Chapter 5: Arrays and Strings

1. **Some notions:**(reminder)

1.1. Identifier:

- An identifier designates the name of a variable, constant, data type, procedure or function…

1.2. Variable:

- A variable has a name, a type, and a value.

1.3. Kind :

- The data can be simple or structured types, in addition there is the possibility of defining new data types.
    - ✓ Simple types:

  Example : integer, real, character, boolean

    - ✓ Structured Types:

Example: Arrays, String, Record…

2. Arrays

- An array is a data structure grouping together a fixed number of variables of the same type.

2.1. Vectors: (one-dimensional array)

<u>Statement :</u>To declare a vector, we use the following syntax:

Nom_vect **[**size**]**: **array of** type

1) **We can specify the size by a positive integer:**

V [20]: array of integers;

2) **Or using positive integer constant:**

CONST n←10;
V [n]: array of integers;

<u>Representation of a Vector:</u>

12.5	3.9	0.8	1.13	2.0	0.0	5.0	1.2	0.1	0.5
i=1	i=2	i=3	i= 4	i=5	i=6	i=7	i=8	i=9	i=10

V[5] = 2.0

The index can be:

- A Value: V [5]
- An integer variable : V [i]

- An expression of integer type : V [i*2+1]

Example :

Write an algorithm that reads the averages of 25 students, then calculates the difference between the average of each student and that of the group average?

```
ALGORITHM Exp_Vect
CONST n ← 25;
VMOY [n]: array of real;
i: integer;
SMOY, MOYG: real;
Begin
// Load (read) the array

For i = 1 to n Do
  Write ("give the average of the student N°", i)
  Read (VMOY [i])
End For

// Calculate the group average

SMOY ← 0;
For i = 1 to n Do
  SMOY ← SMOY+VMOY[i] ;
End for
MOYG ← SMOY/N;
Write ("the average of the group is ", MOYG)

/*Calculate the difference between the average of the group and that of the student*/

For i = 1 to n Do
Write ("the difference of the average of the group and that of the student", i, "is=",
MOYG - VMOY[i]);
End for
END.
```

We can write the first two loops in one; to simplify this algorithm.

Noticed :

- The size of an array is fixed and therefore cannot be changed in a program: this results in two faults:
  - ✓ If you limit the size of an array too much, you risk overflowing.
  - ✓ The reserved memory space is insufficient to receive all the data

2.2. Search methods in a vector:

a) Finding the maximum of a vector:

- It consists in defining the largest element of a vector
- For that, we must traverse the vector by preserving with each iteration the largest element obtained,
- At the end, we obtain the maximum of all the elements.

```
Algorithm Search_max
 Const sizeM ←100; // the maximum size of the array
Vect [sizeM]: real arrays;
Max: real;
i,n: integer; // n represents the actual number of elements
Begin
//after reading n which represents the number of elements that we are going to read
//It is assumed that the n elements of the vector have already been read.
Max ← vector[1]; //we assume that the first element is the maximum
For i =2  to n do // we start from the 2nd element
  If vector[i] > max then
   Max ← vector[i];
  End if
End for;
Write ('the maximum is ', Max);
END.
```

b) Sequential search:

- ➢ One of the first operations on the arrays is the search for an element, its number of appearance, its or their positions.
- ➢ To do this, we must traverse the entire vector element by element and compare it with the value of the element to be sought.

Example:

1. Find the position of the first occurrence of the element 5 in vector V containing n integer elements.

```
Algorithm search1
Const n←10;
V[n]: Array of integer;
i: integer;
```

```
Begin
// assume that the elements of the vector have already been read.
// Find the position of the first occurrence of element 5
i←1;
While (i<= n and V[i] ≠5) do
  i←i+1;
end while
 If (i>n) then
  Write (“Element not found”);
 else
  Write ("The position of the element is:", i);
 End if
END.
```

2. Find the number of occurrences of element **5** in a vector V containing **n** elements, as well as the **positions** of the occurrences of this element?

```
Algorithm search2
Const n←10;
V[n]: Array of integer;
i, nba: integer;
Begin
//read the elements of the array
For i = 1 to n Do
 Write ("give the element N°", i);
 Read (V [i]);
End for
// End of loading
i←1; count←0;
While (i<=n) do
  If ( V[i] = 5 ) then
      count ←count+1;
      write ("the position of occurrence 5 is", i);
  end if
  i←i+1;
end while
```

```
Write ("the number of occurrences of 5 is:", count);
END.
```

c) Dichotomous search:

- This type of search is performed in an **ordered** array:

1) The array is **divided** into **two** roughly equal parts,
2) We compare the value to look for with the element in the **middle**,
3) If they are not equal, we are interested only in the part containing the desired elements and we abandon the other part.
4) We repeat these 3 steps until we obtain the value or we have only one element to compare.

Application :

We assume that we have a vector V of n elements. We want to find the value **Val**?

```
Algorithm rech_dich
Const n←100;
V[n]: Array of integer;
Iinf, Isup, Imil, Val: integer;
Found: Boolean;
Begin
Iinf ← 1; Isup ← n;
Found ← false;
While ((Iinf <= Isup) and (Found = false )) Do
   Imil ← (Iinf+Isup) div 2;
  If (V[Imil] = Val) Then
     Found ← true;
  Else
   If (V [Imil] < Val) Then
     Iinf ← Imil + 1;
   Else
     Isup ← Imil -1;
   End if
 End if
end while
If (Found = true) Then
  Write (Val, "exists at position", Imil);
```

```
Else
  Write (Val, "does not exist in V");
End if
END.
```

2.3. Matrices :(two-dimensional array)

Declaration: we have three ways to declare a matrix:

1) By tow positive integer values

M [5, 10]: array of **real**;

2) By tow positive integer constants

CONST **n**←5, **m**←10;
M [**n, m**]: array of integer;

Representation of a Matrix:

	D=1	D=2	D=3	D=4	D=5	D=6	D=7	D=8	D=9	D=10
I=1	-4	3	14	6	67	4	2	0	7	2
I=2	1	2	3	4	5	6	7	8	9	10
I=3	9	9	3	87	76	5	2	2	2	1
I=4	1	3	2	4	- 5	6	7	8	9	4
I=5	9	9	7	8	9	-7	-1	3	5	17

The element of index [i , j] is that of the intersection of row i with column j; M[4,5] is -5

Example :

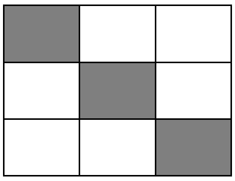
Let Mat(n,m) be a matrix of **n** x **m** real elements. Write an algorithm that calculates the largest (max) and smallest (min) elements of the matrix?

```
Algorithm max_min
Const n←10, m←12;
Mat [n, m]: real array;
max, min: real;
i, j: integer;
Begin
//Read the elements of the matrix
For i = 1 to n do
  For j = 1 to m do
    Read (mat[i,j]);
  End for
End for
```

```
// calculate from largest (max) and smallest (min)
max← mat[1, 1]; min←mat[1.1];
For i = 1 to n do
 For j = 1 to m do
    If (mat[i, j] >max) then
      max ← mat[i, j];
    end if
    If (mat[i][j] <min) then
      min ← mat[i, j];
    end if
  End For
End For
Write ("the largest value of the matrix", max);
Write ("the smallest value of the matrix", min);
END.
```

Noticed :

- ***square matrix***: a matrix whose number of rows is equal to the number of columns.
- Such a matrix has a ***main diagonal*** (all elements for which i=j).
- Elements above the diagonal have their indices i<j and those below the diagonal have their indices i>j.



3. Strings:

- A String is a sequence of characters, that is to say a set of symbols belonging to the character set, defined by the ASCII code, UTF8 etc.
- Some languages (Pascal, Java, Basic...) have a real string type (String).
- In the C++ language, there is no type of variable for strings as there is for integers (int) or for characters (char).
- The strings are in fact stored in an array of char whose end is marked by a Null character, with value 0 and represented by the character '\0'.

Example :

- In memory, the string "GOOD MORNING" is represented as follows:

G	O	O	D	M	O	R	N	I	N	G	\0

- Everything after the character '\0' will be ignored

3.1. string declarations:

- The declaration of a string is as follows:

  <*Identifier*>: String**;**

Example :

S: String; // S is of type string with a maximum size of 255 characters.

3.2. Reading and writing strings:

- In our course, we will use the following notation for reading (resp.) displaying strings:
  - **Reading :**Read (S);
  - **Display :**Write (S)
- You can display several adjoining strings using the +.

Example :

```
Algorithm Exp_string
        S, R, T: chain;
Begin
        S← ' Hello ' ;
        R←'ladies';
        T←'and gentlemen';
        To write(S+R+T); // Will print 'Hello ladies and gentlemen'
END.
```

Noticed : The + operator represents the concatenation and not the sum.