Correction of the Tutorial Series N°1

Exercise 1:

Choose the correct answer:

1) What does the code on the right represent:
   - ☒ Algorithm
   - ☐ Procedure
   - ☐ Function
2) What does line 2 represent:
   - ☐ Declaration of a function
   - ☒ Declaration of variables
   - ☐ Declaration of parameters
3) What does line 3 represent:
   - ☐ An assignment
   - ☐ A function
   - ☒ The header of a function
4) What do **a**, **b**, **c** represent:
   - ☐ Variables of the function
   - ☒ Formal parameters
   - ☐ Actual parameters
5) The word "**real**" in line 3 describes:
   - ☐ The type of the algorithm
   - ☐ The type of the function parameters
   - ☒ The return type of the function
6) The identifier **m** in line 4 is a:
   - ☒ Variable of the function
   - ☐ Formal parameter0
   - ☐ Actual parameter

```
1.  Algorithm calculation;
2.  Var Exam,TW,PW,avg:real;
3.  Function Average(a,b,c:real):real;
4.  Var m:real;
5.  Begin
6.  m ← a * 0.6 + b *0.2 + c * 0.2;
7.  Average ← m;
8.  End;
9.  Begin
10. For i ← 1 To 10 Do
11.     Begin
12.     Read(Exam,TW,PW);
13.     avg ← Average(Exam,TW,PW);
14.     Write(avg);
15.     End;
16. End.
```

7) Line 7 describes:
   - ☐ An assignment
   - ☒ A function return statement
   - ☐ A function call
8) The expression **Average(Exam,TW,PW)** in line 13 is called:
   - ☐ An assignment
   - ☐ A function return statement
   - ☒ A function call

Exercise 2:

1. Write the function **distance** with four real parameters: **xa**, **ya** and **xb**, **yb**, representing the coordinates in a 2D plane of two points A and B. This function should return the distance between these two points. The distance $\overline{AB}$ between two points A and B is given by the following formula:

$$\overline{AB} = \sqrt{(xa - xb)^2 + (ya - yb)^2}$$

2. Write the main algorithm that reads the coordinates of two points and displays the distance between them. The distance between the two points should be calculated using the **distance** function.

   Assume that there is a predefined function called **sqrt** that returns the square root of a given number.

Solution:

1)

```
Function distance(xa,ya,xb,yb:real):real;
Begin
distance ← sqrt((xa-xb)*(xa-xb)+(ya-yb)*(ya-yb));
End;
```

2)

```
Algorithm Ex2;
Var x1,x2,y1,y2,dis:real;
.....
Begin
Write("Enter the coordinates of point 1");
Read(x1,y1);
Write("Enter the coordinates of point 2");
Read(x2,y2);
dis ← distance(x1,y1,x2,y2);
Write("The distance is ",dis);
End.
```

Exercise 3:

1. Write a function that takes two integers as input and returns the maximum of the two.
2. Exploiting the previous function, write another function that returns the maximum of four integers passed as parameters.
3. Test the latter in a main algorithm.

Solution :

1)

```
Function max2Numbers(a, b: integer): integer;
Begin
If a > b Then max2Numbers ← a
Else max2Numbers ← b;
End;
```

2)

```
Function max4Numbers(a, b, c, d: integer): integer;
Var m1, m2: integer;
Begin
m1 ← max2Numbers(a, b);
m2 ← max2Numbers(c, d);
max4Numbers ← max2Numbers(m1, m2);
End;
```

3)

```
Algorithm Ex3;
Var x, y, z, t, max: integer;
.....
Begin
Read(x, y, z, t);
max ← max4Numbers(x, y, z, t);
Write(max);
End.
```

Exercise 4 :

1. Write the function **power** that takes a real number **x** and an integer **y** as input and returns x^y.
2. Write the function **conversion** that takes an integer **x**, composed only of 0s and 1s, considered as a binary number, and returns its equivalent in decimal.
   Example : $(10110)_2 = 0*2^0 + 1*2^1 + 1*2^2 + 0*2^3 + 1*2^4 = 2+4+16 = (22)_{10}$
3. Finally, write the main algorithm that reads a binary number **n** and displays its decimal equivalent. This is achieved by calling the **conversion** function.

Solution:

1)

```
Function power(x: real; y: integer): real;
Var i: integer; p: real;
Begin
p ← 1;
For i ← 1 to y Do
    p ← p * x;
power ← p;
End;
```

2)

```
Function conversion(x: integer): integer;
Var d, ch, i: integer;
Begin
i ← 0;
d ← 0;
While x ≠ 0 Do
    Begin
    ch ← x mod 10;
    d ← d + ch * power(2, i);
    x ← x div 10;
    i ← i + 1;
    End;
conversion ← d;
End;
```

3)

```
Algorithm Ex4;
Var n, dec: integer;
.....
Begin
Write("Enter a binary number: ");
Read(n);
dec ← conversion(n);
Write(n, " is equivalent to ", dec, " in decimal");
End.
```

Exercise 5:

We want to display the multiplication table for all numbers between 1 and a number **n** entered by the user. To do this, you are asked to:

1. Write a procedure that displays the multiplication table of a number **x** passed as a parameter.

2. Write another procedure to display the multiplication tables for all numbers between 1 and a number **y** passed as a parameter. Use the previous procedure.
3. Write the main algorithm that allows entering an integer **n** and displays the multiplication tables for all numbers between 1 and **n**. The display of multiplication tables should be done by calling the previous procedure.

Solution:

1)

```
Procedure displayTable(x: integer);
Var i, result: integer;
Begin
Write("Multiplication table of ", x);
For i ← 1 to 10 Do
    Begin
    result ← x * i;
    Write(x, "*", i, "=", result);
    End;
End;
```

2)

```
Procedure displayAllTables(y: integer);
Var i: integer;
Begin
For i ← 1 to y Do
    displayTable(i);
End;
```

3)

```
Algorithm Ex5;
Var n: integer;
.....
Begin
Write("Enter an integer: ");
Read(n);
displayAllTables(n);
End.
```

Exercise 6:

A rational number in mathematics is a number that can be expressed as the quotient of two integers: the numerator and the denominator.

1. Propose the declaration of the type **Rational** describing a relational number
2. Write the procedure **sumProduct** that takes two rational numbers **R1** and **R2** as input and calculates and returns their sum and product.
3. Test this procedure in a main algorithm.

Solution:

```
Type Rational = Record
                Begin
                Num, Den: integer;
                End;
```

1)

```
Procedure sumProduct(A, B: Rational; Var S, P: Rational);
Begin
S.Num ← A.Num * B.Den + A.Den * B.Num;
S.Den ← A.Den * B.Den;
P.Num ← A.Num * B.Num;
P.Den ← A.Den * B.Den;
End;
```

2)

```
Algorithm Ex6;
Type Rational = Record
                Begin
                Num, Den: integer;
                End;
Var R1, R2, S, P: Rational;
.....
Begin
Write("Enter the first rational number");
Read(R1.Num, R1.Den);
Write("Enter the second rational number");
Read(R2.Num, R2.Den);
sumProduct(R1, R2, S, P);
Write("The sum is ", S.Num, "/", S.Den);
Write("The product is ", P.Num, "/", P.Den);
End.
```

Exercise 7:

Let T be an array of 100 integers. A *peak* is defined as any element in the array that is greater than its predecessor and its successor. Similarly, a *valley* is defined as any element that is smaller than its predecessor and its successor in the array. The array is considered *balanced* if the number of peaks equals the number of valleys.

1. Write the procedure **`nbPeaksValleys`** that takes an array as input and returns the number of peaks and valleys in the array.
2. Write the function **`isBalanced`** that checks if an array passed as a parameter is balanced.
3. Write the main algorithm that fills an array T and displays whether it is balanced or not.

Solution:

```
Const n=100;
Type Tab=Array[n] of Integer;
```

1)

```
Procedure nbPeaksValleys(T: Tab; Var nbPeaks, nbValleys: integer);
Var i: integer;
Begin
nbPeaks ← 0;
nbValleys ← 0;
For i ← 1 to n-2 Do
    If T[i] > T[i-1] and T[i] > T[i+1] Then
           nbPeaks ← nbPeaks + 1
    Else If T[i] < T[i-1] and T[i] < T[i+1] Then
           nbValleys ← nbValleys + 1;
End;
```

2)

```
Function isBalanced(T: Tab): Boolean;
Var nbPeaks, nbValleys: integer;
Begin
nbPeaksValleys(T, nbPeaks, nbValleys);
If nbPeaks = nbValleys Then
    isBalanced ← True
Else isBalanced ← False;
End;
```

3)

```
Algorithm Ex7;
Const n = 100;
Type Tab = Array[n] of integer;
Var T: Tab;
    i, peakNum, valleyNum: integer;
Begin
For i ← 1 to n Do
    Read(T[i]);
If isBalanced(T) = True Then
    Write("The array is balanced")
Else Write("The array is not balanced");
End.
```

Exercise 8:

1. Write the recursive procedure **`display`** that displays numbers from 1 to an integer **`n`** passed as a parameter.
2. Modify the previous procedure so that the display is in reverse order.

Modify the procedure again to display numbers from 1 to **`n`** and then from **`n`** to 1.

Solution:

1)

First solution

```
Procedure display(n, i: integer);
Begin
If i ≤ n Then
     Begin
     Write(i);
     display(n, i+1);
     End;
End;
```

Do not allow to answer question 3

Second solution

```
Procedure display(n: integer);
Begin
If i=1 Then Write(n)
Else Begin
     display(n-1);
     Write(n);
     End;
End;
```

2)

First solution

```
Procedure display(n, i: integer);
Begin
If i ≤ n Then
     Begin
     display(n, i+1);
     Write(i);
     End;
End;
```

Second solution

```
Procedure display(n: integer);
Begin
If i=1 Then Write(n)
Else Begin
     Write(n);
     display(n-1);
     End;
End;
```

3)

First solution

```
Procedure display(n, i: integer);
Begin
If i ≤ n Then
     Begin
      Write(i);
     display(n, i+1);
      Write(i);
     End;
End;
```

Incorrect, display is from n to 1 and then from 1 to n

Second solution

```
Procedure display(n: integer);
Begin
If i=1 Then Write(n)
Else Begin
     Write(n);
     display(n-1);
     Write(n);
      End;
End;
```

Example of call:

```
Algorithm Ex8;
Var x: integer;
.......
Begin
Read(x);
display(x, 1);   // The first number to display is always 1
End.
```

Exercise 9:

1. Write the recursive function **`nbEvenDigits`** to return the number of even digits in a number passed as a parameter.
2. Test this function in a main algorithm.

Solution:

Before tackle this exercise, demand from the students to resolve an additional exercise considered as preparation to this exercise.

The additional exercise is: Write the recursive function **`nbDigits`** to return the number of digits in a number passed as a parameter.

1)

```
Function nbEvenDigits(n: integer): integer;
Var r:integer;
Begin
If n div 10 = 0 Then
     If n mod 2 = 0 then  nbEvenDigits ← 1
     Else nbEvenDigits ← 0
Else Begin
     R ← n mod 10;
     If r mod 2 = 0 Then
          nbEvenDigits ← 1 + nbEvenDigits(n div 10)
     Else nbEvenDigits ← nbEvenDigits(n div 10);
     End;
End.
```

Additional Exercises

Exercise 10:

We would like to write a set of procedures and functions to easily manipulate hours and minutes. Write the following subprograms:

1. The function **Minutes**, which calculates the number of minutes corresponding to a given number of hours and minutes.
2. The function or procedure **HoursMinutes** that performs the inverse transformation of the **Minutes** function. Thus, given a total number of minutes, it calculates the equivalent hours and remaining minutes.
3. The procedure **addTimes** that adds two pairs of times represented in hours and minutes using the two previous functions.

Solution:

1)

```
Function Minutes(h,m:integer):integer;
Begin
Minutes ← h*60+m;
End;
```

2)

```
Procedure HoursMinutes(n:integer; Var h,m:integer);
Begin
h ← n div 60;
m ← n mod 60;
End;
```

3)

```
Procedure addTimes(h1,m1,h2,m2:integer; Var h,m:integer);
Var n1,n2,n:integer;
Begin
n1 ← Minutes(h1,m1);
n2 ← Minutes(h2,m2);
n ← n1+n2;
HoursMinutes(n,h,m);
End;
```

Exercise 11:

Let T be an array of 20 real numbers.

1. Write a procedure **MinMaxSom (T, minT, maxT,somT)** that calculates and returns the smallest element **minT**, the largest element **maxT**, and the sum of all elements **sumT** of the array **T**.
2. The Olympic average of a set of numbers is the arithmetic mean of all the numbers in this set except the smallest and the largest.

   For example, for the set: 2, 3, 13, 7, 8, the *arithmetic mean* is 6.6, and the *Olympic average* is 6.

   Write an algorithm that allows entering the array **T** and calculating and displaying its Olympic average.

Solution:

```
Const n=20;

Type Tab=Array[n] of real;
```

1)

```
Procedure MinMaxSum(T: Tab; Var minT, maxT, sumT: real);
Var i: integer;
Begin
minT ← T[0];
maxT ← T[0];
sumT ← T[0];
For i <- 1 to n-1 Do
     Begin
     sumT ← sumT + T[i];
     If T[i] < minT Then minT ← T[i]
     Else If T[i] > maxT Then maxT ← T[i];
     End;
End;
```

2)

```
Algorithm Ex11;
Const n = 20;
Type Tab = Array[n] of real;
Var T: Tab; i: integer; min, max, sum, avg: real;
......
Begin
For i ← 1 to n Do
     Read(T[i]);
MinMaxSum(T, min, max, sum);
avf ← (sum - min - max) / (n - 2);
Write("The Olympic average is ", avg);
End.
```

Exercise 12:

We want to calculate the greatest common divisor (GCD) between two numbers, **a** and **b**, using the method of successive subtractions. This method operates as follows:

- If **a = b**, the GCD is **a**.
- Otherwise, we calculate the GCD of the pair formed by the difference between **a** and **b** and the smaller of the two.

  GCD(a,b) = GCD(a−b,b) if **a > b**

  GCD(a,b) = GCD(a,b−a) if **b > a**

  GCD(a,b) = a if **a = b**

To achieve this, you are asked to:

1. Provide a recursive function to calculate the GCD of two integers passed as parameters using the method of successive subtractions.
2. Write the main algorithm that reads two numbers and calculates and displays their GCD. The GCD calculation should be performed using the previous function.

Solution:

1)

```
Function GCD(a, b: integer): integer;
```

```
Begin
If a = b Then GCD ← a
Else If a > b Then GCD ← GCD(a - b, b)
Else GCD ← GCD(a, b - a);
End;
```

2)

```
Algorithm Ex12;
Var x, y: integer;
Begin
Write("Enter 2 positive integers: ");
Read(x, y);
If x <= 0 or y <= 0 Then Write("Input error")
Else Write("The GCD is ", GCD(x, y));
End.
```

Exercise 13:

1. Write a recursive procedure to reverse an array of integers passed as a parameter.
2. Write the main algorithm that fills an array of integers, reverses it using the previous procedure, and displays it.

Solution:

```
Const n=6;
Type Tab=Array[n] of Integer;
```

1)

```
Procedure Reverse(T: Tab; i: integer);
Var x: integer;
Begin
If i <= (n - 1) div 2 Then
     Begin
     x ← T[i];
     T[i] ← T[n - 1 - i];
     T[n - 1 - i] ← x;
     Reverse(T, i + 1);
     End;
End;
```

2)

```
Algorithm Ex13;
Const n = 6;
Var T: Array[n] of integer; i: integer;
Begin
For i ← 0 To n-1 Do
     Read(T[i]);
Reverse(T, 0);
Write("After reversal, the array becomes: ");
For i ← 0 To n-1 Do
     Write(T[i]);
End.
```