Assignment 6: Section- Theory **GR NO-21920090**

Lab Assignment No 6: Create a class template to represent a generic vector. Include following member functions:

i. To create the vector.

ii. To modify the value of a given element

iii. To multiply by a scalar value

iv. To display the vector in the form (10,20,30,…)

Aim: Use template class to represent vector and perform operations on vector.

Description: In this task I firstly created one class template to create generic data type. Using that class template I declared one generic variable which then defined to integer vector while creating class object. Using that variable I have created integer vector and performed operations on it.

Oop Concepts Used:

1) **Class Template :** Class Templates Like function templates, class templates are useful when a class defines something that is independent of the data type. Can be useful for classes like LinkedList, BinaryTree, Stack, Queue, Array, etc.

2) **Vector in C++ STL:** Vectors are same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container. Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators. In vectors, data is inserted at the end. Inserting at the end takes differential time, as sometimes there may be a need of extending the array. Removing the last element takes only constant time because no resizing happens. Inserting and erasing at the beginning or in the middle is linear in time.

   Iterators:

1) begin() – Returns an iterator pointing to the first element in the vector
2) end() – Returns an iterator pointing to the theoretical element that follows the last element in the vector
3) rbegin() – Returns a reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
4) rend() – Returns a reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)
5) cbegin() – Returns a constant iterator pointing to the first element in the vector.
6) cend() – Returns a constant iterator pointing to the theoretical element that follows the last element in the vector.

7) crbegin() – Returns a constant reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
8) crend() – Returns a constant reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)

Element access:

1) reference operator [g] – Returns a reference to the element at position 'g' in the vector
2) at(g) – Returns a reference to the element at position 'g' in the vector
3) front() – Returns a reference to the first element in the vector
4) back() – Returns a reference to the last element in the vector
5) data() – Returns a direct pointer to the memory array used internally by the vector to store its owned elements.

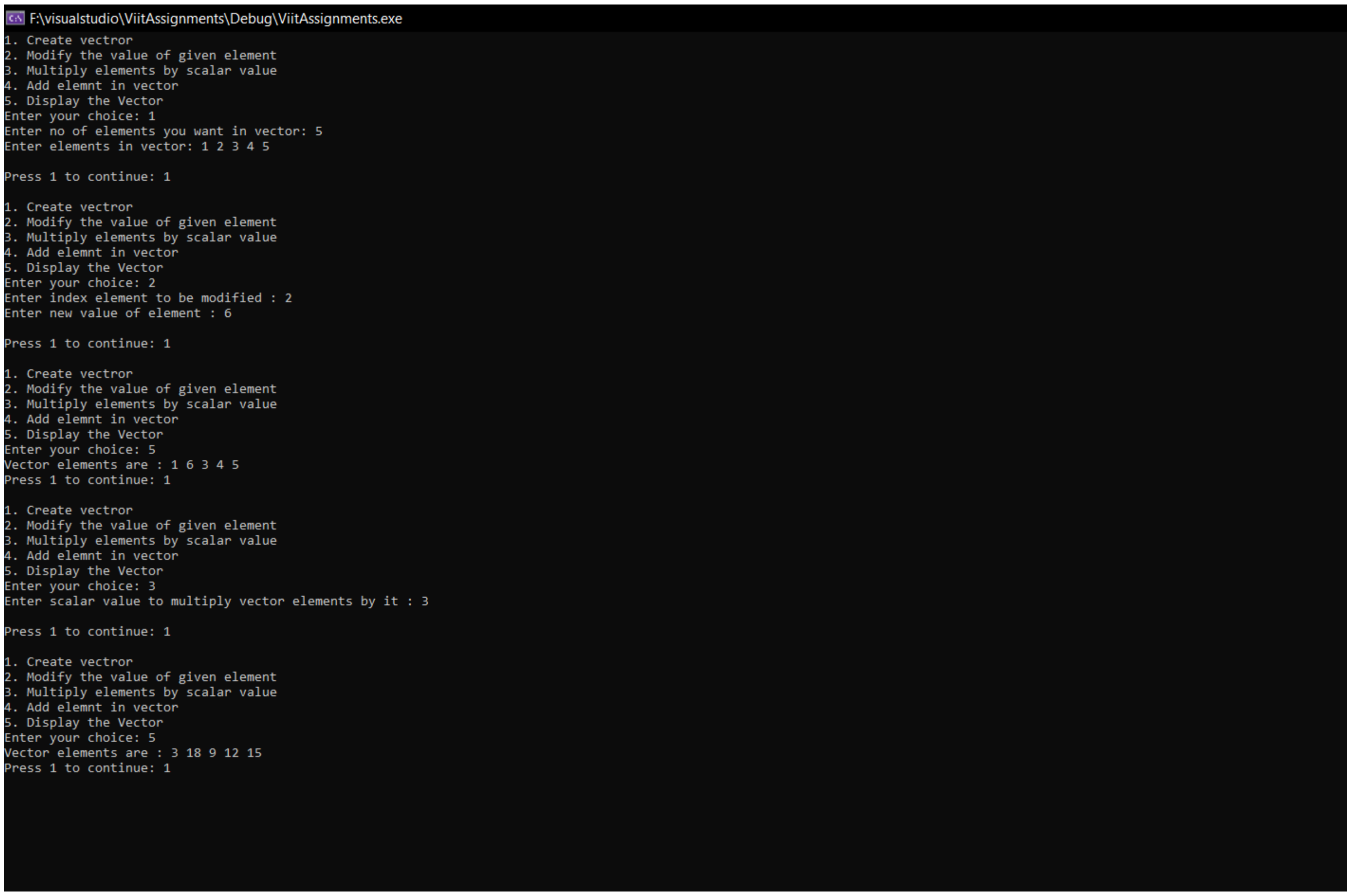
Modifiers:

1) assign() – It assigns new value to the vector elements by replacing old ones
2) push_back() – It push the elements into a vector from the back
3) pop_back() – It is used to pop or remove elements from a vector from the back.
4) insert() – It inserts new elements before the element at the specified position
5) erase() – It is used to remove elements from a container from the specified position or range.
6) swap() – It is used to swap the contents of one vector with another vector of same type. Sizes may differ.
7) clear() – It is used to remove all the elements of the vector container
8) emplace() – It extends the container by inserting new element at position
9) emplace_back() – It is used to insert a new element into the vector container, the new element is added to the end of the vector

Capacity

1) size() – Returns the number of elements in the vector.
2) max_size() – Returns the maximum number of elements that the vector can hold.
3) capacity() – Returns the size of the storage space currently allocated to the vector expressed as number of elements.
4) resize(n) – Resizes the container so that it contains 'n' elements.
5) empty() – Returns whether the container is empty.
6) shrink_to_fit() – Reduces the capacity of the container to fit its size and destroys all elements beyond the capacity.
7) reserve() – Requests that the vector capacity be at least enough to contain n elements.

Output:


F:\visualstudio\ViitAssignments\Debug\ViitAssignments.exe

```
1. Create vectror
2. Modify the value of given element
3. Multiply elements by scalar value
4. Add elemnt in vector
5. Display the Vector
Enter your choice: 1
Enter no of elements you want in vector: 5
Enter elements in vector: 1 2 3 4 5

Press 1 to continue: 1

1. Create vectror
2. Modify the value of given element
3. Multiply elements by scalar value
4. Add elemnt in vector
5. Display the Vector
Enter your choice: 2
Enter index element to be modified : 2
Enter new value of element : 6

Press 1 to continue: 1

1. Create vectror
2. Modify the value of given element
3. Multiply elements by scalar value
4. Add elemnt in vector
5. Display the Vector
Enter your choice: 5
Vector elements are : 1 6 3 4 5
Press 1 to continue: 1

1. Create vectror
2. Modify the value of given element
3. Multiply elements by scalar value
4. Add elemnt in vector
5. Display the Vector
Enter your choice: 3
Enter scalar value to multiply vector elements by it : 3

Press 1 to continue: 1

1. Create vectror
2. Modify the value of given element
3. Multiply elements by scalar value
4. Add elemnt in vector
5. Display the Vector
Enter your choice: 5
Vector elements are : 3 18 9 12 15
Press 1 to continue: 1
```

Conclusion:

Thus we have successfully implemented Vector using class template.