if it were introduced a negative number?
2. What changes would make in the algorithm?

Maria da Conceição Neves

7. What does the following program written in pseudo-code?

```
E.D.  var real  Lado1, Area
ALG
Início
  ler (Lado1)          /* Instrução de entrada de dados */
  Se ( Lado1<= 0)      /* Instrução de decisão  */
    Então
      escrever( "Quadrado impossível") /*Inst saída resultados */
    Senão
      Area <-- Lado1* Lado1  /* Instrução de atribuição*/
      escrever( " Area do quadrado =", Area) /*Inst saída result */
  FimSe                /* Fim de estrutura de decisão*/
Fim
```

Maria da Conceição Neves

7. What does the following program written in pseudo-code?

```
E.D.  var real  Lado1, Area
ALG
Início
  ler (Lado1)
  Enquanto ( Lado1>0)  /* Ciclo com teste à entrada */
    Area <-- Lado1* Lado1
    escrever( " Area do quadrado =", Area)
    ler (Lado1)
  FimEnquanto          /* Fim de ciclo*/
Fim
```

Maria da Conceição Neves

7. What does the following program written in pseudo-code?

```
E.D.   var int   num, quad
ALG
Início
  Repete          /* Ciclo com teste à saída */
    ler (num)
    quad <-- num* num
    escrever( " O quadrado de", num, "=",quad)
  enquanto (quad < 1000)      /*teste à saída */

Fim
```

Maria da Conceição Neves

7. Determining the least common multiple of two positive integers

```
ED int num1, num2, i
Alg
INICIO
  ler(num1, num2)
  SE (num1>num2)
    ENTÃO
      x=num1; num1=num2; num2=x;
  FIMSE
  i=1;
  ENQUANTO (num2*i mod num1 < > 0) FAZER
    i=i+1
  FIMENQUANTO
  escrever ("O menor múltiplo comum entre", num1, "e",
    num2 ,"é ", num2*i)

FIM
```

Maria da Conceição Neves

7. Determining the greatest common divisor of two positive integers

```

ED int num1, num2, x, div, enc;
ALG
INICIO
  ler(num1, num2)
  SE (num1=num2)
    ENTÃO escrever (" O máximo divisor comum é", num1)
    SENÃO
      SE (num1 > num2)
        ENTÃO x=num1; num1=num2; num2=x
      FIMSE
      enc=0
      div=num1
      ENQUANTO (enc=0) FAZER
        SE (num1 mod div =0)
          ENTÃO
            SE (num2 mod div =0) ENTAO enc=1
            SENÃO div= div-1
          FIMSE
        SENÃO div =div -1
      FIMSE
      FIMENQUANTO
      escrever ("O mdc entre", num1, "e", num2 ,"é ", div)
    FIMSE
  FIM

```

Maria da Conceição Neves

7. Determining the greatest common divisor of two positive integers (A more efficient algorithm)

```

ED int num1, num2, x, div;
ALG
INICIO
  ler(num1, num2)
  SE (num1=num2)
    ENTÃO escrever (" O máximo divisor comum é", num1)
    SENÃO
      SE (num1>num2)
        ENTÃO
          x=num1; num1=num2; num2=x
      FIMSE
      div=num1
      ENQUANTO (num1 MOD div <> 0 OR num2 MOD div <> 0)
        div <- div-1
      FIMENQUANTO
      escrever ("O mdc entre", num1, "e", num2 ,"é ", div)
    FIMSE
  FIM

```

Maria da Conceição Neves

7. Problem solving

- Essentially there are two kinds of problems

1. Those for which it is possible to describe a deterministic algorithm which ensures success.

Only this kind of problems are treated in this course


2. Those problems that are not possible to be described by an algorithm that ensures the solution, require a solution search. The resolution of such problems is widely studied in the field of Artificial Intelligence. Several solution search strategies are proposed..

Maria da Conceição Neves

7. Concluding remarks

- The description of real problems includes a set of superfluous details. Must identify the abstract problem.
- Abstraction allows common solutions to problems that are similar.
- The specification of algorithms and data structures can be performed in any language. It must avoid ambiguity.
- We need to select and design the data structures.
- An algorithm defines a process. The process is specified in terms of simple instructions. Each process step has to be implemented via an instruction or some other algorithm.
- Correct algorithm - produces correct "outputs" (final stages) in the presence of "inputs" (initial states) valid.

Maria da Conceição Neves

- 
- Given the complexity of programming languages such as for example JAVA, we must begin by describing the algorithms in pseudo code or flowchart focusing us well in the design of the algorithm.
 - Code or program is implemented in a programming language the data structures and algorithms.
 - A program is not only to run is also to be read. Proper indentation of the code is very important as it helps to display the program structure and better understand the associated semantics.

Maria da Conceição Neves