Algorithms And Data Structure 1

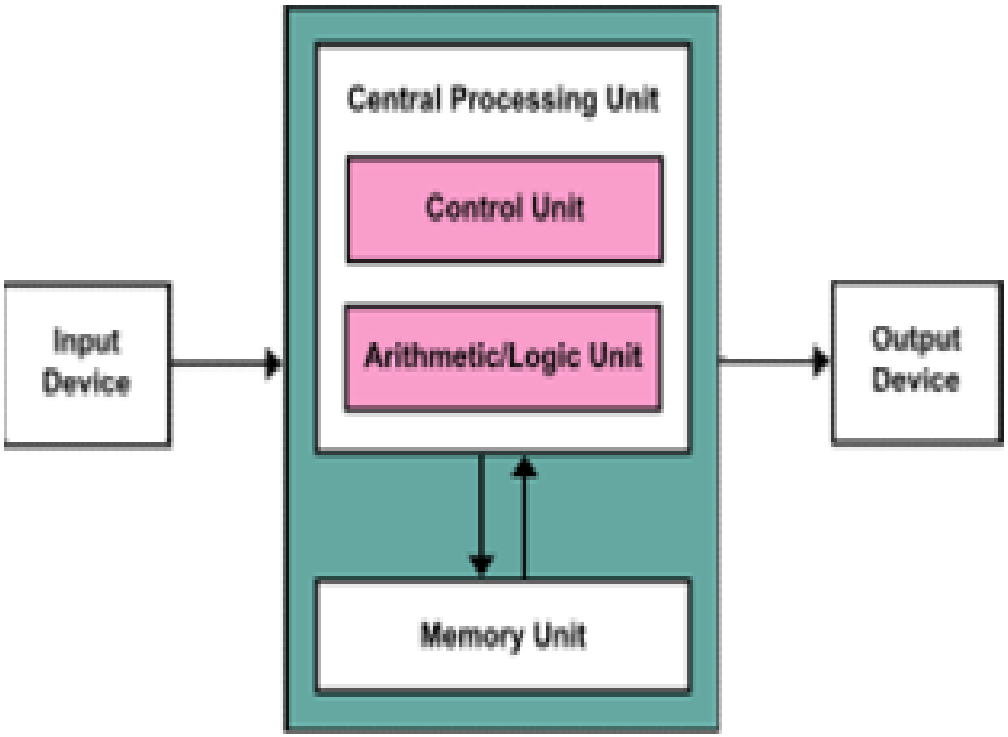
Classroom link Qrcode

Mr Haniche Fayçal

Von Neumann architecture

The **von Neumann architecture** —also known as the **von Neumann model** or **Princeton architecture**—is a computer architecture based on a 1945 description by John von Neumann, and by others, in the *First Draft of a Report on the EDVAC*. The document describes a design architecture for an electronic digital computer with these components:

- A processing unit with both an arithmetic logic unit and processor registers
- A control unit that includes an instruction register and a program counter
- Memory that stores data and instructions
- External mass storage
- Input and output mechanisms



What is algorithm?

- A finite set of instructions which accomplish a particular task
- A method or process to solve a problem
- Transforms input of a problem to output

Algorithm = Input + Process + Output

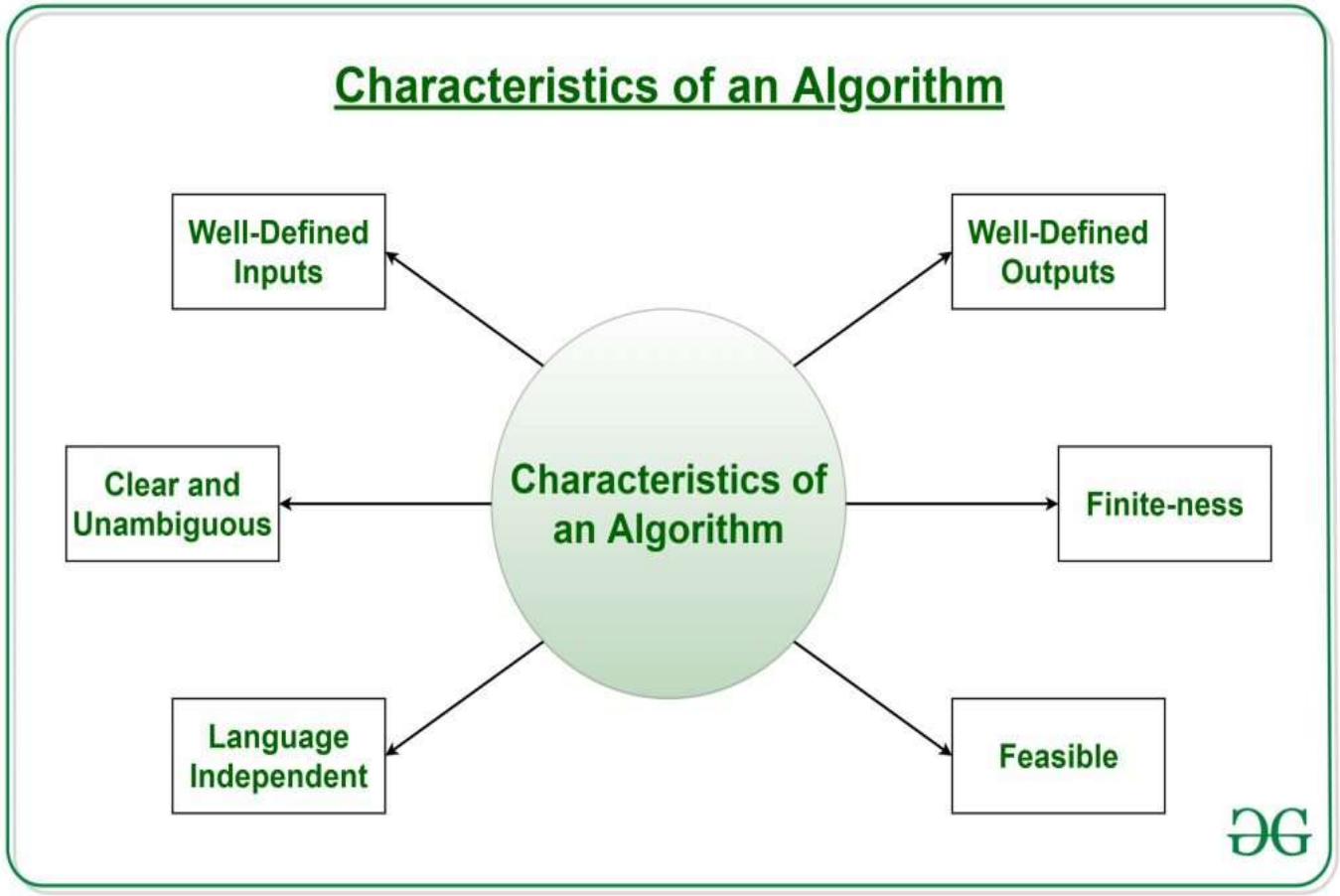
Algorithm development is an art – it needs practice, practice and only practice!

What is a good algorithm?

- It must be correct
- It must be finite (in terms of time and size)
- It must terminate
- It must be unambiguous
  - Which step is next?
- It must be space and time efficient

A program is an instance of an algorithm, written in some specific programming language

Characteristics of a good Algorithm



Algorithm development: Basics

- Clearly identify:
  - what Inputs are required?
  - what is the Output?
  - What steps are required to transform input into output
    - The most crucial bit
    - Needs problem solving skills
    - A problem can be solved in many different ways
    - Which solution, amongst the different possible solutions is optimal?

How to express an algorithm?

- A sequence of steps to solve a problem
- We need a way to express this sequence of steps
  1. Natural language (NL) is an obvious choice, but not a good choice. Why?
     - NLs are notoriously ambiguous (unclear)
  2. Programming language (PL) is another choice, but again not a good choice. Why?
     - Algorithm should be PL independent
  - We need some balance
    - We need PL independence
    - We need clarity
    - Pseudo-code provides the right balance

Steps of Problem Solving in Computer Science

Problem

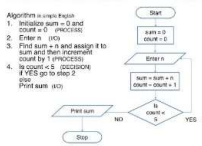
Analyse and Understand the problem statement.

- Identify the expected actions of the mathematical solution.
- Identify the supporting inputs
- Identify the outputs (result data).

An Explicit statement of the problem

Build the Algorithm.

- Write the Algorithmic solution
- Validity Check and correction



Algorithmic Solution

Coding

- Chose à Programing Language
- Rewrite the Algorithm in the chosen PL

Program

Compile and Execute

General structure of an Algorithm

Algorithm <Algorithm id> — Algorithm header

Type
// liste of specific data types
Const
//List of constantes
Var
// Liste of Variables

— data declaration part

Begin

// Liste of Instructions

— instruction part

End.

Standard Data Type

Data is classified into **data types**. e.g. char, float, int, etc.
A data type is (i) a **domain** of allowed values and (ii) a set of **operations** on these values.

1. **Integer int :** الأعداد الصحيحة
2. **Character char : denote Alphabet letters; digits and special character (symbols)**
3. **Real (Floating point) real, float** : represent fractional numbers
4. **Boolean Bool :** this type support two values **True** and **False**
5. **String** : A string data type is a combination of characters

Basic Operations

1- Basic Arithmetic Operations

The four basic arithmetic operations in Maths, for all real and Integer numbers, are:

- Addition (Finding the Sum; '+')
- Subtraction (Finding the difference; '-')
- Multiplication (Finding the product; '×')
- Division (Finding the quotient; '/')

Two additional operation for Euclidean division for integer numbers only:

- Div the quotient of Euclidean division
- Mod the remainder of Euclidean division

Basic Operations

2- **Basic Logical Operations**

Three basic logical operations for boolean data type

. Negation No.

. Conjunction And.

. Disjunction Or

3- **comparison operators** : >, <, <>, =, >=, <=

Comparison operators compare two values of the same type and return True or False. (Such expressions are sometimes called Boolean expressions.)

4- **String concatenation: '+' '.'**

Only whith string data type, which returns the concatenation of its right and left arguments

The Basic Statements

An Algorithm is composed of *statements*, which define the computation by creating and manipulating variables, assigning data-type values to them, and controlling the flow of execution of such operations. Statements are often organized in blocks, sequences of statements within curly braces.

The Basic Statements

Declarations.

A *declaration* statement associates a variable name with a Data type,

The variables are used as data containers to temporally memorize input data, intermediate data and output data,

Var

<ID> **:** <data type> **;**

Exemple :

a: int;

Long,larg : real;

L: char;

in c language :

int a;

float long, larg;

char l;

Identifiers ID

Choosing meaningful names for

the **variables**, **constants** and **subroutines** makes it easier for

the next person to work on the code to understand it.

These names are called **identifiers** and they usually follow

certain rules:

rules

- They can contain letters and numbers but must start with a letter.
- They must contain at least one letter (at the start of the name).
- They must not contain special characters such as !@£$%&* or punctuation characters. However, an underscore '_' can be used.
- Spaces are not allowed.
- They will normally contain lower case letters. However, upper case letters can be used if a variable name comprises more than one word joined together.
- The name should be meaningful - it should represent the value it is holding.

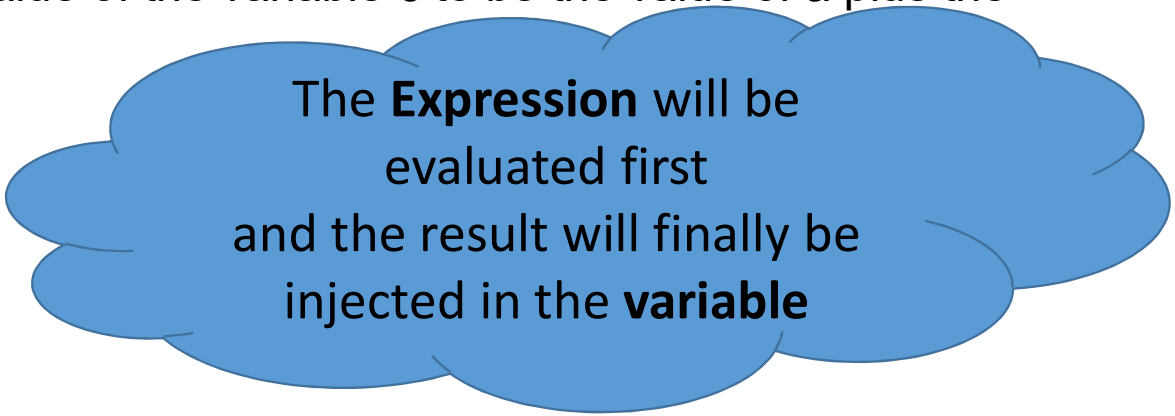
Constants Id

Constants follow the same naming conventions as variables except that they are conventionally written in **upper case**.

Assignments ‘:=‘ ‘←’ (we use ‘ = ‘ in c language)

An *assignment* statement associates a data value (defined by an expression) with a variable. When we write c := a + b , we are not expressing mathematical equality, but are instead expressing an action: set the value of the variable c to be the value of a plus the value of b

<Var_ID> := <Expression> ;



Exp:

a := b;

b := (a-5)/(b+c);

A := 18;

Input Statement

Used for entering data to be manipulated with the Algorithm

Read(<var_id>[,<var_id]);

Exp :

Read(larg): in c language scanf("%f",&larg);

Read(a,b,c); scanf(« %d %d %f",&a, &b, &c);

Output Statement

Used for printing out text messages and Algorithm

results from variable contents,

Write (<text|var_id> [,<text|var_id])

Write('Hello World') ;

in clanguage printf("Hello World");

Write('the result = ' , surf) ;

printf("the result =%f", surf);

CONDITIONAL Statement

In a conditional statement we make a test. The result of the test is a Boolean - either True or False. If the result of the test is True we take a certain course of action and if the result of the test is False you take another course of action.

```
IF <condition >
        THEN <sequence 1 >
        ELSE <sequence 2 >
ENDIF
```

If the condition is True *sequence 1* is executed, otherwise *sequence 2* is executed.

Note : The ELSE sequence is optional.

CONDITIONAL Statement -Example-

```
if moy >= 10
        then
                write('Congratulation, you are passed!')
        else
                write('Sorry, you are failed!')
end if;
```

What will be the output if moy is equal to 15,,2?

In c language :

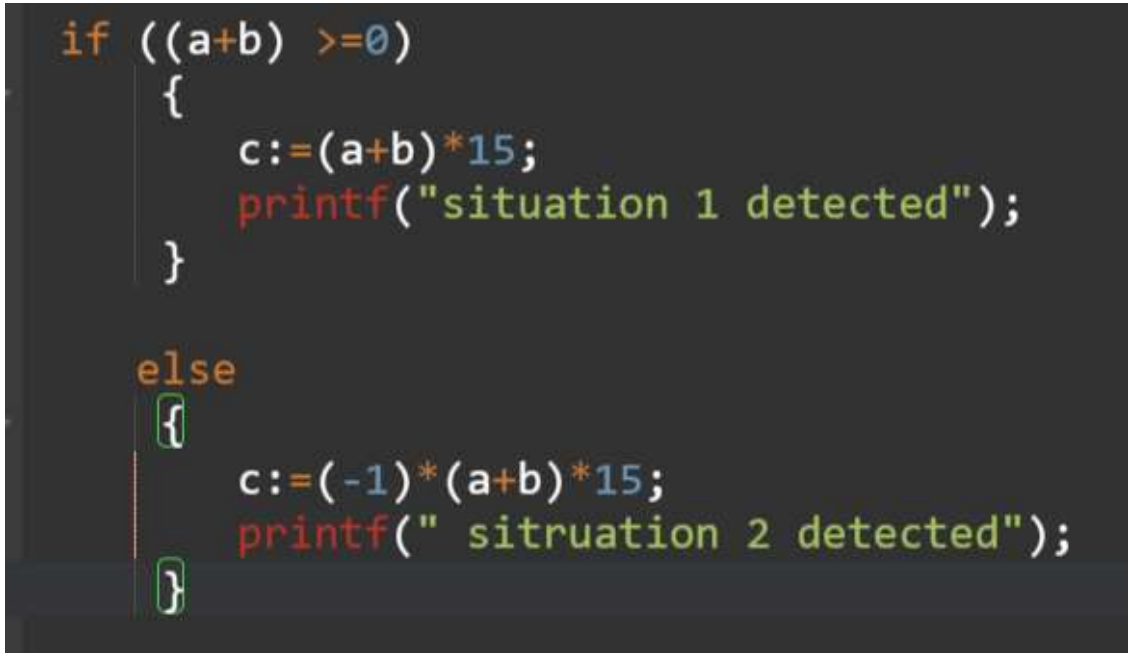
```c
if (moy >=10) printf("Congratulation, you are passed!");

   else printf(" Sorry, you are failed");
```

CONDITIONAL Statement

Notes:

- in c-language, the condition must be written in brackets,
- in c-language, if sequence 1 or sequence 2 have more than one instruction, we use ‘{‘ and ‘}’ to delimit the sequence



```
if ((a+b) >=0)
    {
        c:=(a+b)*15;
        printf("situation 1 detected");
    }

    else
    {
        c:=(-1)*(a+b)*15;
        printf(" sitruation 2 detected");
    }
```

Algorithm representation using Flowcharts

The Flowchart

- (Dictionary) A schematic representation of a sequence of operations, as in a manufacturing process or computer program.
- (Technical) A graphical representation of the sequence of operations in an information system or program.
  - Information system flowcharts show how data flows from source documents through the computer to final distribution to users.
  - Program flowcharts show the sequence of instructions in a single program or subroutine. Different symbols are used to draw each type of flowchart.

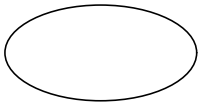
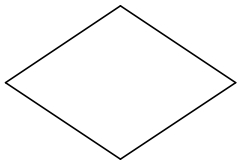
The Flowchart

A Flowchart

- shows logic of an algorithm
- emphasizes individual steps and their interconnections
- e.g. control flow from one action to the next

Flowchart Symbols

Basic

Name	Symbol	Use in Flowchart
Oval		Denotes the beginning or end of the program
Parallelogram		Denotes an input operation
Rectangle		Denotes a process to be carried out e.g. addition, subtraction, division etc.
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of two routes. (e.g. IF/THEN/ELSE)
Hybrid		Denotes an output operation
Flow line		Denotes the direction of logic flow in the program

Example

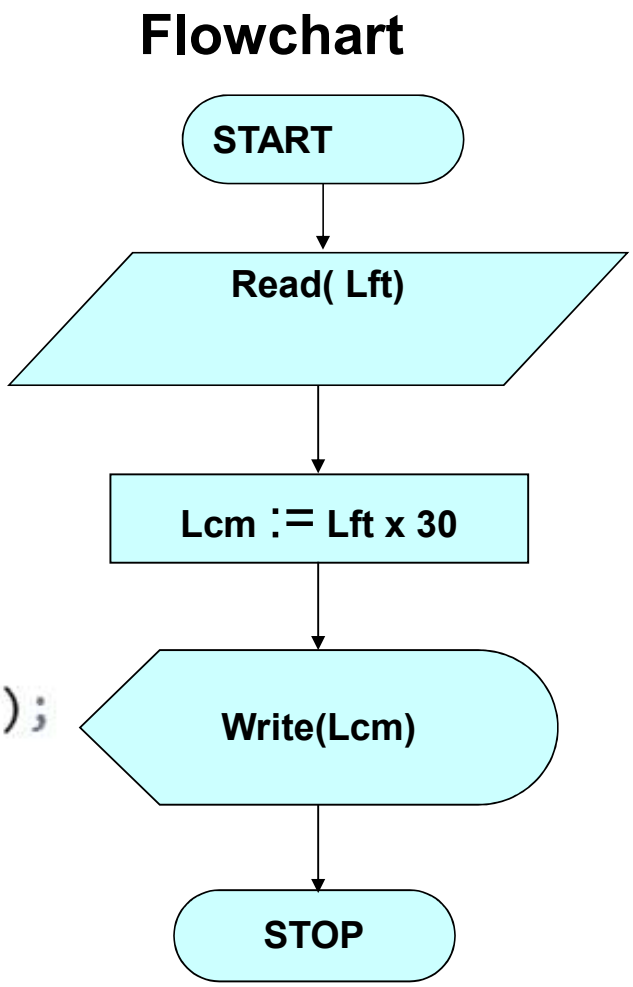
- Write an algorithm and draw a flowchart to convert the length in feet to centimeter.

Pseudocode:

- *Input the length in feet (Lft)*
- *Calculate the length in cm (Lcm) by multiplying LFT with 30*
- *Print length in cm (LCM)*

Example 1

```
Algorithm Transform_unitStep
Var
   Lft,Lcm : real;
Begin
write('give the length in feet :')
  read(Lft);
  Lcm:= Lft*30;
  Write('the length =', Lcm, 'centimeter');
end.
```



Example 2

Write an algorithm and draw a flowchart that will read the two sides of a rectangle and calculate its area.

Pseudocode

- *Input the width (W) and Length (L) of a rectangle*
- *Calculate the area (A) by multiplying L with W*
- *Print A*

Example 2

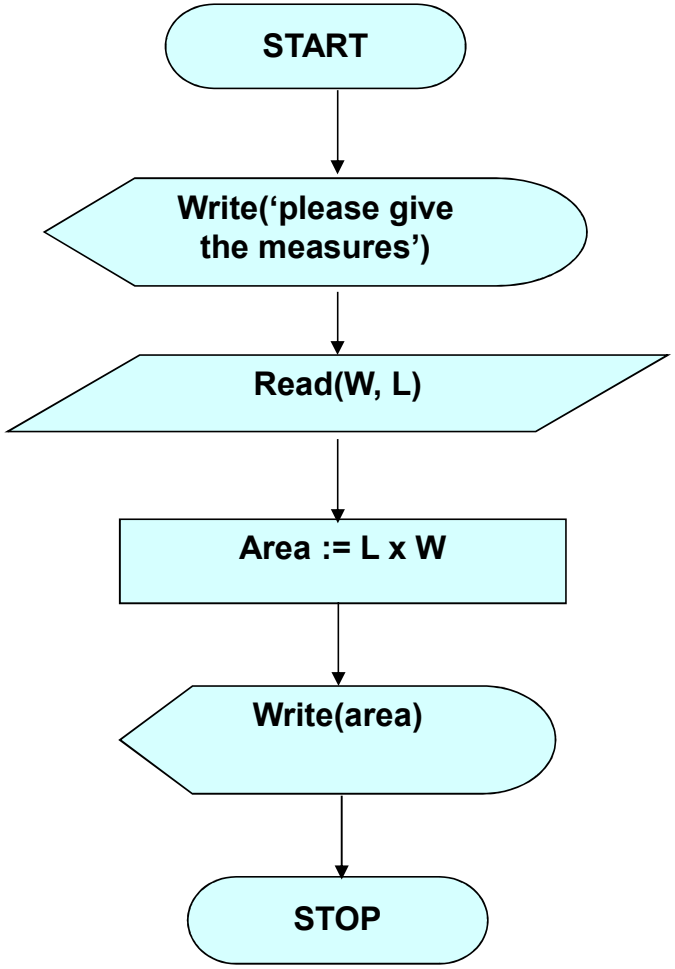
```
Algorithm  Area_Rectangl;
var
  l,w, area : real;

begin

  writeln ('Please enter the width anfd the length');
  readln(l,w);
  area:= l*w;

  writeln ('The calculated surface =', area);

end.
```

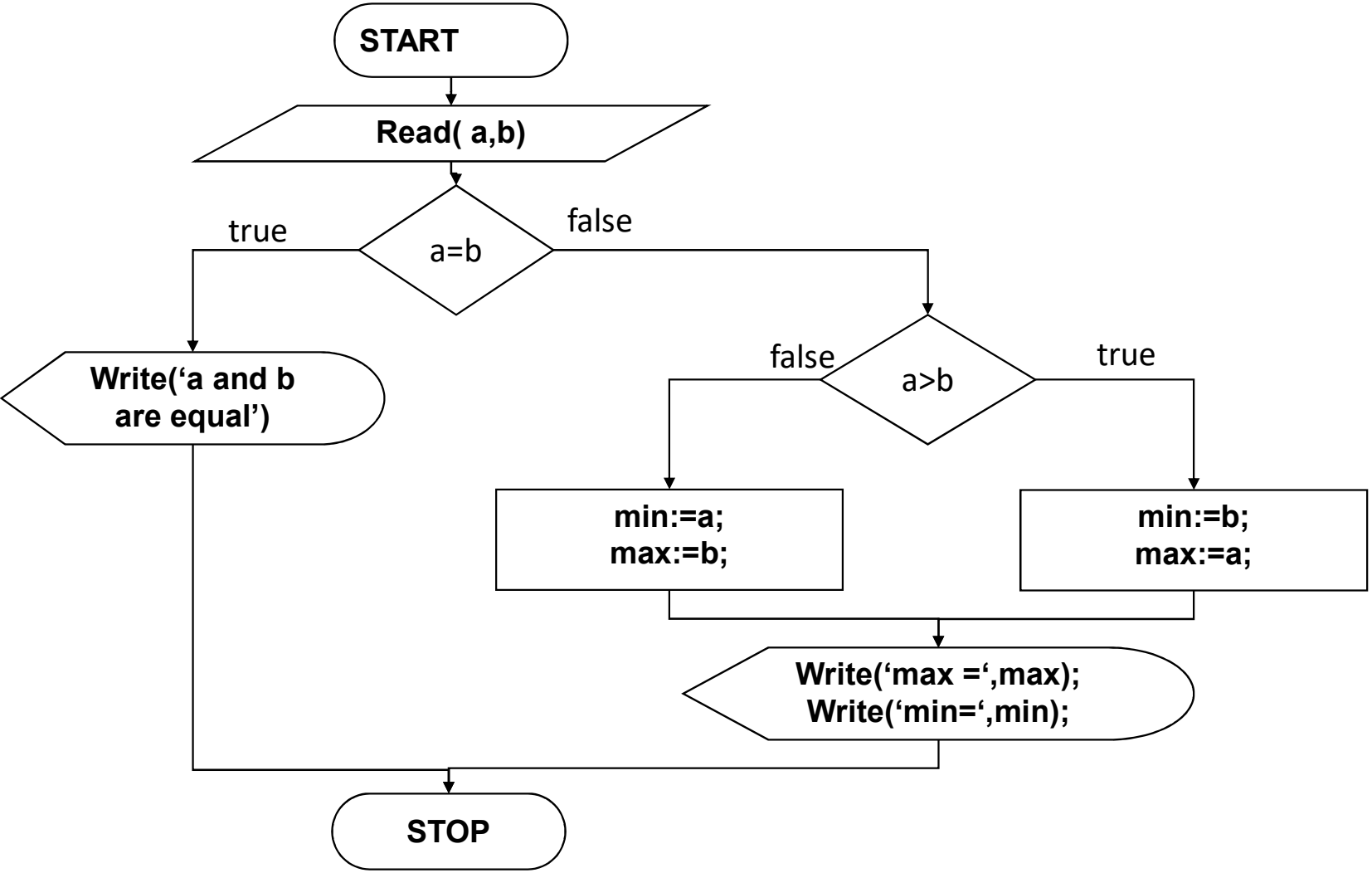


Exemple 3 the min_max Algorithm

```
Algorithm  max_min;
var
  a,b,max,min: integer;
begin
  writeln ('Enter the two numbers a,b :');
  read(a,b);
  if a=b then writeln('the two numbers are equal')
         else if a>b then min:=b;
                          max:=a;
                     else min=a;
                          max:=b;
              endif
              write('the maximum =',max);
              write('the minimum =',min);
  endif

end.
```

Exemple 3


START
Read(a,b)
a=b
true
false
Write(‘a and b are equal’)
a>b
false
true
min:=a;
max:=b;
min:=b;
max:=a;
Write(‘max =‘,max);
Write(‘min=‘,min);
STOP

Switch case statement

Switch case statement evaluates a given variable(the Selector) and based on the evaluated value(matching a certain condition), it executes the statements associated with it. Basically, it is used to perform different actions based on different conditions(cases).

•Switch case statements follow a selection-control mechanism and allow a value to change control of execution.

•They are a substitute for long if statements that compare a variable to several integral values.

•The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the selector.

Syntaxe

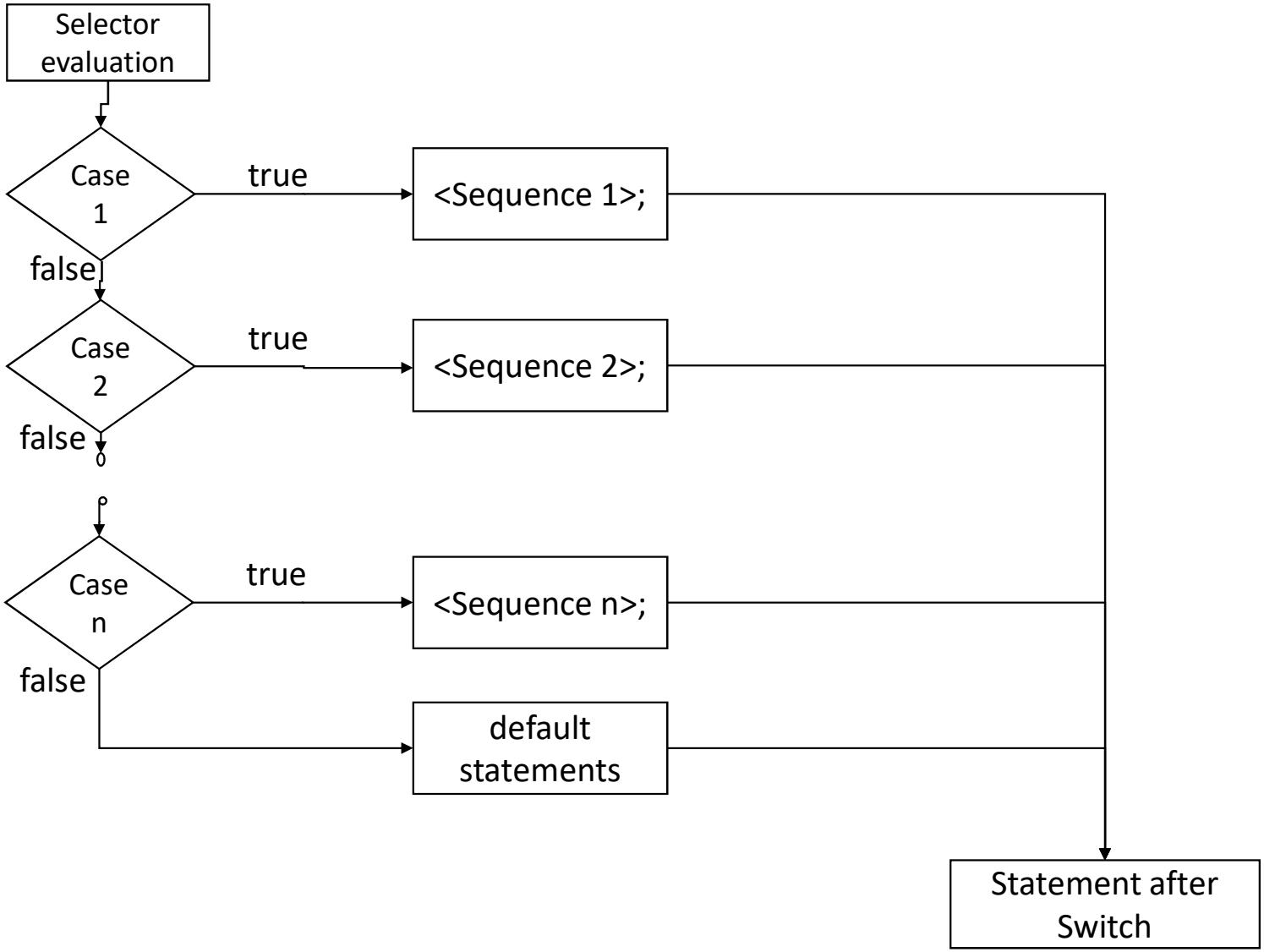
```
Switch  <selector>
  case value1: <sequence_1>;
  case value2: <sequence_2>;
  .
  .
  .
  case value_n: <sequence_n>;
  else        : <default_sequence>;
End switch;
```

Rules of the switch case statement

Following are some of the rules that we need to follow while using the switch statement:

- In a switch statement, the <selector> and "**case value**" must be of "**char**" and "**int**" type (enumerated type in general).
- There can be one or N number of cases.
- The default Statement is optional. The default keyword is used to specify the set of **statements to execute if there is no case match**.

Switch corresponding flowchart

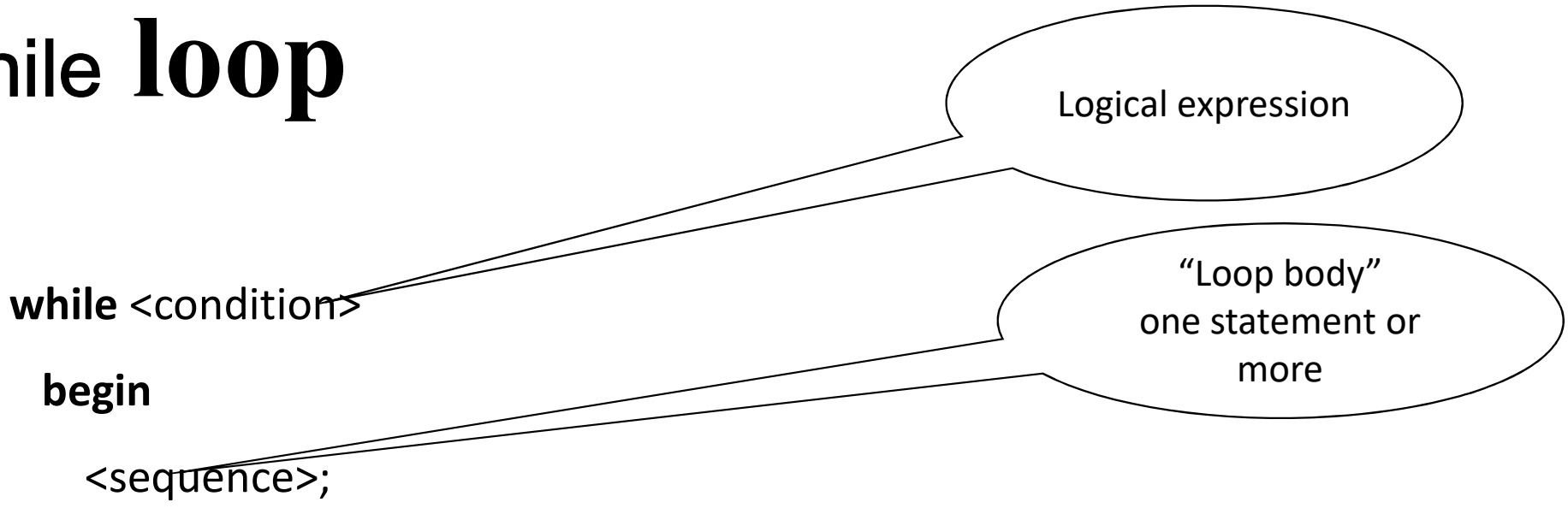


Repetition structures

Repetition structures

- Called *loops*,
- Used to **repeat** the **same code** mulitple times in succession.
- The **number of repetitions** is based on criteria defined in the loop structure, usually a true/false expression
- Three loop structures are:
  - **while** loops
  - **do-while** loops
  - **for** loops

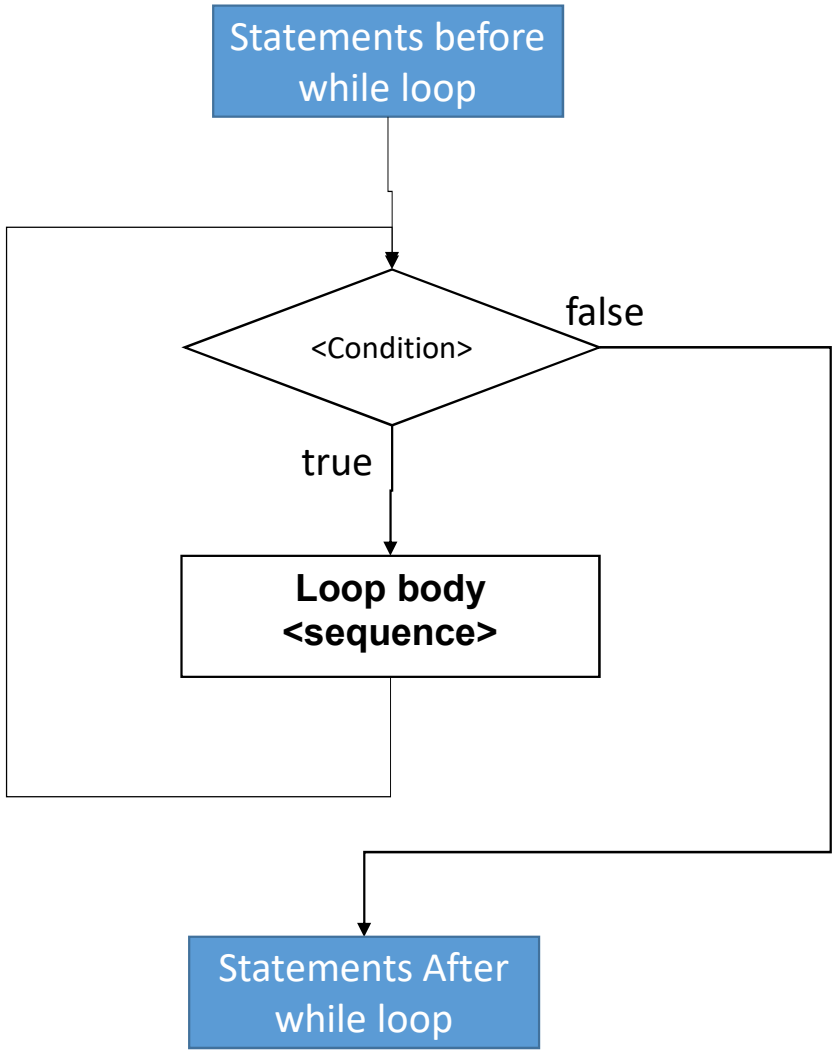
while **loop**



•The condition is evaluated to decide whether the loop takes a new iteration (repetition) or not.

- •**true** means run the loop body again.
- •**false** means quit the loop.

While loop Flowchart



While loop : Example The Average of Readen Measures

We need to calculate the Average of a set of given measures, we continue reading while the entered measure is a positive value

Inputs:

the measures, we need one variable mes of type real in which we read many time,

Output :

the Average of type real

Process :

1- read the first measure
2- initialize the sum to 0
3- initialize the counter to 0
4-If the read m**ea**sure >=0 then goto -5-
else goto -7-
5- add the measure to the sum
6- increment the counter
7- Read an other measure
8- goto -4-
9- calculate the average avg = sum/cnt
10-print out the average

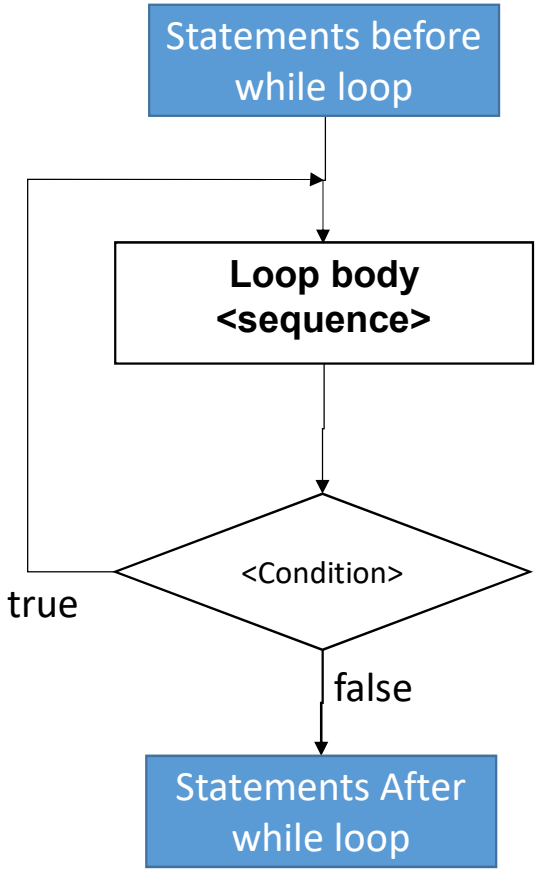
```
Algorithm Average_Measure
var
  mes,Avg,som :real;
  cnt : integer;
begin
  som:=0;
  cnt := 0;
  read(mes);
  while mes>=0 do
   begin
     som:=som+mes;
     cnt:=cnt+1;
     read(mes);
   end;
  avg:=som/cnt;
  write('the Average =',Avg);
 end.
  writeln ('Hello World')
end.
```

do-while loop

The ***do..while*** loop is similar to the ***while*** loop with one important difference :

The body of ***do...while*** loop is executed at least once.

Only then, the test expression is evaluated.



do

<sequence>;

while <condition>

Do- - while loop Exemple

The same last Algorithm is given

here using the do-while loop:

- In this version it is not necessary to

separate the first reading of the measure

and the following readings, due to the fact

that the do-while loop ensures a first

iteration before the first test

```
Algorithm Average_Mesure
var
   mes,Avg,som :real;
   cpt : integer;
begin
   cpt := -1;
   mes:=0;
   do
      som:=som+mes;
      cpt:=cpt+1;
      read(mes);
   While mes>=0 ;
   avg:=som/cpt;
   write('the Average =',Avg);
end.
```

For loop

- When the number of iterations is known we use simply the for loop,
- The loop mechanism is based on a loop counter that allows counting the number of iterations that have been executed

Syntax

```
for   <cnt> := s_v   to  f_v  [step=p]
  begin
      <sequence >;
  end;
```

1- initialize cnt with the starting value

2- if cnt is less or equal to the stopping value

then goto -3-

else goto -5- (exit)

3- execute the <sequence>

4- increment the counter cnt (according to the step)

5 – instructions after the loop

For loop flowchart