C++ Vector

A vector is a sequence container class that implements dynamic array, means size automatically changes when appending elements. A vector stores the elements in contiguous memory locations and allocates the memory as needed at run time.

Difference between vector and array

An array follows static approach, means its size cannot be changed during run time while vector implements dynamic array means it automatically resizes itself when appending elements. Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators.

std::vector in C++ is the class template that contains the vector container and its member functions. It is defined inside the **<vector>** header file. The member functions of std::vector class provide various functionalities to vector containers. Some commonly used member functions are written below:

Iterators

begin() – Returns an iterator pointing to the first element in the vector

end() – Returns an iterator pointing to the theoretical element that follows the last element in the vector

Basic Vector Operations

The vector class provides various methods to perform different operations on vectors. We will look at some commonly used vector operations in this tutorial:

Add elements

Access elements

Change elements

Remove elements

1. Add Elements to a Vector

Using push_back() function

To add a single element into a vector, we use the push_back() function. It inserts an element into the end of the vector. For example,

```
#include <iostream>
#include<vector>
using namespace std;
int main()
{
	vector<int> v={1,2,3,4,5};
	cout<<"Initial Vector: "<<endl;
	for(int i=0; i<v.size(); i++)
	{
		cout<<i<<" ";
	}
	v.push_back(6);
	v.push_back(7);
	cout<<"\nUpdated Vector: "<<endl;
	for(int i=0; i<v.size(); i++)
	{
		cout<<i<<" ";
	}
}
```

Output:

Initial Vector:

1 2 3 4 5

Updated Vector:

1 2 3 4 5 6 7

Inserting values from user

```
#include <iostream>
#include<vector>
using namespace std;
int main()
{
	vector<int> v;
	char choice='Y';
	int value;
	while(choice=='Y' || choice=='y')
	{
		cin>>value;
		v.push_back(value);
		cout<<"\nAnother input?"<<endl;
		cin>>choice;
	}
	cout<<"\nVector: "<<endl;
	for(int i=0; i<v.size(); i++)
	{
		cout<<i<<" ";
	}
}
```

Output:

1
Another input?
Y

```
2
Another input?
Y
0
Another input?
Y
5
Another input?
N
Vector:
1 2 0 5
```

C++ Vector Iterators

Vector iterators are used to point to the memory address of a vector element. In some ways, they act like pointers in C++.

We can initialize vector iterators using the begin() and end() functions.

1. begin() function

The begin() function returns an iterator that points to the first element of the vector. For example,

```
vector<int> num = {1, 2, 3};
vector<int>::iterator iter;

// iter points to num[0]
iter = num.begin();
```

2. end() function

The end() function points to the theoretical element that comes after the final element of the vector. For example,

```
// iter points to the last element of num
iter = num.end() - 1;
```

Here, due to the nature of the end() function, we have used the code num.end() - 1 to point to the last element of the num vector i.e. num[2].

The below code demonstrates the use of iterators in C++ to access and manipulate elements in a vector. Initially, a vector named num is created with five elements: {1, 2, 3, 4, 5}. An iterator iter of type vector<int>::iterator is declared. The iterator is then assigned num.begin(), which returns an iterator pointing to the first element of num. By dereferencing the iterator iter with *iter, we can access and print the value of the first element. Next, the iterator is reassigned to num.begin() + 2, moving it two positions forward from the beginning, making it point to the third element. We print the value of this third element. Lastly, the iterator is reassigned to num.end() - 1, which positions it one element before the end of num. By dereferencing the iterator again, we can print the value of the last element.

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<int> num {1, 2, 3, 4, 5};
  // declare iterator
  vector<int>::iterator iter;
  // initialize the iterator with the first
element
  iter = num.begin();
  // print the vector element
  cout << "num[0] = " << *iter << endl;
  // iterator points to the 3rd element
  iter = num.begin() + 2;
  cout << "num[2] = " << *iter;
  // iterator points to the last element
  iter = num.end() - 1;
  cout << "\nnum[4] = " << *iter;
  return 0;
}
```

Output:

num[0] = 1

num[2] = 3

num[4] = 5

Difference between size and capacity:

Size: The size of a vector refers to the number of elements it currently holds. It represents the actual number of elements present in the vector. The size() function is used to retrieve the size of the vector.

Capacity: The capacity of a vector represents the amount of memory allocated to the vector to store its elements. It is the maximum number of elements that the vector can hold before needing to allocate more memory. The capacity() function is used to retrieve the current capacity of the vector.

In this code, a vector v is created and initialized with three elements {1, 2, 3}. Initially, the size of the vector is 3 because it contains three elements, and the capacity is also 3 because the vector has been allocated enough memory to hold three elements.

Afterward, v.push_back(4) is used to add the element 4 to the end of the vector. At this point, the size of the vector becomes 4 as it now contains four elements. However, the capacity of the vector may change. In this case, the capacity of the vector becomes 6 (double of it’s initial size i-e 3)

```
#include <iostream>
#include<vector>
using namespace std;
int main() {
	vector<int> v= {1,2,3};
	cout<<"Size: "<<v.size()<<endl;
	cout<<"Capacity: "<<v.capacity()<<endl;
	v.push_back(4);
	cout<<"Size: "<<v.size()<<endl;
	cout<<"Capacity: "<<v.capacity()<<endl;
	return 0;
}
```

Output:

Size: 3
Capacity: 3
Size: 4
Capacity: 6

Example: Iterate Through Vector Using Iterators

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<int> num {1, 2, 3, 4, 5};
  // declare iterator
  vector<int>::iterator iter;
  // use iterator with for loop
  for (iter = num.begin(); iter != num.end(); ++iter) {
    cout << *iter << "  ";
  }
  return 0;
}
```

Output:

1 2 3 4 5

Using the insert() Function on Vectors

The insert() method can be used to insert single or multiple elements into a given vector in different ways, for different cases. We can insert a single value at our desired position, we can even insert multiple values into the vector at once, and even we can insert a bunch of values from another vector to it.

So, let us see how we can do that with ease.

We can directly pass an iterator pointing to our desired position and the value to be inserted there to the insert() function to modify a vector.

Look carefully at the example below, here we try to insert a value 10 at the beginning of the vector.

In the following code, the initial vector is printed using a for loop with an iterator. The loop iterates from the beginning of the vector (vec.begin()) to the end of the vector (vec.end()), printing the value of each element.

vec.insert(vec.begin(), 10); inserts the element 10 at the beginning of the vector. This operation shifts the existing elements to the right.

vec.insert(vec.begin()+2, 8); inserts the element 8 at the position 2 (index 1) of the vector. This operation also shifts the existing elements to the right.

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
	vector<int> vec {1,2,3,4,5};
	cout<<"Intially vector: ";
	for(auto i=vec.begin(); i!=v.end(); i++)
	{
		cout<<" "<<*i;
	}
	vec.insert(vec.begin(),10);//Inserting 10 to the vector
	vec.insert(vec.begin()+2,8);//Inserting 10 to the vector
	cout<<"\n\nThe modified vector is: ";
	for(auto i=vec.begin(); i!=vec.end(); i++)
	{
		cout<<" "<<*i;
	}
	return 0;
}
```

Output:

Intially vector: 1 2 3 4 5

The modified vector is: 10 1 8 2 3 4 5

Inserting elements by taking input from the user using insert() method

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
   vector<int> v;
   char choice = 'Y';
   int value;
   vector<int>::iterator iter = v.end(); // initialize the iterator
   while (choice == 'Y') {
      cout << "\nEnter a value:" << endl;
      cin >> value;
      iter = v.insert(iter, value); // insert the value at the current iterator position
      cout << "\nAnother value? (Y/N)" << endl;
      cin >> choice;
      if (choice == 'Y')
         iter++; // increment the iterator only if the user wants to enter another value
   }
   for (auto i : v) {
      cout << i << " ";
   }
   return 0;
}
```

The range-based for loop abstracts away the need for managing iterators or indices manually, making it more convenient and readable when accessing and processing elements of a container, such as a vector.

Access Elements of a Vector

In C++, we use the index number to access the vector elements. Here, we use the at() function to access the element from the specified index. For example.

```
#include <iostream>
#include<vector>
using namespace std;
int main()
{
	vector<int> v={1,2,3,4,5};
	// method 1
	cout<<"\nVector: "<<endl;
	vector<int>::iterator i;
	for(i=v.begin(); i!=v.end(); i++)
	{
		cout<<*i<<" ";
	}

	// method 2
	cout<<"\nVector: "<<endl;
	int index;
	for(index=0; index<v.size(); index++)
	{
		cout<<v.at(index)<<" ";
	}
}
```

3. Update Vector Element

We can change an element of the vector using the same at() function. For example.

```
#include <iostream>
#include<vector>
using namespace std;
int main()
{
	vector<int> v={1,2,3,4,5};
	// method 1
	cout<<"\nVector before: "<<endl;
	vector<int>::iterator i;
	for(i=v.begin(); i<v.end(); i++)
	{
		cout<<*i<<" ";
	}
	v.at(3)=9;
	cout<<"\nUpdated vector: "<<endl;
	for(i=v.begin(); i<v.end(); i++)
	{
		cout<<*i<<" ";
	}
}
```

Output:

Vector before:

1 2 3 4 5

Updated vector:

1 2 3 9 5

4. Delete Elements from C++ Vectors

To delete a single element from a vector, we use the `pop_back()` function. For example.

```
#include <iostream>
#include<vector>
using namespace std;
int main()
{
	vector<int> v={1,2,3,4,5};
	// method 1
	cout<<"\nVector before: "<<endl;
	vector<int>::iterator i;
	for(i=v.begin(); i<v.end(); i++)
	{
		cout<<*i<<" ";
	}
	v.pop_back();
	cout<<"\nUpdated vector: "<<endl;
	for(i=v.begin(); i!=v.end(); i++)
	{
		cout<<*i<<" ";
	}
}
```

Output:

Vector before:

1 2 3 4 5

Updated vector:

1 2 3 4

Removing an **element from** a **particular position**
Example:

```
#include <iostream>
#include<vector>
using namespace std;
int main()
{
	vector<int> v={1,2,3,4,5};
	cout<<"\nVector before: "<<endl;
	vector<int>::iterator i;
	for(i=v.begin(); i<!=v.end(); i++)
	{
		cout<<*i<<" ";
	}
	i=v.begin();
	v.erase(i+2);
	cout<<"\nUpdated before: "<<endl;
	for(i=v.begin(); i!=v.end(); i++)
	{
		cout<<*i<<" ";
	}
}
```

Output:

Vector before:

1 2 3 4 5

Updated before:

1 2 4 5

Removing elements within a range

```
#include <iostream>
#include<vector>
using namespace std;
int main()
{
	vector<int> v={1,2,3,4,5};
	cout<<"\nVector before: "<<endl;
	vector<int>::iterator i;
	for(i=v.begin(); i!=v.end(); i++)
	{
		cout<<*i<<" ";
	}
	i=v.begin();
	v.erase(i+1, i+3);
	cout<<"\nUpdated before: "<<endl;
	for(i=v.begin(); i!=v.end(); i++)
	{
		cout<<*i<<" ";
	}
}
```

Output:

Vector before:

1 2 3 4 5

Updated before:

1 4 5

Erase all Elements in a vector

```
#include <iostream>
#include<vector>
using namespace std;
int main()
{
	vector<int> v={1,2,3,4,5};
	cout<<"\nVector before: "<<endl;
	vector<int>::iterator i;
	for(i=v.begin(); i!=v.end(); i++)
	{
		cout<<*i<<" ";
	}
	v.clear();
	cout<<"\nUpdated vector: "<<endl;
	for(i=v.begin(); i!=v.end(); i++)
	{
		cout<<*i<<" ";
	}
}
```

Output:

Vector before:

1 2 3 4 5

Updated vector:

C++ Vector Functions

In C++, the vector header file provides various functions that can be used to perform different operations on a vector.

Function	Description
size()	returns the number of elements present in the vector
clear()	removes all the elements of the vector
front()	returns the first element of the vector
back()	returns the last element of the vector
empty()	returns **1** (true) if the vector is empty
capacity()	check the overall size of a vector