C++ Foundation with Data Structures

Lecture 6 : Arrays

What are arrays?

In cases where there is a need to use several variables of same type, for storing, example, names or marks of 'n' students we use a data structure called arrays. Arrays are basically collection of elements having same name and same data type. Using arrays, saves us from the time and effort required to declare each of the element of array individually.

Creating an array

The syntax for declaring an array is:

Data_type array_name [array_size] ;

Example :

```
float marks[5];
```

Here, we declared an array, marks, of floating-point type and size 5. Meaning, it can hold 5 floating-point values.

Similarly, we can declare array of type int as follows:

```
int age[10];
```

How are arrays stored?

The elements of arrays are stored contiguously, i.e., at consecutive memory locations. The name of the array actually has the address of the first element of the array. Hence making it possible to access any element of the array using the starting address.

Example:

```
int age[ ] = {10, 14, 16, 18, 19};
```

Suppose the starting address of an array is 1000,then address of second and third element of array will be 1004, 1008 respectively and so on.

10	14	16	18	119
1000	1004	1008	1012	1016

Accessing elements of an array

An element of array could be accessed using indices. Index of an array starts from 0 to n-1, where n is the size of the array.

Syntax for accessing array element is :

Array_name[index]

Suppose we declare array age of size 5.

```
int age[10];
age[2] = 4       //  stores value 4 at index 2
```

		4		
age[0]	age[1]	age[2]	age[3]	age[4]

Here the first element is age[0], second element is age[1] and so on.

Default values

Arrays must always be initialized. If the array is not initialized the respective memory locations will contain garbage by default. Hence, any operation on uninitialized array will lead to unexpected results.

Initializing array at the time of declaration

Elements in an array can be explicitly initialized to specific values when it is declared, by enclosing those initial values in curly braces {}.

Example

```
int age[5] = {5, 2, 10, 4, 12};
```

Alternatively,

```
int age[ ] = {5, 2, 10, 4, 12};
```

5	2	10	4	12
age[0]	age[1]	age[2]	age[3]	age[4]

If array is initialized like this -

```
int age[10] = {5, 2, 10, 4, 12};
```

Then, in memory an integer array of size 10 will be declared. That is, a continuous memory block of 40 bytes (to hold 10 integers) will be allocated. And first 5 values of age are provided, rest will be 0.

Here,

age[0] is equal to 5

age[1] is equal to 2

age[2] is equal to 10

age[3] is equal to 4

age[4] is equal to 12

And age[5] to age[9] is equal to 0.

sizeof operator

sizeof is an operator used to determine the length of the array, i.e., the number of elements in the array .

Example:

```
int main( ){
        int myArray[ ] = {5, 4, 3, 2, 1};
        int size = sizeof(myArray);
        cout << size;
        return 0;
}
```

Output :

```
20
```

Passing arrays to a function

Arrays can be passed to a function as an argument. When an array is passed as an argument, only the starting address of the array gets passed to the function as an argument. Since, only the starting address gets passed to the function as opposed to whole array, the size of the array cannot be determined in function using sizeof operator. Hence, the arrays that are passed as an argument to a function must always be accompanied by its size as another argument.

Syntax of the function call to pass an array :

function_name(array_name, array_size);

Syntax of the function definition that takes array as an argument :

```
return_type function_name(data_type array_name, int size, <other arguments>){
        //function body
}
```

NOTE : Square brackets '[ ]' are not used at the time of function call.

Example:

```
#include <iostream>
using namespace std
```

```
void printAge(int age[ ], int n){
        for(int i=0 ; i < n ; i++){
                cout << age[i] << endl;
        }
}
int main(){
        int age[5] = {11, 14, 15, 18, 20};
        printAge(age, 5);
        return 0;
}
```

Output:

```
11
14
15
18
20
```