WEEK – 13

13.Aim: To Write a program implementing a circular queue class with required operations using STL.

Program Code:

```
#include <iostream>

using namespace std;

int cqueue[5];

int front = -1, rear = -1, n=5;

void insertCQ(int val) {

  if ((front == 0 && rear == n-1) || (front == rear+1)) {

    cout<<"Queue Overflow \n";

    return;

  }

  if (front == -1) {

    front = 0;

    rear = 0;

  } else {

    if (rear == n - 1)

    rear = 0;

    else

    rear = rear + 1;

  }

  cqueue[rear] = val ;

}

void deleteCQ() {

  if (front == -1) {
```

```
      cout<<"Queue Underflow\n";
      return ;
    }
    cout<<"Element deleted from queue is : "<<cqueue[front]<<endl;
    if (front == rear) {
      front = -1;
      rear = -1;
    } else {
      if (front == n - 1)
      front = 0;
      else
      front = front + 1;
    }
}
void displayCQ() {
    int f = front, r = rear;
    if (front == -1) {
      cout<<"Queue is empty"<<endl;
      return;
    }
    cout<<"Queue elements are :\n";
    if (f <= r) {
      while (f <= r){
        cout<<cqueue[f]<<" ";
        f++;
      } } else {
      while (f <= n - 1) {
```

```
        cout<<cqueue[f]<<" ";
        f++;
      }
      f = 0;
      while (f <= r) {
        cout<<cqueue[f]<<" ";
        f++;
      }
    }
    cout<<endl;
}
int main() {
    int ch, val;
    cout<<"1)Insert\n";
    cout<<"2)Delete\n";
    cout<<"3)Display\n";
    cout<<"4)Exit\n";
    do {
      cout<<"Enter choice : "<<endl;
      cin>>ch;
      switch(ch) {
        case 1:
              cout<<"Input for insertion: "<<endl;
              cin>>val;
             insertCQ(val);
            break;
        case 2:
```

```
            deleteCQ();
            break;
        case 3:
            displayCQ();
            break;
        case 4:
           cout<<"Exit\n";
           break;
        default:
            cout<<"Incorrect!\n";
      }
   } while(ch != 4);
   return 0;
}
```

OUTPUT:

```
1)Insert  2)Delete 3)Display 4)Exit
Enter choice : 1 Input for insertion:
Enter choice : 1 Input for insertion:
Enter choice : 1 Input for insertion:
Enter choice : 1 Input for insertion:
Enter choice : 1 Input for insertion:
Enter choice : 2 Element deleted from queue is : 5
Enter choice : 2 Element deleted from queue is : 3
Enter choice : 2 Element deleted from queue is : 2
Enter choice : 1 Input for insertion: 6
Enter choice : 3 Queue elements are :
7 9 6
Enter choice : 4
Exit
```

WEEK-15

15.AIM:Write a program to perform all operations of a single linked list using forward list in STL.

Program Code:

```
#include <iostream>
#include <iterator>
#include <list>
using namespace std;
void showlist(list<int> g)
{
   list<int>::iterator it;
   for (it = g.begin(); it != g.end(); ++it)
      cout << '\t' << *it;
   cout << '\n';
}
int main()
{
   list<int> list1, list2;
   for (int i = 0; i < 10; ++i)
   {
      list1.push_back(i * 2);
      list2.push_front(i * 3);
   }
   cout << "\nList 1 (list1) is : ";
   showlist(list1);
   cout << "\nList 2 (list2) is : ";
   showlist(list2);
   cout << "\ngqlist1.front() : " << list1.front();
   cout << "\ngqlist1.back() : " << list1.back();
   cout << "\ngqlist1.pop_front() : ";
   list1.pop_front();
   showlist(list1);
   cout << "\nlist2.pop_back() : ";
```

```
    list2.pop_back();
    showlist(list2);
    cout << "\ngqlist1.reverse() : ";
    list1.reverse();
    showlist(list1);
    cout << "\nlist2.sort(): ";
    list2.sort();
    showlist(list2);
    return 0;
}
```

OUTPUT:

```
List 1 (list1) is :    0    2    4    6    8    10    12    14    16    18

List 2 (list2) is :    27    24    21    18    15    12    9    6    3    0

gqlist1.front() : 0
gqlist1.back() : 18
gqlist1.pop_front() :  2    4    6    8    10    12    14    16    18
list2.pop_back() :    27    24    21    18    15    12    9    6    3
gqlist1.reverse() :    18    16    14    12    10    8    6    4    2
list2.sort():  3    6    9    12    15    18    21    24    27
```

WEEK-16

16.AIM: Write a program to implement binary search tree using traverse the tree using any traversal schema

PROGRAM CODE:

```
#include <iostream>
using namespace std;
struct node
{
    int key;
    struct node *left, *right;
};
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}
void inorder(struct node *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        cout << root->key << " -> ";
        inorder(root->right);
    }
}
struct node *insert(struct node *node, int key)
{
    if (node == NULL)
        return newNode(key);
    if (key < node->key)
        node->left = insert(node->left, key);
```

```
    else
        node->right = insert(node->right, key);
    return node;
}
struct node *minValueNode(struct node *node)
{
    struct node *current = node;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}
struct node *deleteNode(struct node *root, int key)
{
    if (root == NULL)
        return root;
    if (key < root->key)
        root->left = deleteNode(root->left, key);
    else if (key > root->key)
        root->right = deleteNode(root->right, key);
    else
    {

        if (root->left == NULL)
        {
            struct node *temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL)
        {
            struct node *temp = root->left;
            free(root);
            return temp;
        }
        struct node *temp = minValueNode(root->right);
```

```
        root->key = temp->key;
        root->right = deleteNode(root->right, temp->key);
    }
    return root;
}
int main()
{
    struct node *root = NULL;
    root = insert(root, 8);
    root = insert(root, 3);
    root = insert(root, 1);
    root = insert(root, 6);
    root = insert(root, 7);
    root = insert(root, 10);
    root = insert(root, 14);
    root = insert(root, 4);
    cout << "Inorder traversal: ";
    inorder(root);
    cout << "\nAfter deleting 10\n";
    root = deleteNode(root, 10);
    cout << "Inorder traversal: ";
    inorder(root);
}
```

OUTPUT:

```
Inorder tree traversal:
1->3->4->6->7->8->10->14

After deleting 10
1->3->4->6->7->8->14
```