Lesson:

Arrays in C++

PW SKILLS

Pre-Requisites

- Basic C++ syntax
- Variables
- Loops
- Functions

List of Concepts Involved

- Array Introduction
- Declaration
- Creation
- Array Types
- Operations
- Taking Input in an array
- Problems based on an array

Suppose we have to write a program in which can accept salaries of 50 employees. If we solve this problem by making use of variables, we need 50 variables to store employees' salaries. Managing these 50 variables is not an easy task and will make the program a complex and lengthy one. This problem can be solved by declaring 1 array having 50 elements. Did you notice how convenient the scenario became? If arrays are that useful, why not learn about them?

Topic: Array Introduction - What is an array

Array is a *data structure* (storage, used to store and organize data) to store a group or collection of (homogenous data) items, sequentially, inside memory. Homogenous data is data of the same type, for example, integer or string or floating number, etc. Each array element is identified with an index number.

- The indexing of an array is 0-based i.e. the first element is at index 0, second element is at index 1 and so on. The desired array element can be directly and individually accessed using these numbered indexes.
- The memory allocation in arrays is contiguous i.e. elements are stored one after another.
- An array can be single-dimensional or multi-dimensional based on the utility and application.

Topic: Array Declaration and Creation

Array declaration names the array and specifies the type of elements in it. It can also define the number of elements in the array.

- Syntax for creating a new Array in C++ is

```
data-type array-name[array-size];
```

Example:

```
int arr[10];
```

- Array Literal
  With the help of curly braces, we can initialize the array and add value to it during initialization without defining the size as we did in the previous syntax.

  Example :
  ```
  int arr[] = { 1,2,3,4,5,6,7,8,9,10 };
  ```

You might be wondering that why arrays should be used when we already have the provision of creating as many variables as we want/need.

Look at the scenarios mentioned below which will clear the air around utility of arrays for our good.

Case 1: Scenario without Array
In the example below, we are using five different variables to save our elements one by one.

```
string colour1 = “Red”;
string colour2 = “Green”;
string colour3 = "Blue”;
string colour4 = “Yellow”;
string colour5 = “Purple”;

// To print all the elements to the console
cout << colour1 << endl;// Red
cout << colour2 << endl;// Green
cout << colour3 << endl;// Blue
cout << colour4 << endl;// Yellow
cout << colour5 << endl;// Purple
```

Output:
Red
Green
Blue
Yellow
Purple

Woah ! It was cumbersome and may result in manual mistakes. Isn’t it ?

Case 2: Using the array concept
Now, we will implement the same scenario using an array.

```
// To create an array of colours to store values
string colours[] = {“Red”, “Green”, “Blue”, “Yellow”, “Purple”};

// To print all the elements entered in a array to the console
for (int i = 0; i < 5; i++)
{
 cout << colours [i] << endl;
}
```

Output:
Red
Green
Blue
Yellow
Purple

Thats it ?! Yes, absolutely !
Did you see how arrays made the implementation so convenient and saved a lot of our coding time !

Let us understand how these approaches are different:

- With individual variable approach in case 1, each value is stored in different variables, which results in random memory allocation for each variable; whereas in case 2, when we used the concept of array, the values were stored in contiguous memory locations. Hence, handling and accessing data became extremely convenient.

Arrays have an unlimited potential if we use them correctly. Let us look at its types and see how versatile these can be to handle the most complex scenarios (which will be discussed in the forthcoming lectures)

Topic: Array Types

There can be many classifications in arrays but we will concentrate on the most widely used ones that are based on the dimension of data that it is handling.

1. **Single dimensional or one-dimensional array**
   When we have elements stored in a single dimension or sequentially or linearly. We can declare and allocate memory to a single-dimensional array using a single variable.
   Syntax for declaring an single dimension array -

```
data_type array_name [] = {element_0, element_1, element_2, element_3, ...
elementN};
```

Example :

```
string colours[] = {"Red", "Green", "Blue"};
// To print the elements in the console:
for(string colour : colours)
     cout << colour << “, “;
```

Output:
Red, Green, Blue

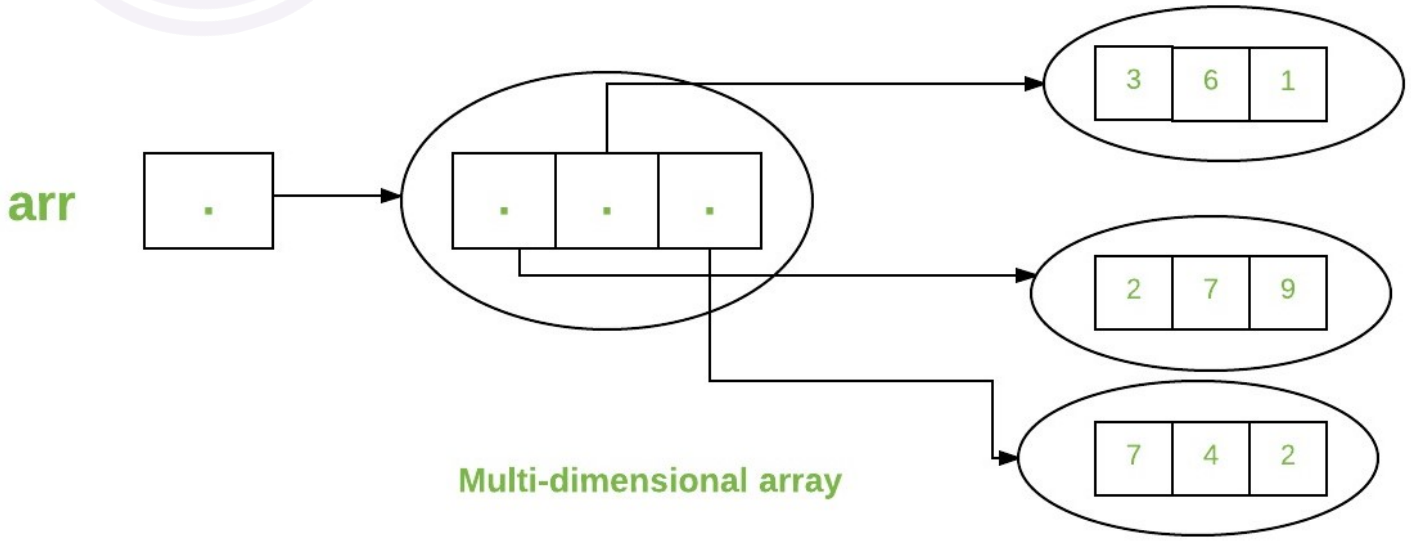
Example to create a single dimensional array:

```
// To create an Single-Dimension array:
int myArray[] = { 1, 2, 3, 4 , 5 };
// first element
cout << myArray[0] << endl;// to print 1
// second element
cout << myArray[1] << endl;// to print 2
// third element
cout << myArray[2] << endl;// to print 3
// fourth element
cout << myArray[3] << endl;// to print 4
// fifth element
cout << myArray[4] << endl;// to print 5
```

Output:
1
2
3
4
5

2. Multi dimensional Array:
A multidimensional array is simply an array that consists of two or more dimensions and is also commonly referred to as an array of arrays.

The diagram below will help you visualize the actual implementation structure of a multi- dimensional array.



To build a two-dimensional array, wrap each array in its pair of "[]" square brackets.
Look at the example below for clarity. Here we are creating a 2-dimensional array

```
int items[2][2] = {
{2, 3},
{5, 6},
};
```

Topic: Simple operations on arrays

Here, we are going to discuss the most basic operations that can be done on arrays .

a. Size of an array

```
string colours[] = {"Red", "Green"," Yellow"," Purple"}
int sz = colours.size();
cout << sz << endl; // prints 4
```

Note: Array index always starts from 0, which means the first element is stored at index 0 and the last element will be stored at index sz-1.

Looping through Array

There are many ways to iterate or loop over the array. The most common ways to do so are:

- **Using For Loop**
  We can iterate through the array using for loop as shown below.

```
// Creating an Array
string colours[] = {“Green”, “Red”, “Purple”, “Yellow”, “Blue”};
// Using loop to iterate over array
for (int = 0; i < colours.size(); i++)
 // Printing array elements using index
  cout << colours [i] << endl;
// Will print
// Green
// Red
// Purple
// Yellow
// Blue
```

Here, we are iterating over the array ‘colours[]’, using a for loop to print the elements.

- **Using For each loop**
  This loop might seem to be new to you but its very easy to understand. It helps in iterating over objects like string, array and so on. It is like the simple loops that we have studied with a little difference, which is actually a limitation. Let us look at the syntax first.

Syntax:

```
for (<data_type> <variable_name> : <array_name> )
{
  statement;
}
```

Limitation of for each loop:

- For each loop can only be used for traversing the whole array and not part of the array.

Let us see the example below to know how for each works.

```
int a[] = { 1, 2, 3, 4, 5, 6, 7, 8 };

        // iterating over an array
        for (int i : a) {
            // accessing each element of array
            cout << i << endl;
        }
```

Output:

1
2
3
4
5
6
7
8

Topic: Taking input in an array

To take input in an array, we simply declare it and use a 'cin' to take input with the help of a loop.

- **Using a basic for loop**

```
for(int i =0 ; i < n; i++) {
 cin >> a[i];
}
```

- **Using while loop**

```
int i = 0;
while(i < n) {
  cin >> a[i];
  i++;
}
```

- **Using for each loop**

```
for(int &i : a) cin >> i;
```

Now that we are equipped with all the basic syntax and nuances of arrays, let us now move to some basic problems.

Topic: Arrays Problems

Let us now discuss some problems based on array application.

Problem 1: Calculate the sum of all the elements in the given array.
Code:

```
int sum = 0;
for(int i = 0; i < a.size(); i++) {
   sum += a[i];
}
```

Problem 2: Find the maximum value out of all the elements in the array.
Code:

```
int mx = a[0];
for(int i = 0; i < a.size(); i++) {
   if(a[i] > mx) mx = a[i];
}
```

Problem3: Search if the given element is present in the array or not and find the index. If not present then return the index as -1. (Linear Search)
Code:

```
int index = -1;
for(int i = 0; i < a.size(); i++) {
  if(a[i] == given_value) index = i;
}
```

That is all for this class ! See you in the next array lecture !! Keep learning ! Keep Exploring !!

Upcoming Class Teaser

- Vectors
- Operations on vectors