ensia The National School of Artificial Intelligence
المدرسة الوطنية العليا للذكاء الاصطناعي

Data Structure & Algorithms 1

CHAPTER1: ALGORITHM FORMALISM

Sep – Dec 2023

Need for Algorithmic Formalism

Once a problem is analyzed, the designer must **express** the **solution** in a **universal formalism**, in the form of an algorithm.

The goal is to use a **common language** to understand algorithms constructed by **others** and vice versa.

→ **Hence, the importance of formalism.**

Need for Algorithmic Formalism

Algorithmic formalism is a set of **conventions** (or rules) in which any **solution** to a given problem is **expressed**.

- Common language.
- Communication.
- Precision - non-ambiguity.

Adopted Formalism

In the following, we will present the adopted formalism (a set of rules) for formulating algorithms that should be readable and understandable by multiple individuals.

Algorithm Structure

An algorithm must follow rules and is composed of a header and a body:

Header, which specifies:

- The name of the algorithm.
- Its purpose (Role) - optional.

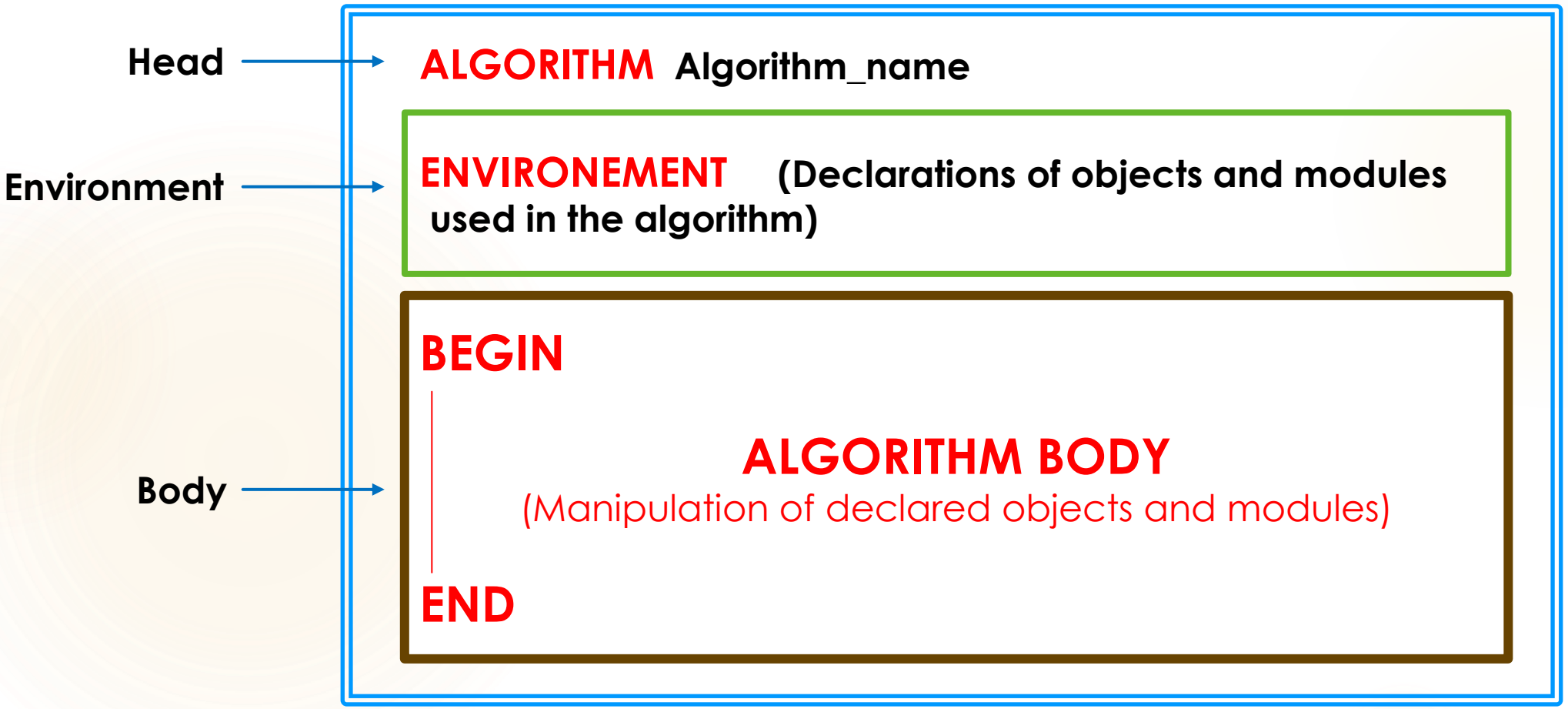
Declarations (Environment):

- Declarations of "input" data, i.e., the elements essential for its proper functioning.
- Declarations of "output" data, i.e., the elements calculated or produced by the algorithm.
- Declarations of local data specific to the algorithm.
- ...

Body, which consists of:

- The keyword **BEGIN**.
  - A sequence of indented instructions.
- The keyword **END**.

Algorithm Structure



Algorithm Structure
Header

The header serves the simple purpose of identifying the algorithm.

The syntax is as follows:

```
Algorithm algorithm_name
// role of the algorithm (optional)
```

- In general, it is advisable to choose a meaningful name to allow the reader to have an idea of what the algorithm will do.
- Examples of headers:

```
Algorithm Calculate_circle_area
Algorithm Integer_sum
```

Algorithm Structure
Environment

The environment (declarative part) contains a comprehensive

list of objects used and manipulated within the algorithm's body.

For each object, you need to specify:

- A **NAME** (Identifier) that allows it to be referred to and distinguished from other elements.
- A **TYPE** that indicates the nature of the set from which the object takes its values.
- A **VALUE** assigned to this object at a given moment.

Algorithm Structure
Environment

Note: **Identifier Concept**
Constructing a **unique** name to designate an object follows precise rules:

- must start with a `lowercase` or `uppercase` letter and
- can consist of `letters`, `digits`, and/or the `underscore` symbol ('_').

- must not contain spaces (`whitespace`), `accented` characters, or certain symbols like `%, ?, *, ., -,` etc.

Examples of correct identifiers:

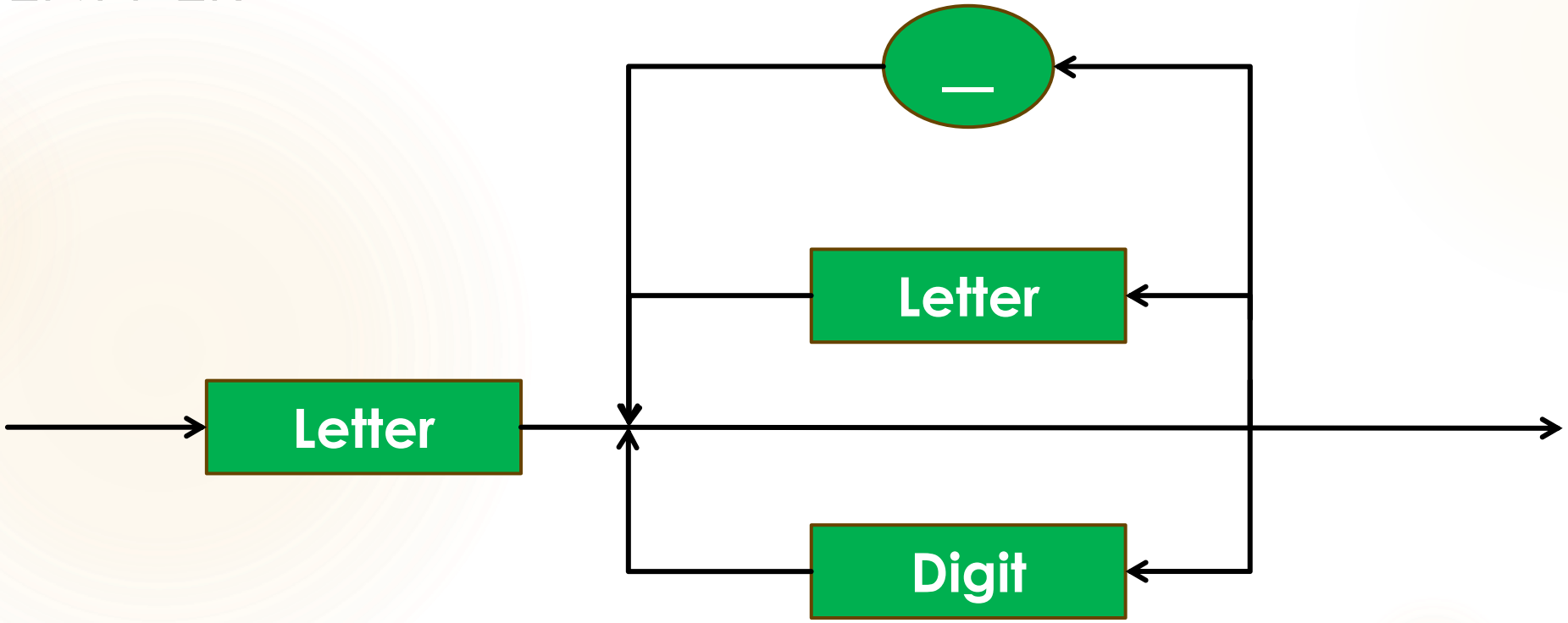
- `x, objectSpeed, Pi, name_`

Examples of incorrect identifiers:

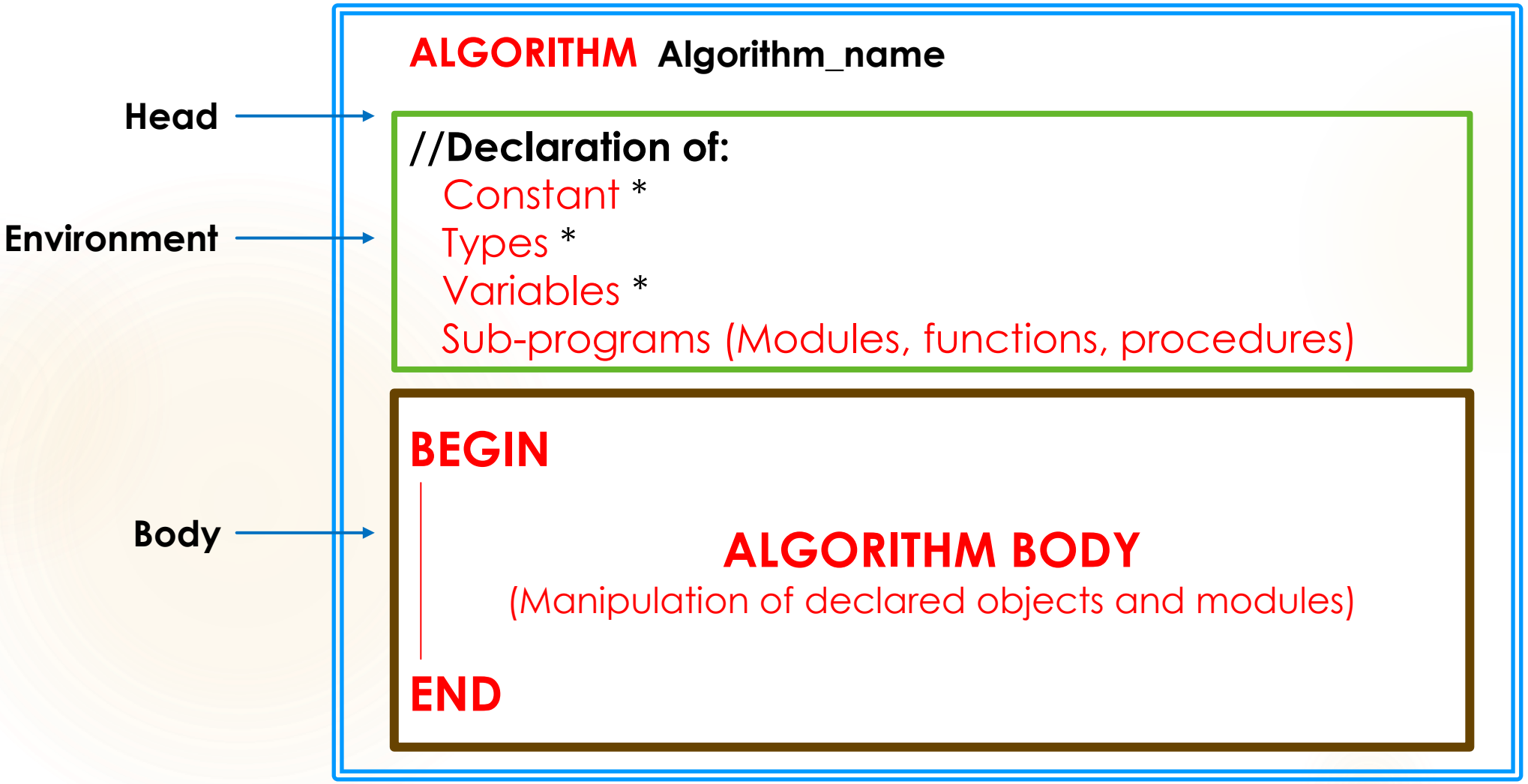
- `34x` (because it doesn't start with a letter)
- `speed object` (because it contains a space between speed and object)

Algorithm Structure
Environment

IDENTIFIER



Algorithm Structure



Algorithm Structure
Declaration of:

Constants

FORMAT:

```
Constant Identifier_Constant = Value
```

Constant Data: Some identifiers have a constant value that does not change throughout the algorithm's execution. These identifiers are called constants. They are declared at the beginning of the algorithm by specifying the identifier's name followed by its value.

Algorithm Structure
Declaration of:

Constants

OPERATION:

Some information manipulated by a program never changes. For example, this is the case with the value of π (PI), the maximum number of students in a class, etc.

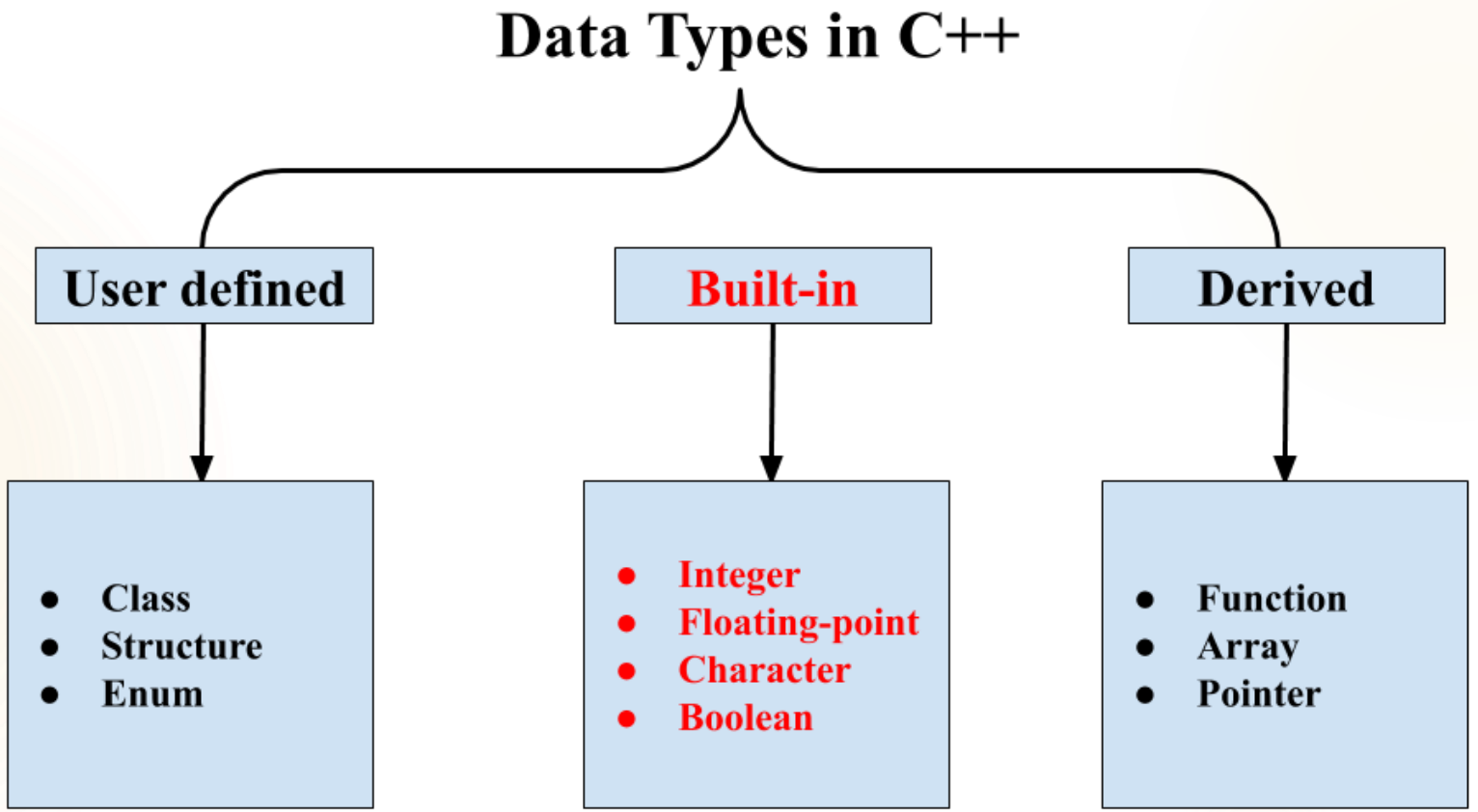
These data are not variable but constant. Instead of explicitly putting their value in the program's text, it is preferable to give them a symbolic (and meaningful) name.

Example:

```
Constant PI = 3.1415 //Value of PI
Constant TVA = 19 //current TVA rate (%)
Constant Capacity = 120 //Max Nb of students in a class
```

Algorithm Structure
Declarations:

Types



Algorithm Structure
Declarations:

Standard Types

Integer Type: This is the set of integer numbers, but it should be noted that while this set is infinite in mathematics, on a computer, the values are limited by the length of machine words.

- The type is designated by the predefined identifier: **INTEGER**

<int in C/C++>

Algorithm Structure
Declarations:

Standard Types

Real Type: This is the set of numbers with a fractional part. This set is also limited, but the limits are broader and depend on the internal representation.

- The type is designated by the predefined identifier: **REAL**

<double in C/C++>

Algorithm Structure
Declarations:

Standard Types

Character Type: It corresponds to a single character. Depending on the systems, the character set may vary, and it includes all alphanumeric characters (letters and numbers), special symbols, and whitespace.

- The type is designated by the predefined identifier: **CHAR**

```
<char in C/C++>
```

Algorithm Structure
Declarations:

Standard Types

Values of the type: The **CHAR** type encompasses all the characters in the character set of the installation. A character is represented by the character itself enclosed in single quotes (apostrophes). The values are ordered according to the internal codes of the characters (ASCII, UNICODE, ...).

Example: `'A' 'c' '1' '0' 'o' '5' ''' ' ' '+' '.'"`

Algorithm Structure
Declarations:

Standard Types

Valid Operators for Integers and Reals:

Integer Type:

- Relational Operators: `< > <> == <= >=`
- Arithmetic Operators: `+ - * DIV MOD`
- Successor Operators: `SUCC and PRED`

Real Type:

- Relational Operators: `< > <> == <= >=`
- Arithmetic Operators: `+ - * /`

Algorithm Structure

Declarations:

Standard Types

Valid Operators for Boolean and Char:

Boolean Type:

- Relational Operators: `< > <> == <= >=`
- Logical Operators: `AND, OR, NOT`

Char Type:

- Relational Operators: `< > <> == <= >=`
- Arithmetic Operators: `+ - * /`
- Successor Operators: `SUCC and PRED`

Algorithm Structure
Declarations:

To Know the different data types used in C++ language refer to the reference card that you can download from the website

https://data-structure1.vercel.app

C++ Libraries and Reference cards:

- C++ Libraries
- Reference Card

Algorithm Structure
Body

The body of an algorithm contains the fundamental tools required to express any algorithm. A block is used to specify how the actions that make up an algorithm should be chronologically arranged.

A block consists of basic actions and control structures.

Algorithm Structure
Body

Basic actions

- Assignment
- Expressions
- Input (read)
- Output (write)

Algorithm Structure
Body

Assignment

Assignment Symbol

FORMAT:

```
Variable = expression
```

The role of assignment is to **assign** (give, attribute) a value to a variable. The value can be a constant, the value of another variable, or the result of evaluating an expression.

Variables and expressions must be compatible.

Algorithm Structure
Body

Assignment

OPERATION:

In assignment, it is necessary to evaluate, if applicable, the entity on the **right side** of the **assignment operator** and then place this result into the entity on the **left side** of the **assignment operator**.

Algorithm Structure

Body

Assignment

⚠ Caution: `A = B + 3`: B must have a value, otherwise, at the beginning of an action, the variables have an undetermined value. B must have been initialized.

EXAMPLES:

`X = 0:`	Set the value zero in X.
`X = Y:`	Set the value of object Y in X.
`X = X + 1:`	Add 1 to X.
`X = X - Y + Z:`	Put in X the result of evaluating the expression X - Y + Z.

Algorithm Structure

Body

Expressions (arithmetic, logical, relational, and mixed)

DEFINITION: An expression is a coherent (possibly parenthesized) set consisting of operands and **operators** that are evaluated to produce a value.

The operands can be:

- Variables: `Moy, A`
- Constants: `Pi, 5, 3.5, etc.`
- Functions: `SQR(), SQRT(), etc.`

Algorithm Structure

Body

Expressions (arithmetic, logical, relational, and mixed)

The operators can be:

- Arithmetic: + - / * MOD, DIV
- Logical: AND, OR, NOT
- Relational: > >= < <= == <>

X = 34 DIV 5
(X will have the value 6: Integer part of the division)
R = 34 MOD 5
(R will have the value 4: reminder of the division)

Algorithm Structure

Body

Expressions (arithmetic, logical, relational, and mixed)

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)`

An **operation** involves two **operands** and one **operator**.

Algorithm Structure

Body

Expression 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**.

Algorithm Structure

Body

Operators Hierarchy

- Operators hierarchy allows to determine how an expression will be evaluated.

We start with operators that have the highest hierarchy and then move on to those with immediately lower hierarchy, and so on.

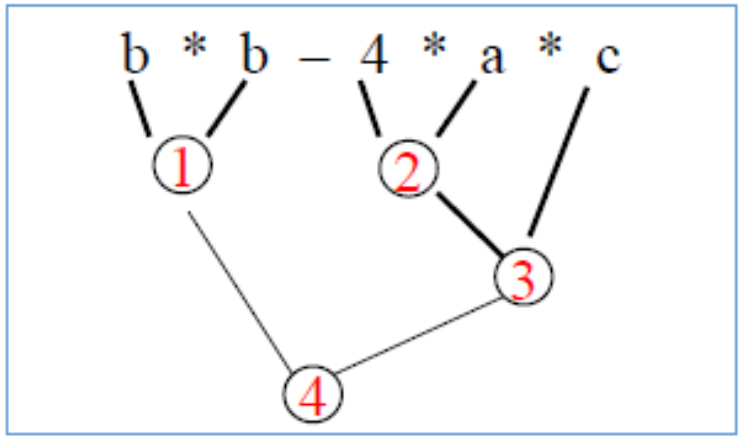
- In the case of an arithmetic expression, we start by performing all multiplications, divisions, integer divisions, and modulos, and then proceed to additions and subtractions.

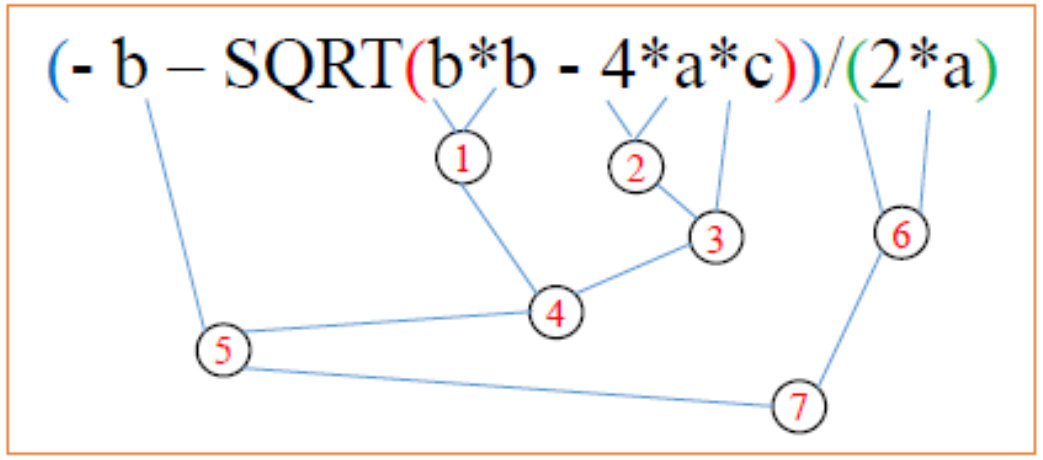
➢ When the hierarchy is the **same**, the expression is evaluated from left to right.

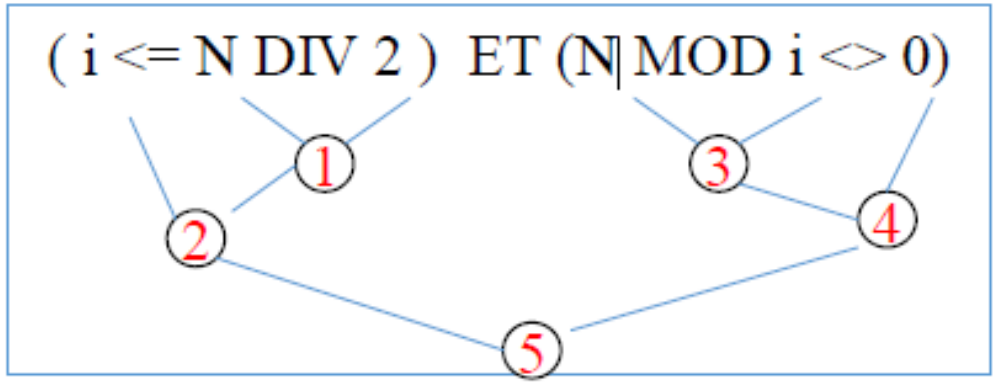
Algorithm Structure

Body

Expression Evaluation (Examples)







Algorithm Structure

Body

Expression Evaluation (Remarks)

The table provides the type to be used for the result of an arithmetic expression based on the **operators** and **operand types**.
It must be followed, as failure to do so may result in errors during program execution.

Operator (op)	Operand types: I: integer R: Real	Result types: I: Integer R: Real
+ - *	I op I R op R R op I I op R	I R R R
/	I op I R op R R op I I op R	R R R R
DIV, MOD	I op I	I

Algorithm Structure

Body

The use of Parentheses

- Complex expressions require the use of **parentheses** to express them in a linear form (on the same line).

- Expressions within parentheses are evaluated **first**, starting with the **innermost** parentheses.

Algorithm Structure

Body

The use of Parentheses

Example

$$Cr = \frac{L.B.F}{\frac{F.B + n}{d} + e}$$

Express it as : Cr = (L * B * F) /((((F* B) +n) /d) + e)

Or: Cr =L * B * F /((F* B +n) / d + e)

Parentheses can be removed due to the operator hierarchy.

Algorithm Structure

Body

Reading

FORMAT:

```
READ ([f], Var1 , Var2 , … , Varn)
```

It allows providing values from the outside to variables of the algorithm because it often happens that an object does not change during the execution of the algorithm. However, the user can change its value between two executions of the same algorithm. This object is called a parameter, and the use of parameters allows generalizing algorithms.

Note: **f** indicates the logical name of the input file. By default **f** is the keyboard.

Algorithm Structure

Body

Reading

OPERATION:

- The values read from the keyboard are assigned to the variables, taking into account the compatibility of the types.

Examples:

- READ (N)
- READ (a, b, c)

Algorithm Structure

Body

Writing

FORMAT:

```
WRITE ([f], r1 , r2 , … , rn)
```

It allows for the output of algorithm results.
`ri` can be: a variable; a label: a string enclosed in single quotes; an expression.

Note: **`f`** indicates the logical name of the output file. By default, it refers to the screen.

Algorithm Structure

Body

Reading

OPERATION:

- The expressions are evaluated, and the values (results) are written or displayed.
- Examples:
  - `WRITE('The discriminant is: ', b * b - 4 * a * c)`
  - `WRITE(N DIV 5 MOD J)`
  - `WRITE('X1 = ', (-b - SQRT(b*b – 4*a*c))/(2*a))`

Algorithm Structure (Example)

Calculate the average of two integer numbers

1. Declare the variables (INPUTs and OUTPUTs)
2. Read the two numbers (introduced by the user)
3. Calculate the average (math formula)
4. Write the result to the screen

Algorithm Structure (Example)

```
ALGORITHM Average_two_numbers

//Variable declaration
int number1
int number2
int avg

BEGIN

    READ (number1, number2)
    avg = number1 + number2
    avg = avg / 2
    WRITE ("The avg of two numbers is: ", avg)

END
```