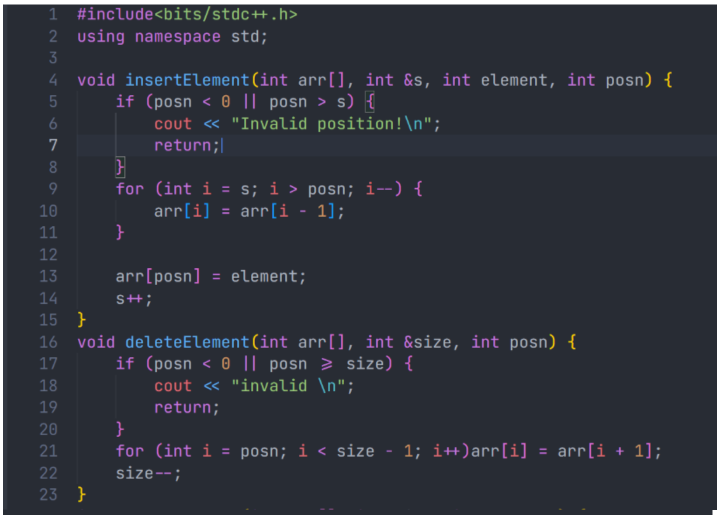
PRACTICALS

SUBJECT : DATA STRUCTURE (COCSC202)

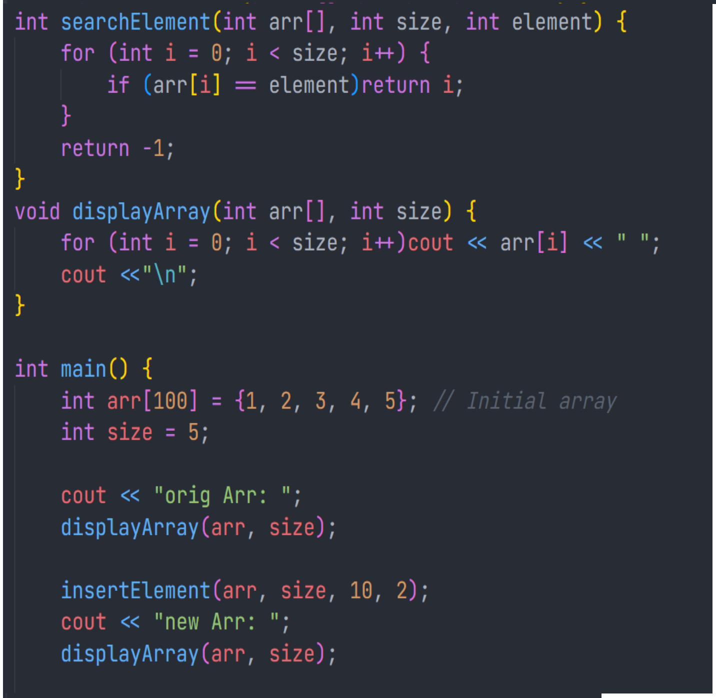
Q1. Write a program to insert one element in an array and delete an element from an array.

Write a program to search for a number in an array.

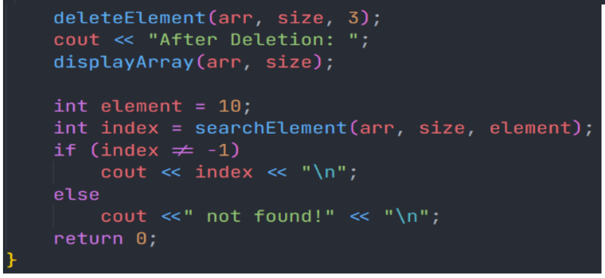
Code:



```cpp
#include<bits/stdc++.h>
using namespace std;

void insertElement(int arr[], int &s, int element, int posn) {
    if (posn < 0 || posn > s) {
        cout << "Invalid position!\n";
        return;
    }
    for (int i = s; i > posn; i--) {
        arr[i] = arr[i - 1];
    }

    arr[posn] = element;
    s++;
}
void deleteElement(int arr[], int &size, int posn) {
    if (posn < 0 || posn >= size) {
        cout << "invalid \n";
        return;
    }
    for (int i = posn; i < size - 1; i++)arr[i] = arr[i + 1];
    size--;
}
```

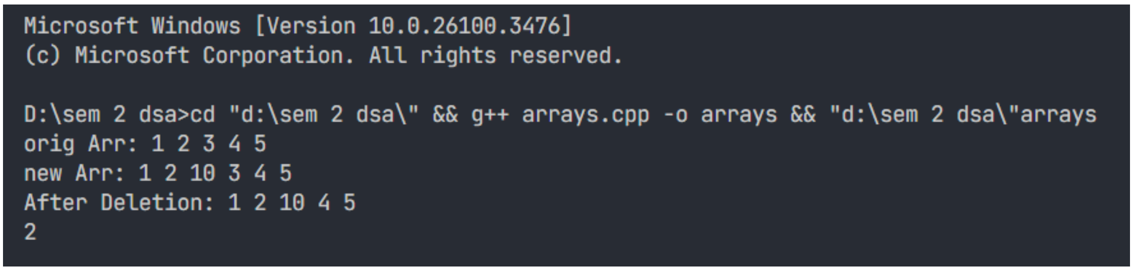


```cpp
int searchElement(int arr[], int size, int element) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == element)return i;
    }
    return -1;
}
void displayArray(int arr[], int size) {
    for (int i = 0; i < size; i++)cout << arr[i] << " ";
    cout <<"\n";
}

int main() {
    int arr[100] = {1, 2, 3, 4, 5}; // Initial array
    int size = 5;

    cout << "orig Arr: ";
    displayArray(arr, size);

    insertElement(arr, size, 10, 2);
    cout << "new Arr: ";
    displayArray(arr, size);
```



```cpp
    deleteElement(arr, size, 3);
    cout << "After Deletion: ";
    displayArray(arr, size);

    int element = 10;
    int index = searchElement(arr, size, element);
    if (index != -1)
        cout << index << "\n";
    else
        cout <<" not found!" << "\n";
    return 0;
}
```

Output:

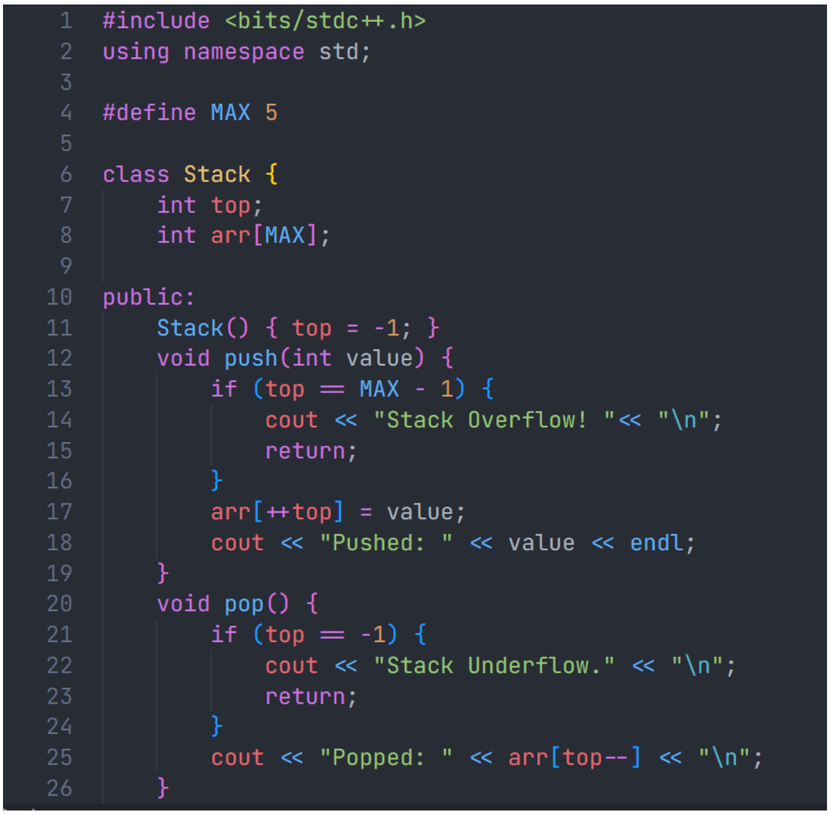


```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

D:\sem 2 dsa>cd "d:\sem 2 dsa\" && g++ arrays.cpp -o arrays && "d:\sem 2 dsa\"arrays
orig Arr: 1 2 3 4 5
new Arr: 1 2 10 3 4 5
After Deletion: 1 2 10 4 5
2
```

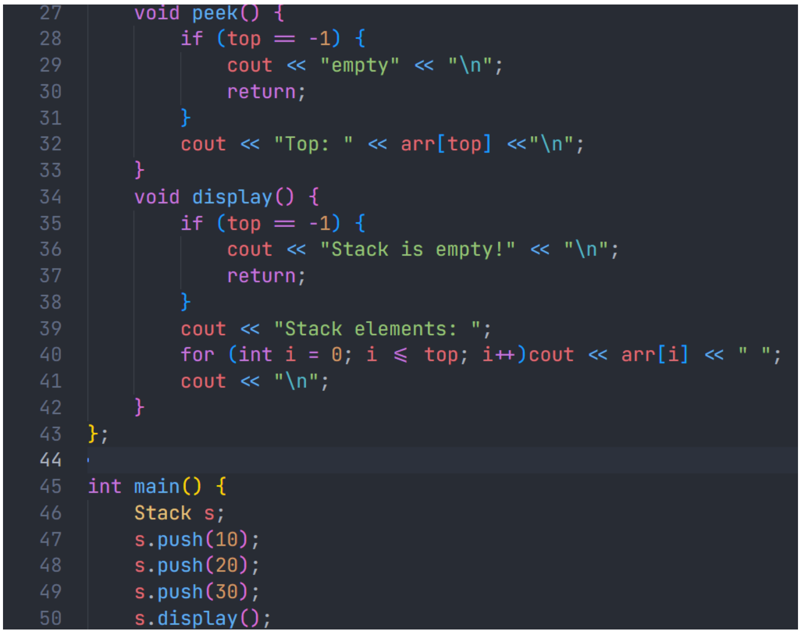
Topic : Stacks

Q2. Write a program to implement a various operations of stack using static and binary data structure.

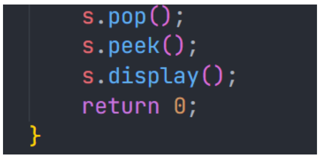
Code:



```cpp
#include <bits/stdc++.h>
using namespace std;

#define MAX 5

class Stack {
    int top;
    int arr[MAX];

public:
    Stack() { top = -1; }
    void push(int value) {
        if (top == MAX - 1) {
            cout << "Stack Overflow! "<< "\n";
            return;
        }
        arr[++top] = value;
        cout << "Pushed: " << value << endl;
    }
    void pop() {
        if (top == -1) {
            cout << "Stack Underflow." << "\n";
            return;
        }
        cout << "Popped: " << arr[top--] << "\n";
    }
```

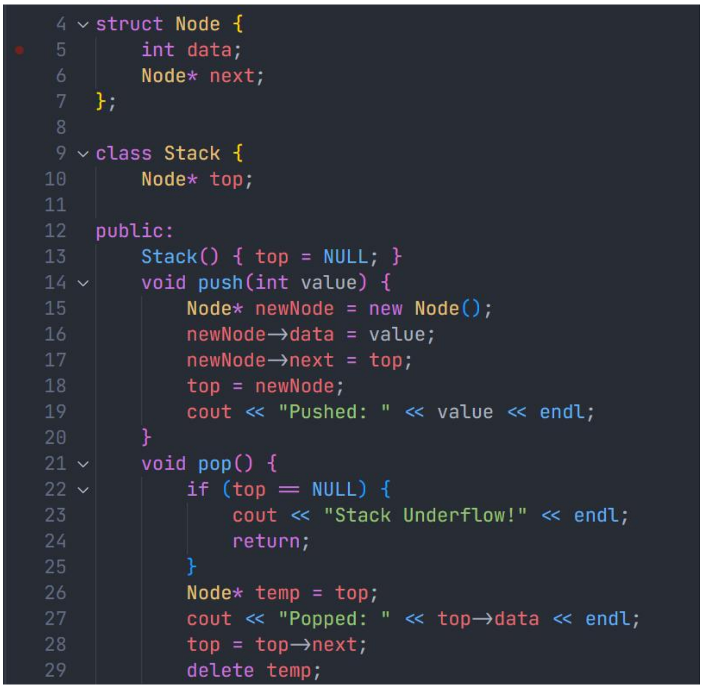


```cpp
    void peek() {
        if (top == -1) {
            cout << "empty" << "\n";
            return;
        }
        cout << "Top: " << arr[top] <<"\n";
    }
    void display() {
        if (top == -1) {
            cout << "Stack is empty!" << "\n";
            return;
        }
        cout << "Stack elements: ";
        for (int i = 0; i <= top; i++)cout << arr[i] << " ";
        cout << "\n";
    }
};

int main() {
    Stack s;
    s.push(10);
    s.push(20);
    s.push(30);
    s.display();
```

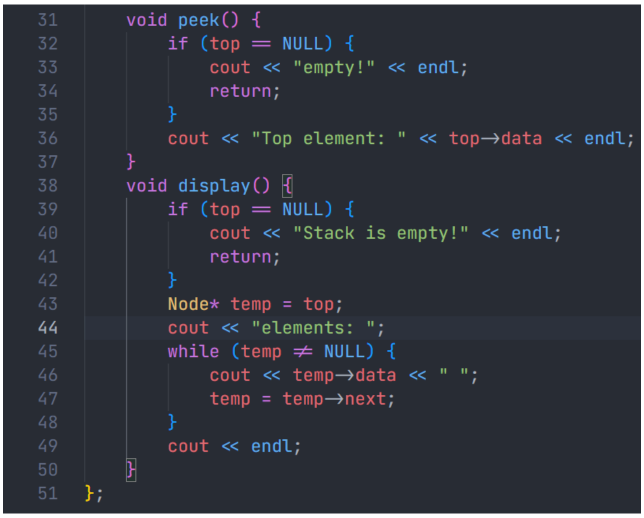


```cpp
    s.pop();
    s.peek();
    s.display();
    return 0;
}
```

Dynamic Representation Using Linked Lists

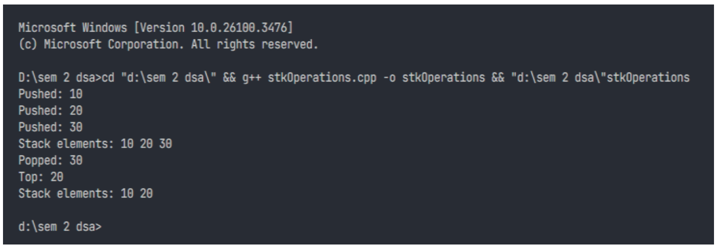


```cpp
struct Node {
    int data;
    Node* next;
};

class Stack {
    Node* top;

public:
    Stack() { top = NULL; }
    void push(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = top;
        top = newNode;
        cout << "Pushed: " << value << endl;
    }
    void pop() {
        if (top == NULL) {
            cout << "Stack Underflow!" << endl;
            return;
        }
        Node* temp = top;
        cout << "Popped: " << top->data << endl;
        top = top->next;
        delete temp;
```

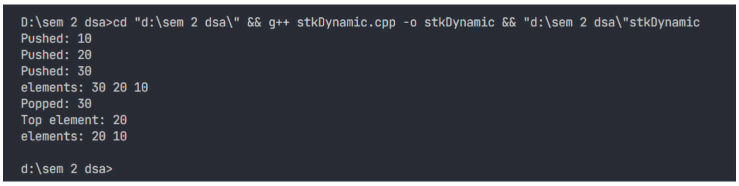


```cpp
    void peek() {
        if (top == NULL) {
            cout << "empty!" << endl;
            return;
        }
        cout << "Top element: " << top->data << endl;
    }
    void display() {
        if (top == NULL) {
            cout << "Stack is empty!" << endl;
            return;
        }
        Node* temp = top;
        cout << "elements: ";
        while (temp != NULL) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};
```

Output:



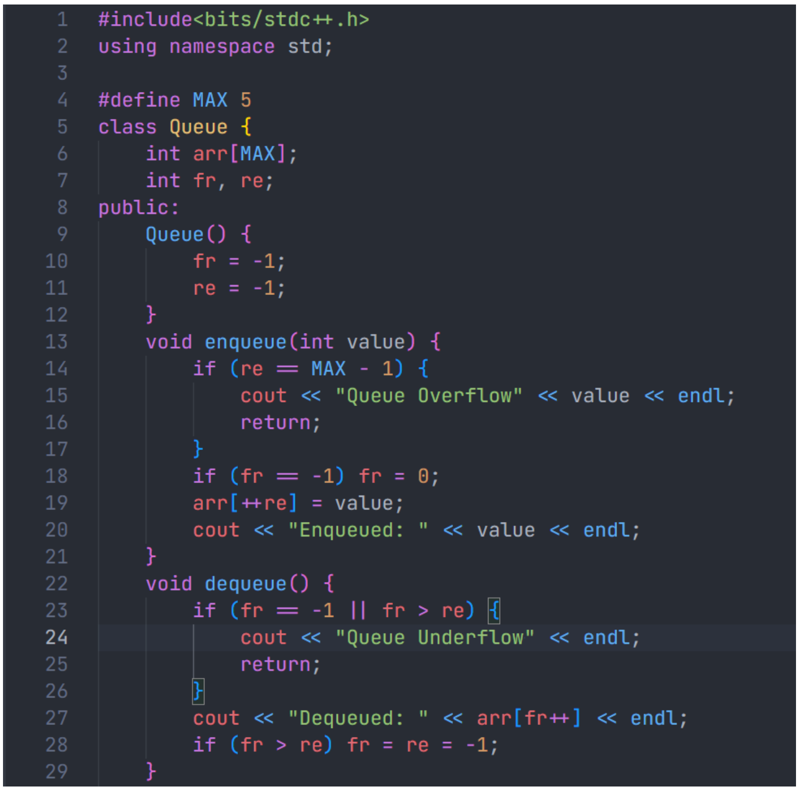
```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

D:\sem 2 dsa>cd "d:\sem 2 dsa\" && g++ stkOperations.cpp -o stkOperations && "d:\sem 2 dsa\"stkOperations
Pushed: 10
Pushed: 20
Pushed: 30
Stack elements: 10 20 30
Popped: 30
Top: 20
Stack elements: 10 20

d:\sem 2 dsa>
```



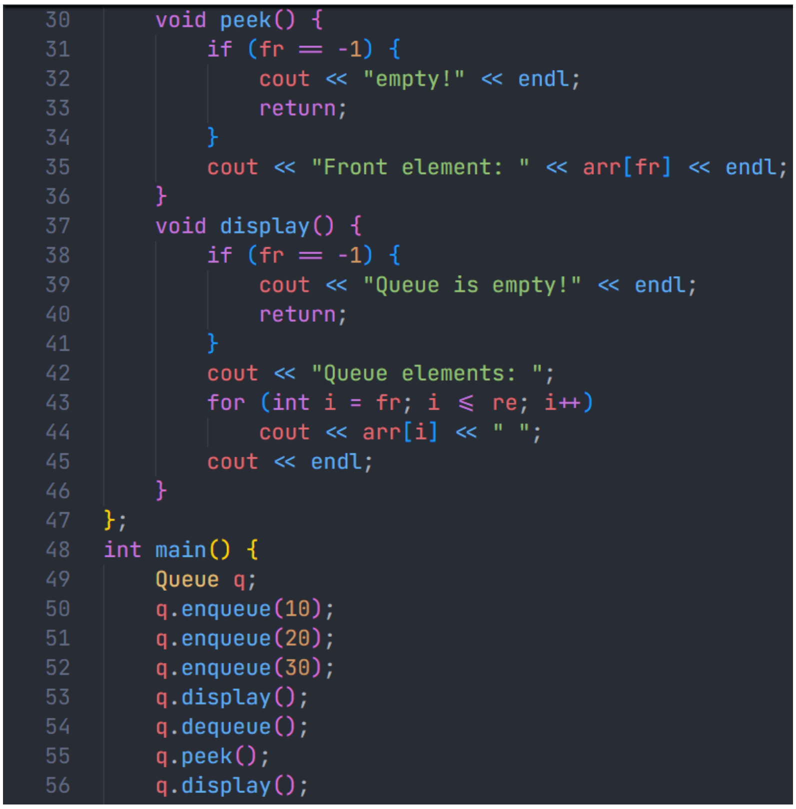
```
D:\sem 2 dsa>cd "d:\sem 2 dsa\" && g++ stkDynamic.cpp -o stkDynamic && "d:\sem 2 dsa\"stkDynamic
Pushed: 10
Pushed: 20
Pushed: 30
elements: 30 20 10
Popped: 30
Top element: 20
elements: 20 10

d:\sem 2 dsa>
```

Q3. Write a program to implement a various operations of queue using static and binary data structure.

Code:

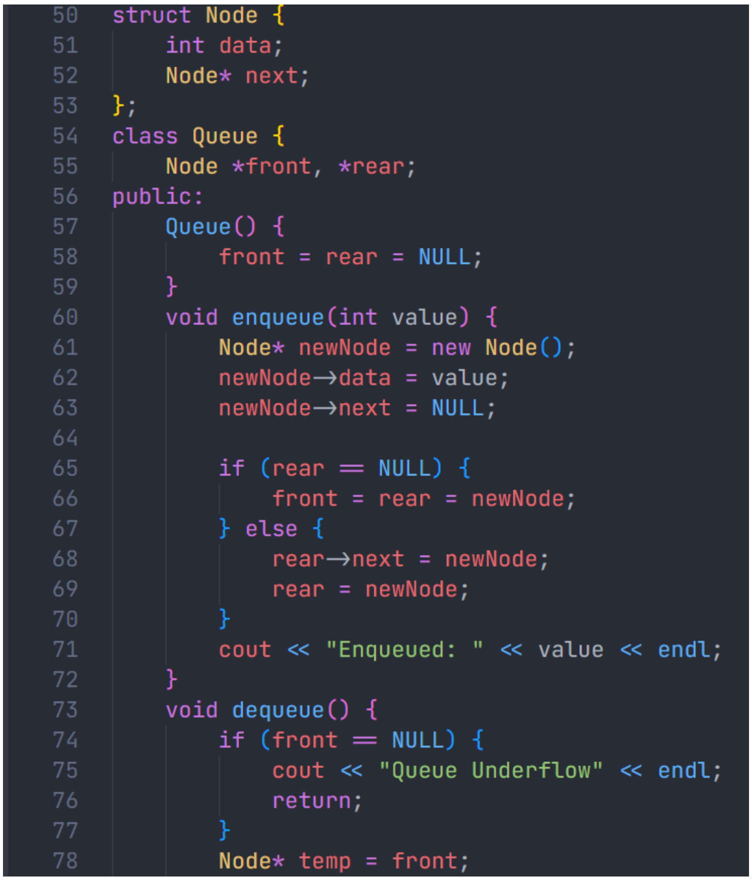


```cpp
#include<bits/stdc++.h>
using namespace std;

#define MAX 5
class Queue {
    int arr[MAX];
    int fr, re;
public:
    Queue() {
        fr = -1;
        re = -1;
    }
    void enqueue(int value) {
        if (re == MAX - 1) {
            cout << "Queue Overflow" << value << endl;
            return;
        }
        if (fr == -1) fr = 0;
        arr[++re] = value;
        cout << "Enqueued: " << value << endl;
    }
    void dequeue() {
        if (fr == -1 || fr > re) {
            cout << "Queue Underflow" << endl;
            return;
        }
        cout << "Dequeued: " << arr[fr++] << endl;
        if (fr > re) fr = re = -1;
    }
```

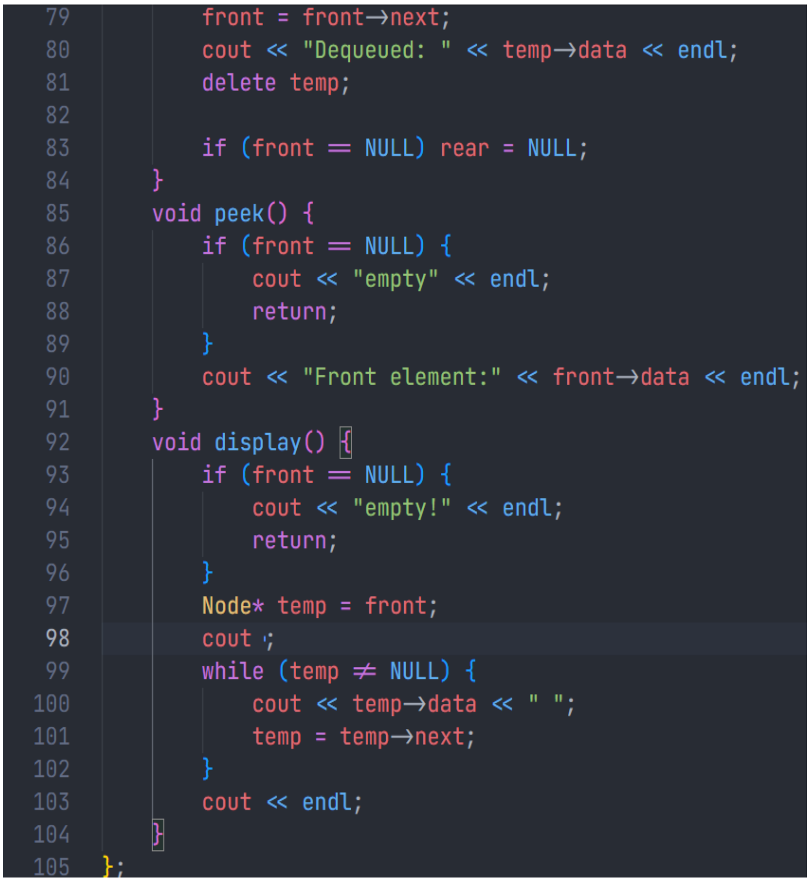


```cpp
    void peek() {
        if (fr == -1) {
            cout << "empty!" << endl;
            return;
        }
        cout << "Front element: " << arr[fr] << endl;
    }
    void display() {
        if (fr == -1) {
            cout << "Queue is empty!" << endl;
            return;
        }
        cout << "Queue elements: ";
        for (int i = fr; i <= re; i++)
            cout << arr[i] << " ";
        cout << endl;
    }
};
int main() {
    Queue q;
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.display();
    q.dequeue();
    q.peek();
    q.display();
```

Dynamic Representation

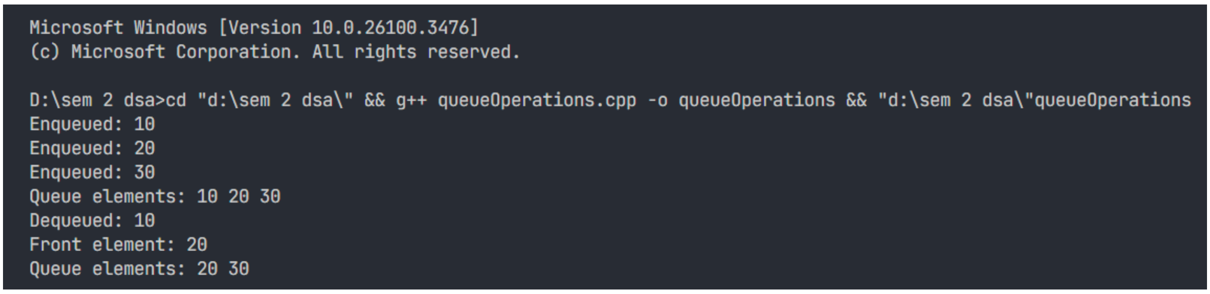


```cpp
struct Node {
    int data;
    Node* next;
};
class Queue {
    Node *front, *rear;
public:
    Queue() {
        front = rear = NULL;
    }
    void enqueue(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = NULL;

        if (rear == NULL) {
            front = rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
        cout << "Enqueued: " << value << endl;
    }
    void dequeue() {
        if (front == NULL) {
            cout << "Queue Underflow" << endl;
            return;
        }
        Node* temp = front;
```



```cpp
        front = front->next;
        cout << "Dequeued: " << temp->data << endl;
        delete temp;

        if (front == NULL) rear = NULL;
    }
    void peek() {
        if (front == NULL) {
            cout << "empty" << endl;
            return;
        }
        cout << "Front element:" << front->data << endl;
    }
    void display() {
        if (front == NULL) {
            cout << "empty!" << endl;
            return;
        }
        Node* temp = front;
        cout ;
        while (temp != NULL) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};
```

Output:

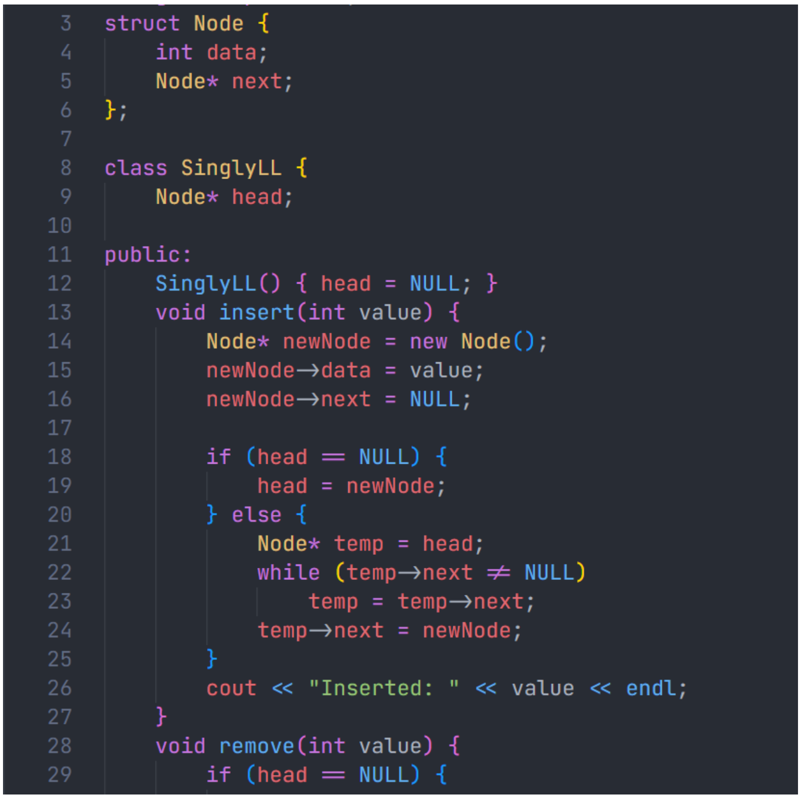


```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

D:\sem 2 dsa>cd "d:\sem 2 dsa\" && g++ queueOperations.cpp -o queueOperations && "d:\sem 2 dsa\"queueOperations
Enqueued: 10
Enqueued: 20
Enqueued: 30
Queue elements: 10 20 30
Dequeued: 10
Front element: 20
Queue elements: 20 30
```

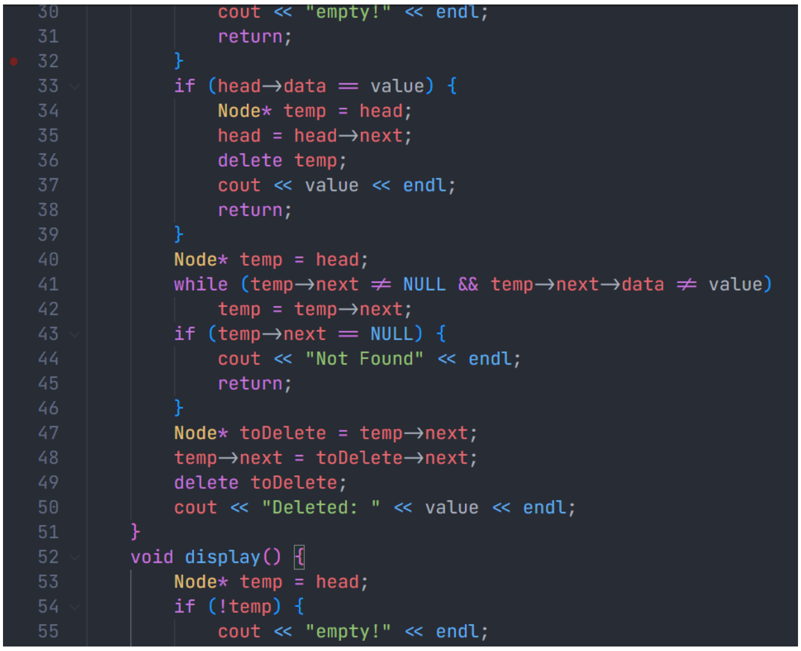
Linked Lists

Q4. Write a program to implement a linked list i.e., singly linked list, doubly linked list. Write a program to insert a node in a linked list and delete a node from a linked list

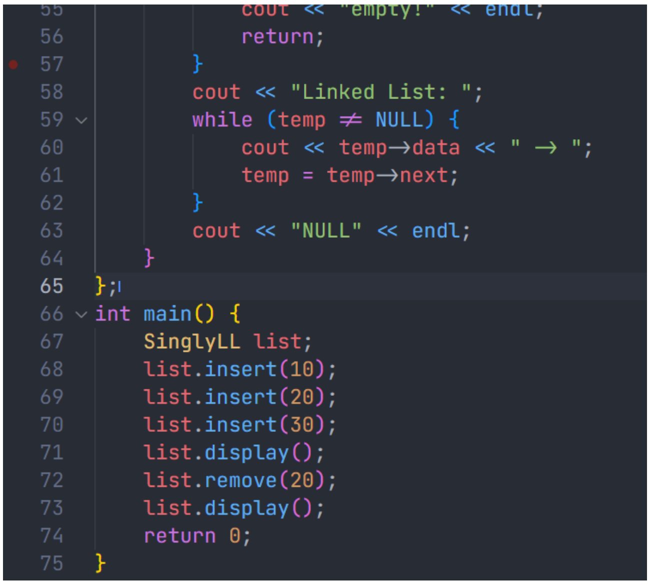
Code:



```cpp
struct Node {
    int data;
    Node* next;
};

class SinglyLL {
    Node* head;

public:
    SinglyLL() { head = NULL; }
    void insert(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = NULL;

        if (head == NULL) {
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->next != NULL)
                temp = temp->next;
            temp->next = newNode;
        }
        cout << "Inserted: " << value << endl;
    }
    void remove(int value) {
        if (head == NULL) {
```

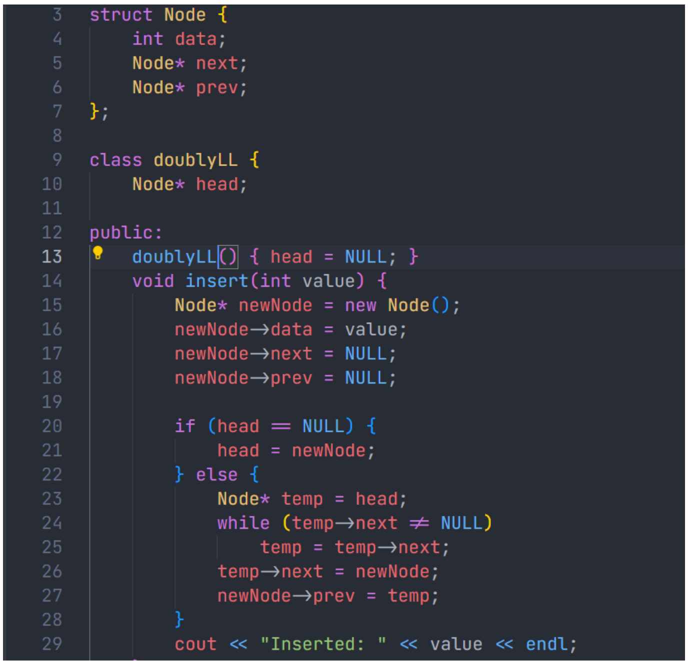


```cpp
            cout << "empty!" << endl;
            return;
        }
        if (head->data == value) {
            Node* temp = head;
            head = head->next;
            delete temp;
            cout << value << endl;
            return;
        }
        Node* temp = head;
        while (temp->next != NULL && temp->next->data != value)
            temp = temp->next;
        if (temp->next == NULL) {
            cout << "Not Found" << endl;
            return;
        }
        Node* toDelete = temp->next;
        temp->next = toDelete->next;
        delete toDelete;
        cout << "Deleted: " << value << endl;
    }
    void display() {
        Node* temp = head;
        if (!temp) {
            cout << "empty!" << endl;
```

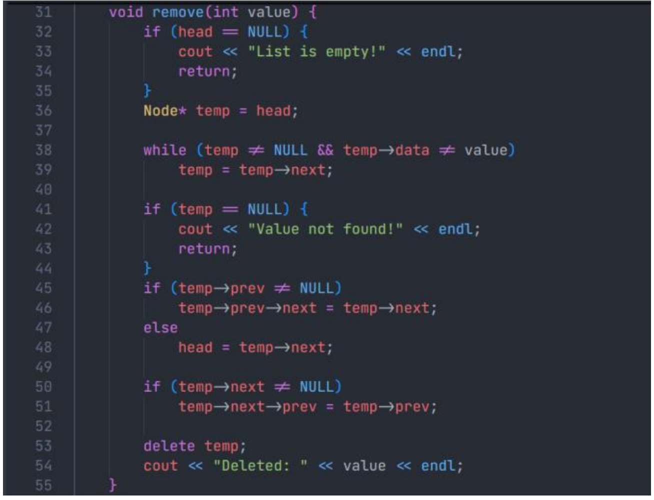


```cpp
            cout << "empty!" << endl;
            return;
        }
        cout << "Linked List: ";
        while (temp != NULL) {
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "NULL" << endl;
    }
};
int main() {
    SinglyLL list;
    list.insert(10);
    list.insert(20);
    list.insert(30);
    list.display();
    list.remove(20);
    list.display();
    return 0;
}
```

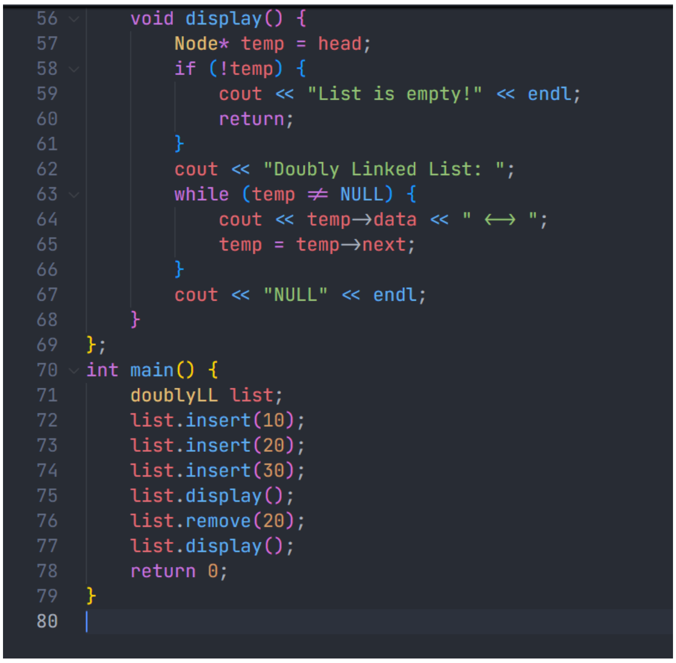
Doubly Linked Lists



```cpp
struct Node {
    int data;
    Node* next;
    Node* prev;
};

class doublyLL {
    Node* head;

public:
    doublyLL() { head = NULL; }
    void insert(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = NULL;
        newNode->prev = NULL;

        if (head == NULL) {
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->next != NULL)
                temp = temp->next;
            temp->next = newNode;
            newNode->prev = temp;
        }
        cout << "Inserted: " << value << endl;
```

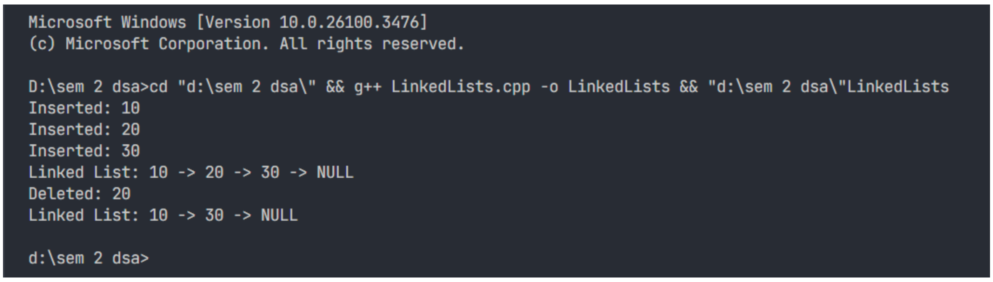


```cpp
    void remove(int value) {
        if (head == NULL) {
            cout << "List is empty!" << endl;
            return;
        }
        Node* temp = head;

        while (temp != NULL && temp->data != value)
            temp = temp->next;

        if (temp == NULL) {
            cout << "Value not found!" << endl;
            return;
        }
        if (temp->prev != NULL)
            temp->prev->next = temp->next;
        else
            head = temp->next;

        if (temp->next != NULL)
            temp->next->prev = temp->prev;

        delete temp;
        cout << "Deleted: " << value << endl;
    }
```

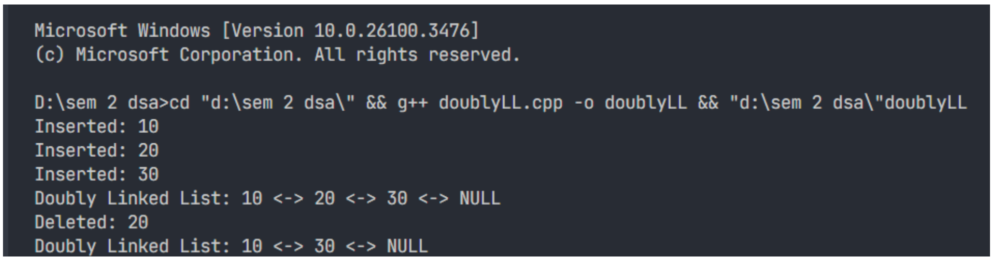


```cpp
    void display() {
        Node* temp = head;
        if (!temp) {
            cout << "List is empty!" << endl;
            return;
        }
        cout << "Doubly Linked List: ";
        while (temp != NULL) {
            cout << temp->data << " <-> ";
            temp = temp->next;
        }
        cout << "NULL" << endl;
    }
};
int main() {
    doublyLL list;
    list.insert(10);
    list.insert(20);
    list.insert(30);
    list.display();
    list.remove(20);
    list.display();
    return 0;
}

```

Output:



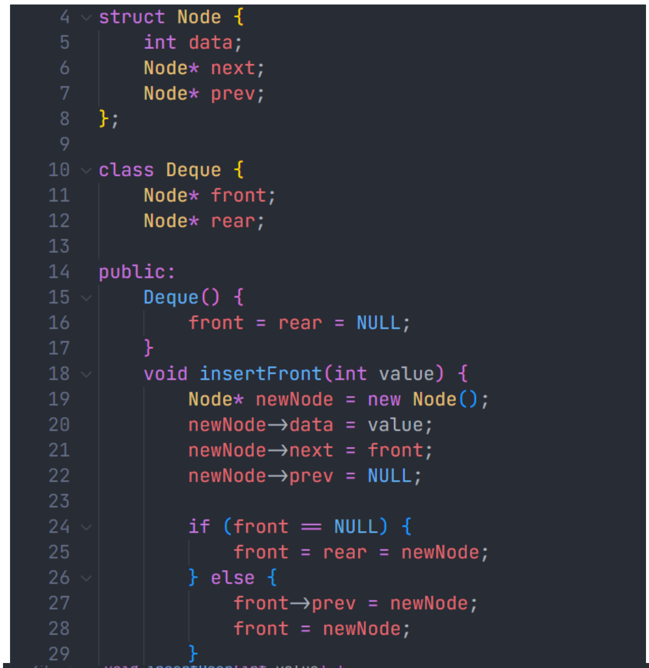
```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

D:\sem 2 dsa>cd "d:\sem 2 dsa\" && g++ LinkedLists.cpp -o LinkedLists && "d:\sem 2 dsa\"LinkedLists
Inserted: 10
Inserted: 20
Inserted: 30
Linked List: 10 -> 20 -> 30 -> NULL
Deleted: 20
Linked List: 10 -> 30 -> NULL

d:\sem 2 dsa>
```



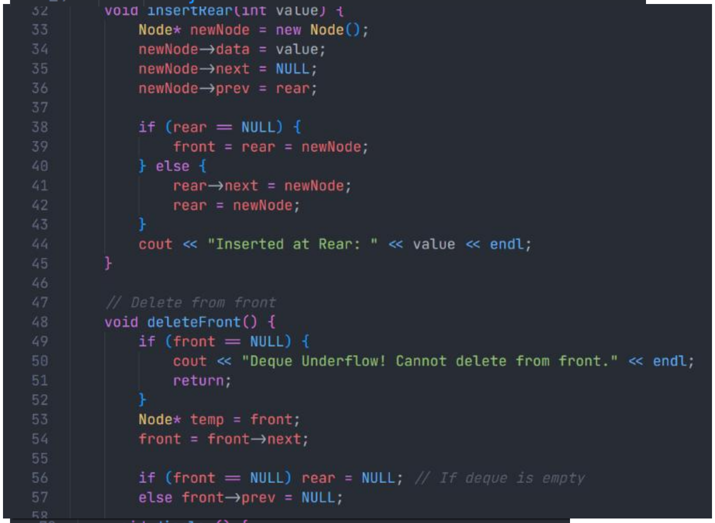
```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

D:\sem 2 dsa>cd "d:\sem 2 dsa\" && g++ doublyLL.cpp -o doublyLL && "d:\sem 2 dsa\"doublyLL
Inserted: 10
Inserted: 20
Inserted: 30
Doubly Linked List: 10 <-> 20 <-> 30 <-> NULL
Deleted: 20
Doubly Linked List: 10 <-> 30 <-> NULL
```

Q5. Write a program to implement a double-ended queue using a linked list

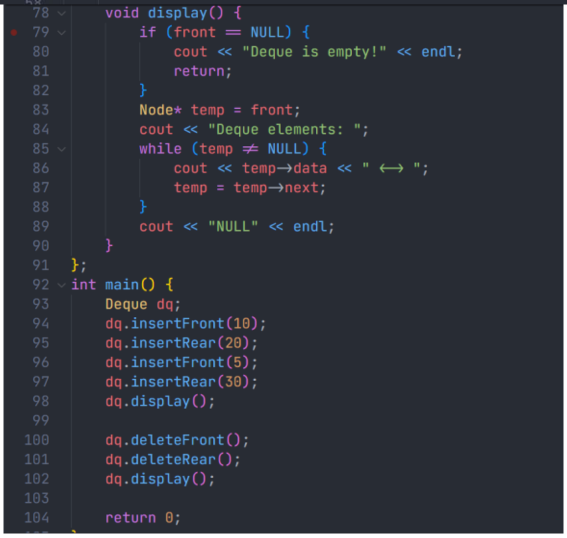
Code:



```cpp
struct Node {
    int data;
    Node* next;
    Node* prev;
};

class Deque {
    Node* front;
    Node* rear;

public:
    Deque() {
        front = rear = NULL;
    }
    void insertFront(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = front;
        newNode->prev = NULL;

        if (front == NULL) {
            front = rear = newNode;
        } else {
            front->prev = newNode;
            front = newNode;
        }
```

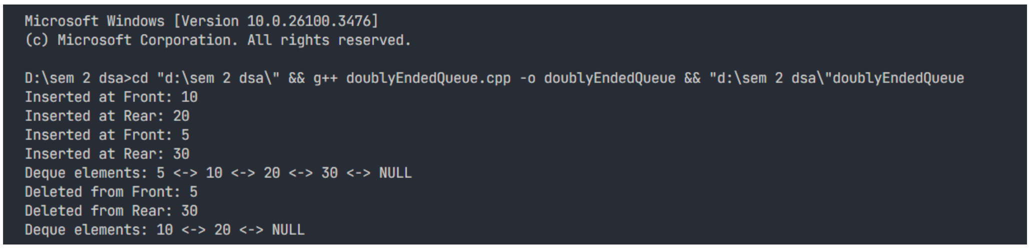


```cpp
    void insertRear(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = NULL;
        newNode->prev = rear;

        if (rear == NULL) {
            front = rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
        cout << "Inserted at Rear: " << value << endl;
    }

    // Delete from front
    void deleteFront() {
        if (front == NULL) {
            cout << "Deque Underflow! Cannot delete from front." << endl;
            return;
        }
        Node* temp = front;
        front = front->next;

        if (front == NULL) rear = NULL; // If deque is empty
        else front->prev = NULL;
```



```cpp
    void display() {
        if (front == NULL) {
            cout << "Deque is empty!" << endl;
            return;
        }
        Node* temp = front;
        cout << "Deque elements: ";
        while (temp != NULL) {
            cout << temp->data << " <-> ";
            temp = temp->next;
        }
        cout << "NULL" << endl;
    }
};
int main() {
    Deque dq;
    dq.insertFront(10);
    dq.insertRear(20);
    dq.insertFront(5);
    dq.insertRear(30);
    dq.display();

    dq.deleteFront();
    dq.deleteRear();
    dq.display();

    return 0;
```

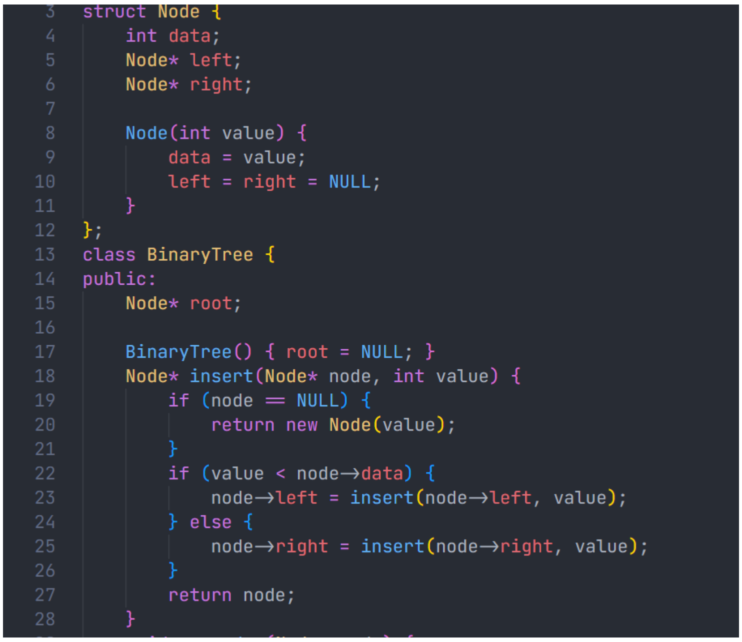
Output:



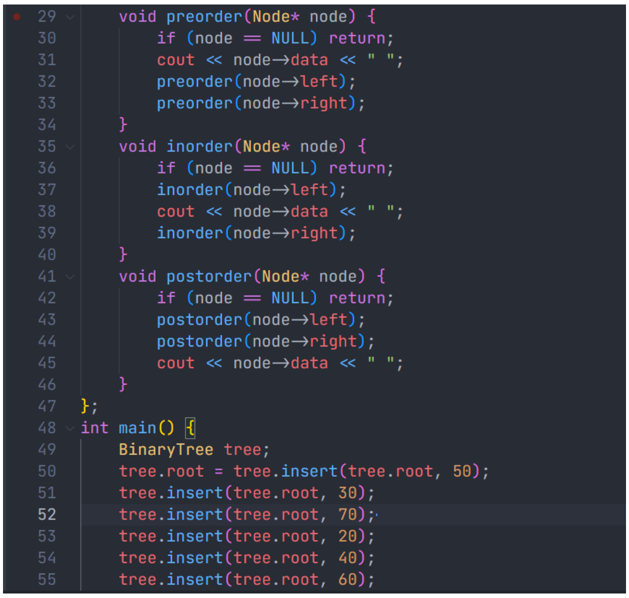
```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

D:\sem 2 dsa>cd "d:\sem 2 dsa\" && g++ doublyEndedQueue.cpp -o doublyEndedQueue && "d:\sem 2 dsa\"doublyEndedQueue
Inserted at Front: 10
Inserted at Rear: 20
Inserted at Front: 5
Inserted at Rear: 30
Deque elements: 5 <-> 10 <-> 20 <-> 30 <-> NULL
Deleted from Front: 5
Deleted from Rear: 30
Deque elements: 10 <-> 20 <-> NULL
```

Q6. Write a program to construct a binary tree and display its preorder, inorder and postorder traversals.

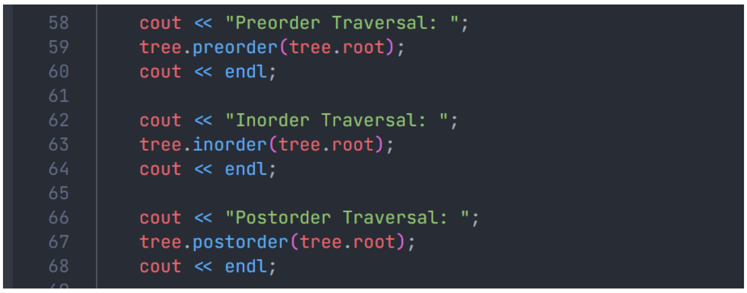
Code:



```cpp
struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int value) {
        data = value;
        left = right = NULL;
    }
};
class BinaryTree {
public:
    Node* root;

    BinaryTree() { root = NULL; }
    Node* insert(Node* node, int value) {
        if (node == NULL) {
            return new Node(value);
        }
        if (value < node->data) {
            node->left = insert(node->left, value);
        } else {
            node->right = insert(node->right, value);
        }
        return node;
    }
```

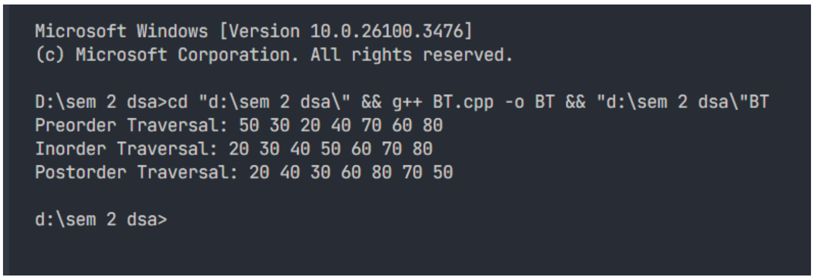


```cpp
    void preorder(Node* node) {
        if (node == NULL) return;
        cout << node->data << " ";
        preorder(node->left);
        preorder(node->right);
    }
    void inorder(Node* node) {
        if (node == NULL) return;
        inorder(node->left);
        cout << node->data << " ";
        inorder(node->right);
    }
    void postorder(Node* node) {
        if (node == NULL) return;
        postorder(node->left);
        postorder(node->right);
        cout << node->data << " ";
    }
};
int main() {
    BinaryTree tree;
    tree.root = tree.insert(tree.root, 50);
    tree.insert(tree.root, 30);
    tree.insert(tree.root, 70);
    tree.insert(tree.root, 20);
    tree.insert(tree.root, 40);
    tree.insert(tree.root, 60);
```



```cpp
    cout << "Preorder Traversal: ";
    tree.preorder(tree.root);
    cout << endl;

    cout << "Inorder Traversal: ";
    tree.inorder(tree.root);
    cout << endl;

    cout << "Postorder Traversal: ";
    tree.postorder(tree.root);
    cout << endl;
```

Output:



```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

D:\sem 2 dsa>cd "d:\sem 2 dsa\" && g++ BT.cpp -o BT && "d:\sem 2 dsa\"BT
Preorder Traversal: 50 30 20 40 70 60 80
Inorder Traversal: 20 30 40 50 60 70 80
Postorder Traversal: 20 40 30 60 80 70 50

d:\sem 2 dsa>
```