People's Democratic Republic of Algeria
Ministry of higher education and scientific research
Larbi Ben M'hidi-Oum El Bouaghi University

Faculty of Exact Sciences and Natural and Life Sciences
Mathematics and Computer Science Department

Educational Handout of Tutorials and Practical Work in Algorithms and Data Structures 2 (ADS2)

Level: 1st year "Computer science"

Proposed by:

Dr. Dehimi Nour El Houda

Academic year : 2024-2025

Table of Contents

Preamble

This document presents tutorials and practical exercises, designed to teach the subject "Algorithms and Data Structure 2," introduced in the second semester in the Department of Mathematics and Computer Science at Oum El Bouaghi University, and intended for first-year computer science students.

First, the pedagogical objectives of this document are:

- ✓ Manipulate sub-algorithms (subroutines): procedures & functions;
- ✓ Understand recursive sub-algorithms;
- ✓ Understand the declaration, syntax and semantics of pointers and linked lists;
- ✓ Allow the student to acquire the fundamentals of programming.

Indeed, and in order to achieve the aforementioned objectives, we have made tremendous efforts to approach this work in several aspects; where we have synthesized the most important and relevant information based on different documentary sources (books, articles, courses, websites, etc.). While respecting the official canevas determined by the Ministry of Higher Education and Scientific Research.

Furthermore, this document is divided into two sections: the first presents tutorials and a set of exercises with their solutions, divided into parts, while the second section provides a set of practical exercises that enable students to acquire the basics of C programming.

However, the document in question will remain partial and not exhaustive. This is why we will constantly update it to enrich its content. However, we would be grateful if readers could notify us of any errors, observations, etc., and also offer us opinions in this regard.

Section I

« Tutorials »

Section 1: Tutorials -Exercises-

Part 1 : Records

Pedagogical objectives

- Understand the usefulness of records and manipulate them to solve different problems.

Exercise n°1

Define a TIME type (record) which contains the fields: hour, minute, second.

1. Write an algorithm which allows you to perform the sum T of two durations T1 and T2 of type TIME.
2. Write an algorithm which allows you to transform a time T of type TIME into an integer S which expresses this time in seconds. Example: for T = 2 hours 10 minutes 37 seconds, **S = 7837 seconds**.

Exercise n°2

A complex number C is defined by its real “a” and imaginary “b” parts (C = a + bi) .

Write an algorithm that reads two complex numbers C1 and C2 and then displays their sum and product.

Exercise n°3

“Ens” is a record defined by two pieces of information (fields):

- T_Pos an array of integers that can contain a maximum of 50 elements;
- N the number of elements of the array T_Pos. Given a string Ch, write an algorithm which allows you to search for the string "ab" and put in a record of type Ens (in the table T_Pos) all the positions of the string “ab” in Ch.

Example: Ch = ' faabaababbaabrs '.

Positions: 3, 6, 8, 12 => T_Pos [1]=3, T_Pos [2]=6, T_Pos [3]=8, T_Pos [4]=12.

Number of elements: 4 = > N=4.

Exercise n°4

A computer is characterized by its serial number, brand, model, and price.

1. Define the structured type Computer.
2. Write an algorithm that stores the information of 20 computers, then displays the computers whose price is greater than 50.000 DA.

Exercice n°5

A car is characterized by its registration_number, brand, model and price.

1- Define the structured type Car.

2- Write an algorithm allowing you to record information about 20 cars and display the most expensive one.

Part 2: Sub-algorithms

Pedagogical objectives

- ✓ Manipulate sub-algorithms (subroutines): procedures & functions;
- ✓ Understand the difference between them;
- ✓ Understand the concepts: local variable, global variable, formal parameter, effective parameter, passing parameters by value and by address.

Exercise n°1

A procedure is declared as follows:

```
Procedure P_Test (A, B, C: integer);
Variable S: integer;
Begin
S← A+B+C;
Write (S);
End;
```

1. Write a main algorithm that calls this procedure.
2. Specify local and global variables, formal and effective parameters.
3. Replace the procedure P_Test with a function F_Test.
4. Call this function in the main algorithm.

Exercise n° 2

Consider the following algorithm:

```
Algorithm exo2;
Variables x, y, z, t: integer;
Procedure my_procedure (a, b, var c, d: integer)
Begin
      c ← a + b ;
      d ← a * b;
End;
Begin
     read (x);
     read (y);
     my_procedure (x, y, z, t);
     write (z);
     write ( t);
END
```

1. Identify the real (effective) parameters and the formal parameters .
2. What does this program display assuming the user enters 2 in *x* and 3 in *y* ? Modify the algorithm to obtain a more logical result .

Exercise n°3

Consider the following algorithm:

```
Algorithm exo3;
Variables T: array [1…100] integer;
                i,N :integer ;

Procedure P1 (T: array [1.. N ] integer);
Variables  i, a: integer;
Begin
   a ←0 ;
    For i ← 1 to N do
         a←a+ T [i] ;
    Endfor
  Write (a);
End;

Procedure P2 (T: array [1.. N ] integer);
Variables  i , b: integer;
Begin
 b← 1 ;
 For i ← 1 to N do
      b←b *T[i] ;
 Endfor
Write (b);
End;
Begin
Repeat
```

```
   Read(N);
Until (N>=1 and N<100)
For i ← 1 to N do
   Read (T[i]);
Endfor
P1(T);
P2(T);
END
```

1. Run the algorithm with the following array:

1	0	2	4	3	1	2	1	2	3

2. What is the role of the procedure P1?
3. What is the role of the procedure P2?
4. In the main algorithm, is it possible to do the following calculation:
   C1 = a/2 and C2 = b/2? Justify your answer.
5. Replace both procedures with functions. In this case is it possible to calculate C1and C2 in the main algorithm? Justify your answer.

Exercise n°4

1. Write a FindVal sub-algorithm that indicates whether a value is contained in a one-dimensional array (with the size N). If so, the sub-algorithm must indicate in which cell the value was found.
2. Write a sub-algorithm which takes as parameters two arrays of real numbers and which returns the value *true* if they are identical, *false* otherwise.
3. Design local and global variables, formal and effective parameters.

Exercise n°5

A positive integer is **perfect** if it is equal to the sum of its divisors (except itself). For example 6 is perfect, because 6 = 1+2+3; similarly 28 is perfect, because 28 = 1 + 2 + 4 + 7 + 14.

1. Write a function **Som_Div** which calculates the sum of the divisors of **n** .
2. Write a **P_perfect_procedure** which uses the **Som_Div** function and indicates whether **n** is perfect or not.
3. Transform this procedure into a Boolean function **F_ perfect.**
4. Use the previous three sub-algorithms in an algorithm.
5. Design local and global variables, formal and effective parameters as well as sub-algorithms calls.

Exercise n°6

Consider an array **T** containing the marks of **N** students (**N ≤ 100**) in a given module. We aim to statistically analyze these results using several procedures:

1. Write a procedure **Admitted_Students (T, N)** that displays all marks greater than or equal to 10 (admitted students).
2. Write a procedure **Non_Admitted_Students (T, N)** that displays all marks strictly less than 10 (non-admitted students).
3. Write a procedure **Class_Average (T, N)** that calculates and displays the overall class average.

*This procedure calls a **recursive** function named **Sum_Grades (T, N)** that calculates the sum of the marks.*

4. Write a procedure **Highest_Mark (T, N)** that identifies and displays the highest mark in the array.
5. Write a procedure **Lowest_Mark (T, N)** that identifies and displays the lowest mark in the array.
6. Write **the main algorithm** that:
   - ✓ Fills the Marks array with input or generated values,
   - ✓ Calls the above procedures in sequence.
7. Is it possible to calculate the difference between the highest and lowest grade in the main algorithm? Clearly justify your answer. If your answer is no, make the necessary changes to ensure this difference can be calculated in the main algorithm.
8. After the initial input of marks for 100 students, some new students enrolled late. This situation highlights the limitation of using an array structure with a fixed size. **What solution would you propose to allow the dynamic addition of new marks?**

Exercise n°7

Consider two vectors V1 and V2 of 20 integers:

1. Write a function "**Product (V1, V2: [1..20] array of integer): integer**" that allows you to calculate **P:** the scalar product of two vectors.
2. Write a procedure "**Sum (V1, V2: [1..20] array of integer)**" that allows you to calculate **S1:** the sum of the elements of the first vector V1 and **S2:** the sum of the elements of the second vector V2.
3. Write the main algorithm in which we call the previous sub-algorithms (**Product** and **Sum**).
4. In the main algorithm, is it possible to compare between **S1** and **S2?** If your answer is no give the appropriate solution.

NB: The scalar product of two vectors V1 and V2 of dimension *n* and coordinates such that:

$$V1(x_1, x_2, ..., x_n) \quad \text{et} \quad V2(y_1, y_2, ..., y_n)$$

$$P = V1.V2 = \sum_{i=1}^{n} x_i \, . \, y_i = x_1y_1 + x_2y_2 + ... + x_ny_n$$

Exercice n°8

Given an array T(N) of integers:

1. Write a procedure Nbr_Positives(T) to:
   - ✓ Find and display all positive numbers in array T,
   - ✓ Calculate and display the sum of these numbers.
2. Write a procedure Nbr_Negatives(T) to:
   - ✓ Find and display all negative numbers in array T,
   - ✓ Calculate and display the sum of these numbers.
3. Write the main algorithm
4. In the main algorithm, is it possible to calculate the total sum of the numbers (positive and negative) in the array? Justify your answer.
5. If your answer is No, make the necessary modifications to ensure that the total sum can be calculated in the main algorithm.

Part 3: Recursive sub-algorithms

Pedagogical objectives

Manipulate recursive sub-algorithms.

Exercise n°1

Run the following recursive function (for n = 8, x = 5) and deduce what it is doing.

```
Function Product (n: integer, x: integer): integer;
Begin
 If (n > 0) then
    Write ("before call n=", n, ", x=", x );
    Product <- Product (n-1, x) + x;
    Write (" after call n=", n, "x= ", x);
 Else
    Product <- 0
 Endif
End;
```

```
Begin /* main algorithm*/
n = 8, x = 5;
Write (n, '*', x, '=' ,Product (n, x));
END
```

Exercise n°2

a. Write an iterative function that returns the quotient of the Euclidean division of an integer **a** by an integer **b** using successive subtraction.
b. Give the corresponding recursive function.

Exercise n°3

Write an algorithm that uses a recursive sub-algorithm to calculate the greatest common divisor (GCD) of two strictly positive integer values using the Euclid method.

Exercise n°4

1. Write a recursive function **Sum_Tab**, allowing you to calculate the sum of the elements of an array of integers.
2. Write a recursive procedure **Inverse_Tab ,** allowing you to reverse the elements of an array of integers.
3. Write an algorithm that uses the Sum-Tab and Inverse_Tab sub-algorithms.

Exercise n°5

Write a recursive function to calculate Xn. (X integer).

2. Write a recursive function to calculate X !. (X integer).

3. Use these two functions in the main algorithm to calculate the following sum:

S = (Xn+1 – (1) !) + (Xn+2 – (2) !) + (Xn+3– (3) !) + ……. + (Xn+m – (m) !)

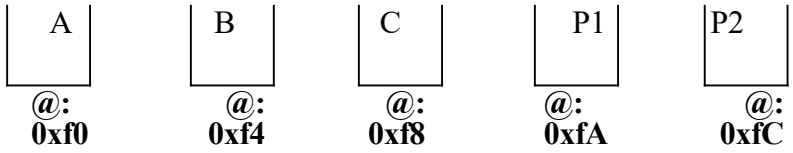
Part 4: Pointers and linked lists

Pedagogical objective

- ✓ Understand the declaration, syntax and semantics of pointers, among others: indirect addressing, the content of a dynamic variable as well as arrays using pointers;
- ✓ Understand dynamic memory allocation and freeing.
- ✓ Manipulate linked lists using different operations.

Exercise n°1

Let three integers A, B, C and two pointers of integer type P1 and P2.



Determine the values of the different elements given in the table for each operation.

	A	**B**	**C**	**&A**	**&B**	**&C**	**P1**	**P2**	***P1**	***P2**
A←1, B←2, C←3 P1←&A, P2←&B										
P2= &C										
***P1 = *P2**										
(*P2)++										
P1= P2										
P2= &B										
***P2 = *P1 - 2* *P2**										
(*P2)--										
C = (P2 == &C)										
***P2= *P1 + A**										

Exercise n°2

Let a pointer P, which points to an array A:

int A[ ]= {12, 23, 34, 45, 56, 67, 78, 89, 90}; int *P; P = A;

What values or addresses do the following expressions provide?

a) *P+2

b) *(P+2)

c) &P+1

d) &A[4]-3

e) A+3

f) &A[7]-P

g) P+(*P-10)

h) *(P+*(P+8)-A[7])

Exercise n°3

Write an algorithm that allows you to:

- Declare pointers to: an integer, a real number, a character, a "Student" record composed of the following fields: *Registration number, Last name, First name and Result* ;
- Then Read() data that corresponds to the variables pointed to by these pointers.

Exercise n°4

Write an algorithm that arranges the elements of an array of N integers in reverse order. Using two pointers P1 and P2 and a numeric variable HELP (intermediate variable) to swap the array elements.

Exercise n°5

An employer is defined by the following information: *last name, first name, date of birth, number of children and qualification.* We want to establish a list of all employers, knowing that we do not know their number beforehand, however, they will not **exceed 300** (<=300).

1. Propose a data structure (records) to be used to manage employers.
2. Write an algorithm to enter all the information.
3. Write an algorithm to sort employers according to increasing order of their ages and only keep in memory 10% of the youngest employers.

Exercise n°6

Consider an array **T** of integers with a maximum size of 100.

Using **only pointers** (without any variable of integer type), write an algorithm that:

1. Reads the elements of the array.
2. Displays **the indices of the elements that are multiples of 3 (divisible by 3).**
3. Calculates and displays **the product of these elements.**

Exercice n°7

We have T an array of integers with a maximum size of 100. Using the pointers, write an algorithm, which allows you to:

1. Read the array;
2. Print the indexes of the odd elements and calculate their sum.

NB. Without using any integer variables just the pointers.

Exercice n°8

We want to build a list of elements defined by real-type data, without knowing their number in advance.

Using the singly linked list data structure, write an algorithm that allows:

a) Creating the list of elements.

b) Displaying the list of elements.

c) Inserting an element at the K-th position in the list.

d) Deleting a given element from the list.

Section 1: Tutorials -Solutions-

Part 1: Records - Solutions-

Pedagogical objectives

- Understand the usefulness of custom types and manipulate them to solve different problems.

Solution of the exercise n°1

```
Type TIME= Record
   Hour, Minute, Sec: integer;
EndRecord
```

```
Algorithm Sum_Time ;
Variables X: integer;
          T, T1, T2: TIME;
Begin
Write ( " Enter the T1 and T2 durations in hours, minutes and seconds " )
Read (T1.Hour);
Read (T1.Minute);
Read (T1.Sec);
Read (T2.Hour);
Read (T2.Minute);
Read (T2.Sec);
```

```
X←T1.Sec+ T2.Sec;
T.Sec←X mod 60;
T.Minute ← X div 60;
X←T.Minute+T1.Minute+ T2.Minute;
T.Minute ← X mod 60;
T.Hour ← X div 60+ T1.Hour+ T2.Hour;
Write ( " The result of the sum of two durations T1 and T2 is ", T.Hour , " Hour ", T.Minute , "
Minute ", T.Sec , " Second " );
END
```

```
Algorithm Transformation;
Variables T: TIME;
          R: integer;
Begin
Write ( " Enter the duration T in hours, minutes and seconds " )
Read ( T.Hour );
Read ( T.Minute );
Read ( T.Sec );
R←T.Sec +( 60* T.Minute )+(3600* T.Hour );
Write ( " The duration in seconds is ", R );
END
```

Solution of the exercise n°2

```
Type Complex = Record
        Preel : real;
        Pimag : real;
EndRecord
```

```
Algorithm Complex_Calculation ;
Variables S, P, C1, C2: Complex;
Begin
Write ("Give the real part of the 1st complex number C1: ");
Read (C1.Preel);
Write ("Give the imaginary part of the 1st complex number C1: ");
Read (C1.Pimag);
```

```
Write ("Give the real part of the 2nd complex number C2: ") ;
Read (C2.Preel);
Write ("Give the imaginary part of the 2nd complex number C2: ") ;
Read (C2.Pimag );
S.Preel ← C1.Preel + C2.Preel;
S.Pimag ← C1.Pimag + C2.Pimag;
Write ("Sum = ", S.Preel , "+ ", S.Pimag , " i ");
P.Preel ← (C1.Preel * C2.Preel)- (C1.Pimag * C2.Pimag);
P.Pimag ← (C1.Preel * C2.Pimag )+ (C1.Pimag * C2.Preel);
Write ("Product = ", P.Preel , "+", P.Pimag , "i");
END
```

Solution to the exercise n°3

```
Type Ens = Record
        T_Pos :array [1..50] integer;
        N: integer;
EndRecord
```

```
Algorithm Pos_ab ;
Variables i, j, k, T :integer ;
                Ch: string;
                Pos: Ens;
Begin
Write ("Enter a string");
Read ( Ch );
T← Length ( Ch );
i← 1 ;
j←1 ;
Pos.N←0 ;
While ( i<T) do
    If ( Ch [i]='a' and Ch [ i+1]='b') then
       Pos. T_Pos [j] ←i ;
       Pos.N←Pos.N+1 ;
```

```
        j←j+1 ;
        i←i+2 ;
    Else
        i←i+1
    End if
Endwhile
Write ("The positions of the string 'ab' in ", Ch , "are");
For k ← 1 to j do
    Write (Pos. T_Pos [k]);
Endfor
Write ("The number of elements is", Pos.N );
END
```

Solution of the exercise n°4

```
Algorithm Exercise_4 ;
Type  Computer=Record
                serial_number: integer;
                brand: string(20); model:
                string(20); price: real;
 EndRecord
Var:
          C : array [1..20] of Computer; i :
          integer;
Begin
Write ("Enter the information about the Computers");
For  1←i to 20 do
          Read (C[i]. serial_number); Read
          (C[i]. brand);
           Read (C[i]. model);
           Read (C[i]. price);
Endfor
```

```
For i←1 to 20 do
 If (C[i].price > 50000) then
    Write (C[i].serial_number, C[i].brand, C[i].model,  C[i].price);
 endif
Endfor

END
```

Solution of the exercise n°4

```
Algorithm car ;
Type  Car=Record
                registration_number: integer;
                brand: string(20);
                 model: string(20);
                 price: real;
  EndRecord
Variables
            C : array [1..20] of Car;
            i, pr, ind: integer;
Begin
Write ("Enter the information about the cars");
For  1←i to 20 do
      Read (C[i]. registration_number);
      Read (C[i]. brand);
       Read (C[i]. model);
       Read (C[i]. price);
Endfor
pr← C[1].price ; ind← 1 ;
For  2←i to 20 do
  If (C[i].price>pr) then
       pr ← C[i].price ;
        ind← i;
  endif
Endfor
Write ("The most expensive car is the", ind, "car");
END
```

Part 2: Sub-algorithms – Solutions-

Pedagogical objectives

- ✓ Manipulate sub-algorithms (subroutines): procedures & functions;
- ✓ Understand the difference between them;
- ✓ Understand the concepts: local variable, global variable, formal parameter, effective parameter, passing parameters by value and by address.

Solution of the exercise n°1

- ✓ Questions 1 and 2

```
Algorithm exo1;                          Global variables
Variables X ,Y,Z : integer;
P_Test Procedure (A, B, C: integer);     Formal parameters
Variable  S: integer;                    Local variable
Begin
S←A+B+C
Write (S);
End;
Begin
Read (X,Y,Z);                            Effective parameters
P_Test (X, Y, Z);
END.
```

✓ Questions 3 and 4

```
Algorithm exo1;
Variables X ,Y,Z, R : integer;
Function F_Test  (A, B, C: integer): integer;
Var S: integer;
Begin
S← A+B+C.
F_Test←S;End;
Begin
Read (X,Y,Z);
R← F_Test (X, Y, Z);Write
(R);
END.
```

Solution of the exercise n°2

- The effective parameters are those of the calling program (of the main algorithm): *x, y, z, t.*
- The formal parameters are those of the sub-algorithm (the procedure) and make it possible to recover the value of the real parameters: *a, b, c, d*

✓ *What does this program display assuming the user enters 2 in x and 3 in y ?*

```
Algorithm exo2;
Procedure ma_ procedure ( a, b, var c, d: integer)
Begin
      c ← a + b ;
      d ← a × b;
End;

Variables  x , y, z, t: integer ;
 Begin
     read (x);
     read (y);
     my_procedure (x, y, z, t);

     write (z);
     write ( t);
END
```

x	y	z	t
2	3	a + b = 2+3= **5**	

a	b	c	d
2	3		a × b = 2×3 = **6**

z	t
5	

After the call to *my_procedure* , ***z* = 5** and ***t* is worth nothing specific**. Indeed, c is a parameter passed by variable, so the modifications made to *c* in the sub-algorithm are reflected in the corresponding effective parameter *z*. On the other hand, *d* being a parameter passed by value,

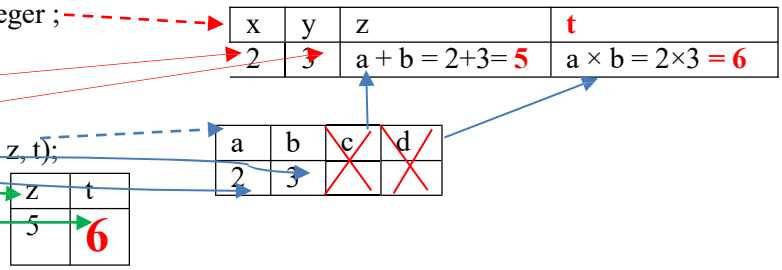
the corresponding effective parameter *t* is not affected by the changes. *t* therefore retains its value before the call to the sub-algorithm.

✓ *Modify the code to obtain a more logical result.*

By "transforming" *d* into a parameter by variable (address), in this case *my_procedure* takes two integers as input and returns via *c* the sum of these two integers and via *d* their product.

Algorithm exo2;

Procedure my_ procedure (a, b, **var** c, **var** d: integer)
Begin
c ← a + b ;
d ← a × b;
End;
Variables x , y, z, t: integer ;
Begin
read (x);
read (y);
my_procedure (x, y, z, t);
write (z);
write (t);
END



Solution of the exercise n° 3

1. Run the algorithm with the following array:

1	0	2	4	3	1	2	1	2	3

The algorithm displays:

19
0

2. What is the role of the procedure P1?

   It calculates the sum of the elements of an array.

3. What is the role of the procedure P2?

   It calculates the product of the elements of an array.

4. In the main algorithm, is it possible to calculate C1= a/2 and C2 = b/2?

   No, we cannot.

Justify your answer

The variables **"a",** declared to calculate the sum, and " **b",** declared to calculate the product, are **local variables.** In fact, they cannot be used in the main algorithm because the space of these local variables is freed at the end of the execution of the procedures.

5. Replace both procedures with functions.

```
Algorithm exo3;
Variables T : array [1..100] integer;
            N,S,P, i:integer;
Function F1 (T: array [1.. N ] integer): integer ;
Variables i , a: integer;
Begin
   a ←0 ;
 For i ← 1 to N do
       a←a+ T [i] ;
End for
   F1←a ;
End;
Function F2 ( T: array [1.. N ] integer ): integer ;
Variables i , b: integer;
Begin
b←1;
For i ← 1to N do
       b←b *T[i];
Endfor
    F2←b ;
End;
Begin
Repeat
   Read(N);
Until (N>=1 and N<100)
 For i ← 1 to N do
     Read (T[i]);
 Endfor
  S← F1(T); Write (S);
  P ←F2(T); Write (P);
END
```

In this case, is it possible to calculate C1and C2 in the main algorithm?

Yes. The use of functions makes it possible to recover the result of the sum “a” in **the global variable S** and the result of the product “b” in **the global variable P** ;

Solution of the exercise n°4

1.

```
Algorithm exo4;
Variables  T: array [1..20] real;
                  val: real;
                  case: integer;
                   b: boolean;
Procedure FindVal (N: Integer,T: array[1.. N ] real,val: real; Var rg : integer, found: boolean );
Variables i: integer;
Begin
found ← false;
rg ← 0 ;
i ← 1 ;
Repeat
 If (T[i] = val) then
    found ←  true;
    rg ← i;
 else
  | i ← i + 1 ;
 endif
Until (found = true OR i > N)
End;
Begin
// We assume that T is already filled
Read (val);
FindVal (20, T, val, case, b);
If (b = true) then
    Write ("The value", val, "was detected at the position", case);
Else
    Write ("Value not found");
Endif
END
```

2.

```
Algorithm exo4;
Variables A1, A2: array [1.. 20 ] real;
                M1, M2: integer;
                b: boolean;
Function Compare (N1, N2: integer, T1: array [1..N1 ] real, T2: array [1..N2] real): boolean;
Variables i: integer;
                identical: boolean;
Begin
```

```
 If (N1 ≠ N2) then
     identical = false; // if the arrays are different sizes, they are not identical.
 else
  i←1;
  identical ← true;
 Repeat
  If (T1[ i] ≠ T2[i] )Then
     identical ← false;
 else
    i ← i+1 ;
  Endif
Until (identical = false OR i > N1)
 Compare ← identical;
End if
END
Begin
Read (M1 , M2); // M1 and M2 <= 20 We assume that A1 and A2 are already filled
    b= Compare (M1, M2, A1, A2 );
    If (b = true) then
       Write ("the two tables are identical");
    else
       Write ("the two tables are not identical") ;
    Endif
END
```

Solution of the exercise n° 5

1.

```
 Function Som_Div (n: integer): integer;
    Variable i, sum: integer;
 Begin
   sum ←0 ;
   For i ←1 to (n div 2) do
        If (n mod i = 0) then
            sum ← sum+i ;
       Endif
    EndFor
  Som_Div← sum;
End;
```

2.

```
 Procedure P_Perfect (n: integer);
  Variable S: integer;
Begin
   S← Som_Div (n);
```

```
    If (S = n) then
        Write ("the number", n, " is perfect");
    Else
        Write ("the number", n, "is not perfect");
    Endif
End;
```

3.

```
Function F_Perfect (n: integer): boolean;
Variable S: integer;
Begin
    S= Som_Div (n);
    If (S = n) then
        F_Perfect ←true;
    Else
        F_Perfect ←false;
    Endif
End;
```

4. Main algorithm:

```
Algorithm Perfect;
Variables Bool : boolean, n: integer;
//Declaration and definition of subroutines 1, 2, 3.
Function Som_Div (n: integer): integer;
    Variable i, sum: integer;
  Begin
    sum ←0 ;
    For i ←1 to (n div 2) do
        If (n mod i = 0) then
            sum ← sum+i ;
        Endif
    EndFor
   Som_Div← sum;
End;
Procedure P_Perfect (n: integer);
  Variable S: integer;
Begin
   S← Som_Div (n);
   If (S = n) then
       Write ("the number", n, " is perfect");
    Else
        Write ("the number", n, "is not perfect");
    Endif
End;
Function F_Perfect (n: integer): boolean;
```

```
Variable S: integer;
Begin
    S= Som_Div (n); If
    (S = n) then
        F_Perfect ←true;
     Else
         F_Perfect ←false;
    Endif
End;
```

```
Begin
Write ("Enter a number"); Read (n);
// Procedure call
P_Perfect (n); // here, "n" is an effective parameter.
// Function call
Bool ← F_Perfect (n); // here, "n" is an effective
parameter. If (Bool = true) then
   Write ("the number", n, " is perfect");
Else
    Write ("the number", n, "is not perfect");
End if
END
```

Solution of the exercise n°6

1.
```
Procedure Admitted_Students (T: [1..100] array of integer, N: integer)

Variables i: integer;
    Begin
        For  i←1 to N do
                If T[i] >= 10  then
                Write ("Student",  i,  " is admitted");
                End If
        End for
    End;
```

2.
```
Procedure Non_Admitted_Students (T: [1..100] array of integer, N: integer)
Var i: integer;
    Begin
        For  i←1 to N do
                If T[i] <10  then
                Write ("Student",  i,  " is not admitted"); End If    End for
                End ;
```

3. **Function Sum_Grades** (T: [1..100] array of integer, N: integer): integer

```
Begin
        If N = 0 then  Sum_Grades ← 0;
        Else Sum_Grades ← T[N] +  Sum_Grades (T, N-1); End if;
 End;
```

Procedure Class_Average (T: [1..100] array of integer, N: integer)

```
Variables S, M: integer;
Begin
S ← Sum_Grades (T, N);
M ← S/N;
Write (" Class_Average is: ", M);
End;
```

4. **Procedure Highest_Grade** (T: [1..100] array of integer, N: integer)

```
Var: i, max: integer;
  Begin
  max ← T[1];
     For  i←2 to N do
        If T[i] >= max then Max ← T[i];
        Endif      Endfor
  Write ("The Highest Grade is: ", max); End;
```

5. **Procedure** Lowest_Grade (T: [1..100] array of integer, N: integer)

```
Var: i, min: integer;
  Begin
  min ← T[1];
     For  i←2 to N do
        If T[i] <= min then
            min ← T[i];
        Endif        Endfor
  Write ("The Lowest Grade is: ", max);
  End;
```

6. **Algorithm Exercise_6;**

Var: T: array [1..100] of integer; N, i integer;

Copy the previous procedures into this section

Begin

Repeat

Write ("Enter the real dimension of the vectors"); Read(N);

Until (N>=1 and N<=100)

Write ("Enter the elements of the vector T");

For 1←i to N do

Read (T[i]);

Endfor;

Admitted_Students(T, N);

Non_Admitted_Students(T, N);

Class_Average(T, N);

Highest_Grade(T, N);

Lowest_Grade(T, N);

END.

7. **No, we can't;**

Justification: The variables "**max** and **min**", declared to identify the Highest Grade and the Lowest Grade, **are local variables** . Indeed, they cannot be used in the main algorithm to compare between **max** and **min** because the space of these local variables is freed when the execution of the procedure is finished.

Solution: there are three possible solutions

a. Use **max** and **min** as parameters and pass them by variable (var max, min).

b. Declare **max** and **min** as global variables.

c. Convert the two procedures: Highest_Grade and the Lowest_Grade into **functions**

8. This case requires the use of **linked lists**

Solution of the exercise n°7

1.

```
Function Product (V1, V2: [1..20] array of integer): integer
Variable P : integer ;
Begin
  P ← 0 ;
For  i←1 to N do
  P ← P +(V1[i] * V2[i]);
Endfor
  Product ←  P ;
End ;
```

2.

```
Procedure Sum (V1, V2: [1..20] array of integer)
Variables S1, S2: integer;
Begin
S1=0; S2=0;
For  i←1 to N do
  S1 ← S1 +V1 [i];
  S2 ← S2 +V2 [i];
Endfor
Write ("the sum of the elements of the vector V1 is",S1);
Write ("the sum of the elements of the vector V2 is",S2);
End;
```

3.

```
Algorithm exercise_7;
Function Product (V1, V2: [1..20] array of integer): integer
Variable P : integer ;
Begin
  P ← 0 ;
For  i←1 to N do
  P ← P +V1 [i] * V2 [i];
Endfor
  Product ←  P ;
End;
```

```
Procedure Sum (V1, V2: [1..20] array of integer);

Variables S1, S2: integer;
Begin
S1=0; S2=0;
For  i←1 to N do
  S1 ← S1 +V1 [i];
  S2 ← S2 +V2 [i];
Endfor
Write ("the sum of the elements of the vector V1 is",S1);
Write ("the sum of the elements of the vector V2 is",S2);
End;
Variables V1: array [1..20] of integer; V2: array [1..20] of integer; prod, N, i integer;

Begin
    Repeat
       Write ("Enter the real dimension of the vectors");
       Read(N);
   Until (N>=1 & N<=20)
  Write ("Enter the elements of the first vector V1");
   For  1←i to N do
      Read (V1[i]);
   Endfor
  Write ("Enter the elements of the second vector V");
   For  1←i to N do
      Read (V2[i]);
   Endfor
prod ← Product (V1, V2);
Write ("Scalar product of the two vectors V1 and V2 is", prod);
Sum (V1, V2);
END
```

4. **No, we can't;**

   Justification: The variables "S1 & S2", declared to calculate the sum of the elements of the array, **are local variables** . Indeed, they cannot be used in the main algorithm to compare between S1 and S2 because the space of these local variables is freed when the execution of the procedure is finished.

   Solution: there are three possible solutions

   a) Use S1 and S2 as parameters and pass them by variable (var S1, S2).

   b) Declare S1 and S2 as global variables.

   c) Use two different function the first one to calculate S1 and the second to calculate S2.

Part 3: Recursive sub-algorithms – Solutions-

Pedagogical objectives

Manipulate recursive sub-algorithms.

Solution of the exercise n°1

1st – call (8.5);

before call n=8, x=5
before call n=7, x=5
before call n=6, x=5
before call n=5, x=5
before call n=4, x=5
before call n=3, x=5
before call n=2, x=5
before call n=1, x=5

In the main algorithm
8*5=40

The sub algorithm makes the product of n*x. The instruction **(" after call n=", n, "x= ", x)**; in the product function is never executed.

Solution of the exercise n°2

a) Iterative function

```
Function quotient_division ( a ,b :integer):integer;
Begin
Variable S: integer; S ← 0;
While (a>=b) do
a←      a- b;
S←      S+ 1;
Endwhile
quotient _division  ← S ; End
```

b) Recursive function

```
Function quotient _ division_rec ( a ,b :integer):integer;
Begin
If (a<b) then
        quotient_division_rec ←0;
Else
       quotient_division_ rec ← quotient_division_rec (a- b,b )+1;
Endif
End;
```

Solution of the exercise n°3

```
Algorithm Calculation ;
Variables X,Y, P : integer ;
Function GCD (a, b : integer) : integer ;
Begin
If (a = b)  then // Particular case = Stopping criterion.
        GCD ← a
Else
  If (a > b) then  // General case
            GCD ← GCD (a-b, b);
  else   // General case
             GCD ← GCD (a, b-a);
     Endif
Endif
End;
Begin
Read (X ,Y);
If (Y≤ 0) Or (X ≤ 0) then
    Write (“Numbers are not strictly positive”);
Else
     P ← GCD (X, Y) ;
     Write (“THE PGCD of”, X, Y, ‘ =', P) ;
Endif
END
```

Solution of the exercise n°4

Type Tab = array [1..50] integer ;

a. Recursive function to calculate the sum of the elements of an array

```
Function Sum_tab ( var T: Tab, n: integer ) : integer ;
/* n is the number of elements in the array
Begin
  If (n=1) then
          Sum_tab ← T[1];
   else
          Sum_tab ← T[n] + Sum_tab (T, n-1);
    Endif
End;
```

b. Recursive procedure to reverse the elements of an array

```
Procedure inverse_ tab (var T:tab; d, f: integer);
Variable x: integer;
Begin
If (d<f) then
   x ← t[d ];
   t[d ]← t[f];
   t[f ]←x;
   inverse_tab (T,d +1,f-1);  // Recursive call
Endif
End;
```

Note

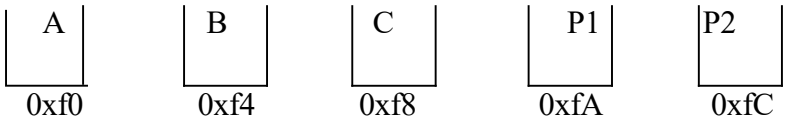
- ✓ **d**: start of the table to be reversed.
- ✓ **f:** end of the table to be reversed

Part 4: Pointers and linked lists – Solutions-

Pedagogical objective

- ✓ Understand the declaration, syntax and semantics of pointers, among others: indirect addressing, the content of a dynamic variable as well as arrays using pointers;
- ✓ Understand dynamic memory allocation and freeing.

Solution of the exercise n°1



	A	B	C	&A	&B	&C	P1	P2	*P1	*P2
A←1, B←2, C←3 P1←&A, P2←&B	1	2	3	0xf0	0xf4	0xf8	0xf0	0xf4	1	2
P2= &C	1	2	3	0xf0	0xf4	0xf8	0xf0	**0xf8**	1	**3**
***P1 = *P2**	**3**	2	3	0xf0	0xf4	0xf8	0xf0	0xf8	**3**	3
(*P2)++	3	2	**4**	0xf0	0xf4	0xf8	0xf0	0xf8	3	**4**
P1= P2	3	2	4	0xf0	0xf4	0xf8	**0xf8**	0xf8	**4**	4
P2= &B	3	2	4	0xf0	0xf4	0xf8	0xf8	**0xf4**	4	**2**
***P2 = *P1 - 2**P2**	3	**0**	4	0xf0	0xf4	0xf8	0xf8	0xf4	4	**0**
(*P2)--	3	**-1**	4	0xf0	0xf4	0xf8	0xf8	0xf4	4	**-1**
C = (P2 == &C)	3	-1	0	0xf0	0xf4	0xf8	0xf8	0xf4	**0**	-1
***P2= *P1 + A**	3	**3**	0	0xf0	0xf4	0xf8	0xf8	0xf4	0	**3**

Solution of the exercise n°2

int A[ ]= {12, 23, 34, 45, 56, 67, 78, 89, 90}; int *P; P = A;

a) *P+2 => the value 14

b) *(P+2) => the value 34

c) &P+1 => the address of the pointer behind the pointer P (rarely used)

d) &A[4]-3 => the address of element A[1]

e) A+3 => the address of element A[3]

f) &A[7]-P => the value (index) 7

g) P+(*P-10) => the address of element A[2]

h) *(P+*(P+8)-A[7]) => the value 23

Solution of the exercise n°3

```
Algorithm Exercise1 ;
Type Student  = Record
      N_Registration: integer;
      Last_name: string (20); First_name: string (20);
      Res: real;
EndRecord
Variables pi: * int; // pointer to an integer.
            pr:*real; // pointer to real.
             pc: * character; // pointer to character.
             pe: * Student; // pointer to a student record.
             i: int; r: real; c: character; e: Student;
Begin
pi ←&i ; pr ← &r ; pc ← &c ; pe ← &e ;
Write ("enter an integer"); Read (*pi);
Write ("enter a real"); Read (*pr);
Write ("enter a character"); Read (*pc);
Write ("enter student's information :");
Write ("enter the registration number"); Read ((*pe). N_Registration );
Write ("enter the name"); Read ((*pe). Last_name);
Write ("enter the first name"); Read ((*pe). First_name);
Write ("enter the result"); Read ((*pe).Res);
End
```

Solution of the exercise n°4

```
Algorithm invert_Array;
Variables tab = Array [1..50] integer;
          N, HELP: integer;
          P1: * integer; P2: *integer; /*helper pointers*/
Begin
/* Data entry */
  Repeat
 Write ("enter the dimension of the array: ");
Read (N);
Until (N>=1) and (N<=50);
For (P1=tab) to (tab+N) do
// Equivalent to (For i=1 to N do)
// Just to show indirect (pointer) addressing manipulation
  Read (*P1);
// we did not use the address operator &, however, we used the pointer P1 which contains
an address to tab[i]; (indirect addressing)
  EndFor
// Display the array
For (P1=tab) o (tab+N) do
Write (*P1);
  EndFor
//Reverse array elements
P1 ← tab; P2 ← tab+N;
While (P1<P2) do
HELP ← *P1 ;
*P1 ← *P2;
*P2 ← HELP;
P1 ← P1+1;
P2 ← P2-1;
  EndWhile
// Display result
```

```
For (P1=tab) to (tab+N) Do
Write (*P1);
EndFor
END
```

Solution of the exercise n°5

1. Data structures

```
Type Date=Record
    dd: integer;
    mm: integer;
    aa: integer;
EndRecord
Type Employer= Record
 Last_name: string(20);
 First_name: string (20);
 Date_B: Date;
 Nb_E: integer;
Qualif: string (30);
EndRecord
Employer_list=Array [1…300] of * Employer;
```

2. Algorithm to enter data

```
Algorithm Employers;
Type Date=Record
    dd: integer;
    mm: integer;
    aa: integer;
EndRecord
Type Employer= Record
 Last_name: string(20);
 First_name: string (20);
 Date_B: Date;
 Nb_E: integer;
 Qualif: string (30);
```

```
EndRecord
Employer_list=Array [1..300] of *Employer; //Array of pointers to employers.
Variables i: integer, Bool: boolean; Char: character; L: Employer_list;
Begin
i ← 1 ; Bool ← true;
While (Bool = true) and (i<=300) do
 Allocate (L[i]);
// L[i] is a pointer to an employer, it is the memory allocation of a dynamic employer type
variable whose pointer is L[ i ].
Write (“"Employer n°", i);
Write ("Give the last name: ");
Read ((*L[ i ]).Last_name);
Write ("Give the first name: ");
Read ((*L[ i ]).First_name);
Write ("Give the date of birth: ");
Write ("Give the day: "); Read ((*L[ i ]). Date_B.dd);
Write ("Give the month: ");Read ((*L[ i ]).Date_B.mm);
Write ("Give the year: "); Read ((*L[ i ]). Date_B.yy);
Write ("Give the number of children: "); Read ((*L[ i ]).Nb_E);
Write ("Give the qualification: "); Read ((*L[ i ]).Qualif);
Repeat
Write (“are there other employers Y/yes/N/no?”);
Read (Char);
Until (Char = ‘Y’) or (Char = ‘N’)
If (Char = 'Y') then
    i ← i +1;
else
    Bool ← false;
Endwhile
END.
```

3. Algorithm for sorting employers

```
Algorithm Sort_Employers;
Employer_list=Array [1..300] of *Employer; //Array of pointers to employers.
Variables N, Pc: integer;
//  N number of employers entered in the previous algorithm =i
        L: Employer_list; P1, P2: *Employer; Aux: Employer;
Begin
For (P1=L)  to (L+N-1) do
    For (P2=P1+ 1) to (L+N) do
      If ((*P1). Date_B .yy) < ((*P2). Date_B .yy) then
        Aux ← (*P1); (*P1)
        ← (*P2);
        (*P2) ← Aux ;
       else
        If ((*P1).Date_B .yy)= ((*P2).Date_B .yy) then
            If ((*P1). Date_B .mm) < ((*P2). Date_B .mm) then
              Aux ← (*P1);
              (*P1) ← (*P2);
              (*P2) ← Aux ;
             else
             If ((*P1). Date_B .mm)= ((*P2). Date_B .mm) then
                 If ((*P1). Date_B .dd) < ((*P2). Date_B .dd) then
                     Aux ← (*P1);
                     (*P1) ← (*P2);
                     (*P2) ← Aux ;
                  End if
               End if
            End if
         End if
       End if
  EndFor
EndFor
// Number of employers selected
```

```
Pc ← (N DIV 10); P1 ← L;
For (P1 = L+Pc)  to (L+N) Do
Free (P1) ;
EndFor
End
```

Solution of the exercise n°6

```
Algorithm Exercise_2 ;

Variables T : array [1..100] integer ; P, P1:*integer ;

Begin

For P=T to (T+100) do (1.5 pt)

Read (*P);

Endfor

Allocate (P1); (1 pt)

*P1←1;

For P=T to (T+100) do

If ((*P) mod 3 = 0) then   (2 pts)

Write("The index is",P-T);

*P1←*P1 * *P;

End If

Endfor

End
```

Solution of the exercise n°7

```
Algorithm Find_indexes_odd ;
Variables T : array [1..100] integer ; P, P1:*integer ;
Begin
For P=T to (T+100) do
    Read (*P);
Endfor
Allocate (P1);
*P1←0;
For P=T to (T+50) do
  If ((*P) mod 2 != 0) then
      Write("index of the odd number is",P-T);
      *P1←*P1+*P;
  Endif
End
```

Solution of the exercise n°8

```
Type Element = Record
   Info: real;
   Next: *Element;
End;

Type List: *Element;
End;

// Declarations
Variables head, P, L: List;
      flag: boolean;
      add_element: char;
      k, i: integer;
      val: real;
```

```
Begin

// Create the list ****************************************************
head ← Nil;
flag ← true;

While (flag = true) do
   Allocate (P); // allocate memory space
   Write ("Enter the value of the element");
   Read ((*P).Info);
   (*P).Next ← head;
   head ← P;

   // Confirm whether to add another element
   Write ("Are there more elements? Y/N");
   Read (add_element);
   If (add_element = "N") Then
      flag ← false;
   EndIf
EndWhile

// Display list elements **********************************************
P ← head; // point to the first element

While (P ≠ Nil) do // traverse from first to last element
   Write ((*P).Info); // display the element
   P ← (*P).Next; // move to next
EndWhile

// Insert an element at the K-th position *****************************
Write ("Enter the position");
Read (k);
Write ("Enter the value to insert");
Read (val);
```

```
L ← head;

If (k = 0) Then // insertion at the head
   Allocate (P);
   (*P).Info ← val;
   (*P).Next ← head;
   head ← P;
Else
   i ← 0;
   While (i < k) and (L ≠ Nil) do
      L ← (*L).Next;
      i ← i + 1;
   EndWhile

   If (i = k) and (L ≠ Nil) Then
      Allocate (P);
      (*P).Info ← val;
      (*P).Next ← (*L).Next;
      (*L).Next ← P;
   Else
      Write ("Position does not exist");
   EndIf
EndIf

// Delete a given element from the list ********************************
If (head ≠ Nil) then
   If ((*head).Info = val) then // delete first element
      P ← head;
      head ← (*P).Next;
      Free (P); // free memory
   Else
      flag ← false;
      Prev ← head;
      P ← (*head).Next;
```

```
      While (P ≠ Nil) and (flag = false) do
         If ((*P).Info = val) Then
            flag ← true;
         Else
            Prev ← P;
            P ← (*P).Next;
         EndIf
      EndWhile

      If (flag = true) Then
         (*Prev).Next ← (*P).Next;
         Free (P);
      Else
         Write ("Value does not exist");
      EndIf
   EndIf
Else
   Write ("The list is empty");
EndIf
End
```

Section II

« Practical Work »

Section 2: Practical work -Exercises-

Part 1 : Records

Pedagogical objectives

- ✓ Handle custom types in C language, in terms of declaration and processing.

Exercise n°1

Write a C program that defines three structures Point (fields: X and Y), Circle (Fields: X, Y, R) and Rectangle (Log, Lag). The program must read and display the respective fields of the Point, Circle and Rectangle structure type variables (point p1, circles c1, rectangles R1).

Exercise n°2

An industrial carpentry manages a stock of wooden panels. Each panel has a width, length and thickness in millimeters, as well as the type of wood which can be pine (code 0), oak (code 1) or beech (code 2).

1. Define a panel record containing all the information relating to a wooden panel;
2. Write a C program that allows you to

- enter and display a wood panel (numeric entry for the type of wood (eg: 0), display in characters of the type of wood (eg: pine);
- Calculate the volume of a panel ((thickness*width*length)/10^{9}).

Exercise n° 3

Consider the following structures (records):

- Date defined by the three fields: day, month and year;
- Address defined by the fields: Number, Street, Municipality, Wilaya, Postal Code;
- Employee defined by the fields: LastName, FirstName, Residence, DateBirth

Write a C program that allows you to enter employee information from the keyboard and check:

→ whether an employee was born before a given year;

→ if an employee resides in a given wilaya.

Part 2: Sub-algorithms

Pedagogical objectives

- ✓ Handle procedures & functions in C;

Exercise n°1

- Write function **int power(int x, int y)** which takes two integer parameters x and y and returns x^y. Also give an example of calling this function by writing a main function in which you display the calculation result.
- Same question, but this time we want to pass the result by variable. The function header therefore becomes **void power(int x, int y, int *r).**
- Write a **division function** that performs integer division and returns both the quotient and remainder by variable, and the main function that calls it.

Exercise n°2

- Write a function ***grab*** that allows you to grab an array of integers.
- Write a function ***display*** that displays the elements of the array.
- Write a function ***calculate_average,*** which allows you to calculate the average of the elements of an array.
- Write a function ***find_min_max,*** which allows you to find the minimum and maximum between the elements of an array.
- Write the main program.

Exercise n°3

F is a numerical function defined by **F(X) = X ^{3} -2X+1** . We want to construct an array of values of this function. The user enters the number N of values as well as the values of X. Write a program that matches this processing.

Example:

Enter an integer between 1 and 100: **9**

Enter 9 real numbers: |-4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 |

X	-4.0	-3.0	-2.0	-1.0	0.0	1.0	2.0	3.0	4.0
F(X)	- 55.0	- 20.0	-3.0	2.0	1.0	0.0	5.0	22.0	57.0

NB : the solution must include subprograms.

Part 3 : Recursive sub-algorithms

Pedagogical objectives

- ✓ Handle recursive procedures & functions in C;

Exercise n°1

- Write in C a recursive function allowing you to calculate X^n for **X** real and **n** natural integer.
- Write a recursive function in C to calculate X! for integer X.

Exercise n°2

We consider the mathematical Ackermann function F of two real variables x and y, defined as follows:

F(x, y) = y+1 if x=0 ;

F(x, y) = F(x-1, 1) if x > 0 and y = 0 ;

F(x, y) = F(x-1, F(x, y-1)) if x And y are different of 0.

Write a recursive program in C which requests the two values x and y and displays the value of F (x, y).

Exercise n°3

We consider the mathematical Fibonacci function F defined as follows:

→ F(0*) = 0; F(1) = 1*

→ *F(n) = F (n−1) + F (n−2) For n > 1*

Write a recursive program in C that asks for an integer value n and displays the value of F(n).

Part 4: Pointers and linked lists

Pedagogical objective

✓ Manipulate pointers in C, pointer arithmetic and addressing the elements of an array;

✓ Apply the “malloc” and “free” functions for dynamic memory allocation and release.

Exercise n°1

Write a C program that uses the notion of pointer to read two integers and calculate their sum.

Exercise n°2

sizeof (T type) function returns the number of bytes necessary to encode, in memory, a dynamic variable of type T.

1. Using this function, write a program that determines the number of bytes occupied by:
   → A character.
   → An integer (short and long).
   → A boolean;
   → A real.
   → The string “1234567”.
2. What do you notice about the size of the string?

Exercise n°3

Consider the following program:

```
main( ){
int x,y; char a,b ; float f ;
int *pi1, *pi2;
char *pc1, *pc2 ;
float *pf1, *pf2;
x=10; y = 9; a='K'; b='S'; f=1.5;
pi1=&x; pi2=& y; pc1=&a; pc2=&b ; pf1=&f; pf2=pf1 ;
printf(" Addresses in hexadecimal: pi1 = %X pi2 = %X \n", pi1, pi2);
printf(" Addresses in decimal: pi1 = %d pi2 = %d \n", pi1, pi2);
printf ("pc1= %d pc2= %d \n", pc1, pc2);
printf ("pf1= %d pf2= %d \n", pf1, pf2);
}
```

1. Write a C program, allowing you to display the result of the following operations:
    a) Increment of pointers pi1, pc1 and pf1;
    b) Decrement of pointers pi1, pc1 and pf1;
    c) Difference between: pi1 and pi2, pc1 and pc2, pf1 and pf2;
    d) Addition of: pi1 and pi2, pc1 and pc2, pf1 and pf2.
2. Which of the above operations are not allowed for pointers.

Exercise n°4

Using the pointer formalism, write a C program that reads two arrays A and B and their dimensions N and M, then adds the elements of B to the end of A.

Exercise n°5

Write a C program that reads an integer X and an array A of type integer and eliminates all occurrences of X in A by packing the remaining elements. The program will use a pointer P1.

Exercise n° 6

Write a C program using pointers that constructs a unitary matrix with dimension N (given by the user).

NB: a unitary matrix is a square matrix whose diagonal elements are equal to 1 and the other elements are zero.

Section 2: Practical work -Solutions-

Part 1 : Records – Solutions-

Pedagogical objectives

- ✓ Handle custom types in C language, in terms of declaration and processing.

Solution of the exercise n°1

```
#include <stdio.h>
typedef struct Point
{ float x,y;
} Point;
typedef struct Circle
{ float xc;
float yc ;
float radius;
} Circle ;
typedef struct Rectangle
{ float Log, Lag ;
} Rectangle;
main ( )
{
Point p1;
Circle c1;
Rectangle R1;
```

```
printf ("Enter the fields of point p1: \n");
scanf ( "%f%f",&p1.x,&p1.y) ;
printf (" Show the fields of point p1: [ %f , %f ]\n", p1.x, p1.y);
printf ( " \n") ;
printf (" Enter the fields of circle c1: \n");
scanf ( " %f %f %f",&c1.xc,&c1.yc,&c1.radius) ;
printf (" Show the fields of circle c1: [%f, %f, %f]\n", c1.xc,
c1.yc,c1.radius);
printf ( " \n") ;
printf (" Enter the fields of rectangle R1: \n");
scanf ( "%f%f",&R1.Log,&R1.Lag) ;
printf ("Show the fields of rectangle R1: [%f, %f]\n", R1.Log,
R1.Lag);
printf ( " \n") ;}
```

Solution of the exercise n°2

```
#include <stdio.h>
#include <math.h>
typedef struct panel{
float width , length, thickness ;
int typeWood ; }panel ;
main
( ){ panel
p; float Vol;
printf ( "\n Enter panel width: ");
scanf ( "%f", &p.width );
printf ( "\n Enter panel length: ");
scanf ( "%f",&p.length );
printf ( "\n Enter panel thickness : ");
scanf ( "%f",&p.thickness );
printf ( "\n give the type of wood: ");
scanf ( "%d", &p. typeWood);
printf ( "\n panel width: %.2f", p.width );
printf ( "\n panel length: %.2f", p.length );
printf ( "\n panel thickness : %.2f", p.thickness );
printf ( "\n wood type: ");
switch( p.typeWood){
```

```
case 0: printf ("pin\n");
 break;
case 1: printf (" oak \n");
 break ;
case 2: printf (" beech \n");
 break ;
default : printf ("unknown\n");
 break ;}
Vol = ( p.width * p.length * p.thickness )/pow(10,9);
printf ( "\n panel volume is: %f", Vol);
}
```

Solution of the exercise n°3

```
#include <stdio.h>
#include <string.h>
typedef struct Date {
 int day, month, year ;
} Date;
typedef struct
 Address{ int
 Number,PostalCode ;
 char Street[20], Commune[20], Wilaya[20];
} Address;
typedef struct NP{
char Lastname[ 20], Firstname [20];
} NP;
typedef struct Employee
{ NP FirstName;
Date BirthDate;
Address Residence;
} Employee ;
main( ){ Empl
oyee E; int
ann ;
char w []="ALGER";
printf ( "\n Employee name : ") ;
 scanf ( "%s",&E.FirstName.Lastname);
```

```
 scanf ( "%s", &E.FirstName.Firstname);
 printf ( "\n Date of birth:\n");
 printf ( "Day: ");
 scanf ( "%d", &E.BirthDate.day);
 printf ( "\n Month:");
 scanf ( "%d", &E.BirthDate.month);
 printf ( "\n Year : ");
 scanf ( "%d", &E.BirthDate.year);
 printf ( "\n Employee residence :\n") ;
 printf (" Number :");
 scanf ( "%d", &E.Residence.Number);
 printf ( "\n Street : ");
 scanf ( "%s",&E.Residence.Street);
 printf ( "\ nCommon : ");
 scanf ( "%s", &E.Residence.Commune);
 printf ( "\n Wilaya : ");
 scanf ( "%s",&E.Residence.Wilaya );
 printf ( "\ n PostalCode : ");
 scanf ( "%d", &E.Residence.PostalCode );
 printf ( "\n Give year : ");
 scanf ( "%d", &ann );
 if( E.BirthDate.year < ann ){
 printf ( "\n employee was born before year %d \n", ann ); }
 else {
 printf ( "\n employee was born after year %d \n", ann );}
 printf ( "Enter Wilaya \n");
 scanf ( "%s",&w);
 if( strcmp
 (E.Residence.Wilaya,w )==0){ printf
 ( "Yes\n");}
 else {
 printf ( "No\n");
}
}
```

Part 2: Sub-algorithms – Solutions-

Pedagogical objectives

- ✓ Handle procedures & functions in C;

Solution of the exercise n°1

1.

```
#include<stdio.h>
int power( int x, int y)
{
int result =1,i;
for( i =0;i< y;i ++)
result = result *x;
return result ;
}
int main()
{
int x=2 ,y =3,r;
r=power( x ,y );
printf( "%d power %d = %d\n", x,y,r );}
```

2.

```
#include <stdio.h>
void power(int x,int y,int*r)
{
int i ;
*r=1;
for( i =0;i< y;i ++)
*r = *r*x;
}
int main()
{
int x=4 ,y =5,r;
power( x ,y ,&r );
printf ("%d power %d = %d\n" , x,y,r );
}
```

3.

```
#include <stdio.h>
void division( int a, int b, int * q, int * r)
{
*r = a % b;
*q = a / b;
}
int main()
{
int a=17, b=3, q, r;
division( a,b,&q,&r);
printf ( "%d=%d*%d+%d\n", a,b,q,r );
}
```

Solution of the exercise n°2

```
#include <stdio.h>
/*function prototypes*/
void grab(int [],int);
```

```
void display(int [],int);
float calculate_average (int[],int);
void find_min_max ( int [], int , int *, int * );
main()
{
int nb_val ;
int min, max, table[100];
float average;
printf ( "Number of elements:"); scanf ("%d", & nb_val);
grab(table, nb_val);
display( table, nb_val );
average = calculate_average ( table , nb_val );
printf ( "Average = %f\n", average);
find_min_max(table, nb_val , &min, &max );
printf ("Min = %d Max = %d\n", min, max );
}
/*  entry of array elements */
void grab(int tab[], int nb)
{
int i ;
printf ( "\n");
for ( i =0; i < nb ; i ++)
{
printf ("Value of tab[%d] = ", i);
scanf ("%d", &tab[i] );
}
}
/* Displaying array elements */
void display ( int tab[], int nb )
{
int i ;
printf ( "\n");
for ( i =0; i < nb ; i ++)
{
```

```
printf ("tab[%d] = %d\n", i , tab[ i ]);
}
printf ( "\n");
}
/* Calculating the average */
float calculate_average ( int tab[ ], int nb)
{
float average;
int sum;
int i;
sum = 0;
for ( i =0; i < nb ; i ++)
{
sum = sum + tab[ i ];
}
average = sum /( double (nb));
return average;
}
/* the min and max of the  array */
void find_min_max ( int tab [], int nb, int *pmin , int *pmax )
{
int val_min , val_max ;
int i ;
val_min = tab[0];
val_max = tab[0];
for ( i =0; i < nb ; i ++)
{
if (tab[i] < val_min)
{
val_min = tab[i];
}
else if (tab[i] > val_max )
{
val_max = tab[i];
```

```
}
}
*pmin = val_min ;
*pmax = val_max ;
}
```

Solution of the exercise n°3

```
#include <stdio.h>
/* Prototypes of functions n */
void ACQUIRE( int *);
float F( float);
void READ_VECTOR (float[], int N);
void CALCULATE_VALUES(float [], float [], int N);
void DISPLAY_TABLE ( float [], float [], int N);
main()
{
/* Declaration of global variables */
float X[100]; /* values of X */
float V[100]; /* values of F(X) */
int Nb;
/* Function calls */
ACQUIRE(&Nb); /* 1 <= Number <= 100 */
READ_VECTOR(X, Nb);
CALCULATE_VALUES(X, V, Nb);
DISPLAY_TABLE(X, V, Nb);
}

/* Definition of the ACQUIRE function */
void ACQUIRE( int *N)
{
do
{
printf ( "Enter an integer between 1 and 100: ");
scanf ("%d",N);
```

```
 } while ((*N<1)||(*N>100));
}
/* Definition of the READ_VECTOR function */
void READ_VECTOR ( float T[], int N)
{
/* Fills an array T of order N with real numbers entered from
the keyboard */
/* Declaration of local variables */
int I;
/* Fill the array */
printf ( "Enter %d real numbers:\n", N);
for (I=0; I<N; I++)
 scanf ( "%f", &T[I]);
}
/* Definition of function F */
float F( float X)
{
/* Returns the numerical value of the polynomial defined by F(X)
= X^3-2X+1 */
return (X*X*X - 2*X + 1);
}
/* Definition of the CALCULATE_VALUES function */
void CALCULATE_VALUES( float X[], float V[], int N)
{
/* Declaration of local variables */
int I;
for (I=0; I<N; I++)
 {
V[ I] = F(X[I]);
}}
/* Definition of the DISPLAY_TABLE function */
void DISPLAY_TABLE(float X[], float V[], int N)
{
/* Declaration of local variables */
```

```
int I;
/* Show table */
printf ( "\nX: ");
for (I=0; I<N; I++)
printf ( "%f", X[I]);
printf ( "\nF(X): ");
for (I=0; I<N; I++)
printf ( "%f", V[I]);
printf ( "\n");}
```

Part 3 : Recursive sub-algorithms -Solutions-

Pedagogical objectives

- ✓ Handle recursive procedures & functions in C;

Solution of the exercise n°1

```
#include <stdio.h>
int factorial (int);
int power(int,int);
main()
{
    int x, n;
    printf ("Enter an integer: x = "); scanf ("%d",&x );
    printf ( "Enter an integer: n = "); scanf ("%d",&n );
    printf ( "%d",factorial (n));
    printf ( "\n %d ",power( x,n ));
}
int factorial(int m)
{
if (m==0)
{
return 1;
}
else
{
 return (m* factorial(m-1));
}
}
int power( int r, int k)
```

```
{
if (k==0)
{
return 1;
}
else
{
return (r*power(r,k-1));
}
}
```

Solution of the exercise n°2

```
#include <stdio.h>
int Ackermann(int,int);
main()
{
int x, y;
printf ( "Enter an integer: x = "); scanf ("%d",&x );
printf ( "Enter an integer: y = "); scanf ("%d",&y );
printf ( "result %d", Ackermann( x,y ));
}
int Ackermann( int M,int N)
{ if( M == 0)
 return N+1;
else
if(N==0 && M>0)
     return (Ackermann(M-1,1));
else
    return (Ackermann(M-1,(Ackermann(M, N-1))));
}
```

Solution of the exercise n°3

```
#include <stdio.h>
int Fibonacci (int);
main()
{
 int x;
 printf ( "Enter an integer: x = ");
scanf("%d",&x);
printf("result %d", Fibonacci(x));
}
int Fibonacci ( int N)
{ if( N == 0) return 0; else
```

```
if( N == 1) return 1;
else return ( Fibonacci (N-1)+ Fibonacci (N-2));
}
}
```

Part 4: Pointers and linked lists – Solutions-

Pedagogical objective

- ✓ Manipulate pointers in C, pointer arithmetic and addressing the elements of an array;
- ✓ Apply the “malloc” and “free” functions for dynamic memory allocation and release.

Solution of the exercise n°1

```
#include<stdio.h>
main()
{
    int a,b,s;
    int *pa,*pb,*ps;
    pa = &a;
    pb = &b;
    ps = &s;
    printf("Give two integers:\n");
    scanf("%d%d",pa,pb);
    *ps = *pa + *pb;
    printf("The sum is %d.\n",s);
}
```

Solution of the exercise n°2

1.

```
#include <stdio.h>
main()
{
printf("The size of a character is %d byte(s)", sizeof(char));
printf("\n"); printf("The size of an integer short is %d
byte(s)", sizeof(short));
printf("\n"); printf("The size of a long integer is %d
byte(s)", sizeof(long));
printf("\n"); printf("The size of a real number is %d byte(s)",
sizeof(float));
printf("\n"); printf(" The size of the character string %s is
%d byte (s)","1234567",sizeof("1234567"));}
```

2. For the string: the number of characters in the string is 7, however the number of bytes to reserve is 8. The last byte is reserved for *the end of string character "\0".*

Solution of the exercise n°3

1.

```
#include <stdio.h>
main(){
int x,y; char a,b ; float f ;
int *pi1, *pi2;
char *pc1, *pc2 ;
float *pf1, *pf2;
x=10; y = 9; a='K'; b='S'; f=1.5;
pi1=& x; pi2=& y; pc1=& a; pc2=& b ;   pf1=& f; pf2=pf1 ;
printf(" Addresses in hexadecimal: pi1 = %X pi2 = %X \n", pi1,
pi2);
printf(" Addresses in decimal: pi1 = %d pi2 = %d \n", pi1, pi2);
printf("pc1= %d pc2= %d \n", pc1, pc2);
printf("pf1= %d pf2= %d \n", pf1, pf2);
// Increment
printf(" pi1 +1 = %d \n", pi1++);
```

```
printf(" pi2 +1 = %d \n", pi2++);
printf(" pc1 +1 = %d \n", pc1++);
printf(" pc2 +1 = %d \n", pc2++);
printf(" pif +1 = %d \n", pf1++);
printf(" pf2 +1 = %d \n", pf2++);
// Decrement
printf(" pi1 -1 = %d \n", pi1--);
printf(" pi2 -1 = %d \n", pi2--);
printf(" pc1 -1 = %d \n", pc1--);
printf(" pc2 -1 = %d \n", pc2--);
printf(" pif -1 = %d \n", pf1--);
printf(" pf2 -1 = %d \n", pf2--);
// Difference
printf(" pi1 - pi2 = %d \n", pi1 - pi2);
printf("pc1 - pc2= %d \n", pc1 - pc2);
printf("pf1 - pf2= %d \n", pf1 - pf2);
}
```

2. *Sum or addition is not allowed*

Adding an integer to a pointer. The result is a pointer of the same type as the starting pointer;

Subtracting an integer from a pointer. The result is a pointer of the same type as the starting pointer;

The difference of two pointers both pointing to objects of the same type. The result is an integer.

The sum of two pointers is not allowed.

Solution of the exercise n°4

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int *A,*B; int n,m,i;
printf("Give the dimension of A:");
scanf("%d",&n);
printf("Give the dimension of B:");
scanf("%d",&m);
```

```
A=(int*)malloc(n*sizeof(int));
for (i=0;i<n;i++)
{
printf("A[%d]=",i);
scanf("%d",&A[i]);
}
B=(int*)malloc(m*sizeof(int));
for (i=0;i<m;i++)
{
printf("B[%d]=",i);
scanf("%d",&B[i]);
}
A=(int*)realloc(A,(n+m)*sizeof(int));
for (i=n;i<n+m;i++)
A[i]=B[i-n];
printf("The array after fusion \n");
for (i=0;i<n+m;i++)
printf ("A[%d]=%d \n",i,A[i]);
free (A);
free (B);
return 0;
}
```

Solution of the exercise n°5

```
#include <stdio.h>
#include <stdlib.h>
main()
{
/* Declaration */
int *A;
int N,i;
int X;
int k=0;
int *P1;
```

```
printf("Give the dimension of the array : ");
scanf("%d", &N );
A=(int*)malloc(N*sizeof(int));
for (i=0; i<N; i++)
{
printf("A[%d] : ", i);
scanf("%d", &*(A+i));
}
printf("Introduce element X to eliminate from the array : ");
scanf("%d", &X );
P1=A;
for (i=0; i<N; i++)
if (A[i] != X)
{
P1[k] = A[i];
k++;
}
/* Display the result */
for (i=0; i<k; i++)
printf("%d ", A[i]);
printf("\n");
free (A);
}
```

Solution of the exercise n°6

```
#include <stdio.h>
#include <stdlib.h>
main()
{
/* Declaration */
int **U; /* unitary matrix  */
int N; /* dimension  */
int I, J;
```

```
printf("Dimension of the square matrix: ");
scanf("%d", &N);
/* Square matrix memory reservation */
U = (int**) malloc (N * sizeof(int*));
for (I = 0; I < N; I++)
U[I] = (int*) malloc (N * sizeof(int));
/* Construction of the unitary square matrix */
for (I=0; I<N; I++)
for (J=0; J<N; J++) if (I==J)
U[I][J]=1;
else
U[I][J]=0;
/* Display the result */
printf("Unitary dimension matrix %d :\n", N);
for (I=0; I<N; I++)
{
for (J=0; J<N; J++)
printf("%7d", U[I][J]); printf("\n");
}
for (I=0; I<N; I++)
free(U[I]); free(U);
}
```

Bibliography

1. Neumann, Günter. "Programming languages in artificial intelligence." Encyclopedia of Information Systems (2002): 31-45.
2. Barnett, Granville, and Luca Del Tongo. "Data structures and algorithms: annotated reference with examples." Pierrefonds, CA, NETSlackers (2008).
3. Karumanchi, Narasimha. Data Structures and Algorithmic Thinking with Python. CareerMonk Publications, 2016.
4. Canning, John, Alan J. Broder, and Robert W. Lafore. Data Structures & Algorithms in Python. Addison-Wesley, 2023.
5. Pierce, Benjamin C., ed. Advanced topics in types and programming languages. MIT press, 2024.
6. LANGLOIS, Ph. (2013). Programmation en C – Exercices. Université de Perpignan Via Domitia
7. Helaoui, M. (2011). Travaux Dirigés : Algorithmique et Structure de Données. 10.13140/2.1.3800.9601.
8. Helaoui, M. (2019). ASDI 2019 2020 v6.pdf. https://www.researchgate.net/publication/337873900_ASDI_2019_2020_v6pdf