Introduction to Arrays

Overview

An array is a collection of elements of the same type placed in contiguous memory locations that can be accessed randomly using an array's indices. They can store collections of primitive data types such as int, float, double, char, etc., of any particular type.

It is used to store multiple values in a single variable instead of declaring separate variables for each value.

Why do we need arrays?

We can use normal variables (v1, v2, v3, etc.) when we have a small number of objects, but if we want to store many instances, it becomes difficult to manage them with normal variables. The idea of an array is to represent many instances in one variable.

Using arrays saves us from the time and effort required to declare each element of the array individually.

Creating an array

To declare an array, define the variable type, specify the array's name followed by square brackets and specify the number of elements it should store.

Syntax:

```
type arrayName [ arraySize ];
```

Example:

```
// Array declaration by specifying size
int arr[10];
```

```
// Declare an array of user specified size
int n;
cin >> n;
int arr[n];

// Taking input in the array
for (int i = 0 ; i < n ; i++) {
    cin >> arr[i]
}
```

Example:

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

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

Dynamically creating array

In C++, we can create an array dynamically using the **new** keyword. The number of items to be allocated is specified within a pair of square brackets. The type name should precede this. The requested number of items will be allocated.

Syntax:

```
pointer_variable = new data_type;
```

Example:

```
// Dynamically Array declaration by specifying size
int * a = new int[100];

// Declare an array dynamically of user specified size
int n;
cin >> n;
int * arr = new int[n];
```

Dynamically Deleting Arrays

A dynamic array should be deleted from the computer memory once its purpose is fulfilled. The delete statement can help you accomplish this. The released memory space can then be used to hold another set of data. However, even if you do not delete the dynamic array from the computer memory, it will be deleted automatically once the program terminates.

Note: To delete a dynamic array from the computer memory, you should use delete[] instead of delete. The [] instructs the CPU to delete multiple variables rather than one variable. The use of delete instead of delete[] when dealing with a dynamic array may result in problems, such as memory leaks, data corruption, crashes, etc.

Syntax:

```
delete [] arr;
```

How are arrays stored?

The elements of arrays are stored contiguously, i.e., at consecutive memory locations. The name of the array 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, 119};
```

Suppose the starting address of an array is 1000, then the address of the second and third element of the 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 the array could be accessed using indices. The index of an array starts from 0 to n-1, where n is the array's size.

Syntax:

```
Array_name[index];
```

Example:

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

int main() {
    string person[4] = {"Aman","Rahul","Riya","Arti"};
    cout << person[1];

    return 0;
}

Output:
Rahul
```

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