First Year Engineer in computer science

# CHAPTER 7:

# Arrays

2023-2024

Sid Ahmed BERRABAH

# Introduction



- Imagine that in a program, we need 10 values at the same time (for example, grades to calculate an average).
  - Obviously, the only solution we have at the moment is to declare ten variables, called N1, N2, N3, etc.
  - After, a succession of ten distinct 'read' instructions, we will have:

    Average = (N1+N2+N3+N4+N5+N6+N7+N8+N9+N10)/10

I

- Imagine now that we need a few hundred or a few thousand values!!!!
  - ✗ In this case, the use of separate variables would be completely impractical.

➔ **Solution: use arrays**

# Definition



- An array is a data structure that stores a collection of values of the same type. Each value in the array is identified by a unique index, which is a number that starts at 0.
- Arrays consist of contiguous memory locations.
- To access an element of an array, we use the name of the array followed by the index of the element in brackets.

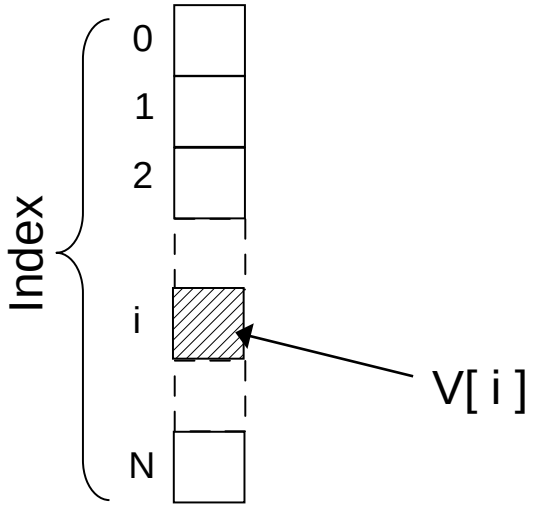
  **Example:**

  **quotes[i]** represents the value at position i in the array **quotes**

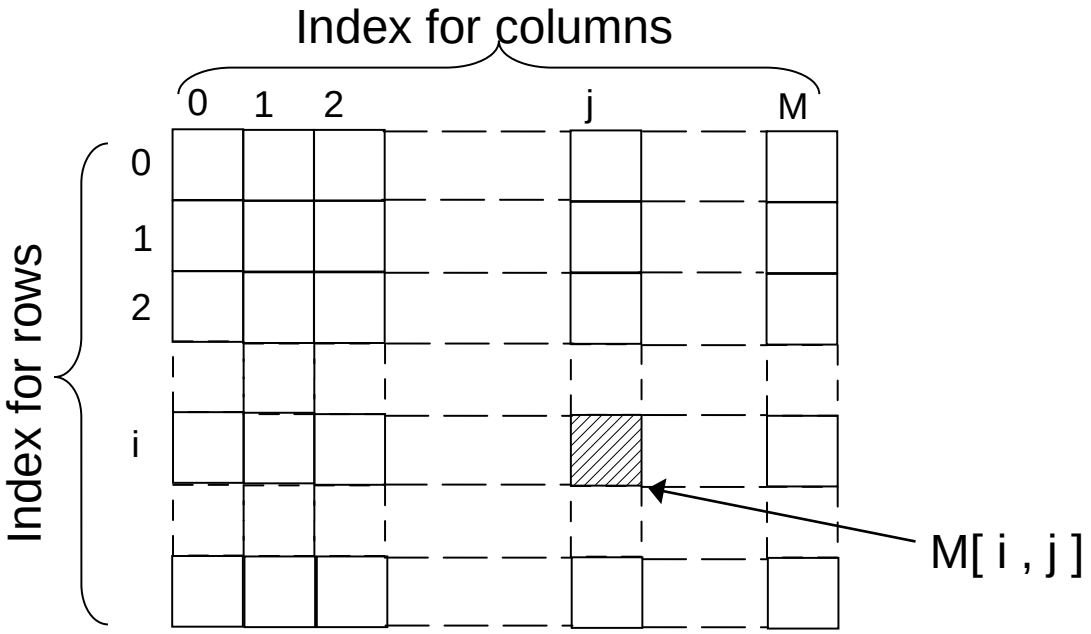
- The most used arrays are:
  - One dimension arrays (vectors)
  - Two dimensions array (matrices)

# Definition






One dimension array




Two dimensions array

- For a 1 dimension array, V[i] represents the value at position i of the array V.
- For a 2 dimensions array, M[i,j] represents the value at row I and column j of the array M.

# Definition



An array is defined by four things:

1. **Name**: The name of the array is an identifier, which is a unique name that is used to refer to the array.

2. **Dimensions**: The number of dimensions of an array determines how many indices are used to access the elements of the array. For example, a 1D array has one index, a 2D array has two indices, and so on.

3. **Size**: The size of an array determines the number of elements that the array can store. The size is specified by a list of numbers, one for each dimension of the array.

4. **Data type**: The data type of an array determines the type of data that the array can store. For example, an array of integers can store only integers.

# Declaring 1D Arrays



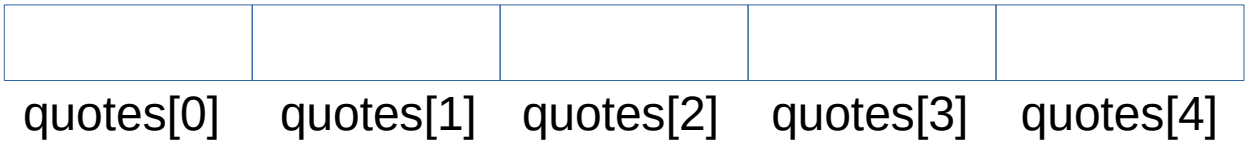
To declare a one dimension (1D) array in C, we specify the type of the elements and the number of elements required by an array as follows:

**type** arrayName [ arraySize ];

- **type** is the type of the elements of the array. type can be any valid C data type.
- arrayName is the identifier of the array
- arraySize is a positive integer number defining the number of the array elements.

**Example :** float quotes[5] ;

The array quotes can hold 5 float values



# Initializing Arrays



- We can initialize an array at the time of declaration by placing the values that you want to store in the array between braces, separated by commas..

**Example :**

```
int x[5] = {1, 2, 3, 4, 5};
char car[3] = {'a', 'b', 'c'};
```

- It is also possible to only mention the first values when initializing an array.

```
int x[5] = {1, 2, 3};
```

In this case, the remaining elements of the array will be initialized to 0.

# Initializing Arrays

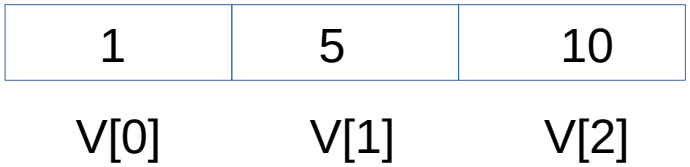


It is possible to omit the size of the array. In this case an array just big enough to hold the initialization is created.

**Example :**

```
int V[] = {1, 5, 10};
```

In this example, an array of 3 integers is created and its elements are initialized to the given values between braces .



# Accessing Array Elements



- you can use an element of an array in any way that you would use a variable of the same type.
- For example:
  - you can assign a value to an element of an array:

```
float X[10];   // declare an array of 10 float elements
X[1] =  20.65; // assign the value 20.65 to the 2nd element of X
```

  - you can read the value of an element of an array:

```
Scanf("%f", &X[5]); //read the value of the 6th element of X
```

  - you can use an element of an array in an expression.

```
X[i]= 3*X[j+1]+2;
```

# iterate through the elements of an array



- you can use a loop to iterate through the elements of an array, one at a time. The loop will use a counter to keep track of the current index of the element being traversed.

**Example** :

- assign the same value to all the elements of an array:

```
int x[10] , i;
for (i = 0; i < 10; i++)
        x[i] = 0;
```

- Reading all the elements of the array:

```
for (i = 0; i < 4; i++) {
        printf("give the value of the element %d : ", i);
        scanf("%d", &x[i]);
}
```

# Important notes on arrays in C:



- It is impossible to manipulate an array globally.

```
int x[5], y[5] ;
x = y ;  // forbidden and will generate error
```

- You cannot access or modify an element of an array that has an index that is negative or greater than the length of the array. The following code will generate an error:

```
int x[10];
x[15] = 0;  //forbidden and will generate error
x[-1] = 0;  //forbidden and will generate error
```

# Exercise 1



- Write a program that reads 5 values and then displays their squares.
- Write a program that calculates the sum of all the elements of an array.
- Write a program to find the largest element of an array as well as its position.

# Exercise 2



Write a C program to read N real elements of a vector V. The program must then replace the negative elements of V with 0 and display the result.

Example output (The numbers in bold are entered by the user):

Enter the number of elements of vector V: **8**

Enter the 8 elements of vector V:

V[1]= **-98.76**

V[2]= **30.55**

V[3]= **-43.78**

V[4]= -**33.54**

V[5]= **20.22**

V[6]= -**0.44**

V[7]= **45.05**

V[8]= -**5.9**

The new vector V = [0.00, 30.55, 0.00, 0.00, 20.22, 0.00, 45.05, 0.00]

# Exercise 3



Write a C program to read N real elements of a vector V. The program builds another vector V2 that contains the difference between the elements of V and the mean value of the vector V. The program must then display the new vector V2.

Example output (The numbers in bold are entered by the user):

Enter the number of elements of vector V: **6**

Enter the 6 elements of vector V:

V[0]= **98.76**

V[1]= **67.55**

V[2]= **105.86**

V[3]= **33.54**

V[4]= **204.22**

V[5]= **90.44**

The vector V2 = [-1.30 , -32.51 , 5.80 , -66.52 , 104.16 , -9.62 ]

# Exercise 4



Write a C program to read N real elements of a vector V and then it builds a second vector V2 whose elements are the decimal parts of the elements of the vector V.

Example output (The numbers in bold are entered by the user):

Enter the number of elements of vector V: **7**

Enter the 7 elements of vector V:

V[0]= **98.76**

V[1]= **67.55**

V[2]= **105.86**

V[3]= **33.54**

V[4]= **204.22**

V[5]= **45.05**

V[6]= **59**

Le new vector V2 = [0.76, 0.55, 0.86, 0.54, 0.22, 0.05, 0.00]

# Passing Arrays as Function Arguments



When we pass the name of an array as an actual argument to a function call, we ultimately transmit the address of the array to the function. This allows the function to perform all the desired manipulations on its elements, whether it involves using their values or modifying them.

# Passing Arrays as Function Arguments



**Example 1 :** A function that sets the value 1 in all elements of a 10-element array.

```
void set_to_one(int t[10])
{
    int i ;
    for (i=0 ; i<10 ; i++) t[i] =1 ;
}
```

Here is an example of how to use the function:

```
int array[] = {1, 2, 3, 4, 5};
set_to_one(array);

for (int i = 0; i < 10; i++) {
    printf("%d\n", array[i]);
}
```

Output:
1 1 1 1 1 1 1 1 1 1

**When we call the function, the address of the array is transmitted to the function.**

# Passing Arrays as Function Arguments



The compiler does not need to know the size of the array in the function declaration. This makes it easy to create a function capable of working with an array of any dimension, as long as you pass its size as an argument.

**Example:** The previous **set_to_one()** function can be defined as:

```
void set_to_one (int t[] , int n)
{
    int i ;
    for (i=0 ; i < n ; i++) t[i] =1 ;
}
```

Here is an example of how to use the function:

```
int V1[10], V2[20];
set_to_one(V1, 10);
set_to_one(V2, 20);
```

# Exercise 5



Write a C function **reverse()** that takes as input a vector T of real numbers and returns another vector containing the elements of T in reverse order.

Use this function in a main program to reverse the order of the elements of a vector V entered by the user.
The output should be as shown in the following example.

```
Enter the 5 elements of the vector V:
V[0]= 9.8
V[1]= 5.5
V[2]= 8.6
V[3]= 3.3
V[4]= 2.1

The new vector is:  [ 2.1   3.3  8.6   5.5   9.8]
```

# Exercise 5 (solution)



```
#include <stdio.h>
#define N 7

void reverse(float T[], float T2[], int n);    // function to reverse the elements of T in T2
void readArray(float T[], int n);    // function to read the N real elements of an array
void printArray(float T[], int n);  // Function to display the N elements of an array

int main()
{
    float V1[N], V2[N] ;
    readArray( V1, N );
    reverse(V1, V2, N) ;
    printf("The entred array : "); printArray(V1, N) ;
    printf("The reversed array : "); printArray(V2, N) ;
   return 0;
}
```

# Exercise 5 ( solution )



```
// function to read the N real elements of an array
void readArray((float T[], int n)
{
    for(int i = 0; i < n; i++) {
        printf("element %d : ", i);
        scanf("%f",&T[i]);
    }
}

 // Function to display the N elements of an array
void printArray(float T[], int n)
{
    printf("[ ");
    for(int i = 0; i < n; i++) {
        printf("%.2f\t", T[i]);
    }
    printf("]\n");
}
```

```
* function to reverse the elements of
T in T2 */
void reverse(float T[], float T2[], int n)
{
    for(int i = 1; i < n; i++) {
        T2[i] = T[n-1-i];
    }
}
```

# Exercise 6



Write a C function that takes as input a vector T of integers and returns another vector with the even elements of T placed at the beginning.

Use this function in a main program.

The output should be as shown in the following example.

```
Enter the 7 elements of the vector V:
V[0]= 9
V[1]= 5
V[2]= 8
V[3]= 3
V[4]= 12
V[5]= 16
V[6]= 3

The new vector is:  [ 8   12   16   9   5   3   3 ]
```

# Exercise 7



Write a C function that takes as input a vector T of real numbers and searches for the smallest element and exchanges its place with the first element of the vector.

Use this function in a main program.

The output should be as shown in the following example.

```
Enter the 7 elements of the vector V:
V[0]= 9
V[1]= 15
V[2]= 8
V[3]= 3
V[4]= 22
V[5]= 16
V[6]= 13

The new vector is:  [ 3   15    8    9    22    16    13  ]
```

# Exercise 7 ( solution )



```
//  Function to exchange the min element with the first element
void exchangeMin(float T[], int n)
{  float min = T[0];
   int k = 0, i;
   for(i = 1; i < n; i++) {
      if(T[i] < min) {
         min = T[i];
         k = i;
      }
   }
   T[k] = T[0];
   T[0] = min;
}
```

# Exercise 8



Write a C function that takes as input a vector T of real numbers and searches for the smallest element and places it at the beginning of the vector.

Use this function in a main program.

The output should be as shown in the following example.

```
Enter the 7 elements of the vector V:
V[0]= 9
V[1]= 15
V[2]= 8
V[3]= 3
V[4]= 22
V[5]= 16
V[6]= 13

The new vector is:  [ 3   9   15   8   22   16   13  ]
```