Second Chapter

Sequential Algorithms

2. Algorithmics: Brief History & Definitions

1.6. Brief History

- The term "algorithm" comes from the name of the 9th-century Muslim Arab mathematician, Al-Khwarizmi (780-850), who was of Persian descent.

- The term "algorithm" is not limited to computer science, and the concept of an algorithm predates that of computer science.

- The earliest discovered algorithms describe methods of mathematical calculation. For example, the Euclidean algorithm, which allows for the calculation of the greatest common divisor (GCD) of two integers a and b:

1) Divide a by b to obtain the remainder, r.

2) Replace a with b.

3) Replace b with r.

4) Continue as long as it is possible. Eventually, the GCD will be obtained.

- Introductory Example: Algorithm (Concrete Preparation).

1) Purchase the raw materials (Cement, crushed stone, sand) - Input
2) Pour two wheelbarrows of crushed stone onto one wheelbarrow of sand. (Step 1)
3) Pour one bag of cement onto the mixture and mix. (Step 2)
4) Pour 60 liters of water onto the mixture and mix. (Step 3)
5) This results in obtaining the concrete. – Output

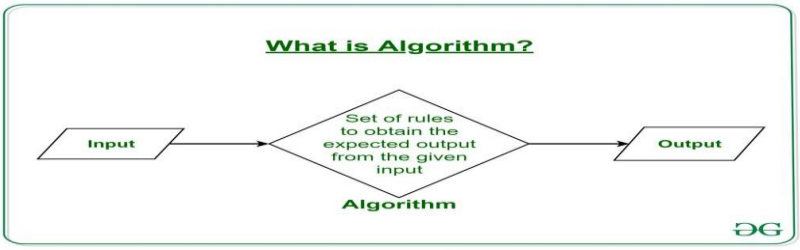
1.7. Definitions

1.7.1. Algorithm :

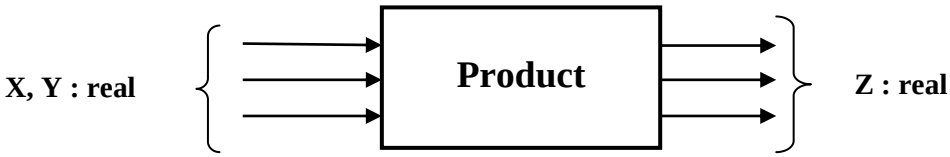
Basically, the word Algorithm means ” A set of finite rules or instructions to be followed in calculations or other problem-solving operations, Therefore Algorithm refers to a sequence of finite steps to solve a particular problem.

Definition : An algorithm is a finite and unambiguous sequence of operations or instructions that allows solving a problem.

- An algorithm is a method, which, in a finite amount of time, leads to a determined result from a given situation. The processing of information by a computer involves having this machine generate information, called results, from information called data.



Example : : calculate the product of two real numbers Z=X*Y ?



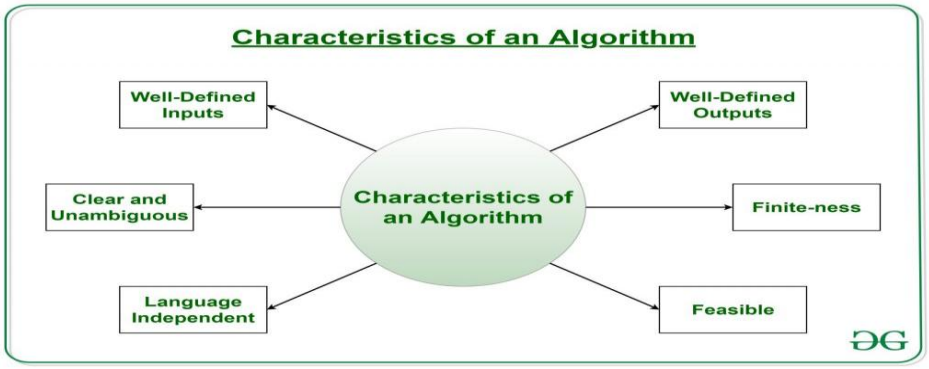
1.7.2 Characteristics of an algorithm :

An algorithm must have the following characteristics:

- ✓ **Clear and Unambiguous**: The algorithm should be unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.
- ✓ **Well-Defined Inputs**: If an algorithm says to take inputs, it should be well-defined inputs. It may or may not take input.
- ✓ **Well-Defined Outputs:** The algorithm must clearly define what output will be yielded and it should be well-defined as well. It should produce at least 1 output.
- ✓ **Finite-ness:** The algorithm must be finite, i.e. it should terminate after a finite time.
- ✓ **Feasible:** The algorithm must be simple, generic, and practical, such that it can be executed with the available resources. It must not contain some future technology or anything.
- ✓ **Language Independent:** The Algorithm designed must be language-independent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be the same, as expected.
- ✓ **Input**: An algorithm has zero or more inputs. Each that contains a fundamental operator must accept zero or more inputs.
- ✓ **Output**: An algorithm produces at least one output. Every instruction that contains a fundamental operator must accept zero or more inputs.
- ✓ **Definiteness:** All instructions in an algorithm must be unambiguous, precise, and easy to interpret. By referring to any of the instructions in an algorithm one can clearly understand

what is to be done. Every fundamental operator in instruction must be defined without any ambiguity.

- ✓ **Finiteness:** An algorithm must terminate after a finite number of steps in all test cases. Every instruction which contains a fundamental operator must be terminated within a finite amount of time. Infinite loops or recursive functions without base conditions do not possess finiteness.
- ✓ **Effectiveness:** An algorithm must be developed by using very basic, simple, and feasible operations so that one can trace it out by using just paper and pencil.



Properties of Algorithm:

- ✓ It should terminate after a finite time.
- ✓ It should produce at least one output.
- ✓ It should take zero or more input.
- ✓ It should be deterministic means giving the same output for the same input case.
- ✓ Every step in the algorithm must be effective i.e. every step should do some work.

1.7.3. Algorithmic Language

This is the language used to describe and define algorithms. By using a set of keywords and structures that allow for a complete and clear description of all operations to be performed on data to obtain results.

1.7.4. Keyword

A keyword is an identifier that has a specific meaning (algorithm, start, end, if, then...).

```
Algorithm Addition;
// variable declaration
num1, num2, sum: integer;
 Begin
// Input
write('Enter the first number (num1): ');
read (num1);
 write('Enter the second number (num2): ');
read (num2);
//Performing  Addition
Sum         num1 + num2;
// Output
 write('The sum is: ', sum);
End.
```

1.7.5. Algorithmics

Algorithmics is the science that studies algorithms.

1.7.6. Program

A computer program is a sequence of instructions written in a language understandable by a computer (programming language). In other words, a computer program is the translation of the algorithm into a programming language.

1.7.7. Programming Language

A set of commands and keywords necessary for writing a program so that it can be understood by the computer.

Examples: Basic, Fortran, C, C++, Pascal, Delphi, Java...

1.8. Structure and Execution Trace of an Algorithm

1.8.1. General Structure of an Algorithm

An algorithm is composed of three parts: the header, the declaration, and the body of the algorithm.

```
ALGORITHM   Algorithm-Name
Constants
Types
Variables
Functions
Procedures

BEGIN
Instruction 1;
Instruction 2;
...
Instruction n;
END.
```

- ✓ **The header**: it simply allows for the identification (naming) of an algorithm.

- ✓ **Declarations:** this is a list of all objects (constants, variables, types, functions, and procedures) used and manipulated within the body of the algorithm.

- ✓ **The body**: The body contains the sequence of instructions to be executed (a set of operations to be performed on the data, i.e., variables).

Example : an algorithm that calculates the product of two integer numbers (First algorithm).

```
Algorithm Product   /* header*/
   Number1, Number2, Product: integer; /* declarations*/

Begin
   Number1 ← 5;
   Number2 ← 4;                          /* body*/
   Product ← Number1 * Number2;
End.
```

1.8.2. Execution Trace of an Algorithm:

- The execution of an algorithm (program) proceeds instruction by instruction in the order specified by the algorithm (program). Execution begins from the first instruction that follows the keyword BEGIN and ends at the last instruction just before the keyword END.

- Initially, the values of variables are unknown (?).

- When a variable receives a value, that value is stored until another instruction changes it.

Step	a	b
Instruction1	10	?
Instruction2	10	15
Instruction3	3	15

```
Algorithm Trace
      a : integer;
      b : integer;
begin
     a←10 ;
     b← a+ 5 ;
     a← 3 ;
end.
```

1.9. Solving a Problem on a Computer:

The creation of a program that can be executed by a computer requires following a process consisting of four phases. The role of the algorithm (Phase 2) is shown in the following figure:

1.9.1. Problem Analysis

Analysis involves:

- ✓ Extracting initial data (inputs),
- ✓ Defining the goal or goals of the problem (outputs or results),
- ✓ Deriving the method to follow to solve and achieve these objectives.

1.9.2. Writing Algorithms :

- Writing the set of corresponding algorithms.

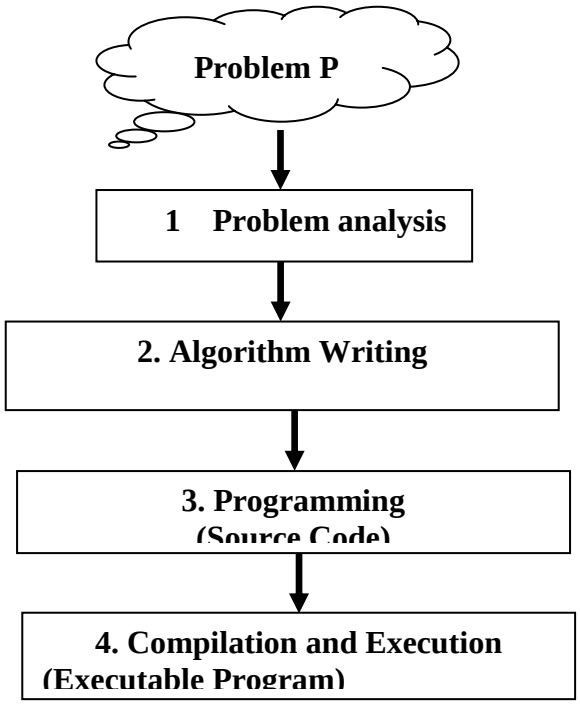
1.9.3. Programming

- In this step, the algorithm is translated into a programming language (C, C++, Pascal, Fortran, Java, etc.). This results in the source code of the program, which is stored in one or more files on the computer.

1.9.4. Compilation and Execution

- In this step, the source code of the program is translated into machine language using special software called a compiler. This process generates the machine code of the program, which is then executed directly by the computer.

- There are several software applications that include a compiler for the C language, such as Dev C++, Turbo C, Microsoft Visual C++ Express, and others.



Example (Problem): Create a program that calculates the electrical current "I" flowing in a closed circuit powered by a source with a potential difference of "U=220 Volts" and containing a resistance "R=12 Ohms."

Solution:

1) Problem Analysis:

- Input Data: U=220 Volts, R=12 Ohms.

- Results or Outputs (Objectives): Calculated current I.

- Method: U=R*I, hence I=

2) Writing Algorithms: (This is where the algorithm for solving the problem would be written.)

```
Algorithm current-intensity-calculation
variables
U,R,I : integers ;
Begin
U←220 ;
R←12;
I← U / R ;
Write (I);
End.
```

1) **Programing : (**see practical work**)**

Program in language *pascal*	**Program in langage *C***	**Program in langage *C++***
Program intensitycalculation ; Uses crt ; Var U,R,I : integer ; **Begin** U **:=**220 ; R **:=**12; I **:=** U/R ; **End.**	#include <stdio.h> int U,R,I ; **void main()** **{** U **=** 220 ; R**=**12; I = U/R ; **}**	#include <iostream> using namespace std; int U,R,I ; **void main()** **{** U **=** 220 ; R**=**12; I = U/R ; **}**

2) **Compiling the program : (**voir **Practical work)**

1.10. Basic Rules for Constructing a Good Algorithm:

✓ Clearly and unambiguously define the given problem.

- ✓ An algorithm must be well-structured.
- ✓ The result must be achieved in a finite number of steps, so there should be no infinite loops.
- ✓ Consider all possible data cases, ...
- ✓ The result must address the problem at hand. Please note that a set of test cases NEVER proves or guarantees that a program is correct. It can only demonstrate that it is incorrect.
- ✓ An algorithm must be commented. Comments are sentences that can be inserted at any level of the algorithm (program). They serve solely to provide explanations for the reader of the algorithm and are completely ignored by the executor. {comment}.
- ✓ If applicable, an algorithm should be modular, meaning it can be broken down into smaller parts.
- ✓ An algorithm should be efficient, considering criteria such as execution time and memory usage.

1.11. Variables:

1.11.1. Concept of a Variable:

- ✓ The variables in an algorithm contain the information necessary for its execution.
- ✓ A variable in a program is a memory space identified by a name, intended to store a value that can be modified during processing (operations).
- ✓ Each variable is characterized by a name (identifier) and a type.

1.11.2. The Identifier (Name):

- ✓ To identify a variable, letters of the alphabet (a, b, c, A, ...) and digits (0, 1, 2, ...) can be used, provided that the first character is an alphabet letter. The identifier can also contain an underscore "_".

- Examples:

  - ✓ X, y, name, price, X1 are variable identifiers (names).
  - ✓ 1x is not an identifier since it starts with a digit.
  - ✓ nom*, Met%un1 are not variable identifiers because they contain special characters.

1.11.3. The Type:

The type corresponds to the kind of information you want to use:

- ✓ Integer for handling integers (1, 115, -7, ...),
- ✓ Real for handling real numbers (11.3, 15.7, -4.3, ...),
- ✓ Boolean for handling Boolean values (true or false),
- ✓ Character for handling alphabetic and numeric characters (a, b, 3, ...),
- ✓ - String for handling strings of characters used to represent words or phrases (mila, omar, ...).

Note:

- ✓ The identifier must be different from all keywords.
- ✓ For code readability, choose meaningful names: e.g., TotalSales2004, Price_Incl_Tax, Price_Excl_Tax, sum, ...

1.11.4. Variable Declaration:

All variables must be declared before they are used.

Syntax: Variable Name: Variable Type;

Example :

```
Algorithm exp_decl
        a : integer;
        c: char;
        Age : integer;
        prénom: string;
        x,y,z: real;
Begin
{Sequence of Instructions}
End.
```

Note: Multiple variables of the same type can be declared on a single line by separating them with commas (for example: a, b, c: integer;).

1.12. Variable Types:

1.12.1. Integer Type (int):

The integer type is equipped with the following operators:

- ✓ Classical arithmetic operators: + (addition), - (subtraction), * (multiplication).
- ✓ Integer division, denoted as div, where n div p gives the integer part of the quotient of the integer division of n by p.
- ✓ Modulus, denoted as mod, where n mod p gives the remainder of the integer division of n by p.
- ✓ Classical comparison operators: <, >, =, ...

- Examples:

- 7 div 2 = 3

- 7 mod 2 = 1

- sqrt(5) = 25

- abs(-17) = 17

1.12.2. Real Type (float and double):

Valid operations on real numbers include:

- ✓ Classical arithmetic operations: + (addition), - (subtraction), * (multiplication), / (division).
- ✓ Classical comparison operators: <, >, =, ...
- ✓ The function that provides the square root (sqrt).

1.12.3. Boolean Type (bool in language C++) :

This is a type with only two values: true or false. The logical operators (operations) on this type are not, or, and. These operators are defined by the following truth tables:

The logical operator (NOT) :

NOT	
True	False
False	True

The logical operator (AND) :

And	true	False
true	true	False
False	False	False

The logical operator (OR) :

OR	True	False
True	True	True
False	True	False

2.9.4. Character Type:

This is a type consisting of alphabetic and numeric characters. A variable of this type can only contain a single character. Predefined operations on characters include:

- ✓ Succ(c): It provides the character that immediately follows the character c.
- ✓ Pred(c): It provides the character that immediately precedes the character c.

Example : Succ('B') = 'C', Pred('z') = 'y'.

2.9.5. String Type:

A string is a sequence of characters enclosed in double quotation marks. Example: "computer," "course," "mila," "-1.6," ....

Predefined operations on strings include:

- ✓ Comparisons: <, >, =, ... based on lexicographic order (ASCII).
- ✓ Concatenation represented by +: It provides the string obtained by concatenating two strings.

Note: Strings are ordered based on lexicographic order.

Example : 'art' < 'course', 'course' < 'courses', 'courage' < 'courses'.

2.10. Constants :

A constant is an algorithmic object that holds a single value throughout its execution.

Example: Const int Max_value = 100; /*cpp; `Max_value` is a constant that is set to 100 and cannot be changed throughout the execution of the program */

```
Algorithm exp_cons
  Const  Pi ← 3,14 ;
  Variable    X : integer ;   Y : real;
  Begin
    X ← 5 ;    Y←X+Pi ;
Pi ← X+1 (false) /*   Pi will always have the value 3.14.*/

  End.
```

Remarks:

- ✓ To declare a variable, you specify its name and type.
- ✓ To declare a constant, you must use the keyword CONST. You define its name and value.
- ✓ Always start with the declaration of constants before variables.

2.11. Expressions:

2.11.1. What is an Expression:

An expression represents a combination of operands and operations performed on these operands.
The operands can be:

- ✓ Variables
- ✓ Values

The operators (operations) can be:

- ✓ Algebraic operators: +, -, *, /, div, mod.
- ✓ Logical operators: and, or, not.

- ✓ - Relational operators: <, <=, >, >=, =, <>.

Every expression is associated with a type, which is the type of the value of that expression.

Example:

"a + 8" is an expression representing an addition operation involving the operands "a" and "8".

Note: An operator that operates on two operands is called a binary operator (e.g., +), while an operator that operates on a single operand is called unary (e.g., not).

2.11.2. Rules for Evaluating an Expression:

The calculation of the value of an expression with more than one operator depends on the interpretation given to that expression.

Example: "5 - 4 div 2" is an ambiguous expression.

Parentheses can resolve this issue:

- ✓ (5 - 4) div 2 = 0.
- ✓ 5 - (4 div 2) = 3.

Note : In the absence of parentheses and to avoid ambiguity, evaluation rules have been established.

There is a descending order of precedence among operators as follows:

1) Unary operators: not.
2) Multiplicative operators: *, /, div, mod, and.
3) Additive operators: +, -, or.
4) Relational operators: <, <=, >, >=, <>.

Example : Therefore, the interpretation assigned to "5 - 4 div 2" is indeed "5 - (4 div 2) = 5 - 2 = 3".

- ✓ If an expression contains operators of the same precedence, then the operators of the same precedence are left-associative.

Example : "5 - 3 - 2" is interpreted as "(5 - 3) - 2 = 2 - 2 = 0".

Note : In algorithmics, to avoid ambiguity, it is always advisable to use parentheses.

2.12. Instructions:

2.12.1. Definition of an Instruction:

An instruction represents one or more actions (operations) involving one or more variables.

There are two types of elementary instructions:

- ✓ Assignment instruction
- ✓ Input/output instruction.

2.12.2. Assignment Instruction:

Assignment is an instruction that stores the value of an expression in a variable.

Syntax of assignment:

- X ⟵ exp; reads as: x receives exp.

Where:

- X: variable,
- ⟵ : The assignment operator, - exp: expression.

- This replaces the initial value of x with the value of the expression.

Example:

Algorithm exp_Aff

a, b : integer;

Begin

a ← 12 ;

b ←a+ 4 ;

End.

Execution: a=12 b=16

2.12.3. Input/Output Instruction:

a) Input Instruction:

This instruction allows reading the value of a variable from the keyboard.

Syntax: Read (variable);

The execution of this instruction involves assigning a value to the variable by taking this value from the input device (keyboard).

Example : Suppose X is an integer variable, then:

Read(X); will assign an integer value typed from the keyboard to the variable X.

b) Output Instruction :

This instruction allows displaying output on the screen.

There are two types of output:

- ✓ Displaying variable values: Syntax: Write (variable);
- ✓ Displaying text: Syntax: Write ('text');

Examples:

- ✓ Write (2 * x + 5): displays the value of the expression 2 * x + 5. If x is 10, then this instruction displays 25.
- ✓ Write ('The result is: '): displays the text "The result is: ".
- ✓ Write ('The result is: ', X): if x is 15, this instruction displays "The result is: 15."

2.13. Comments:

To write the algorithm in a clear and logical manner, it is possible to add comments to algorithmic actions (instructions).

By convention, a comment:

- ✓ Starts with a forward slash (/) followed by an asterisk * and ends with an asterisk * followed by a forward slash: /* multiple lines */
- ✓ Starts with a double slash for a comment that spans a single line: // Single line

A comment is not an instruction. It has no effect on the execution of the algorithm.

Example: Write an algorithm that calculates the sum of two integers?

Algorithm exp_comt

a, b, som : integer ;

Begin

```
Lire (a) ;
Lire (b) ;
 /* "This instruction allows for reading the values of variables a and b from the keyboard." */
Som ← a+b ;     // calculates the sum
 write  (som) ;    // displays the result on the screen.
```

End of the chapter