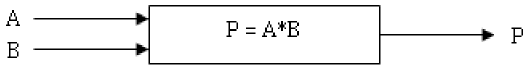
The algorithmic language is a language close to human language and is defined by a set of reserved words and syntax for writing actions.

1. Problem Solving Approach

Problem solving involves three main phases, namely:

1. Study Phase: This is a phase that involves understanding and analyzing the problem to determine the data (inputs) and the results (outputs) to produce.
2. Algorithm Writing Phase: This is a phase of translating the previous phases based on the rules of the algorithmic language.
3. Programming Phase: This is a phase of translating the algorithm into a program in a programming language.

Example: We want to calculate the product of two integers, A and B.



2. Basic Concepts

2.1. Definition of an Algorithm

An algorithm is a structure for solving a given problem. This structure consists of two essential parts:

- The first part (the header) defines the set of data structures and/or objects to use. It can be thought of as a set of tools necessary to solve the problem at hand.
- The second part (the body) contains a sequence of actions to be performed to arrive at a determined result from a given situation in a finite time.

2.2. Structure of an Algorithm

```
ALGORITHM algorithm_name
CONST
   {Set of constants and their values}
TYPE
   {Definition of custom types}
VAR
   {Set of variables to use within the body of the algorithm}
BEGIN
   {Sequence of actions to execute}
END
```

The words ALGORITHM, BEGIN, and END represent keywords that must exist in an algorithm.

The words CONST, TYPE, and VAR are keywords that may or may not exist in an algorithm depending on the requirements.

2.3 Concept of an Object

1. Definition of an Object

An object is a "memory location" characterized by an identifier and a type, capable of holding a value and undergoing specific actions.

2. Object Identifier

The identifier of an object is its name. It is composed of letters of the alphabet, digits, and possibly the character "_", and must start with a letter.

3. Object Type

A type defines the set of values an object of that type can take, as well as the actions allowed on it.

Types can be classified based on several criteria, including whether they are simple or compound and whether they are predefined or custom.

Note: Once a type is fixed for an object in an algorithm, it cannot be changed within the same algorithm.

4. Nature of an Object

An object can be either constant or variable but not both at the same time. It is variable if its value can change during the algorithm, and it is constant if its value remains fixed throughout the algorithm.

5. Use of an Object

An object can be used in one of the following ways:

- Input object: It is an object whose value is used as data for the problem to be solved.
- Output object: It is an object whose value is the solution to the problem.
- Working object: It is a tool that acts as an intermediary between input and output objects.

Example: For a quadratic equation:

- Input objects: a, b, and c
- Output objects: x1 and x2
- Working object: Delta

6. Declaration of an Object

It is the specification of objects to use in the same algorithm. This is presented in the header of the algorithm.

Every object used in an algorithm must be declared one and only once, either in the VAR block or in the CONST block.

Syntax

- **Declaration of a constant:**

Const_id = value

The type of a constant is deduced from its value.

- **Declaration of a variable:**

Variable_id: base_type

Example

```
CONST
   VAT = 0.18
VAR
   Quantity: real
   PriceExcludingVAT: real
   PriceIncludingVAT: real
```

Application 1

Write the header of an algorithm to calculate the perimeter of a circle.

Solution

```
ALGORITHM Perimeter_Circle
CONST
   PI = 3.14
VAR
   Radius: real
   Perimeter: real
```

3. Simple Algorithmic Actions

An action is a command given by the user to the computer. There are three simple algorithmic actions: display, input, and assignment.

3.1. Display

- **Definition**

The display action allows communicating a message or the content of one or more objects to the user on the screen (or printer).

- **Syntax**

WRITE (ARG1, ARG2, ..., ARGn)

Where ARGi can be:

- The identifier of an object (constant or variable).
- An arithmetic expression.
- A message.

Examples

- WRITE (A): displays the content of object A.
- WRITE ("message"): displays the word 'message' in its entirety.
- WRITE (B * B - 4 * a * c): displays the result of the expression; this assumes that the values of a, B, and c are known.
- WRITE ("The square of", A, "is equal to", A * A)

3.2. Input

- **Definition**

This action allows placing a value provided by the user into the memory location associated with a variable. The execution of this action involves temporarily stopping the algorithm's execution, waiting for a value given by the user.

- **Syntax**

READ (VAR1, VAR2, ..., VARn)

Note

You can input each variable separately, or you can input multiple variables with a single READ command.

Example

READ (A)
READ (B) ⟺ READ (A,B,C)
READ (C)

Application 1

Write an algorithm to display the perimeter of a circle.

Solution

```
ALGORITHM Perimeter_Circle
CONST
```

```
   PI = 3.14
VAR
   Radius: real
BEGIN
   WRITE("Enter the value of the radius: ")
   READ(Radius)
   WRITE("The perimeter of the circle is: ", 2 * Radius * PI)
END
```

Application 2

Write an algorithm to display the average of a student for a subject. For each subject, the student has a homework score and an exam score with weights of 0.4 and 0.6, respectively.

Solution

```
ALGORITHM Subject_Average
CONST
   HomeworkWeight = 0.4
   ExamWeight = 0.6
VAR
   HomeworkScore: real
   ExamScore: real

BEGIN
   WRITE("Enter the homework score: ")
   READ(HomeworkScore)
   WRITE("Enter the exam score: ")
   READ(ExamScore)
   WRITE("The student's average is:", HomeworkScore * HomeworkWeight + ExamScore *
ExamWeight)
END
```

3.3. Assignment

- **Definition**

Assignment is an operation that allows placing a value into a memory location associated with a variable. The value can be the content of another variable, the content of a constant, a given value, or the result of an arithmetic or logical expression.

- **Syntax**

VAR_NAME ← EXPRESSION

Where:

- VAR_NAME is the name of the target variable that will receive the value (the left-hand side of the assignment).
- EXPRESSION can be a value, a variable or constant identifier, or an arithmetic or logical expression (the right-hand side of the assignment).

In an assignment operation, the right-hand side is evaluated, and the result of the evaluation is stored in the left-hand side.

Examples

```
A ← 3
B ← A * 5
A ← B - 2
C ← A
C ← B + 6
B ← B + 1
```

Important Remarks

- At the time of assignment, the type of the right-hand expression's result must be the same as that of the target variable (or compatible).
- Assigning values to a variable can be done multiple times, and the variable's value will change with each assignment, losing its previous value. You'll have the last assigned value in the variable.
- A constant can never appear on the left side of an assignment since its value cannot be changed.
- When a variable appears on the right side of an assignment, it's assumed that it contains a value. This value must have been assigned to it previously (through input or assignment); otherwise, the variable is considered "undefined," and the result of the assignment is also undefined.

Exercise :

Write an algorithm to swap two integers A and B :

• using an additional variable C,
• without the use of an additional variable.

Solution 1 :

```
ALGORITHM   Swap_With_Extra_Var
VAR
    A: integer
    B: integer
    C: integer
BEGIN
    WRITE("Enter the value of A: ")
    READ(A)
    WRITE("Enter the value of B: ")
    READ(B)
    C ← A
    A ← B
    B ← C
    WRITE("The new value of A is ", A)
    WRITE("The new value of B is ", B)
END
```

Solution 2 :

```
ALGORITHM   Swap_Without_Extra_Var
VAR
    A: integer
    B: integer
BEGIN
    WRITE("Enter the value of A: ")
    READ(A)
    WRITE("Enter the value of B: ")
    READ(B)
    A ← A + B
    B ← A - B
```

```
   A ← A - B
   WRITE("The new value of A is ", A)
   WRITE("The new value of B is ", B)
END
```

4. Simple Types

Remember that a type defines the set of values an object can take and the actions allowed on it.

4.1. Integer Type

- **Definition**

This type is associated with objects that take their values within a finite range of integers, with predefined lower and upper bounds, included in the set (Z). An object of the integer type can be positive, negative, or zero.

- **Declaration**

identifier1, identifier2, ..., identifierN: Integer

- **Manipulation Operations**

An object of the integer type can undergo the following operations:

- Simple algorithmic actions.
- Standard operations: real division (/), addition (+), subtraction (-), and multiplication (*).
- Exponentiation, noted as "**" where "N**P" gives N^P.
- Integer division, noted as "DIV," where "N DIV P" gives the integer part of the result of N divided by P.
- Modulus, noted as "MOD," where "N MOD P" gives the remainder of the integer division of N by P.
- Square root, noted as SQRT, where SQRT(x) gives the square root.
- Absolute value, noted as ABS, where ABS(x) gives |x|.
- Comparison operators (<, <=, >, >=, <>, =).

Examples

14 MOD 3 returns 2
14 DIV 3 returns 4
14 / 4 returns 3.5

Application 1

Write an algorithm that reads an integer (assumed to be composed of 3 digits) and displays its units, tens, and hundreds digits.

Solution

```
ALGORITHM Decomposition
VAR
   N, U, T, H: integer
BEGIN
   WRITE("Enter a three-digit integer: ")
   READ(N)
   H← N DIV 100
   T ← (N MOD 100) DIV 10
   U ← N MOD 10
   WRITE("The units digit is ", U)
```

```
   WRITE("The tens digit is ", T)
   WRITE("The hundreds digit is ", H)
END
```

4.2. Real Type

- **Definition**

This type corresponds to objects that take their values within a finite set of real numbers, with predefined lower and upper bounds, included in the set (R).

- **Declaration**

identifier1, identifier2, ..., identifierN: real

- **Manipulation Operations**

The allowed operations on this type are the same as those allowed on the "Integer" type, except for the MOD and DIV operators.

Application

Write an algorithm that reads two integer variables, A and B, calculates the sum, difference, average, and product, and displays the results.

4.3. Character Type

- **Definition**

This type allows defining objects representing an element taken from the set of editable characters (uppercase and lowercase letters, punctuation characters, whitespace, digits, etc.).
A value of the character type is always enclosed in single quotes: this allows us to distinguish a character ('3') from the corresponding integer (3) or an alphabetic character ('A') from the name of a variable (A).
The "space" character is represented by two single quotes separated by a space: ' '.

- **Declaration**

identifier1, identifier2, ..., identifierN: char

- **Manipulation Operations**

The allowed operations on the character type are as follows:

- Simple algorithmic actions (input, output, and assignment).

- ORD: so that ORD(C) gives the ASCII code (order in the ASCII table) of the character existing in object C. The result is of the integer type.

Example :

ORD('A') returns 65
A ← 'B'
ORD(A) returns 66

- CHR: so that CHR(N) gives the character corresponding to the ASCII code of the content of object N (N being of the integer type). The result is of the character type.

Example :

CHR(65) returns 'A'.

Note :

The two functions CHR and ORD are reciprocal. That is, for a variable C of the character type and a variable N of the integer type, we have:

CHR(ORD(C)) returns C
ORD(CHR(N)) returns N

- Comparison operators (<, <=, >, >=, =, <>): Comparing two characters is equivalent to comparing their respective ASCII codes.

4.4. Boolean Type

- **Definition**

This type is associated with objects that take their values in the set {True, False}.

- **Declaration**

identifier1, identifier2, ..., identifierN: Boolean

- **Manipulation Operations**
  - Negation (denoted as "NOT").

A	***NOT A***
True	False
False	True

  - Intersection (denoted as "AND").

AND	True	False
True	True	False
False	False	False

  - Union (denoted as "OR").

OR	True	False
True	True	True
False	True	False

Note: Operator Precedence

Each category of operators is associated with a precedence order. In fact, the evaluation of an expression is done by following the ascending order of operator precedence. If parentheses do not exist in an expression to be evaluated, and the operators have the same precedence, the evaluation is done from left to right.

Table 1 illustrates the order of precedence for arithmetic and logical operators.

Precedence Order	Operator Categories	Operators
1	Parentheses	(,)
2	Unary Operators	+ , -
3	Exponentiation	**
4	Multiplicative Operators	* , / , MOD , DIV
5	Additive Operators	+ , -

6	Comparison Operators	= , <> , < , > , <= , >=
7	Logical Negation	NOT
8	Logical AND	AND
9	Logical OR	OR

Table 1: Order of precedence for arithmetic and logical operators

1. The Alternative

1.1. Definition

The alternative is based on the evaluation of a condition. It allows deciding which treatment to execute based on the value returned by the condition (logical expression).

1.2. Conditional Scheme Formats

Reduced Form

```
IF<< Condition(s) >>THEN
  << Treatment 1 >>
END IF
```

Full Form

```
IF<< Condition(s) >> THEN
  << Treatment 1 >>
ELSE
  << Treatment 2 >>
END IF
```

If the condition is TRUE, Treatment 1 is executed in both forms. If it is FALSE, in the first form, nothing is done (it proceeds to the next action immediately after END IF); in the second form, Treatment 2 is executed before moving on to the action immediately after END IF.

Example

Consider the following algorithm:

```
ALGORITHM Try
VAR
        A, B, C: integer
BEGIN
        Read(A, B, C)
        IF (A < B and C > A) THEN
                A ← B + C
                B ← C * 2
        ELSE
                A ← C - B
                C ← 0
        END IF
        Write(A, B, C)
END
```

Manually trace the algorithm for the values:

a- A = 1, B = 4, and C = 2

b- A = 3, B = 2, and C = 4

Application 1

Write an algorithm that reads a real number as a test score and checks if the value is correct or not.

Application 2

Write an algorithm that reads an integer and determines its parity.

Application 3

Write an algorithm that reads a character and checks if it corresponds to an alphabetic letter or not.

1.3. Nesting of Conditional Schemes

Sometimes, Treatment 1 and Treatment 2 can themselves contain conditional schemes. In this case, it is called the nesting of conditional schemes. The general scheme can be presented as follows:

```
IF <<Condition 1>> THEN
        <<Treatment 11>>
        IF <<Condition 2>> THEN
                <<Treatment 21 >>
        ELSE
                <<Treatment 22 >>
        END IF
        <<Treatment 12 >>
ELSE
        <<Treatment 21 >>
        IF <<Condition 3>> THEN
                <<Treatment 31 >>
        ELSE
                <<Treatment 32 >>
        END IF
        <<Treatment 22 >>
END IF
```

Application 1

Write an algorithm that determines the sign of a number.

Application 2

Write an algorithm that compares two numbers.

Application 3

Write an algorithm that reads two integers A and B and checks if A is divisible by B or not.

2. The Selective

2.1. Definition

If, in the case of nested conditional schemes, the condition always pertains to the same variable (or expression), the selective can be used. It involves selecting a treatment from several options based on the value of the selector.

2.2. Syntax

```
SELECT (Selector) DO
        <Value List 1>: <<Treatment 1>>
        <Value List 2>: <<Treatment 2>>
        ...
        <Value List n>: <<Treatment n>>
        [OTHERWISE: <<Treatment n+1>>]
END SELECT
```

Where: <<Selector>> is an identifier of an object or an expression,

<<Treatment i>> is a series of actions to execute,

<<Value List i>> can be given as constants and/or intervals of constants of a type compatible with that of <<Selector>>.

Remarks

• Each value of the selector can appear only once in the value lists.

• The SELECT form triggers the execution of the treatment corresponding to the value list containing the value of the selector. The algorithm execution continues with the action immediately following END SELECT.

• If the value of the selector does not match any value in the value lists, and if the OTHERWISE block exists, the treatment corresponding to that block will be executed. If the OTHERWISE block is not present, no treatment will be executed, and the execution will continue with the action immediately following END SELECT.

• The selector cannot be of a real type or a string type.

• The comparison between the selector and the value lists is done successively by going through the lists one by one until arriving at the value list that matches, the OTHERWISE block, or END SELECT if the OTHERWISE block does not exist.

Application 1
Write an algorithm that reads a month number and displays the month in full words.
Application 2
Write an algorithm that reads a month number and displays the corresponding quarter.
Application 3
Write an algorithm that reads a month number and displays the number of days in that month.
Application 4
Write an algorithm that reads a character and displays whether it is a lowercase alphabetic character, an uppercase alphabetic character, or a digit. (Use both the alternative and the selective)."

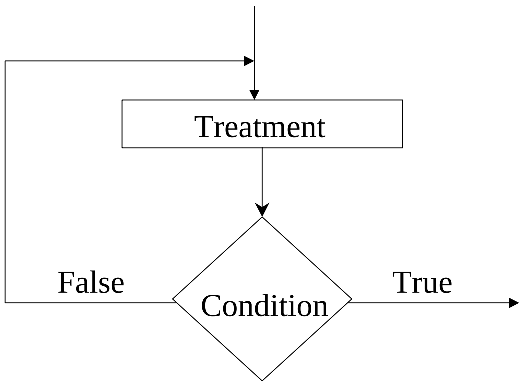
The algorithmic language has three repetitive structures:

- REPEAT ... UNTIL
- WHILE
- FOR

3. The REPEAT ... UNTIL Structure

Syntax

REPEAT
<<Treatment>>
UNTIL (Condition)



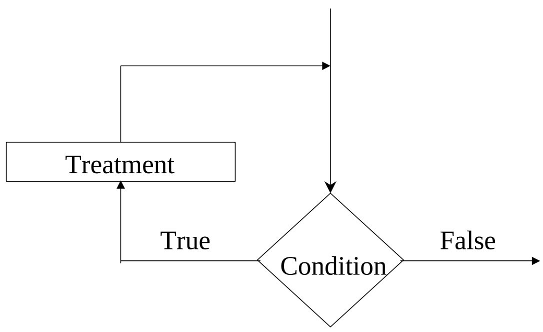
The evaluation and testing of the condition are done after the execution of the treatment (a set of actions composing the loop body). As a result, the treatment is executed at least once, and the number of repetitions is not known in advance. If the condition returns False, the treatment is repeated. There must be at least one action (part of the loop body) that changes the condition's value after a certain number of iterations.

Application 1
Write an algorithm to read a positive integer.
Application 2
Write an algorithm to read an alphabetic character.
Application 3
Write an algorithm to display the multiples of 2 between 2 and 100.
Application 4
Write an algorithm to display the multiples of 2 between two values A and B provided by the user, with the condition A < B and A >= 2.
Application 5
Write an algorithm to calculate the sum of 10 values provided by the user.

4. The WHILE Structure

Syntax

WHILE (Condition) DO
<<Treatment>>
END DO



The evaluation and testing of the condition are done before the execution of the treatment (a set of actions composing the loop body). As a result, the treatment may not be executed (if the first evaluation of the condition returns False), and the number of repetitions is not known in advance. There must be at least one action (part of the loop body) that changes the condition's value after a certain number of iterations.

Note
It is always possible to replace a REPEAT structure with a WHILE structure.

REPEAT
<<Treatment>>
UNTIL (Condition)

<<Treatment1>>
While (not Condition) Do
<<Treatment>>
End Do

Treatment1 is included or equal to Treatment.

Example 1

REPEAT
Read(A)
UNTIL (A > 0)

Read(A)
While (A <= 0) Do
Read(A)
End Do

Example 2

S ← 0
I ← 1
REPEAT
S ← S + I
I ← I + 1
UNTIL (I > 5)

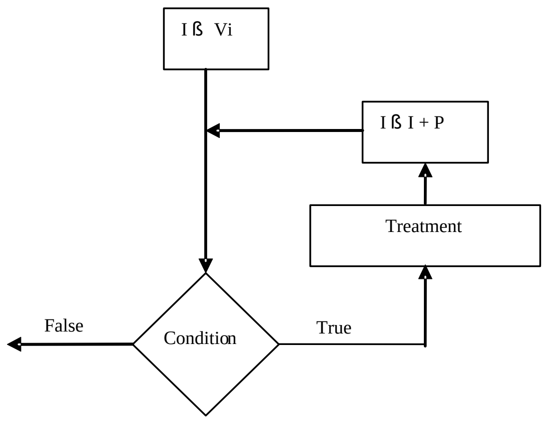
S ← 0
I ← 1
While (I <= 5) Do
S ← S + I
I ← I + 1
End Do

Applications
Revise applications 1, 3, and 5 from the previous section using the WHILE structure.

5. The FOR Structure

Syntax

FOR I **from** Vi **to** Vf **[BY** p**] DO**
<<Treatment>>
END DO



With:

I: the loop control variable
Vi: the initial value of I
Vf: the final value of I
p: the step value of I, defaults to 1.

The initialization of I to Vi and its incrementation by p are two actions performed automatically. The condition to decide whether to iterate or not depends on the sign of the step p:

If p > 0: the condition is I <= Vf.
If p < 0: the condition is I >= Vf.
The number of iterations depends on the values Vi, Vf, and p:

If Vi <= Vf and p > 0: the number of iterations is given by the formula: ((Vf - Vi) div p) + 1.
If Vi >= Vf and p < 0: the number of iterations is given by the formula: ((Vi - Vf) div (-p)) + 1.
If Vi <= Vf and p < 0 or Vi >= Vf and p > 0: the number of iterations is 0.
Note
It is always possible to replace a FOR structure with a WHILE structure, but the reverse is not always straightforward (if the number of repetitions is not known).

In the case of p > 0:

```
FOR I from Vi to Vf by p DO
        Treatment
END DO
```

Can be replaced by:

```
I ← Vi
WHILE (I <= Vf) DO
        Treatment
        I ← I + p
END DO
```

Application 1
Write an algorithm to display all even numbers in the range 0 to 100.

Application 2
Write an algorithm to calculate the sum of 10 integers provided by the user.

Application 3
Write an algorithm to display all odd numbers between 50 and 100 in descending order.

Homework N°1
Simple Data Types and Their Processing

Exercise 1

Consider the following algorithm:

```
Algorithm EX1
VAR
    t, h, m, s: integer
BEGIN
    Write ("Enter an integer")
    Read (t)
    h ← t div 3600
    m ← (t mod 3600) div 60
    s ← t mod 60
    Write ("H=", h, "M=", m, "S=", s)
END
```

1. Perform the manual execution of this algorithm with t=3665.
2. Deduce the purpose of this algorithm.

Exercise 2

Consider the following algorithm:

```
Algorithm EX2
VAR
    A, B, C: integer
BEGIN
    Write ("Enter three integers")
    Read (A, B, C)
    Write ("The values of A, B, and C are: ", A, B, C)
    A ← A + B + C
    C ← A - B - C
    B ← A - B - C
    A ← A - B - C
    Write ("The new values of A, B, and C are:", A, B, C)
END
```

1. Perform the manual execution of this algorithm with the following values :
    - A = 12, B = 5, C = 0
    - A = 7, B = 10, C = 13
2. Deduce the purpose of this algorithm.
3. Propose a second solution to the same problem.

Exercise 3

Write an algorithm that allows entering two integers, each assumed to have two digits, and insert the first integer in the middle of the second integer.

Example: N = 21 and M = 36, the algorithm will display the number 3216.

Exercise 4

Write an algorithm that allows entering a first integer N, assumed to have three digits, and a second integer M, assumed to have two digits, and insert the first integer in the middle of the second integer.

Example: N = 215 and M = 36, the algorithm will display the number 32156.

Exercise 5

Write an algorithm that reads a four-digit integer N and determines its mirror image. The "mirror of N" is the number formed by reversing the digits of N.

Example: N = 2369, so the mirror image of N is 9632.

Homework N°2
Control Flow

Exercise 1

Write an algorithm that allows entering three integers and displays the smallest among them.

Exercise 2

Write an algorithm that reads an integer and checks if it corresponds to the ASCII code of an alphabetic character.

Exercise 3

Write an algorithm that reads two integers and displays the sign of their product without having to calculate it. (The product can be positive, negative, or zero).

Exercise 4

Write an algorithm that allows reading a positive number of up to three digits and checks if it is cubic or not. A number is considered cubic if it is equal to the sum of the cubes of its constituent digits. Example: 153 = 1^3 + 5^3 + 3^3, so it is cubic.

Exercise 5

Write an algorithm that solves a second-degree equation ax^2 + bx + c = 0 in the set of real numbers (IR) and displays the result. The parameters a, b, and c will be provided by the user. All cases should be considered.

Exercise 6

Write an algorithm that allows entering two alphabetic characters, c1 and c2, and checks if these two characters correspond to the same letter, even if one is in uppercase and the other is in lowercase.

Exercise 7

Write an algorithm that reads a character and displays whether the character is an alphabetic letter or not. In the case of a lowercase (respectively uppercase) alphabetic letter, it displays its equivalent in uppercase (respectively lowercase).

Exercise 8

Consider the set of arithmetic operators:

OP $\in$ {+, -, *, /}

Write an algorithm to:

• Read two operands,
• Read an operator OP in the form of a character,
• Perform the corresponding operation,
• Display the result.

The problem should be solved using both the selective and alternative methods.

Exercise 9

Write an algorithm that allows entering a student's overall average and determines their result and distinction. The re-sit conditions are applied from an average of 9.75. In this case, the student will only be allowed to re-sit if the total number of absences does not exceed 20, and the number of teachers in favor of the re-sit is greater than the number of teachers against it.

Exercise 10

Write an algorithm that calculates the amount of overtime for an employee, based on the unit price of an hour and the total number of hours worked according to the following scale:

The first 39 hours have no overtime pay.
Beyond the 39th hour up to the 44th hour, there is a 50% overtime pay.
Beyond the 44th hour up to the 49th hour, there is a 75% overtime pay.
Beyond the 49th hour, there is a 100% overtime pay.
Example: Total hours worked = 53, unit price of an hour = 15 DT. Overtime amount = 5 * (15 + 15 * 0.5) + 5 * (15 + 15 * 0.75) + 4 * 30.

Exercise 11

Write an algorithm that reads a date in the format of day, month, and year and checks the validity of this date. If it's valid, it displays the next date.

Homework N°3
Control Flow

Exercise 1

Write an algorithm that displays the multiples of 5 between 50 and 100.

Exercise 2

Write an algorithm that allows you to:
Enter N non-zero integer values,
Calculate and display their sum and product.

Exercise 3

Write an algorithm that allows you to enter a positive number N, calculate, and display its factorial given by the following formula:
F = 1 * 2 * 3 * ... * (N-1) * N
Knowing that for N = 0, F = 1.

Exercise 4

Write an algorithm that allows you to enter a positive number and check if it is a cubic number or not. A number is considered cubic if it is equal to the sum of the cubes of its constituent digits.

Exercise 5

Write an algorithm that displays all the cubic numbers between two values V1 and V2 given by the user.

Exercise 6

Write an algorithm that allows you to enter a student's grades in 10 subjects and calculate their average. Each subject has a homework grade and an exam grade with weights of 40% and 60%, respectively.

Exercise 7

Revisit Exercise 6 to calculate the averages of N students and calculate and display:
The overall class average,
The highest average,
The lowest average,
The number of students with an average score greater than or equal to 10.

Exercise 8

Write an algorithm that allows you to convert a non-zero positive number into binary, using successive divisions by 2.

Exercise 9

Write an algorithm that allows you to determine if a number is prime or not. A number is prime if it is divisible only by 1 and itself.

Exercise 10

Write an algorithm that allows you to display all prime numbers between two positive values V1 and V2 provided by the user.

Exercise 11

Write an algorithm that allows you to:
Read two strictly positive integers N and P,
Calculate and display the quotient of the integer division of N by P without using the DIV function,
Calculate and display the remainder of the integer division of N by P without using the MOD function.