WEEK 12

Program No:12.1

Develop a C++ program of List and Vector Containers

Aim: To develop a C++ program List and Vector Containers.

Description:

In C++, a **vector** is a dynamic array that can automatically resize when elements are added or removed.

- ✔ **Random Access:** Access elements using [] or at().
- ✔ **Fast at End:** Insertion/deletion is fast at the end, slower in the middle.
- ✔ **Memory:** Stores elements in contiguous memory.
- ✔ **Functions:** push_back(), pop_back(), size().

A **list** is a doubly linked list where elements are stored in non-contiguous memory.

- ✔ **Fast Insertion/Deletion:** Anywhere in the list.
- ✔ **Sequential Access:** No random access, use iterators.
- ✔ **Memory:** Non-contiguous memory storage.
- ✔ **Functions:** push_back(), push_front(), pop_back(), pop_front().

Syntax:

```
═════ VECTOR ═════
vector<int> v;      // Declare vector
v.push_back(10);    // Add element at end
v.pop_back();       // Remove last element
v[0];               // Access element
v.size();           // Get size

// ═════ LIST ═════
list<int> l;        // Declare list
l.push_back(100);   // Add element at end
l.push_front(200);  // Add element at beginning
l.pop_back();       // Remove last element
l.pop_front();      // Remove first element
l.size();           // Get size
```

Program:

```
#include <iostream>
#include <list>
#include <vector>
using namespace std;
int main() {
    cout << "=== VECTOR OPERATIONS ===" << endl;
    vector<int> v; // declare a vector
    // Insertion
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);
    cout << "Vector elements after insertion: ";
    for (int x : v)
    cout << x << " ";
    // Deletion (remove last element)
    v.pop_back();
    cout << "\nVector after deletion: ";
    for (int x : v)
    cout << x << " ";
    // Access element
    cout << "\nFirst element: " << v.front();
    cout << "\nLast element: " << v.back() << endl;

    cout << "\n=== LIST OPERATIONS ===" << endl;
    list<int> lst; // declare a list
    // Insertion
    lst.push_back(100);
    lst.push_back(200);
    lst.push_front(50); // insert at beginning
    cout << "List elements after insertion: ";
    for (int x : lst)
    cout << x << " ";
    // Deletion
    lst.pop_front(); // remove first element
    cout << "\nList after deletion: ";
    for (int x : lst)
    cout << x << " ";
    // Traversal using iterator
    cout << "\nList traversal using iterator: ";
    for (list<int>::iterator it = lst.begin(); it != lst.end(); ++it)
    cout << *it << " ";
    cout << endl;
    return 0;
}
```

Output:

```
=== VECTOR OPERATIONS ===
Vector elements after insertion: 10 20 30
Vector after deletion: 10 20
First element: 10
Last element: 20
=== LIST OPERATIONS ===
 List elements after insertion: 50 100 200
List after deletion: 100 200
List traversal using iterator: 100 200
```

Date: Roll No.: | | | | | | | | | | |

Program No:12.2

Develop a C++ program of Deque

Aim: To develop a C++ program of Deque.

Description:

 A deque (double-ended queue) is a sequence container in C++ STL that allows insertion and deletion at both ends (front and back).
 It is like a dynamic array, but more flexible than a vector for operations at the beginning.
 Supports random access to elements using [] or at().
 Useful when you need to add or remove elements from both ends efficiently.

- ✓ Can add/remove from front and back.
- ✓ Random access supported.
- ✓ Automatic resizing.
- ✓ Functions: push_back(), push_front(), pop_back(), pop_front(), size(), at(), [].

syntax:

```
  {    deque<int> d;          // Declare a deque

      d.push_back(10);       // Add element at back
      d.push_front(20);      // Add element at front

      int x = d[0];          // Access element using index
      int y = d.at(1);       // Access element using at()

      d.pop_back();          // Remove element from back
      d.pop_front();         // Remove element from front

      int size = d.size();   // Get size
}
```

programm:

```
 #include <iostream>
 #include <deque>
 using namespace std;
 int main() {
     deque<int> dq;
     cout << "=== DEQUE OPERATIONS ===" << endl;
     dq.push_back(10);
     dq.push_back(20);
     dq.push_front(5);
     cout << "Deque elements after insertion: ";
     for (int x : dq)
     cout << x << " ";
```

```
    dq.pop_front();
    cout << "\nDeque after deleting front element: ";
    for (int x : dq)
    cout << x << " ";
    dq.pop_back();
    cout << "\nDeque after deleting last element: ";
    for (int x : dq)
    cout << x << " ";
    cout << "\nFront element: " << dq.front();
    cout << "\nBack element: " << dq.back();
    dq.push_front(1);
    dq.push_back(50);
    cout << "\nDeque after adding 1 (front) and 50 (back): ";
    for (int x : dq)
    cout << x << " ";
    cout << endl;
    return 0;
}
```

Output:

```
=== DEQUE OPERATIONS ===
Deque elements after insertion: 5 10 20
Deque after deleting front element: 10 20
Deque after deleting last element: 10
Front element: 10 Back element: 10
Deque after adding 1 (front) and 50 (back): 1 10 50
```

program No:12.3

Develop a C++ program of Map and demonstrate operations such as insertion, deletion, access, and searching

Aim: To develop a C++ program of of Map and demonstrate operations such as insertion, deletion, access, and searching

Description:

A map is an associative container in C++ that stores key-value pairs with unique keys and keeps them sorted in ascending order. It allows fast access, retrieval, and modification of values using keys, making it ideal for situations where you need efficient lookups. Maps are usually implemented using balanced binary search trees, which ensures logarithmic time complexity for insertion, deletion, and searching.

Key Features and Operations:

- Insertion: Add elements using map[key] = value or insert().
- Access: Retrieve or update values using map[key].
- Searching: Check if a key exists using find(key).
- Deletion: Remove a key-value pair using erase(key).
- Traversal: Display all elements in sorted order of keys.
- Useful for storing unique keys, fast lookups, and dynamic data management.

program:

```
#include <iostream>
#include <map>
using namespace std;
int main()
{
  map <int, strings> students;
  cout << "=== MAP OPERATIONS ===" << endl;
  students[101] = "Alice"; students[102] = "Bob";
  students[103] = "Charlie";
  students.insert({104, "David"});
  cout << "Students after insertion:" << endl;
  for (auto x : students)
  cout << "Roll No: " << x.first << " Name: " << x.second << endl;
  cout << "\nAccess element with key 102: " << students[102] << endl;
  int key = 103;
  auto it = students.find(key);
  if (it != students.end())
    cout << "Found student with Roll No " << key << ": " << it->second << endl;
  else
    cout << "Student with Roll No " << key << " not found!" << endl;
  students.erase(101);
  cout << "\nAfter deleting key 101:" << endl;
  for (auto x : students)
  cout << "Roll No: " << x.first << " Name: " << x.second << endl;
  cout << "\nTotal students: " << students.size() << endl;
  return 0;
}
```

Output:

```
 === MAP OPERATIONS ===
Students after insertion:
Roll No: 101 Name: Alice
Roll No: 102 Name: Bob
Roll No: 103 Name: Charlie
Roll No: 104 Name: David
Access element with key 102: Bob
Found student with Roll No 103: Charlie
After deleting key 101:
Roll No: 102 Name: Bob
Roll No: 103 Name: Charlie
Roll No: 104 Name: David Total students: 3
```