Ministry of Higher Education and Scientific Research
Djilali BOUNAAMA University - Khemis Miliana(UDBKM)
Faculty of Matter Science and Computer Science
Department of Mathematics

Chapter : 2

Simple Sequential Algorithm

MI-L1-UEF121 : Algorithms and Data Structures I

Ali Khalfi

Khalfiali.udbkm@gmail.com

Course Topics

1. Notion of language and algorithmic language
2. Part of an algorithm
3. Data: variables and constants
4. Data types
5. Basic operations
6. Basic instructions
7. Construction of a simple algorithm
8. Representation of an algorithm by a flowchart
9. Translation into C language

Reminder : The problem

Goal

- ✓ Getting the “**machine**” to do work for us.

Problem

- ✓ explain to the "**machine**" how it should do it.
  - ➢ *how to tell it?*
  - ➢ *How to teach it?*
  - ➢ *How do we make sure it does this job as well as we do?*
  - ➢ *Even, Better than us?*

Objectives

- ✓ ***solve*** problems “like” a **machine**
- ✓ know how to ***explain*** your reasoning
- ✓ know how to ***formalize*** your reasoning
- ✓ ***design*** (and write) **algorithms**

Concepts covered

- ✓ **Basic Concepts**
  - ➢ "***basic***" algorithms for elementary problems
- ✓ **Learning a language**
  - ➢ Algorithmic ***formalism***,
  - ➢ ***programming*** languages: C, Pascal
- ✓ **Data Structures**
  - ➢ from the ***simplest*** to the most ***complex***
- ✓ **Complex problem solving**
  - ➢ ***clever*** and ***efficient*** algorithms

Algorithms

Definition

- ✓ An **ALGORITHM** is a ***sequence of instructions*** which, once ***executed correctly***, leads to a ***given result*** (desired).

- ✓ **Examples : Algorithms**

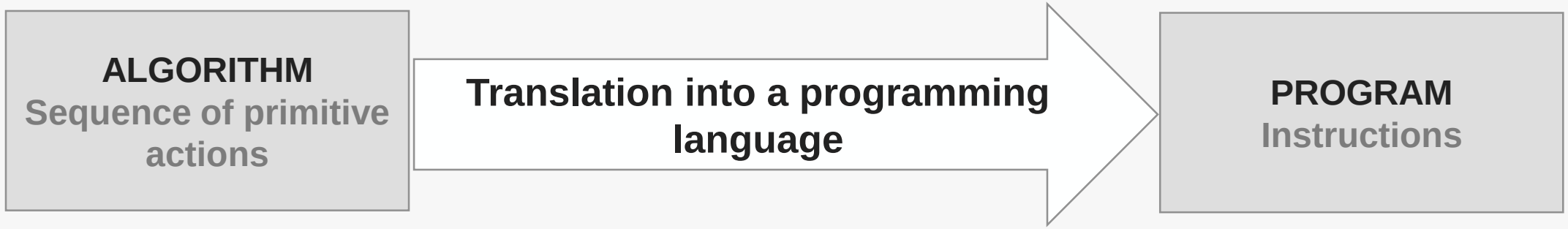
  - ➢ show a way to a lost tourist;
  - ➢ write a cooking recipe;
  - ➢ Dispense drinks automatically;
  - ➢ Play a video on YouTube

1. Notion of language and algorithmic language

Program

- ✓ A **program** is a ***sequence of instructions*** written in a ***programming language*** **translating** an algorithm

- ✓ Each of its instructions specifies the operation that the computer must execute.



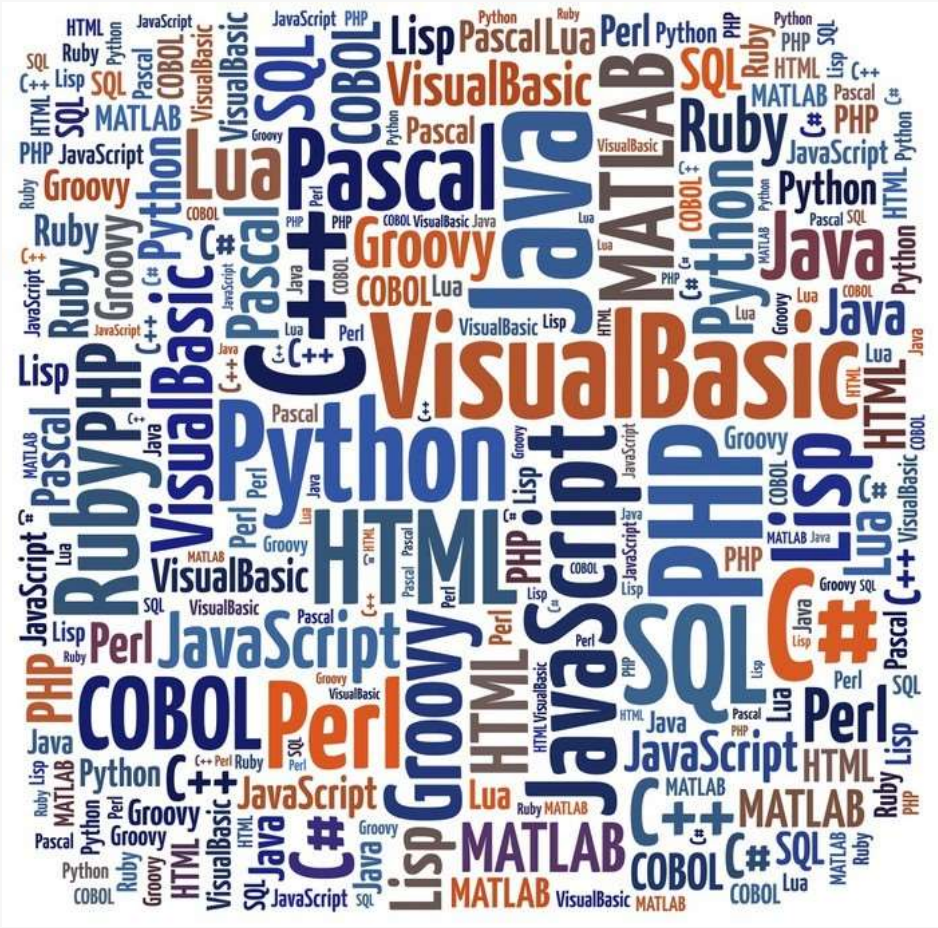
Algorithm vs. Program

Algorithm	Program
No machine required	Machine required
Written according to a formalism or an algorithmic description	Written in a programming language
Takes place	Runs or executed
Logical solution	Physical solution
result of the analysis	translation of an Algorithm

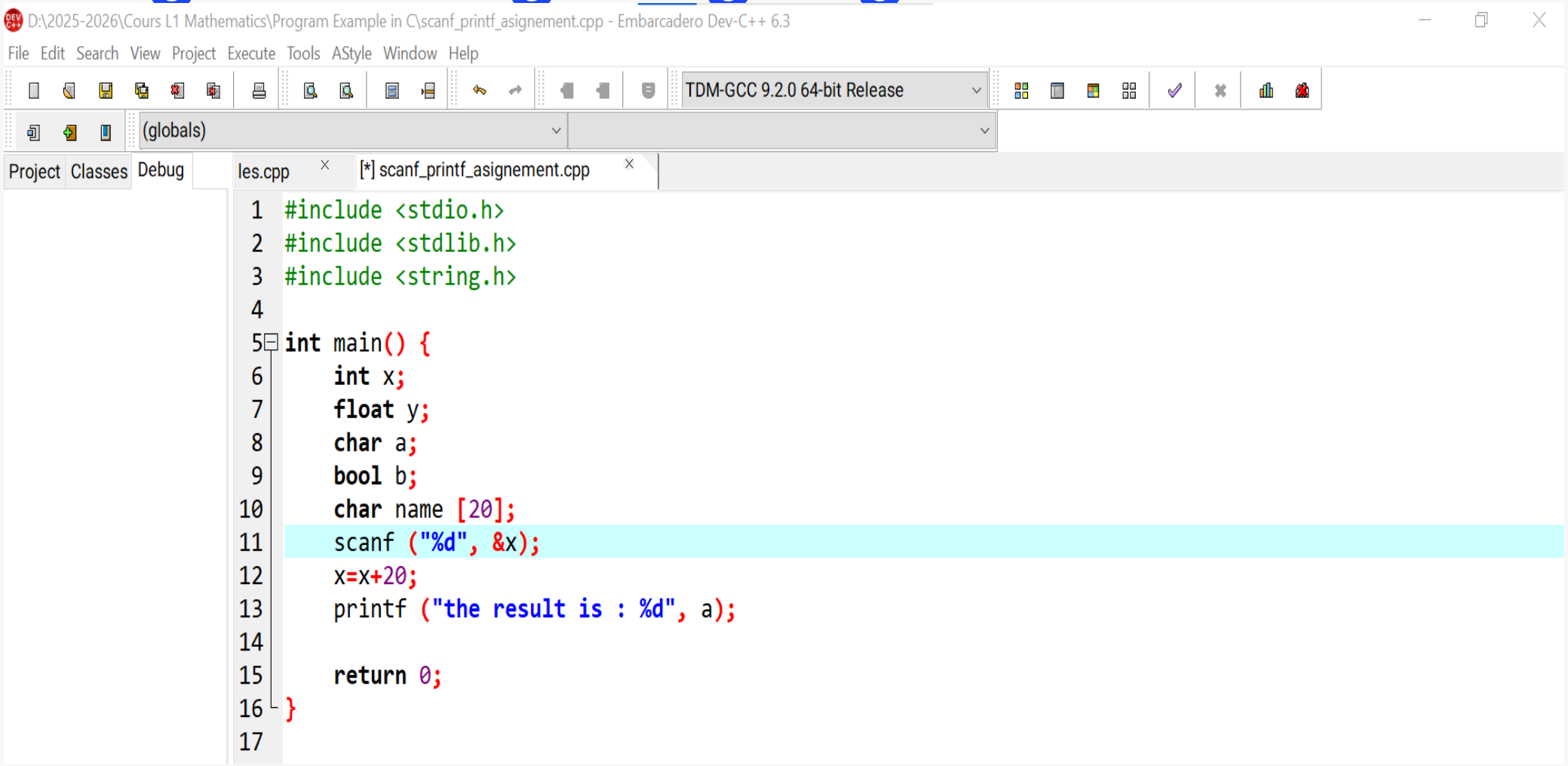
Programming language

- ✓ A **programming language** is a conventional notation intended for ***formulating*** (translating) algorithms and ***producing*** (developing) programs
- ✓ it is an **abstraction** of operations that can be carried out on a computer

- ✓ **Examples :**
  - ✓ Pascal, C, Python, Java, C++, C#, PHP, JavaScript



Programming language : C or C++



Programming Tools

1. ***Editor***:

   ✓ A text or ***source code*** editor is software intended for creating and editing text files (program source files).

   Examples: NotePad++, Sublime Text, Atom, Brackets ...

2. ***Compiler***:

   ✓ It is a program that transforms ***source code*** (written in a programming language) into ***object code*** to create a machine-***executable program***.

   ✓ **Examples:**

   ➢ ***C Langage***: GCC, Borland C,

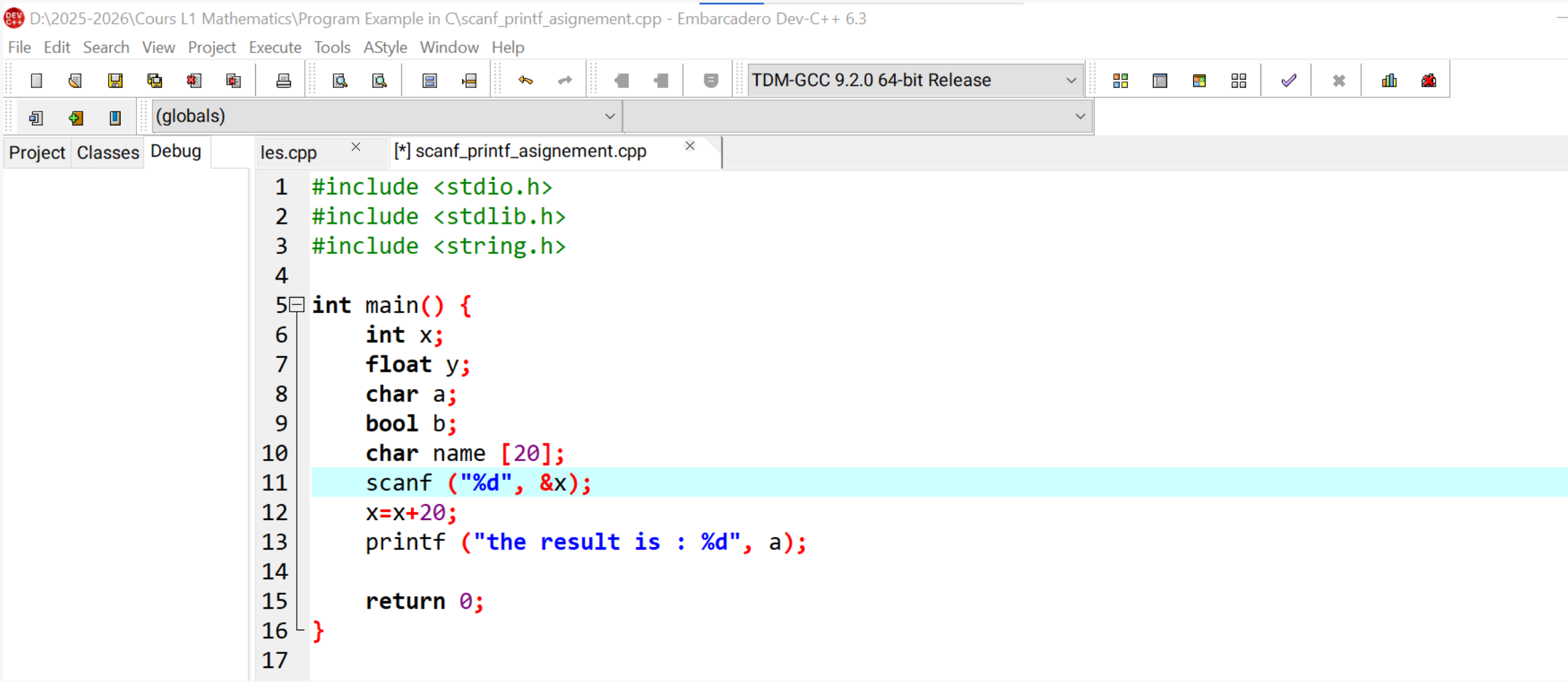
   ➢ ***Pascal Langage***: Turbo Pascal, Free Pascal ...

Programming Tools

3. ***IDE***:

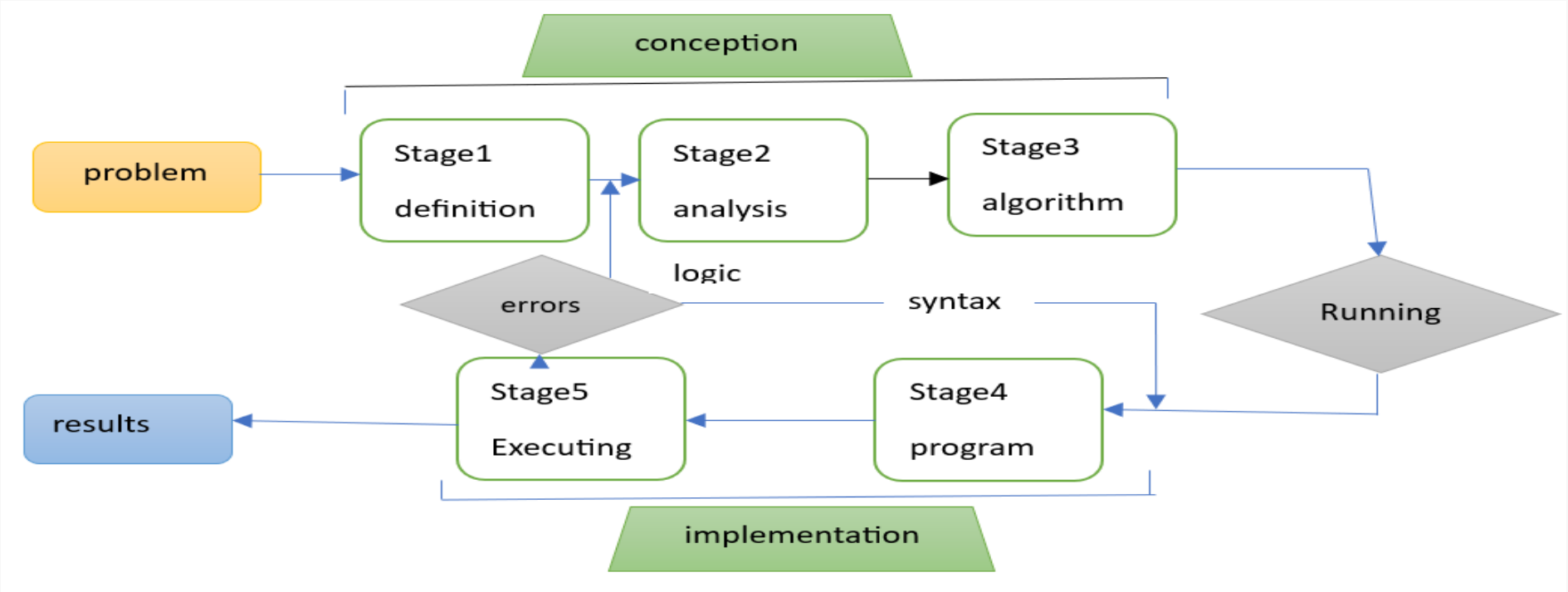
- ✓ The Integrated Development Environment (IDE) brings together a set of tools specific to program development. It can contain :
  - ➢ A text editor
  - ➢ A compiler
  - ➢ A debugger
  - ➢ A GUI creator ....etc
- ✓ **Examples:**
  - ➢ ***C/C++ Langage***: DevC++, Code::Blocks, Visual Studio Code, Eclipse + CDT ...
  - ➢ ***Pascal Langage***: Lazarus, Free Pascal ...
  - ➢ ***Python*** : PyCharm, Spyder, Visual Studio Code, Jupyter Notebook ...

Programming Tools



From Problem to Solution

Solving a problem in computer science



Step 1 : Problem definition

- It is about :
  1. ***Determine*** all ***available information***
  2. ***Define*** the shape of the ***desired results***

- In this step, we ask the following questions:
  1. ***What*** is ***given*** as ***input*** (***input*** or ***initial data***)?
  2. ***What outputs*** are ***requested*** (***output*** or ***results***)?

Step 2 : Problem Analysis

- It consists of :
  - ***a.*** ***Find*** the way to get from ***data*** to ***results***.
    - a. Natural language, scientific formula, diagram or drawing, ...
  - ***b.*** ***Acquire*** the reflex to ***propose adequate solutions*** to a given problem
    - ***a. Solution by theory:*** mathematical, physical problem, etc.
    - ***b. Solution by synthesis:*** store management, sales management, etc.
    - ***c. Solution by experience:*** storage and search problem, ...
- In this step, we ask the following questions :
  1. ***How*** to solve the ***problem*** ? How do we get to the ***results***?
  2. ***What*** is the solution to this ***problem***? What is the form of ***result?***

Step 3 : Writing an Algorithm

- You have to :
  1. ***Write*** a ***clear*** and ***unambiguous*** solution.
  2. ***Represent*** the solution by ***an algorithm*** written in "***Algorithmic Language***" or "***Algorithmic Formalism***"
  3. ***Run*** the Algorithm and check its operation (***general*** case, ***special*** cases)

- In this step, we ask the following questions :
  1. ***How*** to represent the solution in an algorithm?
  2. ***Which control structures*** to use?
  3. ***What variables*** and ***data structures*** to use?

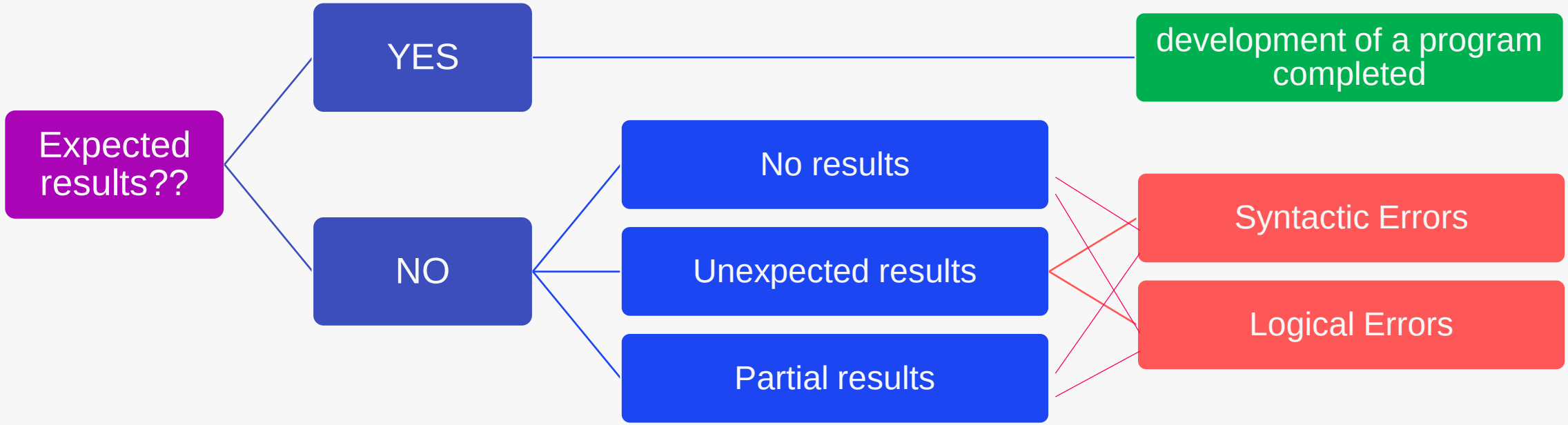
Step 4 : Program Implementation

- To concretely implement algorithm, we must :
  1. ***Choose*** a ***programming language***, and ***install*** its ***tools*** on a machine (PC)
  2. ***Translate*** Algorithm into this programming language

- In this step, we ask the following questions :
  1. ***Which programming language*** to use? (procedural, object-oriented, ...)
  2. ***Which IDE*** or ***code editor*** to use? (Paid, open source, free, ...)

Step 5 : Program maintenance (Test & Debug)

- When running the program on a machine :
  1. The machine ***checks*** the ***syntax*** (spelling) of the ***program***
  2. The machine ***translates*** and ***executes*** the meaning expressed by the ***program***



Example : Dividers of a Number

Problem: Return the list of dividers of a number

Problem : Return the list of dividers of a number

Step 1 – Definition : Let N be an integer; find a solution that allows you to display these dividers.

Step 2 – Analysis :

- We **divide** **N** by i=1, i=2, i=3, ...until i=N/2 (half of N)
  - Each time we **verify** the rest of the division of **N** by i
    - If it's « 0 », we **display** the divider (i)

From problem to solution

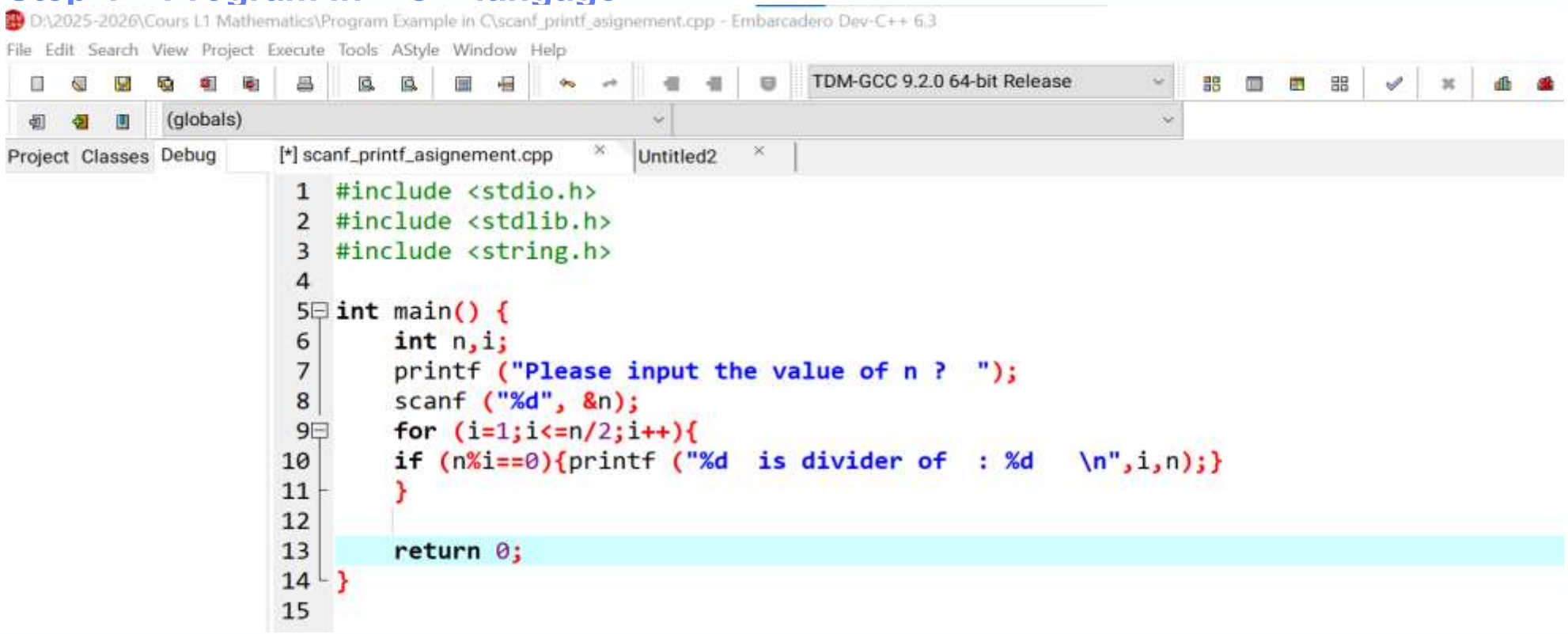
Example : Dividers of a Number

Step 3 – Algorithm

```
Algorithm  Divider;
Var  N,i : integer ;

begin
     //Inputs
   read (N);

      //manipulation of data
   for i from 1 to N DIV 2 do
         begin
            if N MOD i = 0 then
                  begin
                        //Outputs
                        write (i,' is divider ', N);
                  end
         end
end.
```

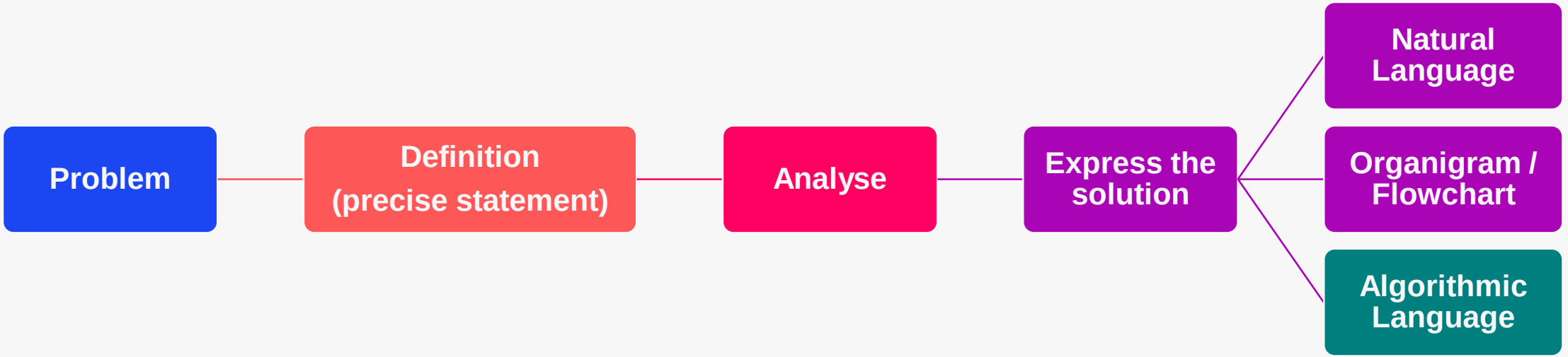
Exemple : Dividers of a Number

Step 4 – Program in « C » langage



2. Parts of an algorithm

Need for an algorithmic language



- ✓ **Natural Language** (literary descriptions): imprecision, ambiguity, own rules and conventions, different explanations of the same concept ...
- ✓ **Flowchart :** clutter, hard to modify and update, ...

Algorithmic Language

- ✓ An **algorithmic language** (or formalism) is a set of ***conventions*** (or ***rules***) in which we can express any solution to a given problem.
- ✓ Also called: **Pseudo Code**,
- ✓ **Properties :**
  - ➢ A ***common*** language;
  - ➢ Principle of ***communication***;
  - ➢ ***Precision*** and clarity (non-ambiguity)

Parts of an Algorithm

Header

Environment

Body

Algorithm name_of_algorithm

Type
Const
Var
$\left\{ \begin{array}{c} Declaration\ of \\ variables(objets) \\ and\ functions \end{array} \right\}$

begin

\- $\{Initializations\}$

\-

\- $\left\{ \begin{array}{c} ojects \\ processing\ and \\ manipulation \end{array} \right\}$

\-

\-

\- $\{Results\ and\ output\}$

end.

Comments

- ✓ Gives a human description in machine code.
- ✓ Simplified code maintenance and therefore, accelerate debugging
- ✓ Important when writing functions for other users

Syntax

```
//        a Single line comment
/*        Comment on
multiple lines */
```

Example : Square of a Number

Header

Environment Declaration

Body

```
Algorithm  Square;

Var  N, square : entier ;

begin
     //Inputs
   read (N);

      //manipulation des données
      square ← N*N

   //Outputs
      write ( ' the square of N ',N,' is', square);

end.
```

Example : Square of a Number

Programme en langage C

```c
#include <stdio.h>

int main (){

    int N, square;

    // Input reading
    scanf("%d", &N);

    square = N*N;

    /* result displaying  */
    printf(« the square of  %d is %d", N, square);

    return 0;
}
```

Library declaration

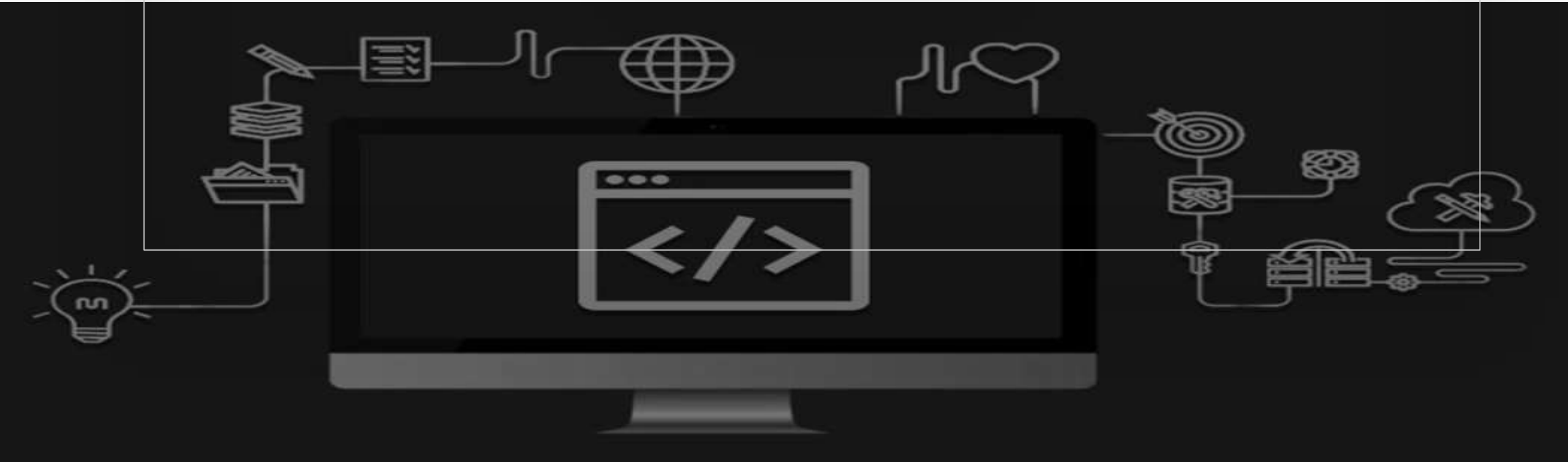
Declaration part

body

Main Fonction

3. Data: variables and constants

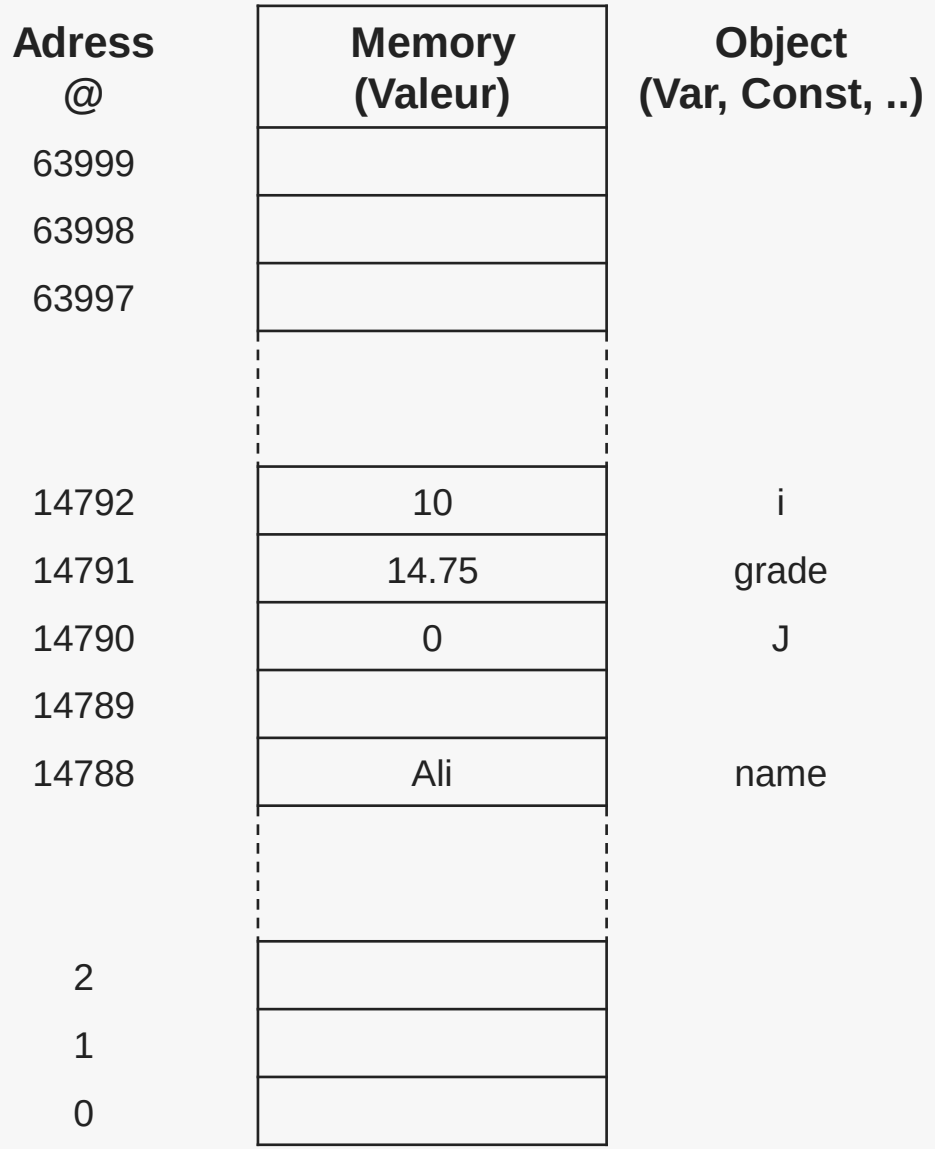
4. Data types



Objects: Memory Representation

- ✓ Any **object** manipulated by an **algorithm** (or a **program**) is ***stored*** in ***central memory*** (***RAM***).
- ✓ ***Central memory*** is made up of a ***series of contiguous boxes*** called memory "**boxes**" or "**cells**".
- ✓ Each **memory box** is characterized by :
  - ➢ An ***address***: ***unique*** which references the box
  - ➢ A ***space***: to store object ***values***

Objects: Memory Representation



Objects: Definition

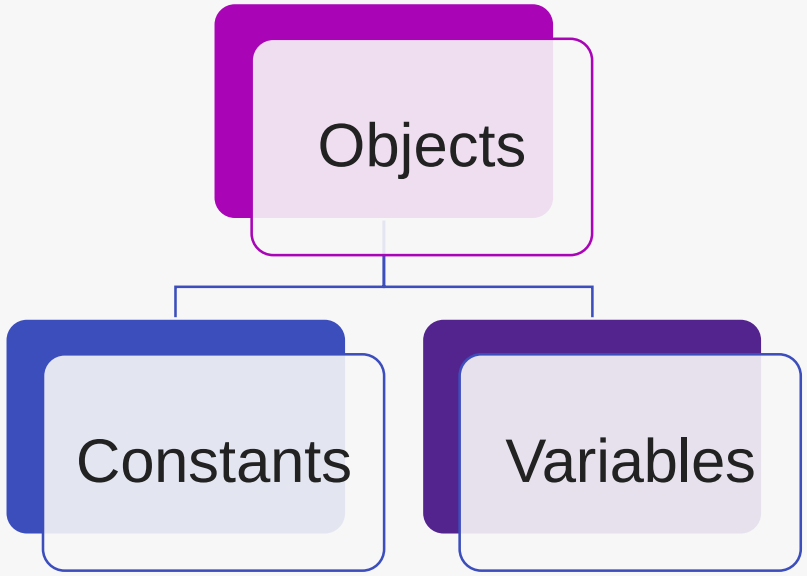
- ✓ All constituent **objects** of an **algorithm** must be ***described*** or ***declared*** in the ***environment*** (or the "***declaration***" part).
- ✓ Each object is characterized by :
  - ➢ A **Name**: "***unique identifier***": a series of ***alphanumeric characters*** which allows it to be designated and distinguished
  - ➢ A **Type**: which indicates the ***nature*** of the set in which it takes its values
  - ➢ A **value**: which indicates the ***size*** taken by an object at a given moment

Objects: identifiers

- ✓ A **name** or **identifier** is a sequence of ***alphanumeric characters*** whose first character is ***alphabetic***
  - ➢ The identifier can be: program name, variable and constant names, function names
  - ➢ Can contain letters and numbers
  - ➢ No (most) punctuation marks
- ✓ **Examples :**
  - ➢ **Valid identifiers**: product, i, j, T1, L_21, surface, student_name,
  - ➢ **Invalid identifiers**: 5 students, release date, x+y, T1, ...

Objets: Categories

- ✓ **Objects** are used to ***store data*** manipulated by the ***algorithm***
- ✓ There are two categories of objects
  - ➢ **Constant** : it is an object whose value is ***invariable***
  - ➢ **Variable** : it is an object that ***can vary*** during the execution of an algorithm .



Constants

Declaration

Const *name_constant* **=** *Value*

Examples

```
Algorithm  example_const;
    Const       pi = 3.14
                number = 100
                Lettre = 'M'
                Title = ‘Result :'
begin
    ...
    ...
end.
```

Variables, Constants and Types

Déclaration

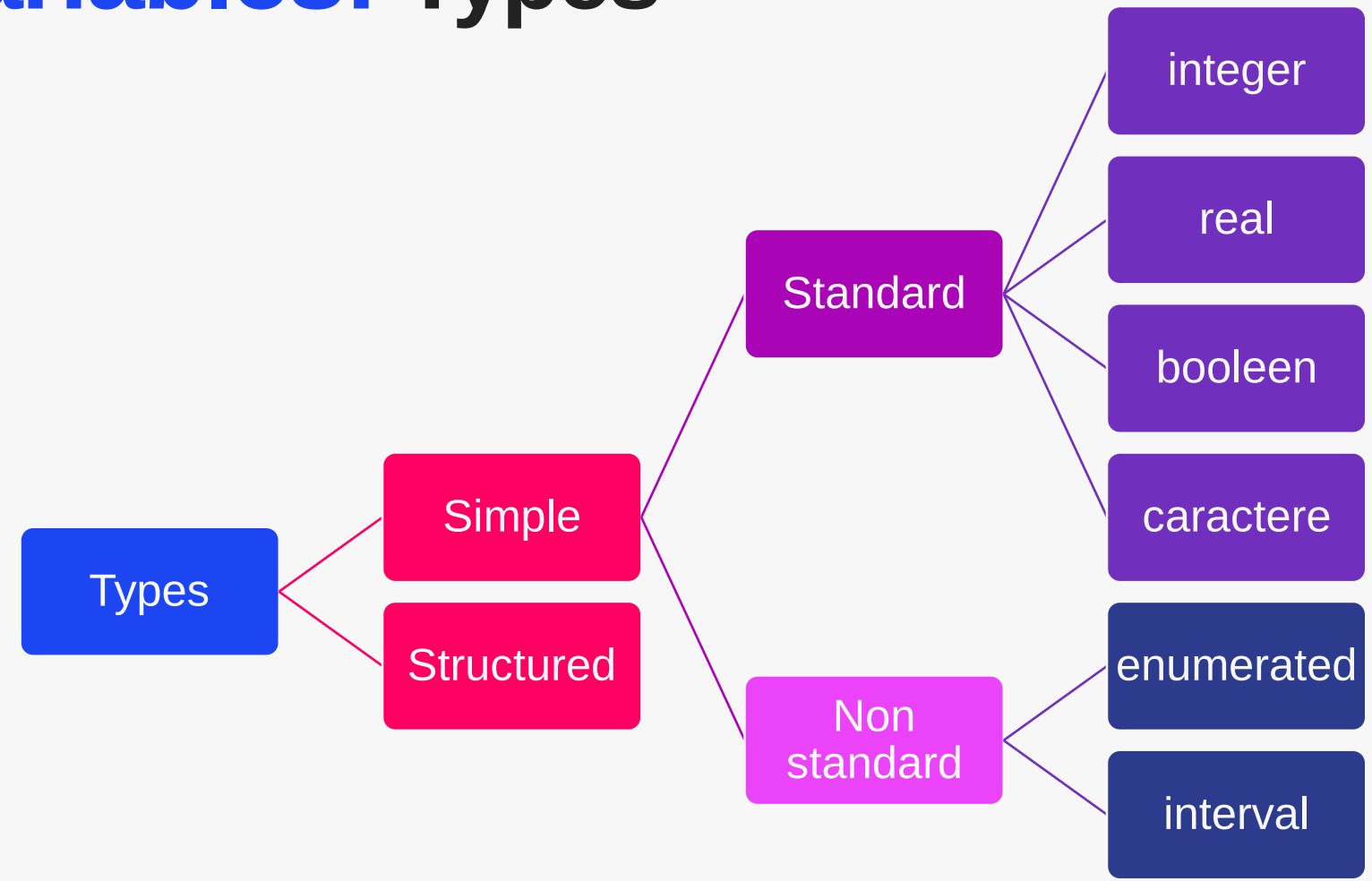
Exemples

C

#define Name_CONST value

```c
#include <stdio.h>

#define PI 3.14
#define Number 100
#define LETTRE 'M'
#define TITLE ’Result :'

int main (){

    /*
          ...
    */
    return 0;
}
```

Variables: Types



Variables

Declaration

```
Var       name_variable : Type
```

Examples

```
Algorithme  example_vars;
    Var          C : character;
                 N, i, j, day, month: integer
                x, y, square_root : real
                find : boolean
begin
    ...
    ...
end.
```

Numerical Types

- **Integer** : it is the set of relative integers.
- **Real** : It is the set of numbers having a fractional part.

Type	Octets	range
Byte (octet)	1	0 to 255
simple integer	2	-32 768 to 32 767
long integer	4	-2 147 483 648 to 2 147 483 647
simple real	4	$-3{,}40\times10^{38}$ to $-1{,}40\times10^{45}$ for negative values $1{,}40\times10^{-45}$ to $3{,}40\times10^{38}$ for positive values
Double real	8	$1{,}79\times10^{308}$ to $-4{,}94\times10^{-324}$ for negative values $4{,}94\times10^{-324}$ to $1{,}79\times10^{308}$ for positive values

Numerical Types

Declaration

```
Var     name_variable1: integer
        name_variable2: real
```

Examples

```
Algorithm exemple_vars_num;
    Var         N, i, j, day, month: integer;
                x, y, square_root : real;

begin
    ...
    ...
end.
```

Numerical Types

Déclaration

Exemples

C

integer : short, int, long, double long
real : float, double, long double

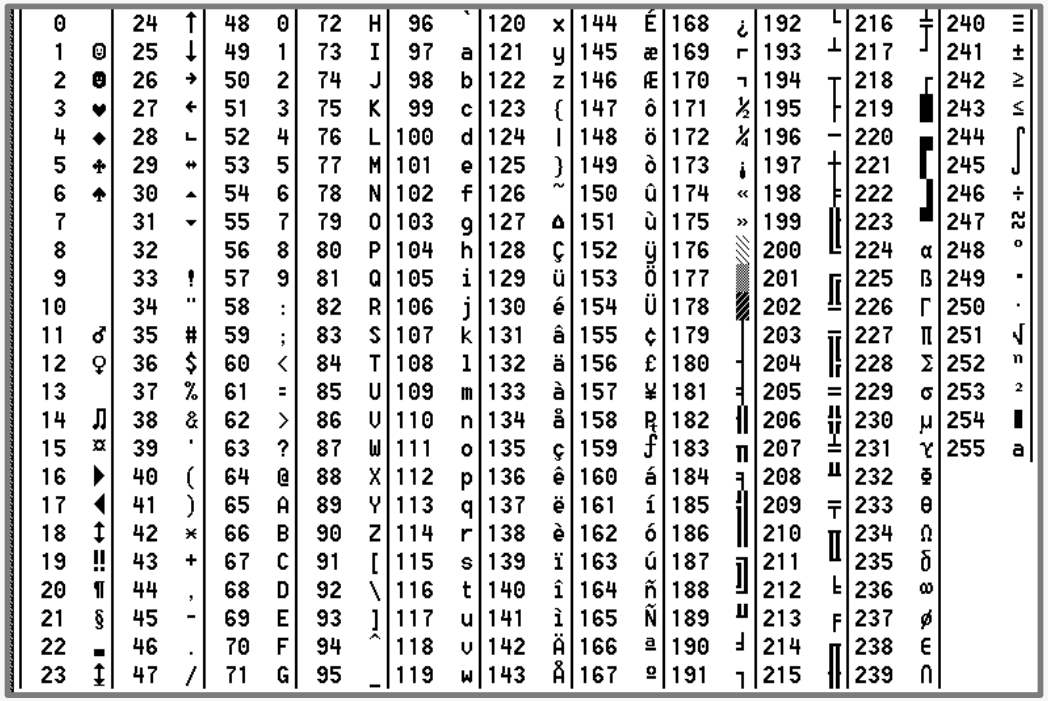
```c
#include <stdio.h>

int main (){
    int x, y, z;
    int day, year;
    short age;
    float lenght, witdh;
    float surface;
    double total;

    /*
          ...
    */
    return 0;
}
```

Alphanumeric Types

- **CHARACTER** : it matches a single character (depending on the system). Character sets may vary.
  - Placed between two single quotes.
  - It includes all alphabetical, numerical characters, punctuation marks, special signs, space (blank), empty character, etc.
- **STRING** : It is a set (group) of characters.

Alphanumeric Types

- Each character has an ASCII code
- The ASCII code is a table of 0 to 255 characters which contains: lowercase and uppercase letters, numbers, punctuation, special symbols and graphics ...



ASCII Code

Alphanumeric Types

Declaration

```
Var     name_variable1: character
        name_variable2: string
```

Examples

```
Algorithme  example_vars_aplha;
    Var         c, aplha: character;
                day, month, name: string;

begin
    ...
    ...
end.
```

Alphanumeric Types

Declaration

Examples

C

character : char
String : table if characters

```c
#include <stdio.h>

int main (){
    char c;
    char word[];
    char name[25], last_name[25];

    //
    c = code;
    printf("c = %c", c); // display : c = R

    c = 'T';
    printf("c = %d", c);
    /* dispplay : c = 84 */


    //
    return 0;
}
```

Boolean Type

- **BOOLEAN** : it is the set of values {TRUE, FALSE}.

Declaration

```
Var       nom_variable1: Boolean
```

Examples

```
Algorithme  example_vars_bool;
    Var          find, continue: boolean;

begin
    ...
    ...
end.
```

Boolean Type

Declaration

Examples

C

Boolean : need to « **stdbool.h** » library else , all values **no null** correspond to « **true** » and **0** correspond to « **false** »

```c
#include <stdio.h>

int main (){

    int find = 0;

    if (find){
        printf(« find is true");
    }else{
        printf(« find is false FAUX");
    }

    return 0;
}
```

5. Basic operations

Basic Operations

1. Assignment
2. Reading (input)
3. Writing (output)

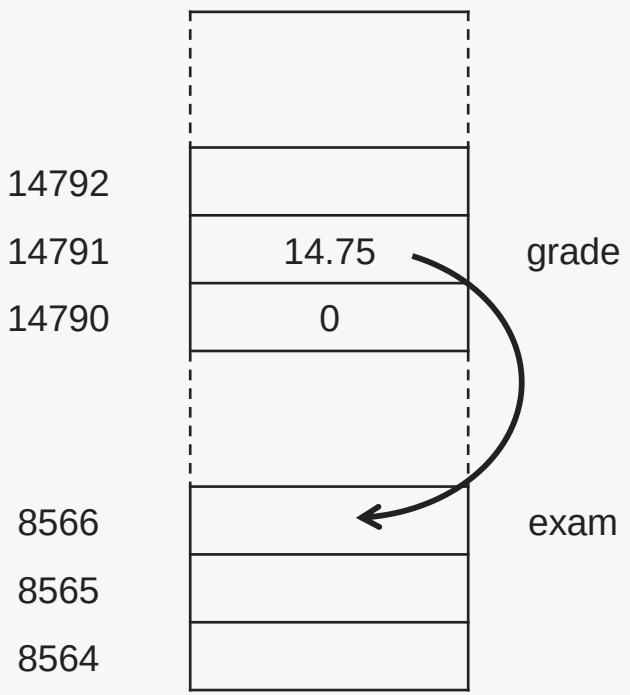
6. Basic Instructions

Assignment

- ✓ Its role is to ***assign*** (give, attribute) a **value** to a variable
- ✓ This value can be :
  - ➢ A constant
  - ➢ The value of another variable or constant
  - ➢ an expression

Syntax

variable ← *expression*



exam ← grade

Assignment

- ✓ Dual **roles** of assignment :
  1) Calculate and evaluate the expression to the ***right*** of the symbol: ←
  2) Assign and store the result in the variable to the ***left*** of the symbol ←

Examples

```
Algorithm  asign_example;
    Var          n, m, l: integer
begin
    ...
    n ← 10
    m ←  n
    l ← n*2 + m*3
    ...
end.
```

Assignment

Basic Instructions

Declaration

Examples

C

variable = expression

```c
#include <stdio.h>

int main (){

    int a, b, c;

    a = 5;
    b = a + 3;
    c = a * b;

    printf("a = %d", a); // display : a = 5
    printf("b = %d", b); // display : b = 8
    printf("c = %d", c); // display : c = 40

    return 0;
}
```

Expressions

- ✓ An ***expression*** is a set of values (***operands***), linked by ***operators***, and equivalent to a single value.
- ✓ An ***operator*** is a sign that connects two values to produce a result

Syntax

*expression***=** *operand* ***operator*** *operand*

Expressions : Arithmetic

- ✓ ***Operands :*** **integer**, **real**.
- ✓ ***Operator*** :

+ : addition

- : soustraction

***** : multiplication

/ : division

DIV: quotient in Euclidean division

MOD: remainder in Euclidean division (modulo)

Expressions : Arithmetic

Déclaration

Algorithm

Operator : **+**, **-** , ***** , **/** , **DIV** (quotient in Euclidean division), **MOD** remainder in Euclidean division (modulo) (Modulo)

C

Operators : **+**, **-** , ***** , **/** (quotient in Euclidean division), **%** (Modulo), **++** (Increment),**--** (Decrement)

Exemples

```
Algorithm Example;

var     a, b, c : Integer;


begin
    //...
    a := 15;
    b := a + 1;
    c := b MOD 3;

    Write('a = ', a); // a = 15
    Write('b = ', b); // b = 16
    Write('c = ', c); // c = 1

    //...

end.
```

```c
#include <stdio.h>

int main (){

    int a, b, c;

    a = 15;
    b = a++;
    c = b % 3;

    printf("a = %d", a); // display : a = 15
    printf("b = %d", b); // dispaly : b = 16
    printf("c = %d", c); // display : c = 1

    return 0;
}
```

Expressions : Alphanumeric

- ✓ ***Operands :*** **caracters**, **strings**.
- ✓ ***Operator*** :

  + : concatenation

Examples

```
Algorithm  example_alpha;
    Var          c1, c2: String;
begin
    ...
    c1 ← "name" + "last_name"  // result  : "namelast_name"
    c2 ← "name + " " + "last_name"  // result : "name last_name"
    ...
end.
```

Expressions : Alphanumeric

Basic Instructions

Déclaration

Exemples

C

Operaors :

```c
#include <stdio.h>
# include <string.h>

int main (){

    char str1[50] = "Hello "; // Make sure destination has enough space
      char str2[20] = "World!";
      strcat(str1, str2);
     printf("Result: %s\n", str1); // Output: "Hello World!"




    return 0;
}
```

Expressions : logic

- ✓ ***Operands :*** **boolean (true, false).**
- ✓ ***Operator*** :

  NON : negation

  AND : logic « AND »

  OR : logic « OR »

Expressions : logic

C

Opérateurs : **&&** (AND), **||** (OR), **!** (NOT)

```c
#include <stdio.h>

int main (){


    return 0;
}
```

Expressions : relational

- ✓ ***Operands :*** **integer**, **real, character**, **string**.
- ✓ ***Operator*** :

< : less than

> : greater than

= : equal

<= : less than or equal

>= : greater than or equal

<> : different

Expressions : relational

C

Opérateurs : **<, <=, >, >=, ==, !=**

```c
#include <stdio.h>

int main (){


    return 0;
}
```

Expressions

Mathematical Expression	Algorithmic Expression
$b^2 - 4ac$	`b * b - 4 * a * c`
$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$	`(-b - SQRT(b*b - 4*a*c))/(2*a)`
$(i <= n/2)\ \text{AND}\ (n\ \text{MOD}\ i <> 0)$	`(i <= n DIV 2) AND (n MOD i <> 0)`

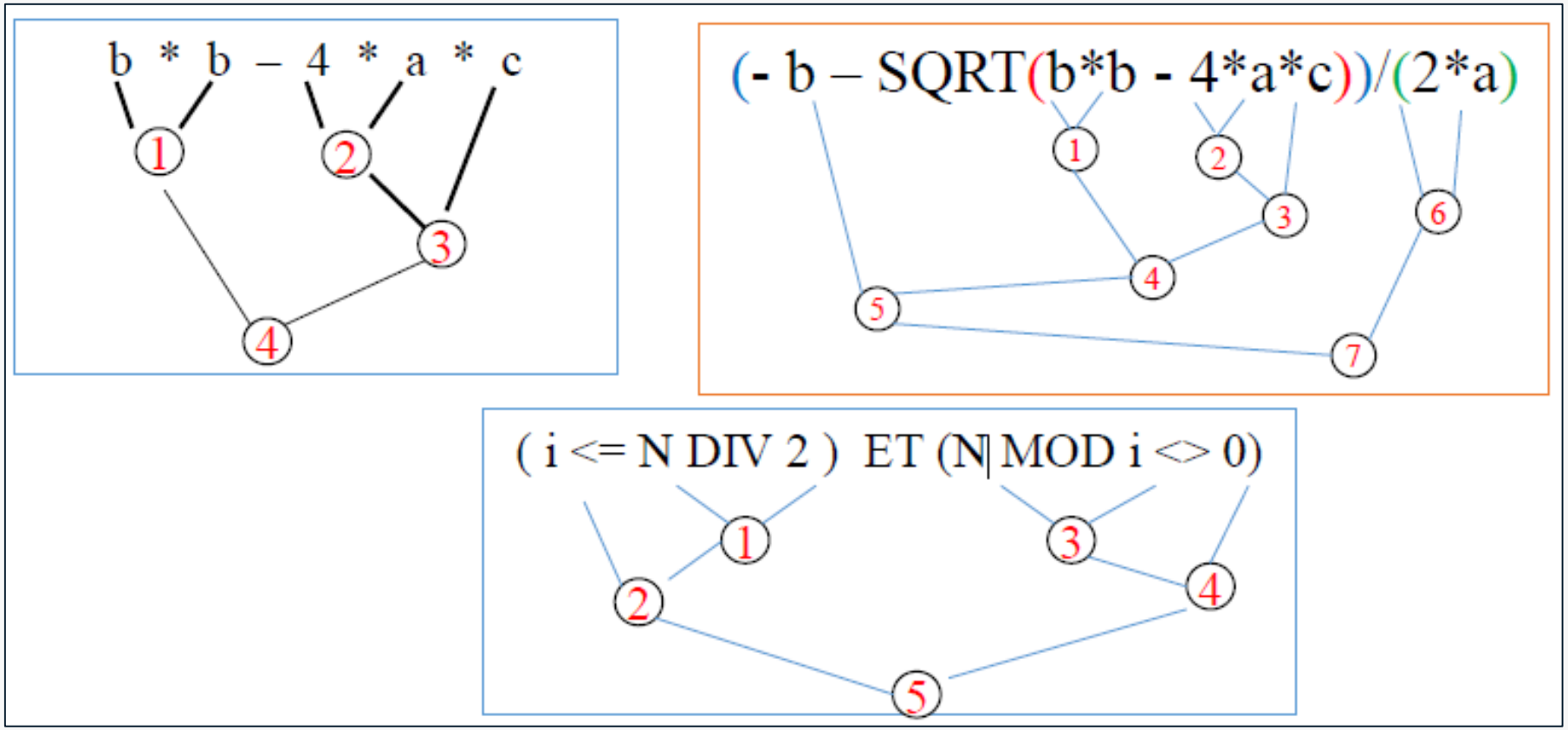
Expressions : Evaluation

- ✓ The evaluation of an expression is performed from left to right, taking into account ***priorities***:
  - ➢ Level 1: NOT
  - ➢ Level 2: * / DIV MOD AND
  - ➢ Level 3: + - OR
  - ➢ Level 4: Relational operators
- ✓ Parentheses can be used to alter the ***priority***.

Expressions : Operators Hierarchy

- ✓ **Operators hierarchy** allows to determine how an expression will be evaluated.
  1. We start with operators that have the ***highest hierarchy*** and then move on to those with ***immediately lower hierarchy***, and so on.
  2. In the case of an arithmetic expression, we start by performing all multiplications, divisions, integer divisions, and modulos, and then proceed to additions and subtractions.
- ✓ When the **hierarchy is the same**, the expression is evaluated from ***left to right***.

Expressions : Evaluation



Inputs / Outputs : Reading

- ✓ Allows you to ***provide*** values from outside
  - ➢ Values are entered using the ***keyboard***
  - ➢ The P1, P2, ...Pn are variables

Syntax

read *(P1, P2, ..., Pn)*

Examples

```
Algorithm  asign_example;
    Var          n, m, l: integer
begin
    read (n, m)
     read (l)
    ...
end.
```

Inputs/outputs: Reading

Déclaration

Exemples

Algorithm

Syntaxe: **Read**

```
algorithm Example;

var     a1, a2 : Integer;
        b : Real;
        c : Char;

begin
    //...
    Read(a1, a2);
    Read(b);
    Read(c);

    //...

end.
```

C

Syntaxe: **scanf** (<stdio.h)

```
#include <stdio.h>

int main (){

    int a;
    float b;
    char c;

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

    //...

    return 0;
}
```

Inputs / Outputs : Writing

- ✓ Allows you to ***display*** the results of an algorithm
  - ➢ The values are displayed on the ***screen***
  - ➢ E1, E2, ...En can be: variables, strings or expressions

Syntax

Write *(E1, E2, ..., En)*

Examples

```
Algorithm  asign_example;
    Var          n, m, l: integer
begin
    write (n, m)
    write ('Addition =', n+m);
    ...
end.
```

Inputs/outputs: Writing

Basic Instructions

Déclaration

Exemples

Algorithm

Syntaxe: **Write**

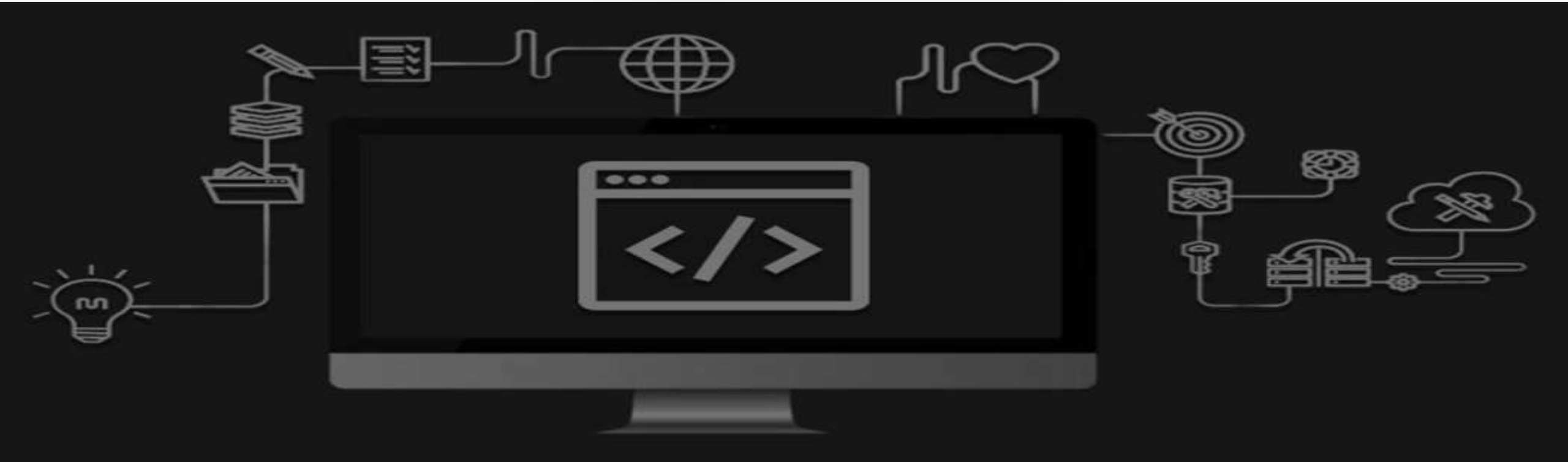
```
program Example;

var     a1, a2 : Integer;
        b : Real;
        c : Char;

begin
    //...

    Write('Hello, World');
    Write(a1, a2);
    Write(b);
    Write(c);
    Write('The result is ', a1+a2);

    //...

end.
```

C

Syntaxe: **printf** (<stdio.h)

```
#include <stdio.h>

int main (){

    int a;
    float b;
    char c;

    //...
    printf("Hello, World");
    printf("%d", a);
    printf("%f", b);
    printf("%c", c);
    printf(« The result is  %d", a);
    //...

    return 0;
}
```

7. Construction of simple algorithm



Example

Example

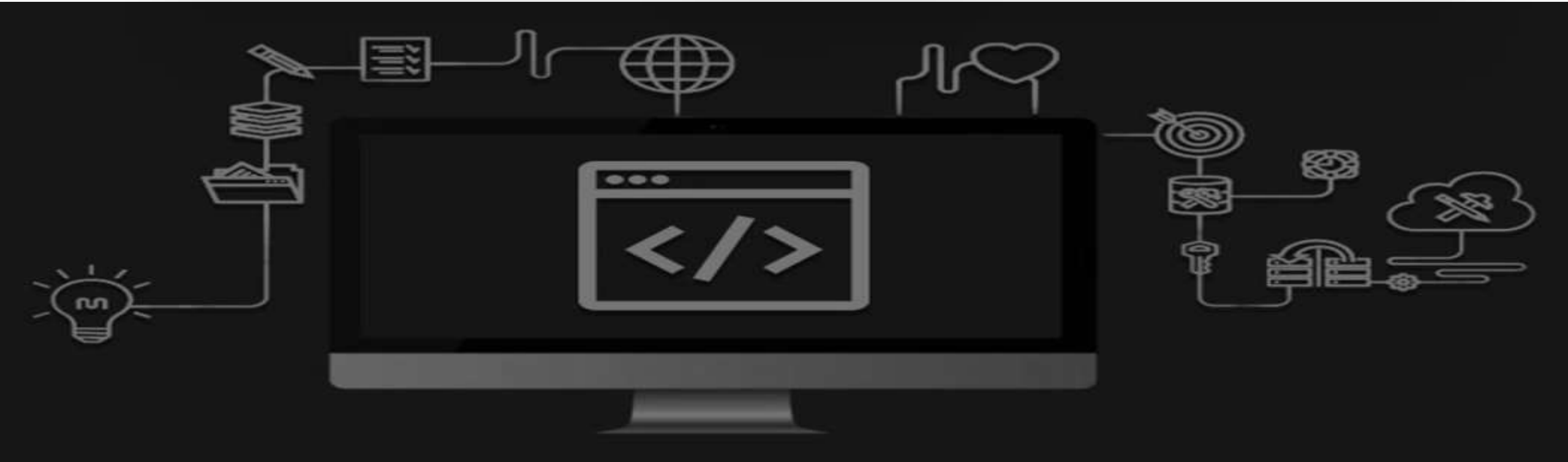
Write an algorithm that calculates and displays the area of a rectangle, where the width and height will be given by the user.

Example

Example

Solution ????

8. Representation of an algorithm by a flowchart

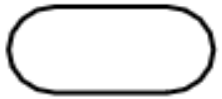
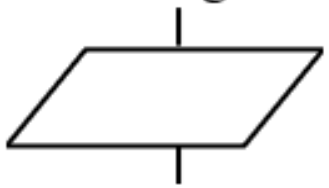
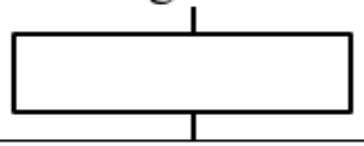
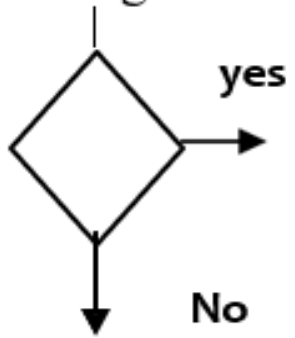


Flowchart

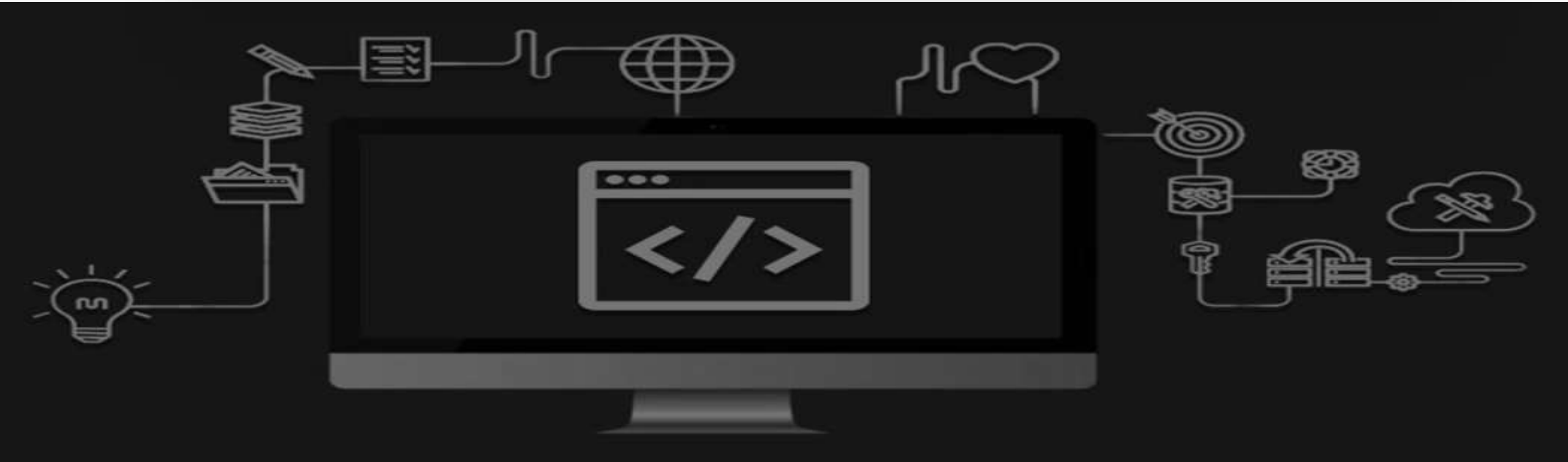
Flowchart

A flowchart is a graphical representation of an algorithm, it allows you to graphically schematize the solution to a problem. A flowchart includes five simple standardized symbols that are connected to each other by connecting lines. These symbols are:

Flowchart

SYMBOL	*SIGNIFICANCE*
Oval 	It expresses the beginning or end of flowchart
Parallelogram 	It is used for input/output operations (Instruction or group of instructions to read data or write results)
Rectangle 	It is used for operations or groups of processing operations (calculations) on data.
Lozenge 	It is used to verify a condition (a test). Conditional statement. The condition is evaluated to be able to take the corresponding path.

8. Translation into C Language



Translation into C language

scanf_printf_asignement.cpp | Untitled2

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    float lenght;
    float width;
    float area;
    printf ("Please input the lenght ?   ");
    scanf ("%f", &lenght);
    printf ("please input the width ?   ");
    scanf ("%f", &width);

    area=lenght * width;
    printf ("the erea is : %f", area);

    return 0;
}

```

Ministry of Higher Education and Scientific Research
Djilali BOUNAAMA University - Khemis Miliana(UDBKM)
Faculty of Matter Science and Computer Science
Department of Mathematics

Chapter : 2

Simple Sequential Algorithm

MI-L1-UEF121 : Algorithms and Data Structures I

Ali Khalfi

Khalfiali.udbkm@gmail.com