Lab 07

Objectives

After performing this lab, students will be able to

- Understand arrays and use them in programs

Arrays

In C++, an array is a variable that can store multiple values of the same type. Because of this, array is sometimes called as the ***collection of homogenous (same type of) data items***. For example, suppose we need to save the **age** of 37 persons Then, instead of creating 27 separate variables, it will be easy and convenient to simply create an array. The values stored in an array are also called its ***elements*** or ***members***.

Declaring an array

The general syntax is for declaring an array is:

```
dataType arrayName[arraySize];
```

For example

```
int age[37];
```

Here

- **int** - is type of elements to be stored (means only integers can be stored in this array)
- **age** - is name of the array (as array is a variable, all the rules applied to naming a normal variable will also apply here)
- **37** - is size of the array

One thing to keep in mind is that one array can store many values, **but** it actually is a single variable having only one ***name***. So, if it has a single name, then how it saves or allow access to multiple values? The answer is through **indices** (multiple of index).

Accessing Elements of an Array

Each element in an array is associated with a number. The number is known as an array ***index***. We can access elements of an array by using those indices.

```
// syntax to access array elements
array[index];
```

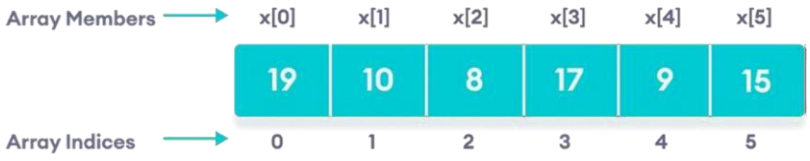
Suppose we have an array named **x**, then it's represented as:

Some important things to keep in mind

- The array indices start with 0. Meaning **x[0]** is the first element stored at index 0.
- If the size of an array is n, the last element is stored at index (n-1). In this example, **x[5]** is the last element.
- Elements of an array have consecutive addresses.
- For example, suppose the starting address of **x[0]** is 2120d. Then, the address of the next element **x[1]** will be 2124d, the address of **x[2]** will be 2128d and so on.
- Here, the size of each element is increased by 4. This is because we consider the size of an **int** as 4 bytes.

Array Initialization

In C++, it's possible to initialize an array during declaration. For example:



Another method to initialize array during declaration is:

```
// declare and initialize an array
int x[] = {19, 10, 8, 17, 9, 15};
```

Here, we have not mentioned the size of the array. In such cases, the compiler automatically computes the size.

In C++, if an array has a size n, we can store upto n number of elements in the array. However, what will happen if we store less than n number of elements. For example,

```
// store only 3 elements in the array
int x[6] = {19, 10, 8};
```

Here, the array x has a size of 6. However, we have initialized it with only 3 elements. In such cases, the compiler assigns random values to the remaining places. Oftentimes, this random value is simply 0.

There are different ways of initializing an array while creating it.

```
// Access particular index. Like, change 4th element to 9
mark[3] = 9;
```

```
// take input from the user
// store the value at third position
cin >> mark[2];
```

Ranged for loop

C++11 introduced the **ranged for** loop. This **for** loop is specifically used with collections such as **arrays** and **vectors**. For example:

```
// initialize an int array
int num[] = {1, 2, 3, 4, 5};

// use of ranged for loop
for (int var : num) {
    cout << var << "  ";
}
cout << endl;
```

Output:

```
1 2 3 4 5
```

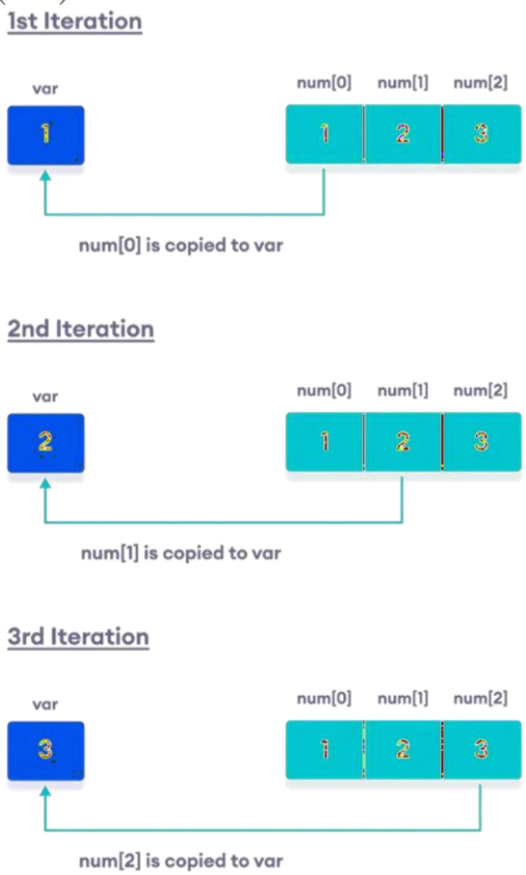
Here, the ranged for loop iterates the array **num** from beginning to end. The **int** type variable **var** stores the value of the current array element in each iteration. Means, the value of **var** is automatically updated in each iteration to contain the current array element. Its general syntax is:

```
for (rangeDeclaration : rangeExpression) {
    // code
}
```

In the example discussed earlier,

- **rangeDeclaration** - int var
- **rangeExpression** - num

For simplicity, you can consider **rangeDeclaration** as *the variable to hold the array value in each iteration* and **rangeExpression** as *the name of the array being used/accessed*.



Multidimensional Arrays

A multidimensional array is also called as ***an array of array***. Means one array contains the other array (or one array is part of (resides in) another array). Multidimensional means many dimentions like 2D (two-dimensional), 3D (three-dimensional), and so on. A 2D array can be considered as a ***table*** or a ***matrix***. One array denotes the ***rows*** and the other denotes the ***columns***. For example:

```
int x[3][4];
```

Here, **x** is a two-dimensional array. It can hold a maximum of 12 elements. We can think of this array as a table with 3 rows and each row has 4 columns as shown in the figure.

	Col 1	Col 2	Col 3	Col 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Three-dimensional arrays also work in a similar way. For example:

```
float x[2][4][3];
```

This array x can hold a maximum of 24 elements.

We can find out the total number of elements in the array simply by multiplying its dimensions:

```
2 x 4 x 3 = 24
```

2D array

```
int  test[2][3] = { {2, 4, 5}, {9, 0, 19}};
```

This array has 2 rows and 3 columns, which is why we have two rows of elements with 3 elements each.

	Col 1	Col 2	Col 3
Row 1	2	4	5
Row 2	9	0	19

Similarly, we can initialize a 3D array as well

```
int test[2][3][4] = {
                        { {3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2} },
                        { {13, 4, 56, 3}, {5, 9, 3, 5}, {5, 1, 4, 9} }
                    };
```

Notice the dimensions of this three-dimensional array. The first dimension has the value 2. So, the two elements comprising the first dimension are:

```
Element 1 = { {3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2} }
Element 2 = { {13, 4, 56, 3}, {5, 9, 3, 5}, {5, 1, 4, 9} }
```

The second dimension has the value 3. Notice that each of the elements of the first dimension has three elements each:

```
{3, 4, 2, 3}, {0, -3, 9, 11} and {23, 12, 23, 2} for Element 1.
{13, 4, 56, 3}, {5, 9, 3, 5} and {5, 1, 4, 9} for Element 2.
```

Finally, there are four int numbers inside each of the elements of the second dimension:

```
{3, 4, 2, 3}
{0, -3, 9, 11}
... .. ...
... .. ...
```

Taking input in 2D array

Imagine there is a 2D array named ***numbers***. Then, we can save the values into it by taking input from a user as:

```
for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 3; ++j) {
        cin >> numbers[i][j];
    }
}
```

Exercises

1. Write a C++ program that asks the user to input an N number of values in an array (means, the size of the array as well as all the values should be input by the user) and then displays those values in reverse order.

Sample Output

```
Enter the size of array: 6
Enter value number 1: 34
Enter value number 2: 11
Enter value number 3: 88
Enter value number 4: 56
Enter value number 5: 23
Enter value number 6: 77
The numbers in reverse order are:
77  23  56  88  11  34
```

2. Write a program that should ask the user to input the size of an array and then take that much number of values as input from the user and save them in the array. Finally, print the sum of all those values of the array.
3. Write a program to declare and initialize an array with 10 values (means give the values to array elements within the code, no need to take input from the user) and then display the average of the values of that array.
4. Write a program to copy the elements of one array into another array. And then, print the values of both of the arrays. You may give fixed values to the first array elements (means, it is not necessary to get input from the user).

Sample Output

```
The values of the first array are:
3  41  88  11  2  69
The values of the second array are:
3  41  88  11  2  69
```

These are the same values but internally they should be from two arrays. The values of first the array should be copied in the second (you should not give fixed values to the second array).

5. Write a program that gets array size and elements as input from a user and then prints only the EVEN values that are present in the array.

Sample Output

```
Enter the size of array: 7
Enter value number 1: 13
Enter value number 2: 42
Enter value number 3: 66
Enter value number 4: 77
Enter value number 5: 89
Enter value number 6: 50
Enter value number 7: 35
The EVEN numbers are:
42  66  50
```

6. Write a program that gets array size and elements as input from a user and then prints only those values that are present on ODD indices (whose index number is ODD). **Sample Output**

```
Enter the size of array: 8
Enter value number 1: 32
Enter value number 2: 55
Enter value number 3: 21
Enter value number 4: 90
Enter value number 5: 18
Enter value number 6: 74
Enter value number 7: 43
Enter value number 8: 38
The values on the ODD indices are:
55  90  74  38
```

7. Write a program that gets the array size and elements as input from the user. Then, create another array of the same size as the first array and copy the values of the first array in reverse order in the second array (means the value on the first index of the first array should be saved on the last index of the second array, and so on). Then, display the values of both arrays, as shown below:

Sample Output

```
Enter size of array: 6
Enter value 1: 10
Enter value 2: 20
Enter value 3: 30
Enter value 4: 40
Enter value 5: 50
Enter value 6: 60

Values of the first array are
10 20 30 40 50 60

Values of the second array are
60 50 40 30 20 10
```

8. Write a program where you should sum the elements of two same-sized arrays and the result should be saved in the third array. Then, print the values of all of the three arrays.

   The values of the first two arrays may be given fixed (no need to get input) but each value of the third array should be the sum of respective values from the first and the second arrays.

Sample Output

```
The values of the first array are:
21  8  11  4  33  50
The values of the second array are:
7  35  6  23  14  10
The values of the third array are:
28  43  17  27  47  60
```

9. Write a program that creates two integer arrays of sizes N and M, respectively. These arrays can be initialized within the code (no need to get values as input from the user). Then, create a third array of size N+M (i.e., whose size should be equal to the sum of the sizes of the first and the second array) and copy the elements of both (first and second) arrays into the third array. Finally, print the values of all the three arrays, as shown below:

Sample Output

```
Values of the first array are
23 10 15 7 42

Values of the second array are
45 77 12 89 53 20

Values of the third array are
23 10 15 7 42 45 77 12 89 53 20
```

Explanation: Here, the first array contains 5 elements and the second array has 6 values. The third array contains 11 values (equal to the total number of values from both arrays). First 5 values in the third array are copied from the first array and its 6 to 11 values are copied from the second array.

10. Write a program that handles students' records/mark sheets. First, ask the user to input the number of students for which records are to be entered (this value will be used as the **size** of different arrays to be created in the next steps). Then, create 8 arrays having the size gotten in the previous step.
    a. One array should save the **names** of the students. The values for this array will be input by the user (i.e., you will ask the user to enter the names for all the students).
    b. Four arrays should save the students' **marks** in four subjects. For this array also, the values will be obtained from the user (i.e., you will ask the user to input the marks for each subject for every student).
    c. One arrays should save each student's **obtained marks**. The value of this array will NOT be input by the user. Instead, it will be computed as the sum of marks gotten in the four subjects by each student and then saved in this array.
    d. One array should save the **percentages** of students. Again, values for this array will also NOT be entered by the user. Instead, they will be computed within the program by looking at the **obtained marks** of each student (calculated in the previous step) and the **total marks** (assume total marks are 400 for the four subjects).
    e. The last array should save the **grades** of students. Like the previous two arrays (obtained marks and percentage), the values of this array also will NOT be entered by the user. Instead, its values will be decided in the code by looking at every

student's **percentage**. We are assuming that a grade will be one-character long (i.e., A, B, C etc.). The rules for deciding a grade are:

i. If the percentage is between 80 and 100 (inclusive), the grade will be **A**. ii. If the percentage is between 70 and 79, the grade will be **B**. iii. If the percentage is between 60 and 69, the grade will be **C**. iv. If the percentage is less than 60, the grade will be **F**.

At the end, print the results in the format shown below:

```
================================================================================
Name     Programming    Math    English    P.S.    Total Obtained    Percentage    Grade
================================================================================
Abc      80             70      75         85          310           77.5           B
Xyz      85             75      80         85          325           81.25          A
...      ...            ...     ...        ...         ...           ...            ...
```

11. Write a program that gets array elements as input from a user and then prints only the smallest value.

```
Enter the size of array: 7
Enter value number 1: 45
Enter value number 2: 21
Enter value number 3: 89
Enter value number 4: 66
Enter value number 5: 8
Enter value number 6: 73
Enter value number 7: 32
The smallest value is: 8
```

12. Write a program that gets array elements as input from a user and then prints only the largest value.

Sample Output

```
Enter the size of array: 8
Enter value number 1: 32
Enter value number 2: 61
Enter value number 3: 47
Enter value number 4: 8
Enter value number 5: 2
Enter value number 6: 15
Enter value number 7: 24
Enter value number 8: 99
The largest value is: 99
```

13. Write a program that gets array elements as input from a user and then asks the user to input any value to search from the array. After that, if the value entered by the user is present in the array, it should display the index where that value is present. If the value is NOT present in the array, then it should print a message saying that the given value is not present in the array.

Sample Output 1

```
Enter the size of array: 7
Enter value number 1: 87
Enter value number 2: 12
Enter value number 3: 56
Enter value number 4: 79
Enter value number 5: 23
Enter value number 6: 45
Enter value number 7: 16
Enter the value which want to search from this array: 23
23 is present at the 4 index
```

Sample Output 2

```
Enter the size of array: 6
Enter value number 1: 66
Enter value number 2: 22
Enter value number 3: 55
Enter value number 4: 11
Enter value number 5: 77
Enter value number 6: 99
Enter the value which want to search from this array: 35
35 is NOT present in the array
```