PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION

AND SCIENTIFIC RESEARCH

UNIVERSITY OF MOSTEFA BENBOULAID BATNA 2

FACULTY OF TECHNOLOGY

DEPARTMENT OF COMMON CORE IN SCIENCE AND TECHNOLOGY

INFORMATICS 2 COURSE

First Year (S2) Common Core in Science and Technology

Dr. Salhi Hicham

Associate Professor, Class A

Chapter III : The Recordings and files

General introduction

Computer science is a discipline that encompasses art, technique, and science. It utilizes the computer as a tool to define algorithms, solve complex problems, and extract useful knowledge from disorganized data. It offers a creative, practical, and scientific approach to addressing various fields and challenges.

The content of this educational document follows a classical approach. In the first chapter, we focus on presenting two parts. The first part deals with one-dimensional arrays, covering memory representation and associated operations. The second part concerns two-dimensional arrays, encompassing memory representation and specific operations related to these arrays.

In the second chapter of the document, the focus shifts to functions, addressing different types of functions, function declaration, and function calls. It also explores procedures, including concepts of global and local variables, simple procedures, and procedures with arguments.

In the third chapter of the document, the emphasis is on records and files, covering the following points :

- Structure of heterogeneous data and a record.
- Manipulation of record structures and an introduction to files.

Chapter I : The Indexed Variables

1.1. One-Dimensional Arrays:

1.1.1. Definition of an Array:

A one-dimensional array is a data structure in computer science that allows for storing and organizing elements sequentially. It is a linear collection where elements are arranged one after another, forming a continuous sequence.

1.1.2. Main Characteristics of a One-Dimensional Array:

Sequential Storage: Elements are stored sequentially in memory. Each element is placed one after another, allowing for easy traversal from the first to the last element.

Fixed Size: Once defined, the size of the array remains constant throughout its lifetime. This means the number of elements it can hold is predetermined and cannot be changed dynamically during execution.

Direct Access: Elements are accessed using their index position. Each element in the array has a unique index starting from 0 (for the first element) to n−1n-1n−1 (for the nth element, where nnn is the size of the array).

Homogeneous Data Type: All elements in a one-dimensional array must be of the same data type (e.g., integer, float, character). This ensures that each element occupies the same amount of memory and simplifies memory management and access operations.

Contiguous Memory Allocation: Elements of the array are stored in contiguous memory locations. This means that the memory addresses of consecutive elements in the array are adjacent to each other, which allows for efficient memory access and traversal.

1.1.3. Declaration of a one-dimensional array :

In Pascal, the declaration of a one-dimensional array follows this syntax:

Var name: array [<min index> **..** <max index>] of <component type>;

Exemple:

Write a Pascal program that declares a one-dimensional array, initializes its elements, and then displays some values on the screen.

I=1	I=2	I=3	I=4	I=5	I=6
20	25	40	55	69	76

```
program Table;
var  tab: array[1..6] of Integer;
begin
  tabl [1] := 20; tabl [2] := 25;
```

```
  tabl [3] := 40; tabl [4] := 55;
  tabl [5] := 69;  tabl [6] := 76;
  writeln(' The first element of the array is: ', tabl[1]);
  writeln(The second element of the array is: ', tabl[3]);
end.
```

1.1.4. Creation and writing of a one-dimensional array:

In Pascal, creating and writing to a one-dimensional array is done as follows:

I=1	I=2	I=3	I=4	I=5	I=6
20	25	40	55	69	76

```
program CreationTable;
var          T: array[1..6] of Integer;
             i: Integer;
begin
  for i := 1 to 6 do
    begin
      writeln ('T=' );
       Read (T[i]);
    end;
End.
```

1.1.5. Displaying a one-dimensional array:

To display a one-dimensional array in Pascal, you can use a for loop to iterate through the elements of the array and print them.

Exemple

Write a program to create an Integer array T1 of size 6 through user input. Additionally, create another array T2 of the same size, where each element T2[i] is three times T1[i].

Solution:

This Pascal program accomplishes the following steps:

- Asks the user to input values for array T1.
- Creates array T2 by tripling each corresponding element of T1.
- Displays the values of T2.

```
program CreationTable;
var      T1, T2: array[1..6] of Integer;
          i: Integer;
```

```
begin
  writeln(' Input for array T1:');
  for i := 1 to 6 do
     begin
        write(' Enter the value for T1[' , i,  '] : ' );
         readln(T1[i]);
      end;
  writeln('Création  T2 :');
  for i := 1 to 6 do
  begin
    T2[i] := 3 * T1[i];
    writeln('T2[' , i,  '] = ' ,  T2[i]);
  end;
end.
```

1.1.6. Operations on One-Dimensional Arrays:

Operations on arrays are commonly used in programming to perform mathematical calculations on the elements of an array, such as computing the sum and average of the elements, finding the minimum and maximum values, and testing if an element belongs to the array.

1.6.1. Calculation of Sum and Average of Elements:

In this program, the user enters the size of the array. Then, the program dynamically allocates the array based on this size. The elements of the array are inputted, and subsequently, the sum and average are computed. Finally, the program displays the results.

```
program Sum_av;
var      T: array[1..6] of Integer;
 var       som,i: Integer;
           moy: Real;
Begin
  for i := 1 to 6 do
     begin
        write('Enter the value for T[', i, '] : ');
         readln(T[i]);
      end;
som := 0;
for i := 1 to 6 do
begin
```

```
  som := som + T [i];
end;
writeln('som= ', som);
moy := som / 6;
writeln(' moy= ', moy);
end.
```

1.1.6.2. Calculation of Minimum and Maximum:

In this program, the user first enters the size of the array. Then, the program dynamically allocates an array based on the specified size. The elements of the array are inputted, and afterward, the program searches for the minimum and maximum values.

```
program max_min;
var       T: array[1..6] of Integer;
 var       min, max,i: Integer;
Begin
  for i := 1 to 6 do
      begin
         write('Enter the value for T[', i, '] : ');
          readln(T[i]);
       end;
min := T[1];
max := T[1];
for i := 2 to 6 do
begin
  if T[i] < min then
    min := T[i];
  if T[i] > max then
    max := T[i];
end;
writeln('Min= ', min);
writeln('Max= ', max);
End.
```

1.1.6.3. Testing Membership of an Element in the Array:

To test the membership of an element in an array in Pascal, you can use a loop to iterate through the array elements and check if the searched element equals any of the elements in the array.

```
program Test_Membership;
```

```
var      T: array[1..6] of Integer;
          x,i: Integer;
          y: Boolean;
begin
   for i := 1 to 6 do
    begin
       write('Enter the value for T[', i, '] : ');
        readln(T[i]);
      end;
  write(' Enter the element to search for: ');
  readln(x);
  y := False;
  for i := 1 to 6 do
  begin
   if ( T[i] = x ) then
         begin
            y := True;
            writeln(' The element is present in the array.')
          end
      else
           begin
              y := False;
              writeln('LThe element is not present in the array.');
            end;
   end;
End.
```

1.1.7 Application Exercises:

Exercise 1:

Write a Pascal program that calculates the sum and average of elements in a real array T of size N.

Solution:

```
program Sum_Aver;
var      T: array[1..N] of real;
          sum, Ave: real;
          I,N: integer;
Begin
Write(' Enter the size of the array:');
Read(N);
```

```
  writeln(' Enter the elements of the array:');
  for i := 1 to N do
  begin
    write('T[', i, '] = ');
    readln(T[i]);
  end;
  sum := 0;
  for i := 1 to N do
  begin
    sum := sum+ T[i];
  end;
  Ave := sum / N;
  writeln(The sum of the elements of the array is: ', sum);
  writeln('The average of the elements of the array is: ', Ave);
end.
```

Exercise 2:

Complete the program that asks the user for 10 real values corresponding to grades between 0 and 20, stores them in an array, and displays how many grades are greater than or equal to 10. (Assume all entered values are valid between 0 and 20.)

Solution:

```
program Ex2;
var i , n : .....;
tab : .....;
begin
for i:=..... to ..... do
 begin
writeln ('Enter note number',i );
.....;
end;
n := .....;
for i:=..... to ..... do
if ..... then
 .....;
writeln('There are ', ..... ,' note greater than or equal to 10');
end.
```

Solution:

```
program Ex2;
var i , n : integer;
tab : array [ 1..10 ] of real;
begin
for i:=1 to 10 do
begin
writeln ('Enter note number',i );
readln (tab[i]);
 end;
n := 0;
for i:=1 to 10 do
if tab[i] >= 10 then
n := n+1;
writeln('There are ', n ,' note greater than or equal to 10');
end.
```

Exercise3:

Complete the program that asks the user for 20 integer values, stores them in an array T. Then, the program places the even values from T into another array T1 and the odd values into array T2. Finally, the values of T1 and T2 are displayed at the end.

```
program even_odd;
 var    i , j,k : .....;
        T, T1, T2 : .....;
begin
for i:=..... to ..... do
begin
writeln (' Enter value number',i , ‘of  T’);
end;
j:=.....; k:=.....;
for i:=..... to ..... do
begin
if (.....) then
   begin
   .....;
  T1[j]:= .....;
  End
    Else
```

```
        begin
          T2[k]:= .....;
         End;
j:= .....;
K:= .....;
End;
for i:=..... to ..... do
writeln (' The value number' , i ,' of T1 is: ', .....) ;
 for i:=..... to ..... do
writeln (' The value number' , i ,' of T2 is: ', .....) ;
end.
```

Solution:

```
program even_odd;
var     i , j,k : integer;
          T, T1, T2 : array [ 1..20 ] of integer;
begin
for i:=1 to 20 do
begin
writeln (' Enter value number",i , 'of T');
readln ( T[i] );
end;
j:=1; k:=1;
for i:=1 to 20 do
begin
if (T[i]mod 2=0)then
begin
T1[j]:= T[i];
 j:= j+1;
End
Else
begin
T2[k]:= T[i];
 K:= k+1;
End; End;
for i:=1 to j-1 do
```

writeln (' The value number' , i ,' of T1 is: ', T1[i]) ;

for i:=1 to k-1 do

writeln ('The value number' , i ,' de T2 is: ', T2[i]) ;

End.

Exercise 4:

Complete the program that asks the user for 10 real values, stores them in an array T, then asks for a real value n, and finally displays the number of occurrences of n in T.

program occurrences**;**

var **i , occ : .....;**

T : .....;

n: .....;

begin

for i:..... to ..... do

begin

writeln ('Entre value number' ,i);

readln (.....);

end;

writeln ('Entre the value n ');

readln (.....);

occ := .....;

for i:.....to .....do

if.....then

occ:=.....;

writeln('The number of occurrences of n in T is:',.....);
end.

Solution:

program occurrences**;**

var **i , occ : integer;**

T : array[1..10] of real;

n:real;

begin

for i:=1 **to** 10 **do**

begin

writeln ('Entre value number', i);

readln (T[i]) ;

end;

```
writeln ('Entre the value n ');
 readln ( n );
occ := 0;
for i:=1 to 10 do
if (T[i] = n) then
occ:= occ+1;
writeln('The number of occurrences of n in T is: ', occ );
end.
```

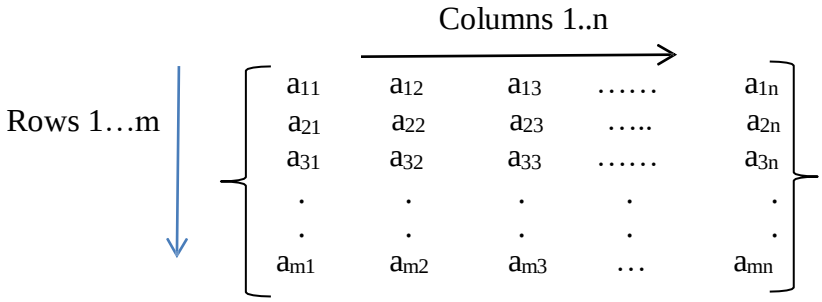
1.2. Two-Dimensional Arrays (Matrices)

1.2.1. Definition of a Matrix:

A matrix is a two-dimensional data structure composed of elements arranged in rows and columns. Each element of the matrix is identified by its row and column indices. It can be viewed as a rectangular array of elements, where each element is accessible by two indices.

The general notation for a matrix is as follows:

La notation générale d'une matrice est la suivante :



where:

- m is the number of rows,
- n is the number of columns.

For example, in the matrix above, (a_{32}) represents the element located in the third row and second column.

Matrices can have different sizes, such as square matrices (same number of rows and columns) or rectangular matrices (different numbers of rows and columns).

Matrices are commonly used in various fields of mathematics, physics, computer science, and other disciplines to represent and manipulate structured data.

1.2.2. Main Characteristics of a Matrix:

The main characteristics of a matrix include its order, elements, transpose, operations (addition and multiplication), diagonal, and determinant (for square matrices).

Ordre : The order of a matrix refers to the number of rows and columns it has. It is generally expressed as m×n, where m is the number of rows and n is the number of columns.

Éléments : Les éléments d'une matrice sont les valeurs individuelles situées à chaque intersection de ligne et de colonne. Ils sont généralement notes a_{ij}, où i est le numéro de ligne et j est le numéro de colonne.

Elements: The elements of a matrix are the individual values located at each intersection of a row and a column. They are typically denoted as a_{ij}, , where i is the row number and j is the column number.

Transpose*:* The transpose of a matrix is obtained by swapping its rows and columns. If A is an m×n matrix, then its transpose, denoted as A^T, is an n×m matrix where the rows of A become the columns of A^T and vice versa.

Matrix Operations:

- **Addition**: Two matrices can be added if they have the same dimensions. Addition is performed element-wise.
- **Multiplication**: The rules for matrix multiplication depend on their dimensions. In general, the product of an m×p matrix A with a p×n matrix B results in an m×n matrix C.

Diagonal: A square matrix has a main diagonal running from the top-left to the bottom-right corner. The elements on this diagonal are those where i=j.

Determinant: The determinant is a characteristic of square matrices. It is used in various applications, particularly to determine if a matrix is invertible. The computation of the determinant depends on the size of the matrix.

1.2.3. Declaration of a Matrix:

In Pascal, a matrix is declared as follows. Suppose you want to declare a matrix of size M×N:

```
program Declaration_Matrix;
var     Matrix: array[1..M, 1..N] of integer;
begin
Your code
end.
```

1.2.4. Creation and Displaying of a Matrix:

To display a matrix in Pascal, we can use two nested loops to iterate through each element of the matrix, prompt the user to enter values, and then display the matrix on the screen. You can adjust the values of M and N based on the desired size for your matrix.

```
program Creation_Matrix;
var     A: array[1..M, 1..N] of integer;
        i, j: integer;
begin
```

```
readln (M , N);
  writeln(' Enter the elements of the matrix.:');
  for i := 1 to M do
  begin
   for j := 1 to N do
   begin
     write('A[' , i,  ',' , j,  '] = ');
     readln(A[i, j]);
   end;
  end;
```

Displaying the matrix

```
  writeln('The matrix you entered is:');
  for i := 1 to M do
  begin
   for j := 1 to N do
   begin
     write (A[i, j]:4, ' ');
   end;
  end;
end.
```

1.2.5. Application Exercises:

Exercise 1:

Consider a matrix M of 8 rows and 5 columns of real numbers. We propose completing the Pascal program that allows reading the elements of M, then determining the maximum value of this matrix as well as the row and column to which this maximum value belongs.

```
Program Matrix;
Var     M :..................................... ;
        i, j, imax, jmax :..... ;
        valmax :..... ;
begin
for i :=.....to.....do
for j:=.....to.....do
readln(.....);
valmax:=.....; imax:=.....; jmax:=.....;
for i := 1 to.....do
for j:=.....to.....do
```

```
 begin
if ..... then
begin
imax:=.....; jmax:=.....; valmax:=.....;
end;
end;
writeln('The maximum value of M is: ',……., ', located in row ',……, ' and column ',……..');
End.
```

Solution:

```
Program Matrix;
Var          M :array [1..8,1..5] of real ;
             i, j, imax, jmax :integer ;
             valmax :real;
begin
for i :=1 to 8 do
for j:= 1 to 5 do
readln(M[i,j]);
valmax:=M[1,1];   imax:= 1;     jmax:= 1 ;
for i := 1 to 8 do
for j:= 1 to 5 do
 begin
if (M[i,j] > valmax) then
begin
imax:= I ; jmax:= j ; valmax:= M[i,j];
end;
end;
writeln('The maximum value of M is: ', M[i,j], ', located in row ', imax, ' and column ', jmax );
End.
```

Exercise 2:

A is a square matrix of order 10 with real coefficients (10 rows and 10 columns). Complete the program that prompts for input of the coefficients of matrix A, displays it, and then calculates and displays the sum of the diagonal elements of A.

```
PROGRAM Matrix ;
VAR      A, B :..... ;
         i, j : ..... ;
```

```
        s : ..... ;
BEGIN
FOR i := ..... TO ..... DO
FOR j := ..... TO ..... DO
BEGIN
WRITE (' A[',i,',' ,j,']=');
READLN (.....);
END;
FOR i := ..... TO ..... DO
BEGIN
FOR j:= ..... To ..... DO
WRITE(.....,'—');
END;
s := .....;
FOR i := ..... TO ..... DO
s:= .....;
Writeln('The sum of the diagonal elements is: ', .....);
 END.
```

Solution:

```
PROGRAM Matrix ;
VAR     A :array [1..10,1..10] of real;
        i, j : integer;
        s : real;
BEGIN
FOR i := 1TO 10 DO
FOR j := 1TO 10 DO
BEGIN
WRITE (' A[',i,',' ,j,']=');
READLN (A[i,j]);
END;
FOR i := 1TO 10 DO
BEGIN
FOR i := 1TO 10 DO
WRITE(A[i,j],'—');
WRITELN;
END;
```

```
s := 0;
FOR i := 1 TO 10 DO
s:= s+ A[i,i];
Writeln (‘The sum of the diagonal elements is:’, s);
 END.
```

Exercise 3:

Write a Pascal program that calculates the sum of two matrices A and B, each containing 5 rows and 4 columns.

Solution:

```
program Sum_Matrix;
var.    A, B, Sum: array[1..5, 1..4] of integer;
        i, j: integer;
begin
  writeln(' Enter the elements of matrix A:');
    for i := 1 to 5 do
      for j := 1 to 4 do
        begin
          write('A[', i, ',', j, '] = ');
          readln(A[i, j]);
        end;
  writeln(' Enter the elements of matrix B :');
  for i := 1 to5 do
    for j := 1 to 4 do
      begin
        write('B[', i, ',', j, '] = ');
        readln(B[i, j]);
      end;
  for i := 1 to 5 do
    for j := 1 to 4 do
      Sum[i, j] := A[i, j] + B[i, j];
  writeln('sum of matrix A and B :');
  for i := 1 to 5 do
  begin
    for j := 1 to 4 do
      write(Sum[i, j]:4, ' ');
    writeln;
```

```
  end;
End.
```

Exercise 4:

Write a Pascal program that calculates the transpose of a matrix A with 3 rows and 4 columns.

Solution:

```
program Tran_Matrix;
var     A, Tr : array[1..3, 1..4] of integer;
            i, j: integer;
begin
  writeln(' Enter the elements of matrix A:');
  for i := 1 to 3 do
    for j := 1 to 4 do
    begin
      write('A[', i, ',', j, '] = ');
      readln(A[i, j]);
    end;
  for i := 1 to 3 do
    for j := 1 to 4 do
      Tr [j, i] := A[i, j];
  writeln(The transpose of matrix A is:');
  for i := 1 to 4 do
  begin
    for j := 1 to 3 do
      write(Tr[i, j]:4, ' ');
    writeln;
  end;
End.
```

Exercise 5:

Write a Pascal program that calculates the sum of the elements on the diagonal of a square matrix of order 4.

Solution:

```
program Diagonal;
var     M: array[1..4, 1..4] of integer;
        i, j, SD: integer;
begin
```

```
  writeln(' Enter the elements of matrix M:');
  for i := 1 to 4 do
    for j := 1 to 4 do
    begin
      write('M[', i, ',', j, '] = ');
      readln(M[i, j]);
    end;
  SD:= 0;
  for i := 1 to 4 do
    SD:= SD + M[i, i];
  writeln('The Matrix M:');
  for i := 1 to 4 do
  begin
    for j := 1 to 4 do
      write(M[i, j]:4, ' ');
    writeln;
  end;
  writeln('sum of the elements of  the diagonal is : ', SD);
End.
```

Exercise 6:

Write a Pascal program that calculates the product of two matrices A and B with dimensions 4×4.

Solution:

```
program Prod_Matrix;
var    A, B, P: array[1..4, 1..4] of integer;
           i, j, k: integer;
begin
  writeln(' Enter the elements of matrix A :');
  for i := 1 to 4 do
    for j := 1 to 4 do
    begin
      write('A[', i, ',', j, '] = ');
      readln(A[i, j]);
    end;
  writeln(' Enter the elements of matrix B :');
  for i := 1 to 4 do
```

```
  for j := 1 to 4 do
  begin
    write ('B[', i, ',', j, '] = ');
    readln (B[i, j]);
  end;
 for i := 1 to 4 do
  for j := 1 to 4 do
    P [i, j] := 0;
 for i := 1 to 4 do
  for j := 1 to 4 do
    for k := 1 to 4 do
      P[i, j] := P[i, j] + A[i, k] * B[k, j];
 writeln(' The product of matrices A and B is :');
 for i := 1 to 4 do
 begin
  for j := 1 to 4 do
    write(P [i, j]:6, ' ');
  writeln;
 end;
End.
```

Chapter II: Functions and Procedures

2.1. Definition of a Subroutine:

The definition of a subroutine involves declaring a distinct and reusable portion of code within a main program. This code segment, often referred to as a function or procedure, performs a specific task and is characterized by input and output parameters. Using subroutines enhances code modularity and clarity, as specific functionalities can be isolated and handled independently of the rest of the program.

The structure of a subroutine typically includes a header detailing the name, return type (for functions), and parameters, followed by an executable code block. This approach facilitates code maintenance, reusability, and comprehension.

2.2. Functions:

In a programming language, a function is a self-contained block of code designed to perform a specific task and return a result. A function is characterized by a unique name, a return type that specifies the data type of the value returned by the function, and optionally input parameters that allow values to be passed into the function.

The structure of a function is defined as follows:

```
function function_name(parameters: parameter_types): return_type;
var
  local_variables: types_local_variables;
  result: return_type;
begin
  { ... in the body, perform operations on result ... }
  { assign the result value }
  function_name := result;
end;
```

In this structure:

function_name: The name of the function.

parameters: parameter_types: The input parameters and their data types.

return_type: The data type of the value the function will return.

local_variables: Local variables used within the function, with their data types.

result: A variable of the return type where the function's result is stored.

function_name := result;: The result of the function is assigned to the function name, which is how the function returns its value.

Example:

A simple example of a function in Pascal. This program asks the user for two numbers, calls a function to calculate their sum, and then displays the result.

```
program Fonction_sum;
# Function declaration
function Summ (a, b: Integer): Integer;
var
  sum: Integer;
begin
  sum := a + b;
 Summ:= sum;
End;
Var         x1, x2, R: Integer;
begin
  write('x1= ');
  readln(x1);
  write('x2= ');
  readln(x2);
# Calling the function
  R := Summ(x1, x2);
  writeln('The sum is : ', R);
end.
```

Another example of a function in Pascal. This time, the function takes a number as input, performs a mathematical operation, and returns the result:

```
program Fonction_carre;
# Function declaration
function Square (a: Integer): Integer;
begin
  Square:= a * a ;
end;
var         x, res: Integer;
begin
  write('x=');
  readln(x);
# Calling the function
 res:= Square (x);
```

```
  writeln('Square of ', x, ' is : ', res);
end.
```

2.3.The procedures:

A procedure in Pascal is a subroutine that performs a specific task without returning an explicit value. Unlike a function, which returns a value, a procedure is designed to execute actions without producing a specific result.

The structure of a procedure is defined as follows:

```
program Exe_Procedure;
 #Procedure Declaration
procedure nam_Procedure(a1: Integer; a2: Integer);
var      sum: Integer;
begin
  sum := a1 + a2;
  writeln(' The sum is : ', sum);
#No explicit return is necessary for a procedure
end;
var   x1, x2: Integer;
begin
  write (' Enter the first number: ');
  readln (x1);
  write ('Enter the second number: ');
  readln (x2);
Calling the procédure
  nam_Procedure(x1, x2);
End.
```

Example:

A simple example of a procedure in Pascal that asks the user for two numbers, calls the procedure to calculate their product, and then displays the result:

```
program Procedure_Prod;
procedure    Product (a, b: Integer);
var       P : Integer;
begin
  P := a * b;
  Writeln ('The  product is : ', P);
end;
var     x, y: Integer;
```

```
begin
  write(' Enter the first number ');
  readln(x);
  write(' Enter the second number: ');
  readln(y);
  Product (x, y);
End.
```

Another example of a procedure in Pascal that takes a number as input, calculates its factorial, and then displays the result:

```
program Procedure_Facto;
procedure Factorial (x: Integer);
var         i, R: Integer;
begin
 R := 1;
  for i := 2 to x do
    R := R * i;
  writeln('the Factorial of ', x, ' is : ',R);
end;
var           y: Integer;
begin
  write ('Enter the number : ');
  readln (y);
  Factorial(y);
End.
```

2.4. Parameter Passing Modes:

In programming, the parameter passing mode refers to the method by which values are transmitted to a function or procedure. There are several parameter passing modes, each with implications for how data is handled. The main parameter passing modes are as follows:

2.4.1. Pass-by-Value (default):

- The actual value of the parameter is passed to the function or procedure.
- Changes made to the parameter inside the function do not affect the original variables.
- This is the most commonly used parameter passing mode.

Example in Pascal:

```
procedure PassByValue(x: Integer);
begin
  x := x + 1;  { Changes are not reflected outside the procedure }
end;
program PassByValueExample;
{ Declaration of the procedure with a parameter passed by value }
procedure ModifyValue(x: Integer);
begin
  x := x + 1;  { Changes are not reflected outside the procedure }
end;
var
  n: Integer;
begin
  n := 10;
  { Calling the procedure with pass-by-value }
  ModifyValue(n);
  { Display the result }
  writeln('The unmodified value is: ', n);
end.
```

2.4.2. Pass-by-Reference:

The memory address of the parameter is passed to the function or procedure.

Changes made to the parameter inside the function affect the original variables.

Example in Pascal:

```
procedure PassByReference(var x: Integer);
begin
  x := x + 1;  { Changes are reflected outside the procedure }
end;
program PassByReferenceExample;
{ Declaration of the procedure with a reference parameter }
procedure ModifyReference(var x: Integer);
begin
  x := x + 1;  { Changes are reflected outside the procedure }
```

```
end;
var
  n: Integer;
begin
  n := 10;
  { Calling the procedure with pass-by-reference }
  ModifyReference(n);
  { Display the result }
  writeln('The modified value is: ', n);
end.
```

2.5. Application Exercises:

Exercise 1:

Write a program that calculates the first 10 factorials using a function.

Solution:

```
program factorials;
function Factorial(n: Integer): Integer;
var     i, r: Integer;
begin
  r := 1;
  for i := 2 to n do
    r := r * i;
  Factorial:= r;
end;
var      i: Integer;
begin
  writeln('The first 10 factorials are:');
  for i := 1 to 10 do
    writeln(' factorials of ', i, ' : ', Factorial(i));
End.
```

Exercise 2:

Write a procedure in Pascal that prompts the user to enter a number N between 1 and 10 until a valid number is provided.

Solution:

```
program Request_Number;
procedure number (var N: Integer);
begin
```

```
  repeat
    Write('enter a number between 1 and 10:');
    Readln(N);
  until (N >= 1) and (N <= 10);
end;
var     N1: Integer;
begin
number (N1);
  writeln(' You have chosen the number:', N1);
end.
```

Exercise 3:

Write a function in Pascal that requests a number X between 10 and 20 until a valid number is provided. If the response is greater than 20, it should display a message: "Smaller!", and conversely, "Larger!" if the number is less than 10.

Solution:

```
program Request_Number;
procedure Number (var X: Integer);
begin
  repeat
    Write('enter a number between 10 et 20 : ');
    Readln(X);
    if (X < 10) then
      writeln(' Larger!')
    else if (X > 20) then
      writeln(' Smaller!');
  until (X >= 10) and (X <= 20);
end;
var     n: Integer;
Number (n);
  writeln(' You have chosen the number: ', n);
end.
```

Exercise 4:

Write a function in Pascal that takes a real number x and a non-negative integer n, and then calculates x^n.

Solution:

```
program Calculate_Power
function Power (x: Real; n: Integer): Real;
var    r: Real;
       i: Integer;
begin
 r := 1.0;
 if n >= 0 then
 begin
  for i := 1 to n do
  begin
   r:= r * x;
  end;
 end
 else
 begin
  writeln('The exponent must be a non-negative integer.');
 end;
 Power := r;
end;
var     x: Real;
        n: Integer;
       P: Real;
begin
 Readln(x,n);
 P := Power (x, n);
 writeln(x, '^', n, ' = ', P);
end.
```

Exercise 5:

1-Complete the absolute function, which takes a real parameter x and returns the absolute value of x

```
Function absolute(…… :……..……..):……..;
Var R : real;
Begin
  If…….……then R:=……..
  Else  R:=……;
  absolute:=……..;
```

End.

2- Rewrite this function as a procedure.

Solution:

1-Complete the absolute function, which takes a real parameter x and returns the absolute value of x

```
Function absolute(x:real):real;
Var R : real;
Begin
  If   x < 0  then
       R:=  -x
  Else  R:= x ;
  absolute:= R ;
End.
```

2- **procedure** absolute(x:real; var R:real);

```
Begin
  If   x < 0  then R:=  -x
  Else  R:= x ;
End.
```

Exercise 6:

Write a program that finds all prime numbers between 1 and 50 and displays them on the screen. Reminder: a number is prime if and only if it has no divisors other than 1 and itself. Use a Prime function that takes an integer as a parameter and returns true or false indicating whether the integer is prime or not.

Solution:

```
Program ex6;
  Var i: integer;
   Function Prime (n: integer) : Boolean;
     Var j: integer;
         b:Boolean;
   Begin
      b:= true;
      for j:= 2 to (n-1)  do
          if (n mod i)=0 then
           b:=false;
     Prime:=b;
end;
begin
```

```
    for i:= 2  to  50  do
      if  Prime (i)  then  writeln (i);  (* if  Prime (i) is equivalent to if Prime (i) = true.*)
end.
```

Exercise 7:

Write a procedure named product that calculates the product of two complex numbers. This procedure takes as parameters the real numbers a,b,c,d,e,f, where the first four numbers define the two complex numbers a+i*b and c+i*d whose product is to be computed, and e and f define the complex number e+i*f which will store the result of the operation.

Solution:

```
Procedure product ( a,b,c,d : integer; var e,f :integer);
Begin
  e:= a*c – b*d;
  f:= a*d + b*c;
end;
```

Exercise 8:

Write a program that prompts the user to enter 4 real values corresponding to the coordinates of 2 vectors and displays whether these vectors are orthogonal to each other or not. (Two vectors are orthogonal if their dot product is equal to 0.) Use a function named dot_Product that returns the dot product of the two vectors.. $\vec{u}.\vec{v}=xx'+yy'$ avec: $\vec{u}(x;y)$ et $\vec{v}(x',y')$

Solution:

```
program vecteurs ;
   var x1,x2,y1,y2 : real ;
   function dot_Product (a1,a2,b1,b2 : real) : real;
   begin
      dot_Product:= a1*a2 + b1*b2 ;
   end ;
begin
  writeln (‘Enter the coordinates of the first vector’) ;
   readln(x1) ; readln(y1) ;
   writeln (‘Enter the coordinates of the second vector) ;
   readln(x2) ; readln(y2) ;
   if   dot_Product (x1,x2,y1,y2) = 0   then
        write (‘The vectors are orthogonal’)
    else
 write (‘The vectors are non-orthogonal’);
end.
```

Exercise 9:

1-Write a function named power that takes two integer parameters a and n and returns the result of a^n. Then, write a program that calculates the sum s= 1+22+33+…+kk, where k is a given integer.

2-Rewrite the same program, but this time using a procedure named power.

Solution:

```
1- Program same;
  Var i,som,k : integer ;
  Function power (a, n : integer) : integer ;
    Var j, s : integer ;
  begin
     s:=1;
     for j= 1 to n do  s:= s*a;
    power := s;
  end;
begin
  writeln ('Enter an integer') ;
  readln(k) ;
   sum:=0;
  for i := 1 to k do
   sum := sum + power (i,i);
  writeln('The same is:', sum) ;
end.
2- Program same ;
  Var i, sum, k, pow: integer ;
  procedure power (a, n : integer; var p: integer );
    Var j : integer ;
  begin
    p:=1;
     for j= 1 to n do
        p:= p*a;
  end;
begin
  writeln ('Enter an integer') ; readln(k) ;
  sum:=0;
  for i := 1 to k do
```

```
    begin
      power(i , i, pow);
       som := sum + pow;
    end ;
  writeln(‘The same is : ’, sum) ;
end.
```

Chapter III : The Recordings and files

3.1. Structure of Heterogeneous Data:

When it is necessary to store data of various types within the same entity, such as product details, choosing the appropriate data structure is crucial. Unlike arrays, which require all elements to be of the same type, they are not suitable for heterogeneous data. An effective solution is to use a data structure of the 'record' type. This structure can contain both numeric data, such as quantity, unit price, and total price, as well as alphanumeric data, such as reference and description. This approach allows for grouping different types of data within a single unit, making it easier to represent and manage complex data. This heterogeneous structure is particularly useful for applications where flexibility and diversity in data types are required, providing an effective solution for various needs in data storage and processing.

3.2. Manipulation of Record Structures:

3.2.1 Definition:

Records are essential data structures in programming, distinct from arrays in their ability to store fields of varying types within a single entity. Each record, also known as a 'record' in English, consists of a fixed number of fields, each defined by the user with a specific type. This characteristic allows for the organized representation of complex information. For example, a 'date' record may include fields such as 'day,' 'month,' and 'year,' where 'day' and 'year' are of integer type and 'month' is of string type.

Field	Type
Day	Integer
Month	String
Year	Integer

Each field is identifiable by a clear name and is associated with a specific type that determines how the data is stored and manipulated. This structure allows for efficient data management by facilitating access to and modification of information relevant to a given application. Records are particularly useful in database management systems and in application development where the diversity and complexity of information require a structured and coherent organization.

3.2.2 Declaration:

In Pascal, a record is declared as follows, specifying the details of each field with its name and type:

Type <id_Record> = RECORD

<field1> : <type1>;

<field2> : <type2>;

...

<fieldN> : <typeN>;

End;

Var <record_name> : <id_Record>;

With:

- <id_Record> is the identifier for the record type.
- <field1>, <field2>, ..., <fieldN> are the names of the fields.
- <type1>, <type2>, ..., <typeN> are the respective types of these fields.
- <record_name> is a variable of the type <id_Record>.

Example :

```
program Record_Example;
{ Declaration of the "Date" record }
Type
  Date = record
    day: Integer;          { Field "day" of type integer }
    month: String[10];     { Field "month" of type string with a length of 10 characters }
    year: Integer;         { Field "year" of type integer }
  end;
var
  myDate: Date;            { Declaration of a variable "myDate" of type "Date" }
begin
  { Using the record }
  myDate.day:= 8;          { Assigning the value 8 to the "day" field }
  myDate.month:= 'July';   { Assigning the value 'July' to the "month" field }
  myDate.year:= 2024;      { Assigning the value 2024 to the "year" field }
  {Displaying the values}
  writeln('Date: ', myDate.day, ' ', myDate.month, ' ', myDate.year);
end.
```

Detailed Explanation:

Type Date: Defines a new record named Date.

Fields:

- **day:** A field of type Integer used to store the day.
- **month:** A field of type String[10] used to store the month, with a maximum length of 10 characters.
- **year:** A field of type Integer used to store the year.

Variable Declaration: myDate is declared as a variable of type Date, allowing it to store an instance of the Date record.

Usage: The fields of the record are accessed through the myDate variable using the dot operator (.). For example, myDate.day allows you to access and modify the value of the day field.

Display: The values of the fields are printed to the screen to demonstrate the practical use of the record.

3.2.3 Accessing Fields of a Record:

In Pascal, accessing the fields of a record differs from accessing elements in an array. While arrays are indexed by integers to access their elements, records use field names to reference their data. The dot operator (.) is used to access a specific field of a record from its variable:

<record_name>.<field_name>

For example, to access the 'age' field of a person represented by the variable P, you use the syntax P.age.

This approach provides an intuitive and direct way to handle complex data structures, allowing for clear and direct access to each relevant element. This method is particularly useful in object-oriented programming and data management, where code readability and maintainability are crucial. In summary, accessing the fields of a record in Pascal is straightforward and efficient, offering a well-organized method for managing information in software applications

Example :

```
program Record_Example;
type
  Person = record
    name: String;
    age: Integer;
    gender: Char;
  end;
var
  P: Person;
begin
  P.name := 'Aicha';    { Assigning the value 'Aicha' to the field "name" }
  P.age := 30;          { Assigning the value 30 to the field "age" }
  P.gender := 'F';      { Assigning the value 'F' to the field "gender" }
  writeln('Name: ', P.name);   { Displaying the value of the "name" field }
  writeln('Age: ', P.age);     { Displaying the value of the "age" field }
  writeln('Gender: ', P.gender); { Displaying the value of the "gender" field }
end.
```

3.2.4 Operations on Records:

Operations on records in Pascal primarily include initialization, assigning values to fields, comparison, and copying.

3.2.4.1 Initialization: Records can be initialized either at the time of declaration or later by assigning values to their fields. For example:

```
type
  Person = record
    nom: String;
    age: Integer;
  end;
var
  P: Person;
begin
  P.nom := 'Aicha';
  P.age := 30;
end.
```

3.2.4.2 Assigning Values: The fields of a record can be populated with specific values using the dot operator (.):

```
P.nom := 'Aicha';
P.age := 30;
```

3.2.4.3 Comparison: Records can be compared field by field to determine their equality or difference. For example:

```
var
  P1, P2: Person;
begin
  {Initializing records }
  P1.name := 'Aicha';
  P1.age := 30;
  P2.name := 'Ahmed';
  P2.age := 25;
  { Comparing records }
  if (P1.name = P2.name) and (P1.age = P2.age) then
    writeln('The persons are identical.')
  else
```

```
  writeln('The persons are different.');
end.
```

3.2.4.4 Copying: To copy a record into another, you can use the assignment operator ':=.' For example:

```
var
  P1, P2: Person;
begin
  P1.name := 'Aicha';
  P1.age := 30;
  { Copying P1 into P2 }
  P2 := P1;
  { Displaying information of P2 }
  writeln('Name: ', P2.name);
  writeln('Age: ', P2.age);
end.
```

3.2.5 A table as a field of a record:

Using an array as a field in a record allows you to represent structured and complex data, such as a person's information with a list of friends. This provides increased flexibility in handling and organizing data within Pascal programs.For example, suppose we want to extend the Student type to include grades for the student in 10 modules. The student's notes will be stored in an array.

```
Type TabNotes = array [1..10] of real ;
    Student = record
        name, surname: String;
```

```
        section: Char; group: String; average: Real;
        Notes: TabNotes;
    end ;
```

Var E **:** Student

Accessing Specific Grades:

To access the 3rd Note of student E, use: E.Notes[3].

To access the 7th Note of student E, use: E.Notes[7].

Example:

```
program Record_Array;
type
  { Definition of the "Person" record }
  Person = record
    name: String;
    age: Integer;
    friends: array[1..5] of String;   { "friends" field is an array of strings }
  end;
var
  P: Person;   { Declaration of a variable of type "Person" }
  i: Integer;
begin
  { Initialization of the record fields }
  P.name := 'Aicha';
  P.age := 30;
  { Assigning values to the "friends" array }
  P.friends[1] := 'Ahmed';
  P.friends[2] := 'Adel';
  P.friends[3] := 'Walid';
```

```
    { Displaying the person's information and their friends }
    writeln('Name: ', P.name);
    writeln('Age: ', P.age);
    writeln('Friends: ');
    for i := 1 to 5 do
    begin
      if P.friends[i] <> '' then
        writeln('- ', P.friends[i]);
    end;
  end.
```

3.2.6 An Array of Records :

Using an array of records allows for the efficient management of collections of structured data in Pascal. This approach is especially useful for representing sets of similar objects, such as students, employees, products, etc. It simplifies the management and manipulation of complex data in computer programs.

Example:

```
program Array_of_Records;
type
  { Definition of the "Student" record }
  Student = record
    name: String;
    age: Integer;
    grades: array[1..5] of Real;
  end;
var
  students: array[1..3] of Student; { Array of 3 Students }
  i, j: Integer;
begin
  { Initialize the first student }
  students[1].name := 'Aicha';
  students[1].age := 20;
  students[1].grades[1] := 85.0;
```

```
  students[1].grades[2] := 90.5;
  { Initialize the second student }
  students[2].name := 'Khaled';
  students[2].age := 22;
  students[2].grades[1] := 78.0;
  students[2].grades[2] := 82.5;
  { Initialize the third student }
  students[3].name := 'Hicham';
  students[3].age := 21;
  students[3].grades[1] := 88.0;
  students[3].grades[2] := 91.0;

  { Display the students' information }
  for i := 1 to 3 do
  begin
    writeln('Student ', i, ':');
    writeln('Name: ', students[i].name);
    writeln('Age: ', students[i].age);
    writeln('Grades:');
    for j := 1 to 5 do
    begin
      if students[i].grades[j] <> 0 then
        writeln('  Grade ', j, ': ', students[i].grades[j]:0:2);
    end;
  end;
end.
```

3.3. The Files:

The data used in all the programs we have written primarily comes from two sources: either it is directly embedded within the program itself or it is entered by the user during execution. Once the program terminates, this data is lost. If the program is restarted, the same data or new data must be re-entered. All this information has a common characteristic: it resides in the computer's main memory. Therefore, any deletion of this memory, whether intentional or accidental, results in the loss of this data, including the data used in the Pascal program.

Sometimes, it is essential to retain certain data after the program has finished executing, whether for archiving purposes or for future use. In everyday life, such as in a library, a doctor's office, or in

administrative settings, a file represents a collection of similar records, with no limit on the number. These considerations lead us to introduce the concept of a file.

3.3.1. Concept of a File

The concept of a file refers to a computer entity used to store data persistently on a storage medium such as a hard drive, USB flash drive, or any other storage device. A file can contain various types of data, including text, images, videos, or computer programs. It is an organized collection of data that can be read, written, modified, and deleted by a computer program.

In computing, files allow information to be preserved beyond the duration of a program's execution, making them essential for archiving, sharing data between applications, and backing up important documents. Each file has a unique name and an extension that indicates the type of data it contains. Operating systems provide standardized methods for handling files, including creating, opening, reading, writing, and closing them.

3.3.2. File Access Modes

The various methods by which a program can read or write records in a file are referred to as access modes. When opening a file, it is essential to specify the desired access mode, which determines how the data will be organized and handled. A file can be accessed in several ways depending on its initial organization defined at creation.

There are three main file access methods:

- **Sequential Access**: Records are processed in the sequential order in which they are stored. This mode is ideal for handling text files or logs where data is read or written in sequence.
- **Direct Access**: Records can be accessed directly by their number or physical address within the file. This allows for fast and efficient operations, particularly useful for structured data files such as databases.
- **Indexed Access**: This mode allows access to records based on the order of their access keys, facilitating search and manipulation operations based on index keys. It is often used in relational databases to optimize data retrieval and management.

It is important to note that some file types, such as sequential files, only support sequential access, while other file types may support multiple access modes, such as both direct and sequential access. In Pascal, you can create and manipulate different types of files, including **FILE OF** for records of the same type, **TEXT** files for records of various types, and **FILE** for files without a specific type.

3.3.3 Reading from and Writing to a File :

In Pascal, reading from and writing to a file is accomplished through specific procedures integrated into the language. Here’s how to perform these operations with details:

- **Opening a File**: Before performing any operations, you need to open the file using the assign and reset (for reading) or rewrite (for writing) procedures.

```
assign(fileVar, 'filename.txt');  #Associate the file variable with the file
reset(fileVar);                   # Open the file for reading
// or
rewrite(fileVar);                 # Open the file for writing
```

- **Reading from a File**: To read data from a file, you use the read or readln procedures. The read procedure reads data without advancing to the next line, while readln reads data and advances to the next line.

```
read(fileVar, variable);      # Read a value from the file into a variable
readln(fileVar, variable);    # Read a value from the file and move to the next line
```

- **Writing to a File**: To write data to a file, use the write or writeln procedures. The write procedure outputs data without moving to the next line, whereas writeln outputs data and moves to the next line.

```
write(fileVar, value);      #Write a value to the file.
writeln(fileVar, value);    #Write a value to the file and move to the next line.
```

- **Closing a File**: After you have finished reading from or writing to a file, it's important to close the file using the close procedure to free up system resources.

```
close(fileVar);  # Close the file
```

These steps demonstrate how to read from and write to a file in Pascal using the standard procedures **Assign, Reset, Rewrite, Read, ReadLn, Write, WriteLn** and **Close.** Ensure proper error handling and follow best practices for file manipulation to maintain data security and integrity.

Example1:

```
program FileExample;
var
  inputFile, outputFile: text;
  line: string;
begin
  #Opening files
  assign(inputFile, 'input.txt');
  reset(inputFile);
  assign(outputFile, 'output.txt');
  rewrite(outputFile);
# Reading from input file and writing to output file
  while not eof(inputFile) do
```

```
  begin
    readln(inputFile, line);
    writeln(outputFile, line);
  end;
#Closing files
  close(inputFile);   close(outputFile);
end.
```

Example 2 :

Student practical work notes:

```
PROGRAM RESULTS;
TYPE
  INDIVIDUAL = RECORD
    Name: string[20];
    Surname: string[20];
    Lab1: real;
    Lab2: real;
  END;
VAR
  STUDENT: array[1..100] OF INDIVIDUAL;
  Average: real;
  I, N: integer;
  EndOfInput: Boolean;
  File1: FILE OF INDIVIDUAL;
BEGIN
  ASSIGN(File1, 'D:\students.dat');
  REWRITE(File1);
  Writeln('ENTERING DATA INTO File1');
  EndOfInput := False;
  I := 1;
  WHILE NOT EndOfInput DO
  BEGIN
    Writeln('Press ENTER to finish entering data');
    Write('NAME: ');
    READLN(STUDENT[I].Name);
    # Check to end input if Name is empty
```

```
  If (STUDENT[I].Name = '') then
    EndOfInput := True
  else
  begin
    Write('SURNAME: ');
    READLN(STUDENT[I].Surname);
    Write('LAB1 SCORE: ');
    READLN(STUDENT[I].Lab1);
    Write('LAB2 SCORE: ');
    READLN(STUDENT[I].Lab2);
    WRITE(File1, STUDENT[I]);
    I := I + 1;
  end;
END;
 Close(File1);
 Writeln('READING FILE AND CALCULATING LAB AVERAGE');
 # Reinitialize file for reading
 RESET(File1);
 I := 1;
 # Reading and calculating the average
 WHILE NOT EOF(File1) DO
 BEGIN
  READ(File1, STUDENT[I]);
  I := I + 1;
 END;
 N := I - 1;     # N is the number of students
 # Displaying results
 Writeln('DISPLAYING RESULTS: PRESS ANY KEY');
 Readln;
 FOR I := 1 TO N DO
 BEGIN
  Average := (STUDENT[I].Lab1 + STUDENT[I].Lab2) / 2;
  writeln(STUDENT[I].Name, ' ', STUDENT[I].Surname, ' ', STUDENT[I].Lab1:6:2, ' ',
STUDENT[I].Lab2:6:2, ' ', Average:6:2);
 END;      Close(File1);
```

```
  Readln;
END.
```

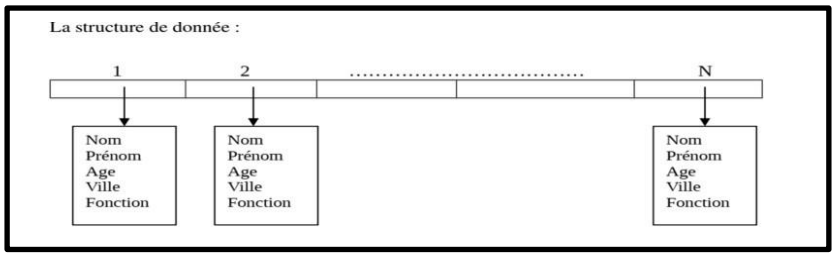
3.4 Application exercises:

Exercise 1:

Let's consider a database containing individuals described by the following information:

Name, Surname, Age, City, Position

We want to organize this information in such a way that we can select individuals based on a specific age, city, or position, etc. In general, we need to manage the database.:



We propose: T : array[1..100] of Individu;

Solution:

```
PROGRAM Ex1;
TYPE
  Individual = RECORD
    Name, Surname: string[20];
    Age: integer;
    City, Position: string[25];
  END;
VAR
  T: array[1..1000] of Individual;
  I, N: integer;
BEGIN
  Writeln('Enter the number of individuals: ');
  Readln(N); { N is the number of individuals }
  Writeln('Enter Name, Surname, Age, City, Position for each individual:');
  For I := 1 to N do
  begin
    Readln(T[I].Name);
```

```
  Readln(T[I].Surname);
  Readln(T[I].Age);
  Readln(T[I].City);
  Readln(T[I].Position);
 end;
 Writeln('Displaying the information of the individuals:');
 For I := 1 to N do
 begin
  Write('Name: ', T[I].Name, ', ');
  Write('Surname: ', T[I].Surname, ', ');
  Write('Age: ', T[I].Age, ', ');
  Write('City: ', T[I].City, ', ');
  Write('Position: ', T[I].Position);
  Writeln;
 end;
END.
```

Exercise 2:

Write a Pascal program that allows:

1. Creating a file containing the following information:
   - Identification number (integer)
   - Name (string of 20 characters)
   - Average (real number)
2. Viewing the list of students.
3. Viewing a student by identification number.
4. Viewing the list of students in descending order of their averages.

Solution:

```
function menu: integer;
var
 choice: integer;
begin
 writeln('1. Create the file.');
 writeln('2. View the list of students.');
 writeln('3. View a student by ID number.');
 writeln('4. View the list of students in descending order of averages.');
```

```
  writeln('5. Exit the application.');
  writeln('Enter your choice:');
  readln(choice);
  menu := choice;
end;
type
  student = record
    id_num: integer;
    name: string[20];
    average: real;
  end;
var
  stu_file: file of student;
  s: student;
  response: char;
  choice, num, count, i, j: integer;
  students: array[1..10] of student;
begin
  choice := menu;
  repeat
    case choice of
      1:
      begin
        # Create the file
        assign(stu_file, 'student.dat');
        rewrite(stu_file);
        repeat
          writeln('Enter the student ID number:');
          readln(s.id_num);
          writeln('Enter the student name:');
          readln(s.name);
          writeln('Enter the student average:');
          readln(s.average);
          write(stu_file, s);
          writeln('Enter another? (y/n)');
```

```
    readln(response);
  until response = 'n';
  close(stu_file);
end;
2:
begin
  #View the list of students
  assign(stu_file, 'student.dat');
  reset(stu_file);
  while not eof(stu_file) do
  begin
    read(stu_file, s);
    writeln('ID = ', s.id_num, ', Name: ', s.name, ', Average: ', s.average:0:2);
  end;
  close(stu_file);
end;
3:
begin
  #View a student by ID number
  writeln('Enter the student ID number:');
  readln(num);
  assign(stu_file, 'student.dat');
  reset(stu_file);
  while not eof(stu_file) do
  begin
    read(stu_file, s);
    if s.id_num = num then
      writeln('ID = ', s.id_num, ', Name: ', s.name, ', Average: ', s.average:0:2);
  end;
  close(stu_file);
end;
4:
begin
  # View the list of students in descending order of averages
  assign(stu_file, 'student.dat');
```

```
    reset(stu_file);
    count := 1;
    # Store records from the file into an array for sorting
    while not eof(stu_file) do
    begin
      read(stu_file, students[count]);
      count := count + 1;
    end;
    close(stu_file);
   #Sort the array in descending order of averages
    for i := 1 to count - 1 do
    begin
      for j := i + 1 to count - 1 do
      begin
        if students[j].average > students[i].average then
        begin
          s := students[i];
          students[i] := students[j];
          students[j] := s;
        end;
      end;
    end;
    # Display the sorted array
    for i := 1 to count - 1 do
      with students[i] do
        writeln(id_num, ' ', name, ' ', average:0:2);
  end;
  5:
  begin
    # Exit the application
  end
  else
    writeln('Invalid choice. Please try again.');
end;
if choice <> 5 then
```

```
    choice := menu;
  until choice = 5;
end.
```

Exercise 3:

Write a Pascal program that stores the list of students from the file **student** into a second file **student2**, after sorting the students in ascending order of their averages.

Solution:

```
program Sort_File;
type
  student = record
    num_id: integer;
    name: string[20];
    average: real;
  end;
var
  file1: file of student;
  file2: file of student;
  s: student;
  count, i, j: integer;
  students: array[1..10] of student;
begin
  # Open the original file for reading
  assign(file1, 'student');
  reset(file1);
  count := 0; // Initialize count to 0, since it will be incremented before storing in the array
  # Store records from the 'student' file into an array for sorting
  while not eof(file1) do
  begin
    count := count + 1; // Increment count before storing in the array
    read(file1, students[count]);
  end;
  close(file1);
  #Sort the array by ascending average
  for i := 1 to count - 1 do
  begin
```

```
  for j := i + 1 to count do
  begin
    if students[j].average < students[i].average then
    begin
      s := students[i];
      students[i] := students[j];
      students[j] := s;
    end;
  end;
end;
#Store the sorted array into a new file 'student2'
assign(file2, 'student2');
rewrite(file2); // This will create the file 'student2' if it doesn't exist
for i := 1 to count do
  write(file2, students[i]);
close(file2);
#Display the list of students from the file 'student2'
writeln('List of students in the file student2:');
assign(file2, 'student2');
reset(file2);
while not eof(file2) do
begin
  read(file2, s);
  writeln('ID = ', s.num_id, ', Name: ', s.name, ', Average: ', s.average:0:2); #Display average with 2 decimal places
end;
close(file2);
end.
```

Exercise 4:

Write a Pascal program that inserts a student into the **student2** file while maintaining the order of the averages in ascending order.

Solution:

```
program Insert_File;
type
  student = record
```

```
    num_id: integer;
    name: string[20];
    average: real;
  end;
var
  file1: file of student;
  s: student;
  count, i, j: integer;
  students: array[1..10] of student;
  inserted: boolean;
begin
  # Display the list of students from the file student2 before insertion
  writeln('List of students in the file student2 before insertion:');
  assign(file1, 'student2');
  reset(file1);
  while not eof(file1) do
  begin
    read(file1, s);
    writeln('ID = ', s.num_id, ', Name: ', s.name, ', Average: ', s.average:0:2);
  end;
  close(file1);
  #Store records from the file student2 into an array
  assign(file1, 'student2');
  reset(file1);
  count := 0;
  while not eof(file1) do
  begin
    count := count + 1;
    read(file1, students[count]);
  end;
  close(file1);
  # Read the information of the student to be inserted
  writeln('Enter the student to be inserted:');
  writeln('Enter the student ID:');
  readln(s.num_id);
```

```
  writeln('Enter the student name:');
  readln(s.name);
  writeln('Enter the student average:');
  readln(s.average);
  # Insert the student into the array
  i := 1;
  inserted := false;
  while (i <= count) and (not inserted) do
  begin
    if (s.average < students[i].average) then
      inserted := true
    else
      i := i + 1;
  end;
  for j := count downto i + 1 do
    students[j] := students[j - 1];
  students[i] := s;
  count := count + 1;
  #Write the array back to the file student2
  assign(file1, 'student2');
  rewrite(file1);
  for i := 1 to count do
    write(file1, students[i]);
  close(file1);
  #Display the list of students from the file student2 after insertion
  writeln('List of students in the file student2 after insertion:');
  assign(file1, 'student2');
  reset(file1);
  while not eof(file1) do
  begin
    read(file1, s);
    writeln('ID = ', s.num_id, ', Name: ', s.name, ', Average: ', s.average:0:2);
  end;
  close(file1);
end.
```

Exercise 5:

Write a Pascal program to delete or modify the information of a student based on a number read from the keyboard. This involves modifying or deleting a record in the previously created student file. If the entered number does not exist, it should be reported.

Solution:

```
program SchoolManagement;
function menu: integer;
var
  choice: integer;
begin
  writeln('1. View the list of students.');
  writeln('2. Modify a record.');
  writeln('3. Delete a record.');
  writeln('4. Exit the application.');
  writeln('Enter your choice:');
  readln(choice);
  menu := choice;
end;
type
  student = record
    num_id: integer;
    name: string[20];
    average: real;
  end;
var
  studentFile: file of student;
  s: student;
  choice, num, count, i, j: integer;
  students: array[1..20] of student;
  exists: boolean;
begin
  choice := menu;
  repeat
    case choice of
      1:
```

```
begin
  #View the list of students
  assign(studentFile, 'student');
  reset(studentFile);
  while not eof(studentFile) do
  begin
    read(studentFile, s);
    writeln('ID = ', s.num_id, ', Name: ', s.name, ', Average: ', s.average:0:2);
  end;
  close(studentFile);
end;
2:
begin
  # Modify a record
  assign(studentFile, 'student');
  reset(studentFile);
  count := 1;
  # Store records from the file into an array
  while not eof(studentFile) do
  begin
    read(studentFile, students[count]);
    count := count + 1;
  end;
  close(studentFile);
  writeln('Enter the ID of the student to modify:');
  readln(num);
  i := 1;
  exists := false;
  while (i <= count - 1) and not exists do
  begin
    if students[i].num_id = num then
      exists := true;
    if not exists then
      i := i + 1;
  end;
```

```
  if exists then
  begin
    writeln('Enter new information:');
    writeln('Enter the ID of the student:');
    readln(s.num_id);
    writeln('Enter the name of the student:');
    readln(s.name);
    writeln('Enter the average of the student:');
    readln(s.average);
    students[i] := s;
    # Store the array back into the file
    assign(studentFile, 'student');
    rewrite(studentFile);
    for j := 1 to count - 1 do
      write(studentFile, students[j]);
    close(studentFile);
  end
  else
    writeln('No student exists with the ID: ', num);
end;
3:
begin
  # Delete a record
  assign(studentFile, 'student');
  reset(studentFile);
  count := 1;
  # Store records from the file into an array
  while not eof(studentFile) do
  begin
    read(studentFile, students[count]);
    count := count + 1;
  end;
  close(studentFile);
  writeln('Enter the ID of the student to delete:');
  readln(num);
```

```
      i := 1;
      exists := false;
      while (i <= count - 1) and not exists do
      begin
        if students[i].num_id = num then
          exists := true;
        if not exists then
          i := i + 1;
      end;
      if exists then
      begin
        for j := i to count - 2 do
          students[j] := students[j + 1];
       # Store the array back into the file
        assign(studentFile, 'student');
        rewrite(studentFile);
        for j := 1 to count - 2 do
          write(studentFile, students[j]);
        close(studentFile);
      end
      else
        writeln('No student exists with the ID: ', num);
    end;
    4: begin   end;   # Exit the program
  else
    writeln('Invalid choice. Please try again.');
  end;
  if choice <> 4 then
    choice := menu;
 until choice = 4;
end.
```

Exercise 6:

Ecrire un programme Pascal permettant de copier un fichier texte **texte.txt** dans un deuxième fichier **essai.txt**, ensuite de fusionner ces deux fichiers dans un troisième nommé texteglob.txt.

Write a Pascal program to copy a text file **texte.txt** into a second file **essai.txt**, then merge these two files into a third one named texteglob.txt.

Solution:

```
program concat_copy_file;
var
  file1, file2, file3: text;
  line: string;
begin
  assign(file1, 'texte.txt');
  reset(file1);
  if (IOResult = 0) then
  begin
    # Copy texte.txt to essai.txt
    assign(file2, 'essai.txt');
    rewrite(file2);
    while not eof(file1) do
    begin
      readln(file1, line);
      writeln(file2, line);
    end;
    close(file1);
    close(file2);
    # Concatenate texte.txt and essai.txt into texteglob.txt
    assign(file1, 'texte.txt');
    reset(file1);
    assign(file2, 'essai.txt');
    reset(file2);
    assign(file3, 'texteglob.txt');
    rewrite(file3);
    while not eof(file1) do
    begin
      readln(file1, line);
      writeln(file3, line);
    end;
    close(file1);
```

```
while not eof(file2) do
begin
  readln(file2, line);
  writeln(file3, line);
end;
close(file2);
close(file3);
# Display the contents of texte.txt
writeln('Contents of the file texte.txt:');
assign(file1, 'texte.txt');
reset(file1);
while not eof(file1) do
begin
  readln(file1, line);
  writeln(line);
end;
close(file1);
# Display the contents of essai.txt
writeln('Contents of the file essai.txt:');
assign(file2, 'essai.txt');
reset(file2);
while not eof(file2) do
begin
  readln(file2, line);
  writeln(line);
end;
close(file2);
# Display the contents of texteglob.txt
writeln('Contents of the file texteglob.txt:');
assign(file3, 'texteglob.txt');
reset(file3);
while not eof(file3) do
begin
  readln(file3, line);
  writeln(line);
```

```
    end;
    close(file3);
  end
  else
    writeln('File texte.txt does not exist.');
end.
```

Bibliography

1. Jean MAYSONNAVE, « Introduction à l'algorithmique générale et numérique DEUG Sciences : Résumés de cours », Edition Masson, 1996.
2. Support de cours en Informatique 02, Algorithmique et programmation, R. BENGHEZAL, Universite de batna2, 2020.
3. Ives GRANJON, « Tavaux dirigés - Informatique : Algorithmique en Pascal et en langage C - DEUG Sciences : Résumés de cours », Edition Dunod, 1999.
4. Cormen, Thomas H., and Hervé Soulard. Algorithmes: notions de base. Dunod, 2013.
5. Salah FENNI, « Exercices en Turbo Pascal », Edition Chebba, 2000.
6. Cormen, Thomas H., et al. Algorithmique: cours avec 957 exercices et 158 problèmes. Dunod, 2010.
7. Olivier LECARME, « Pascal : Langages d'écriture de systèmes », Techniques de l'Ingénieur, traité Informatique, H 2260.
8. Mueller, John Paul, and Luca Massaron. Les algorithmes pour les Nuls grand format. Pour les nuls, 2017.
9. Adeline CREPIEUX, « Introduction à l'informatique et à la programmation », Cours de DEUG U1, Université de la méditerranée, 2002.
10. Hugo ETIEVANT, « Cours de Turbo Pascal 7.0 : Le cours aux 100 exemples », 2004.
11. Support de cours en Informatique 02, Algorithmique et programmation, li Wided, Universite Larbi Tebessi de Tébessa, 2022.
12. Philipe TRIGANO, Dominique LENNE, « Algorithmique et programmation », Université de Technologie de Compiègne, 2008.
13. ChristopheDARMANGEAT,«Algorithmiqueetprogrammationpour nonmatheux : Cours complet », Université Paris 7, 2008.