C++ Course Notes | Arrays | Week 5

Hello everyone welcome to the weekly lecture notes

Topics to be covered:

- Array Introduction
- Declaration
- Creation
- Array Types
- Operations
- Taking Input in an array
- Vectors in C++
- Basic STL(Standard Template Library) operations
- Looping in Vector
- Problems in vector

Array Introduction

- Arrays can be described as a type of data structure used to store a group or collection of items or elements sequentially inside a memory.
- The main function of an Array is to store a collection of homogenous data of the same type. For example, integers, strings, floating numbers, etc
- The indexing of an Array is 0 based indexing ie the first element is at index 0 and the second element is at index 1 and so forth. We can directly access the required elements using the indexes.
- The memory allocation in Arrays is contiguous in nature.
- We can create both single-dimensional and multidimensional arrays.

Array Declaration and Creation

- Syntax for creating a new Array in C++ is `data-type array-name[array-size];`
- Some examples:

`int arr[10];` , `float array[15];`

- Array Literal

With curly braces we can initialize the array and add value to it during initialization without defining the size.

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

Topic 3: Array Types

- Single dimensional or one-dimensional array
- Multidimensional Array

Single dimensional Array:

When we have elements stored in a single dimension sequentially, they are called single dimensional arrays.

We can declare and allocate memory to a single-dimensional array using a single variable in C++.

Here is an example,

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

Output:

`Red, Green, Blue`

Syntax for declaring an single dimension array -

`int arrayName[] = {element_0, element_1, element_2, element_3, ...`

`elementN};`

Since we have come this far in arrays, it is worth mentioning here that there is a provision in C++ wherein we can store the elements/data dynamically and sequentially. That means the length of this container is dynamic. These are called vectors ! Let's learn about them.

Vectors in C++

Vectors are dynamic arrays that can be resized whenever an element is inserted or deleted. Vectors also have contiguous storage like arrays but their size can change dynamically according to the requirements of user. Another important point to note here is that vectors are not compulsorily bound by index values.

Declaration Syntax:

`vector < data_type >  name;`

Example: `vector<int>myVec;`

Note: We can even declare a vector of fixed size also with the following syntax:

`vector<int> v(n);`

Here, a vector of size n is declared.

Basic Operations on Vectors

1. Just like in arrays, it returns the size of vector.

Example:

```
vector<int>v = {10,20,30,40,50};
cout << v.size() << endl;
```

Output :

5

2. **Resizing operation in vector:**

Unlike the fixed size approach of arrays, we can resize the vector according to our requirement.

Example:

```
vector<int> v;
v.resize(n);
```

Note: This is extremely useful when we do not know the size initially (at the time of creating the vector). It can be resized anytime as per need.

Vector `capacity()` function: It returns the size of the storage space allocated to the vector. This capacity is greater than or equal to the size of the vector catering the need for extra space to allow growth without the need to reallocate on each insertion.

Note: This capacity is not the limit for growth and can be automatically expanded.

Code Example:

```
vector<int> v;
v.resize(10);
cout << v.capacity() << endl;
v.resize(90);
cout << v.capacity() << endl;
```

Output:

`16`

`128`

Note: Whenever we resize and increase the size of the vector, the total capacity gets to the next power of 2 or the remaining memory. For example, in the above case, the vector size is 10 but its capacity is the closest next power of 2 i.e.16. When increased to 90, its capacity went to the closest next power of 2 i.e. 128.

3. **Inserting an element at the back of the vector using push_back operation:**

Code:

```
vector<int> v = {10,20,30,40};
v.push_back(50);
```

Now the new array will have values: `{10,20,30,40,50}`.

4. **Popping the last element of the vector using pop_back operation:**

Code:

```
vector<int> v = {10,20,30,40,50};
v.pop_back();
```

Now the new array will have values: `{10,20,30,40}`.

6. **Insert an element somewhere in the vector's middle/ (in between the elements):**

Syntax:

`vector<int> v;`

`v.insert(position, value);`

Position specifies to the iterator which points to the position where the insertion is to be done.

Example Code:

```
vector<int> v = {10,20,30,40,50};
v.insert(v.begin() + 2, 100);
```

The new vector will look like this:

`{10,20,100,30,40,50}`

7. **Deleting an element in a vector at a specific position:**

Syntax:

`v.erase(position);`

Example Code:

```
vector<int> v = {10,20,30,40,50};
v.erase(v.begin() + 2);
```

The new vector will look like this:

`{10,20,40,50}`

8. **Clearing the vector : clear() function is used to remove all the elements of the   vector container, thus making its size 0.**

Code:

```
vector<int> v = {10,20,30,40,50};
v.clear();
```

Output:

```
Now v will be empty.
```

Looping in Vector

There are many ways to traverse through the elements of a vector. The most common and widely used ways of looping are:

- **For Loop:**

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

int main(){
  vector<int> v;
  for(int i = 0; i < 5; i++) {
    v.push_back(i);
  }
  // now by the above loop v will have the values : {0,1,2,3,4}
  for(int i = 0; i < v.size(); i++){
    cout << v[i] << " ";
  }
}
```

Output :

```
0 1 2 3 4
```

- **For each loop:**

This loop helps in iterating over iterable objects like string, array and so on.

```
vector<int> a = { 1, 2, 3, 4, 5, 6, 7, 8 };
for (int i : a) {
// accessing each element of the vector
cout << i << endl;
}
```

Output:

1

2

3

4

5

6

7

8

Explanation : Here, all the elements present in the array will be printed.

- **While loop:**

```
int main(){
  int i = 5;
  vector<int> v;
  while(i > 0) {
    v.push_back(i--);
  }
  i = 0;
  while(i < v.size()){
    cout<<v[i]<<" ";
    i++;
  }
}
```

Output : `5 4 3 2 1`

Explanation: Here, we initialize the value of ‘i’ to be 5 and keep decrementing it using the post-decrement operator and at each step insert it at the back of the vector. Therefore, in the end, the vector will contain elements from 5 to 0.
