School of Engineering & Technology

VIPUL

BTECH CSE FSD

2401350017

FACULTY NAME:- Dr. Vandana Batra

Data Structures Lab Manual

ENCS205, ENCA 201

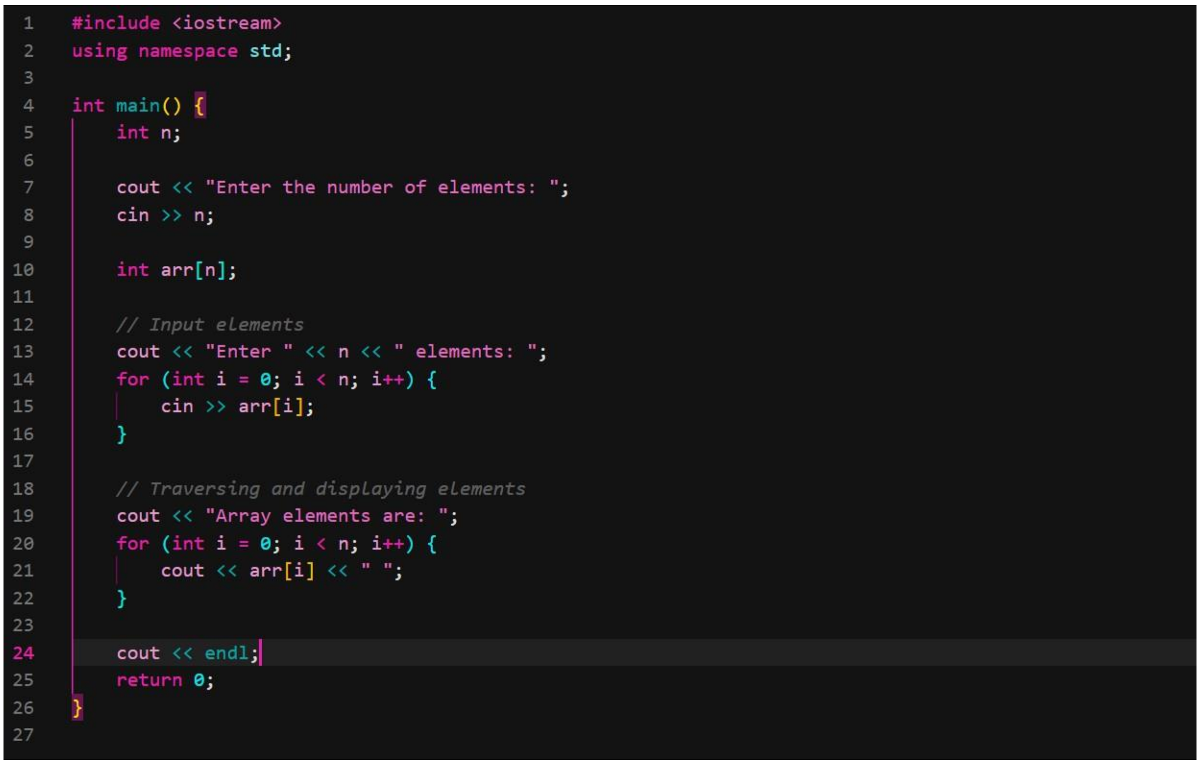
(2025-2026)

Index

	edge cases such as duplicate values and nearly sorted arrays.	
13.	Given preorder and in-order traversal of a tree, construct the binary tree.	
14.	Perform the traversal of graph (DFS, BFS)	
15.	Implement Prim's and Kruskal’s algorithm to find the minimum spanning tree of a graph.	
16.	Implement Dijkstra’s algorithm to find the shortest path from a source vertex to all other vertices in a weighted graph. Use both adjacency matrix and adjacency list representations for the graph. Ensure the algorithm handles negative weights appropriately.	

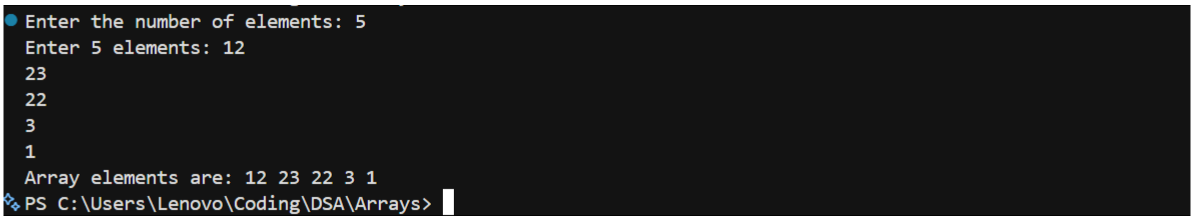
Problem1: Given an array of integers, perform the following operations: traversing, insertion, deletion

Input (traversal code):



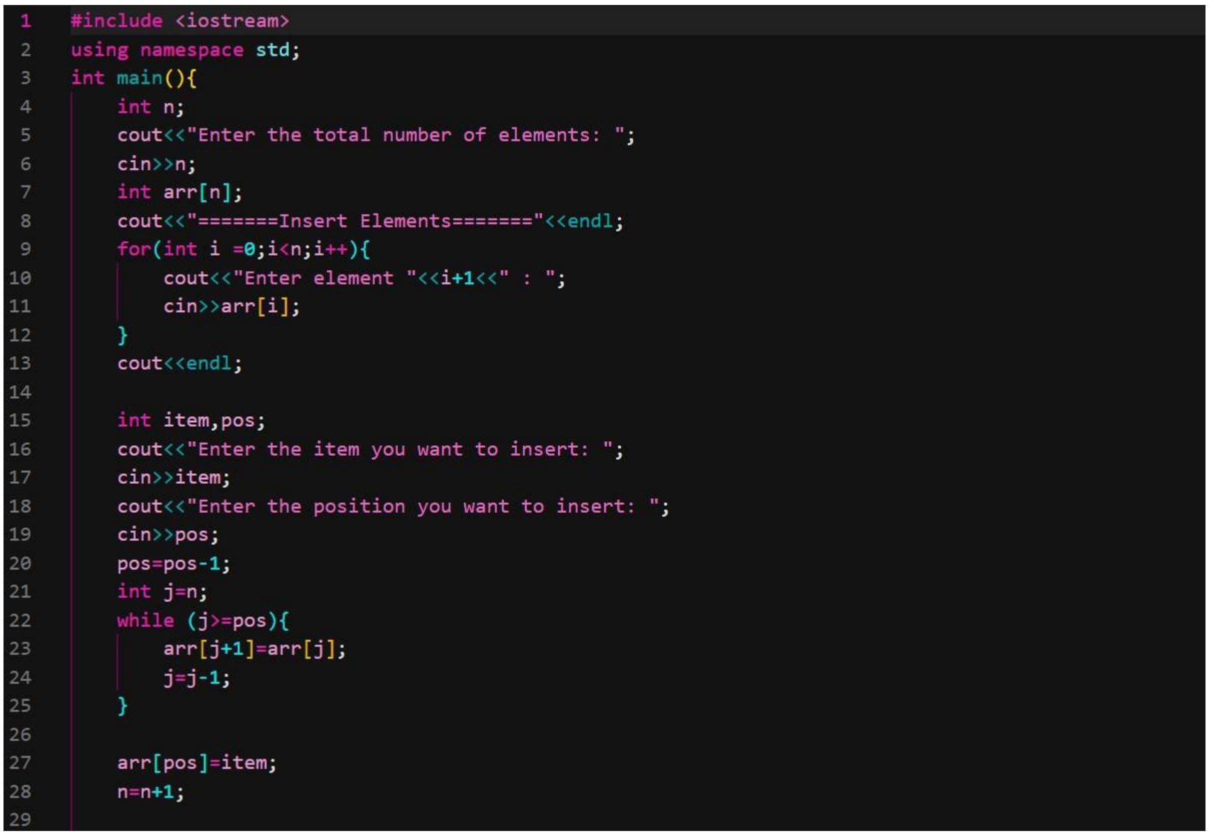
```cpp
#include <iostream>
using namespace std;

int main() {
    int n;

    cout << "Enter the number of elements: ";
    cin >> n;

    int arr[n];

    // Input elements
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    // Traversing and displaying elements
    cout << "Array elements are: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }

    cout << endl;
    return 0;
}

```

Output (traversal code):

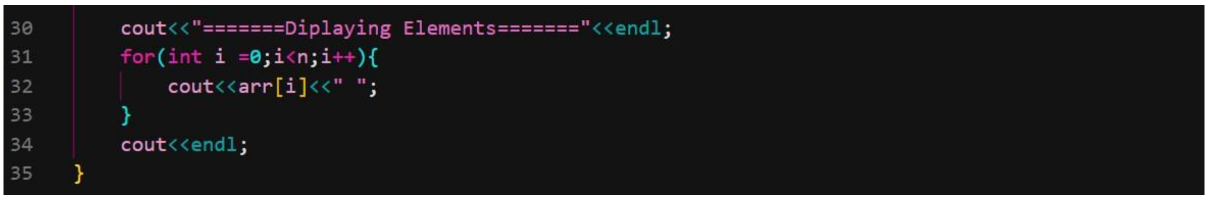


```
Enter the number of elements: 5
Enter 5 elements: 12
23
22
3
1
Array elements are: 12 23 22 3 1
PS C:\Users\Lenovo\Coding\DSA\Arrays>
```

Input (Insertion code):

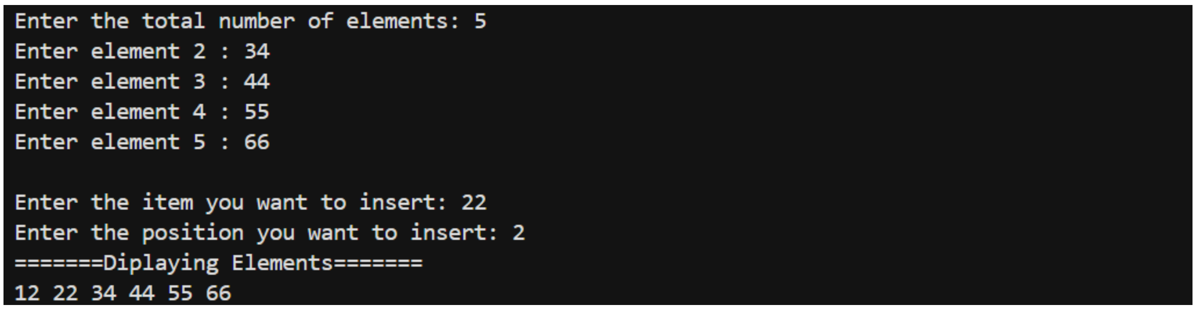


```cpp
#include <iostream>
using namespace std;
int main(){
    int n;
    cout<<"Enter the total number of elements: ";
    cin>>n;
    int arr[n];
    cout<<"=======Insert Elements======="<<endl;
    for(int i =0;i<n;i++){
        cout<<"Enter element "<<i+1<<" : ";
        cin>>arr[i];
    }
    cout<<endl;

    int item,pos;
    cout<<"Enter the item you want to insert: ";
    cin>>item;
    cout<<"Enter the position you want to insert: ";
    cin>>pos;
    pos=pos-1;
    int j=n;
    while (j>=pos){
        arr[j+1]=arr[j];
        j=j-1;
    }

    arr[pos]=item;
    n=n+1;

```



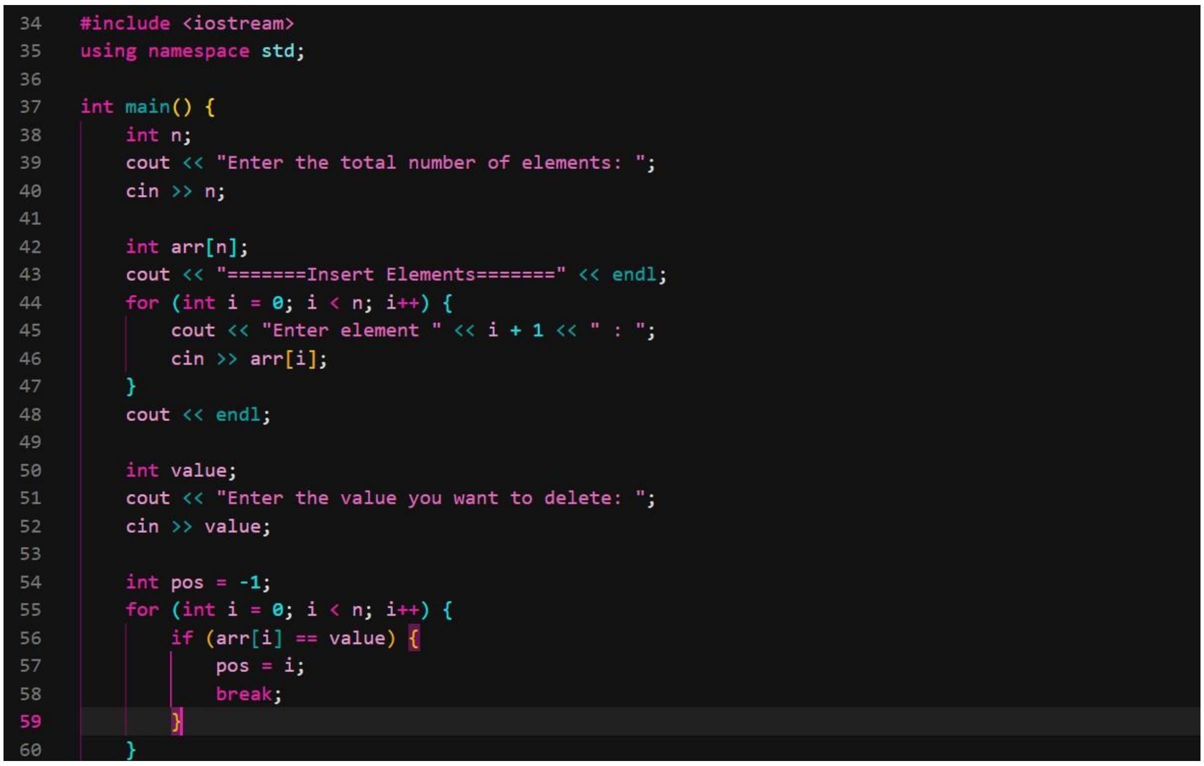
```cpp
    cout<<"=======Diplaying Elements======="<<endl;
    for(int i =0;i<n;i++){
        cout<<arr[i]<<" ";
    }
    cout<<endl;
}
```

Output (insertion code):

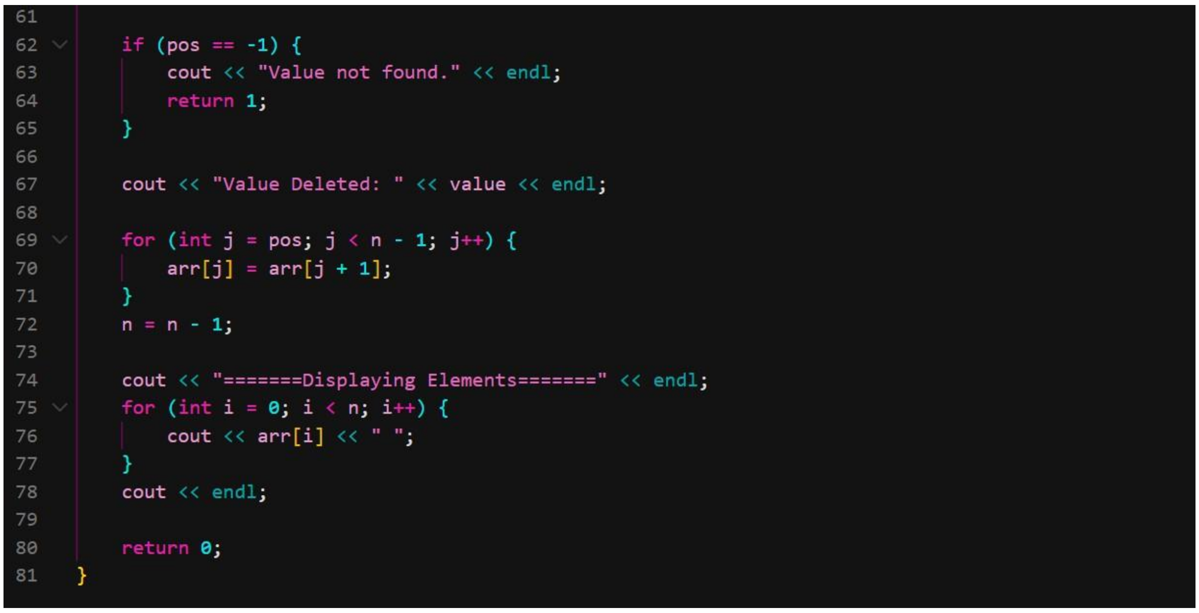


```
Enter the total number of elements: 5
Enter element 2 : 34
Enter element 3 : 44
Enter element 4 : 55
Enter element 5 : 66

Enter the item you want to insert: 22
Enter the position you want to insert: 2
=======Diplaying Elements=======
12 22 34 44 55 66
```

Input (deletion code):

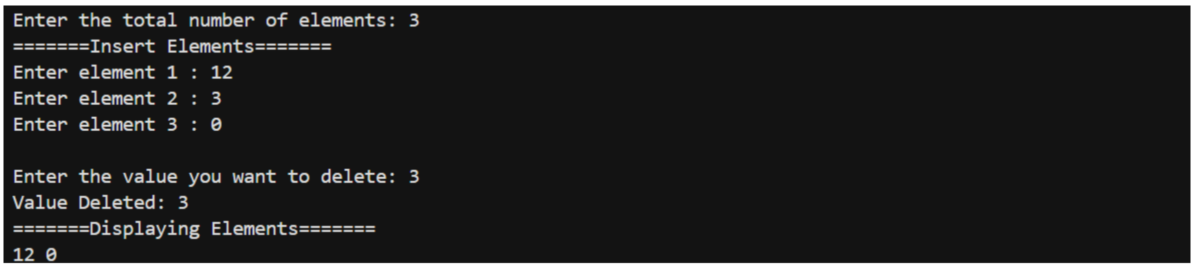


```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter the total number of elements: ";
    cin >> n;

    int arr[n];
    cout << "=======Insert Elements=======" << endl;
    for (int i = 0; i < n; i++) {
        cout << "Enter element " << i + 1 << " : ";
        cin >> arr[i];
    }
    cout << endl;

    int value;
    cout << "Enter the value you want to delete: ";
    cin >> value;

    int pos = -1;
    for (int i = 0; i < n; i++) {
        if (arr[i] == value) {
            pos = i;
            break;
        }
    }
```



```cpp

    if (pos == -1) {
        cout << "Value not found." << endl;
        return 1;
    }

    cout << "Value Deleted: " << value << endl;

    for (int j = pos; j < n - 1; j++) {
        arr[j] = arr[j + 1];
    }
    n = n - 1;

    cout << "=======Displaying Elements=======" << endl;
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

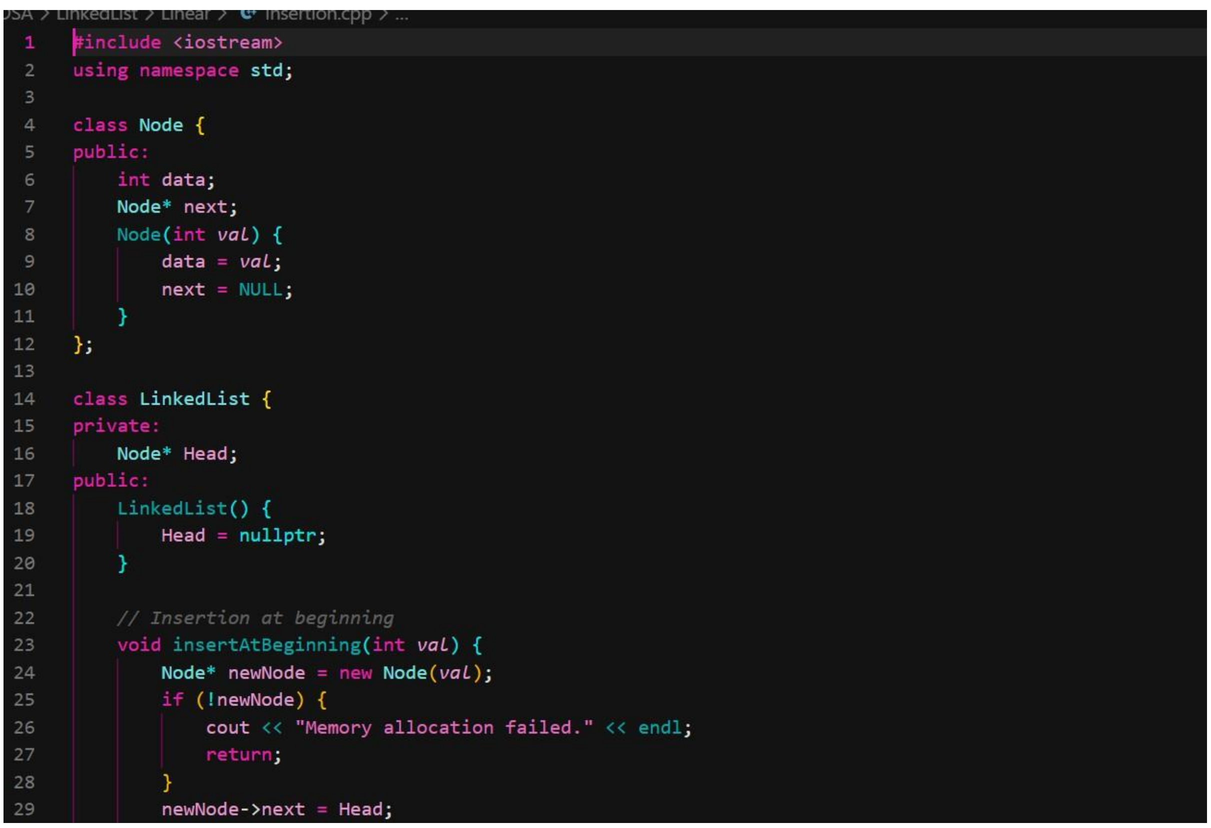
Output (deletion code):



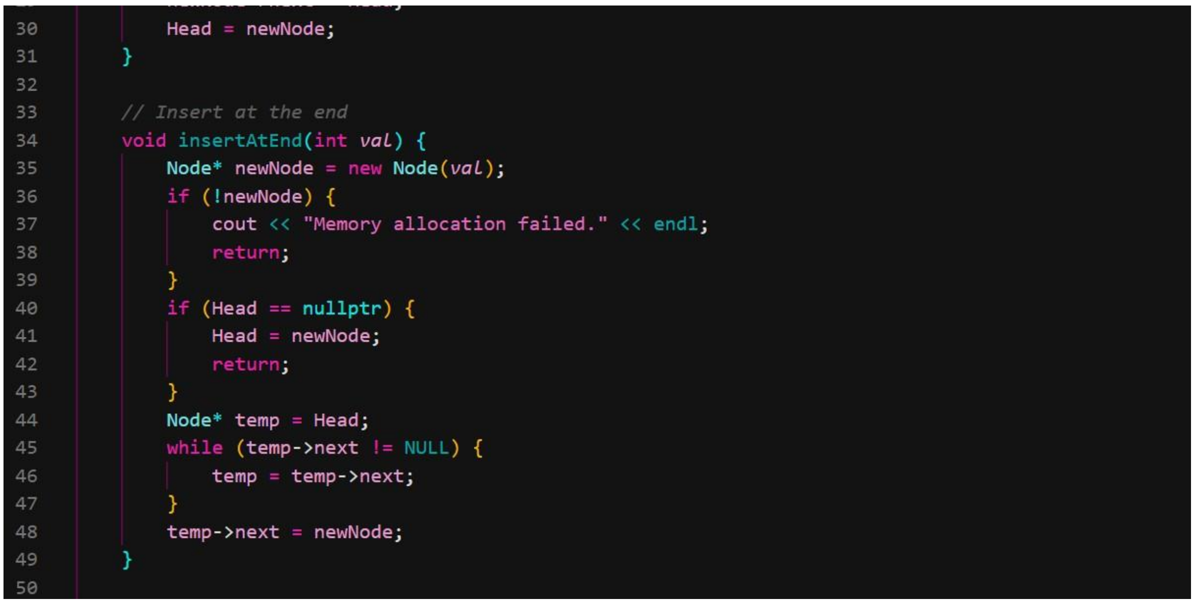
```
Enter the total number of elements: 3
=======Insert Elements=======
Enter element 1 : 12
Enter element 2 : 3
Enter element 3 : 0

Enter the value you want to delete: 3
Value Deleted: 3
=======Displaying Elements=======
12 0
```

<u>Problem2</u>: Write a program to implement a singly linked list with methods to insert an element at the head, insert an element at the tail, delete an element by value, and traverse the list to print all elements.

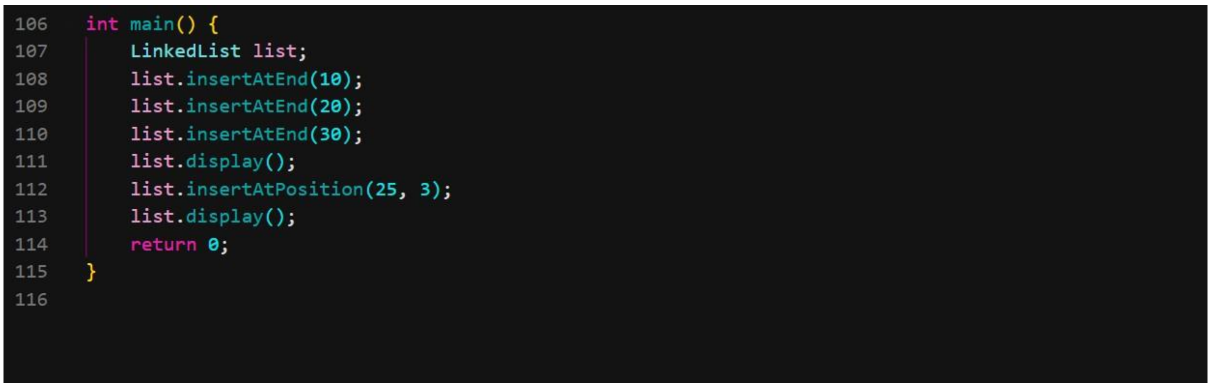
Input (Insertion code):



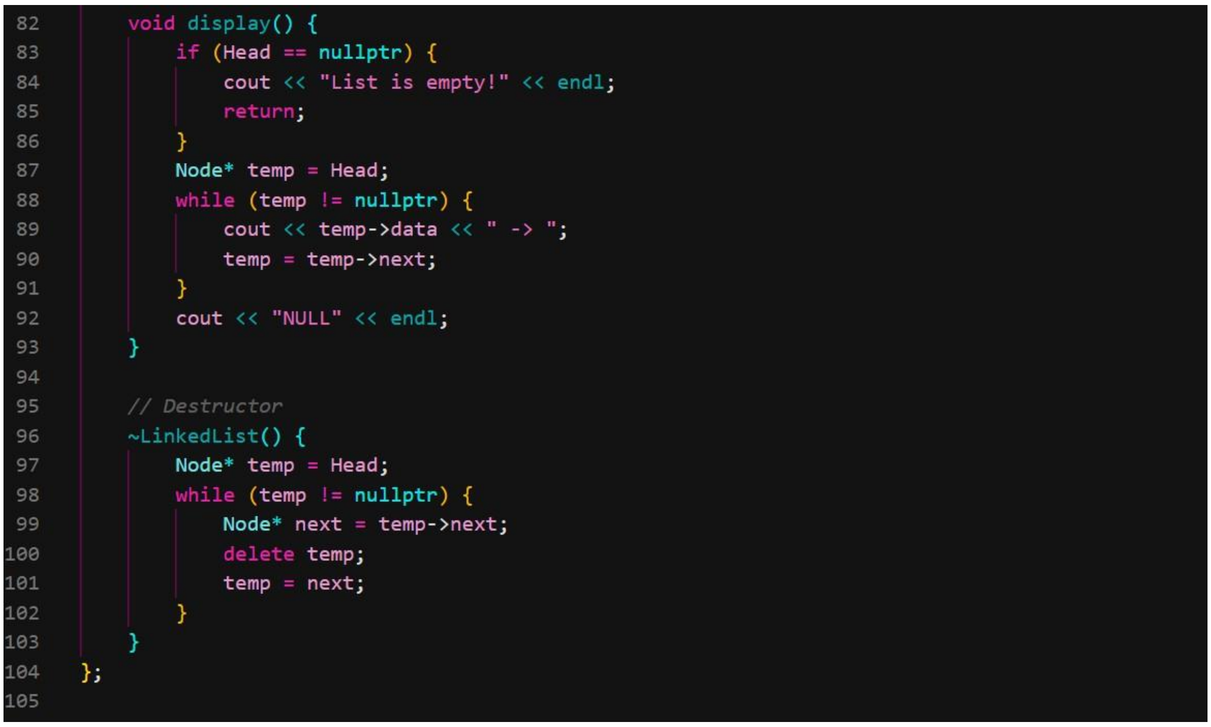
```cpp
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = NULL;
    }
};

class LinkedList {
private:
    Node* Head;
public:
    LinkedList() {
        Head = nullptr;
    }

    // Insertion at beginning
    void insertAtBeginning(int val) {
        Node* newNode = new Node(val);
        if (!newNode) {
            cout << "Memory allocation failed." << endl;
            return;
        }
        newNode->next = Head;
```



```cpp
        Head = newNode;
    }

    // Insert at the end
    void insertAtEnd(int val) {
        Node* newNode = new Node(val);
        if (!newNode) {
            cout << "Memory allocation failed." << endl;
            return;
        }
        if (Head == nullptr) {
            Head = newNode;
            return;
        }
        Node* temp = Head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }

```

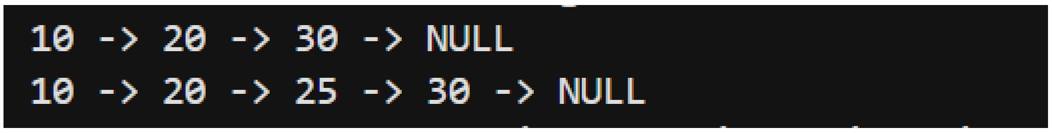


```cpp
int main() {
    LinkedList list;
    list.insertAtEnd(10);
    list.insertAtEnd(20);
    list.insertAtEnd(30);
    list.display();
    list.insertAtPosition(25, 3);
    list.display();
    return 0;
}

```



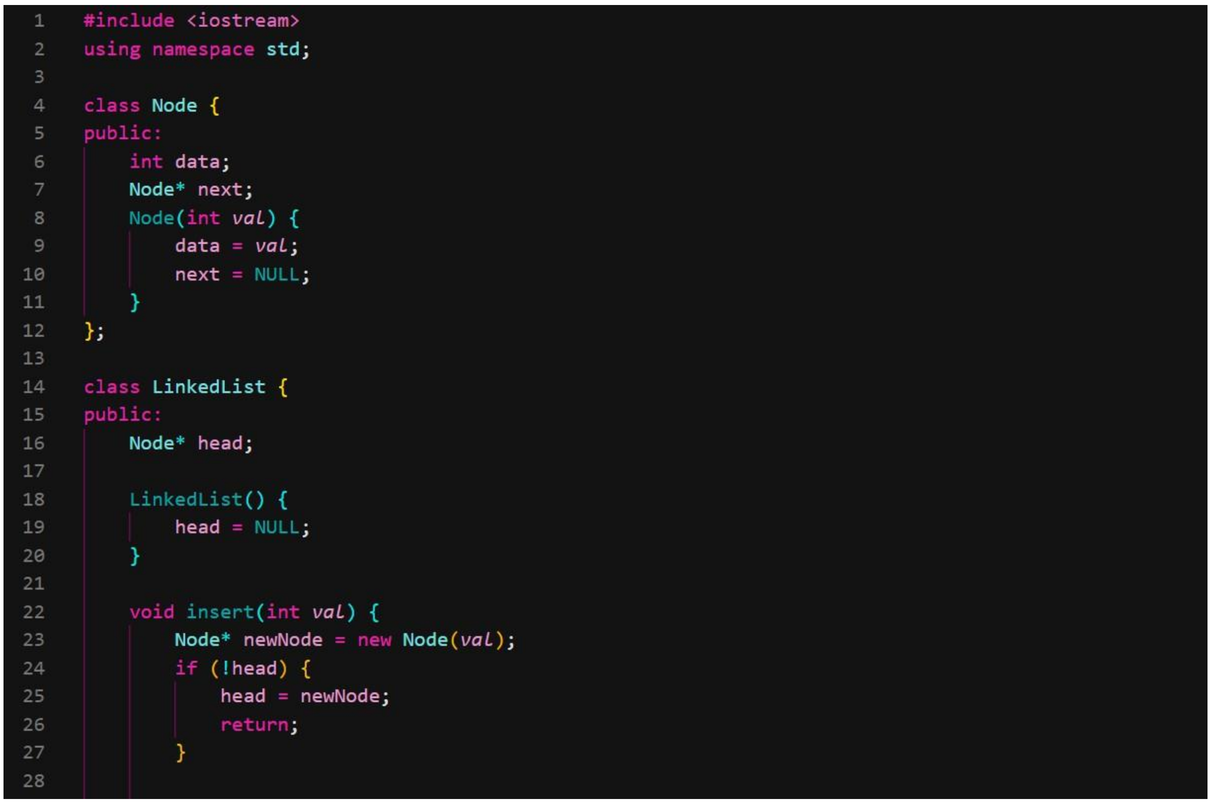
```cpp
    void display() {
        if (Head == nullptr) {
            cout << "List is empty!" << endl;
            return;
        }
        Node* temp = Head;
        while (temp != nullptr) {
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "NULL" << endl;
    }

    // Destructor
    ~LinkedList() {
        Node* temp = Head;
        while (temp != nullptr) {
            Node* next = temp->next;
            delete temp;
            temp = next;
        }
    }
};

```

Output (Insertion code):

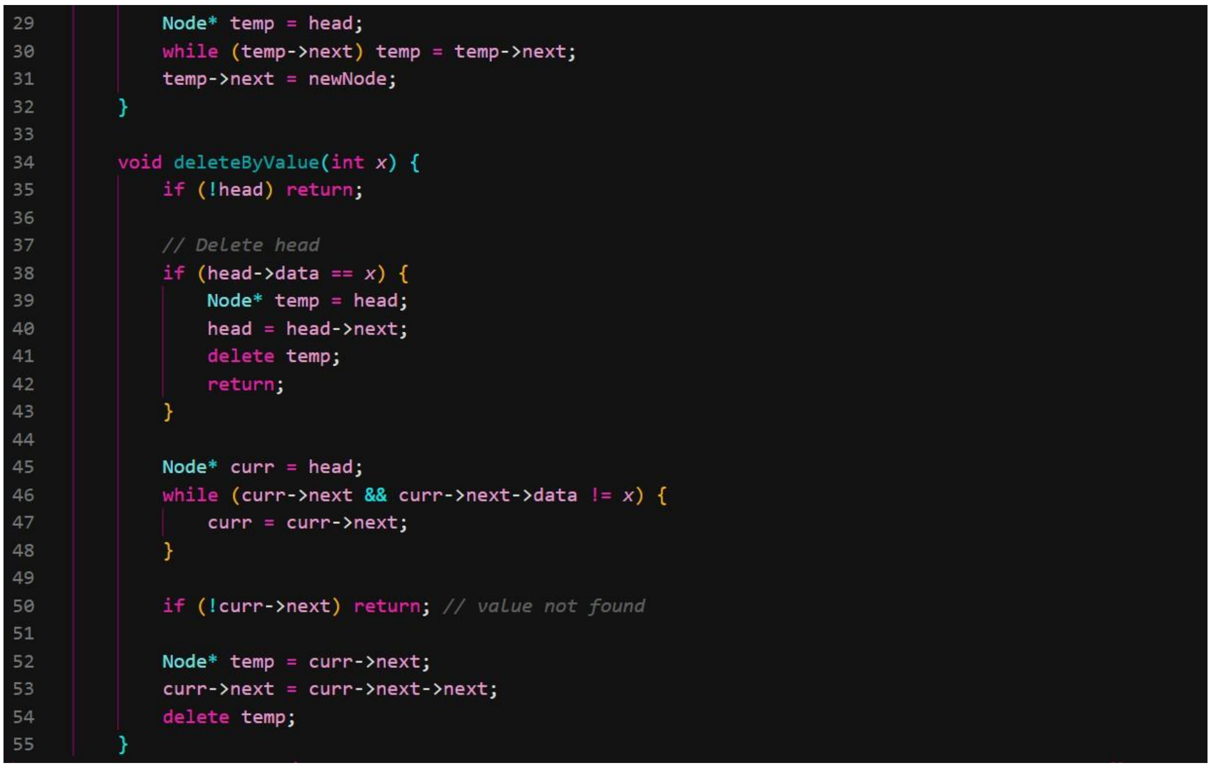


```
10 -> 20 -> 30 -> NULL
10 -> 20 -> 25 -> 30 -> NULL
```

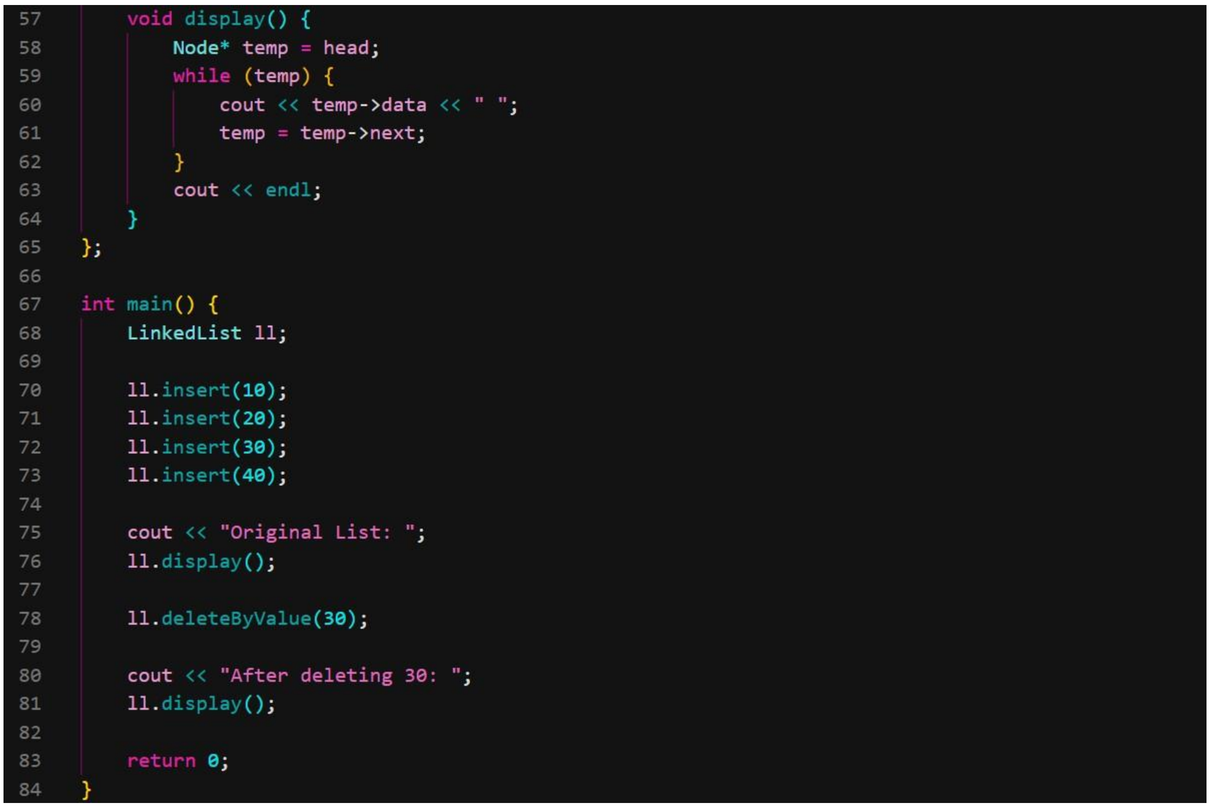
Input (Deletion code):



```cpp
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = NULL;
    }
};

class LinkedList {
public:
    Node* head;

    LinkedList() {
        head = NULL;
    }

    void insert(int val) {
        Node* newNode = new Node(val);
        if (!head) {
            head = newNode;
            return;
        }

```

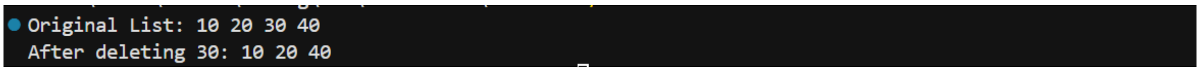


```cpp
        Node* temp = head;
        while (temp->next) temp = temp->next;
        temp->next = newNode;
    }

    void deleteByValue(int x) {
        if (!head) return;

        // Delete head
        if (head->data == x) {
            Node* temp = head;
            head = head->next;
            delete temp;
            return;
        }

        Node* curr = head;
        while (curr->next && curr->next->data != x) {
            curr = curr->next;
        }

        if (!curr->next) return; // value not found

        Node* temp = curr->next;
        curr->next = curr->next->next;
        delete temp;
    }
```



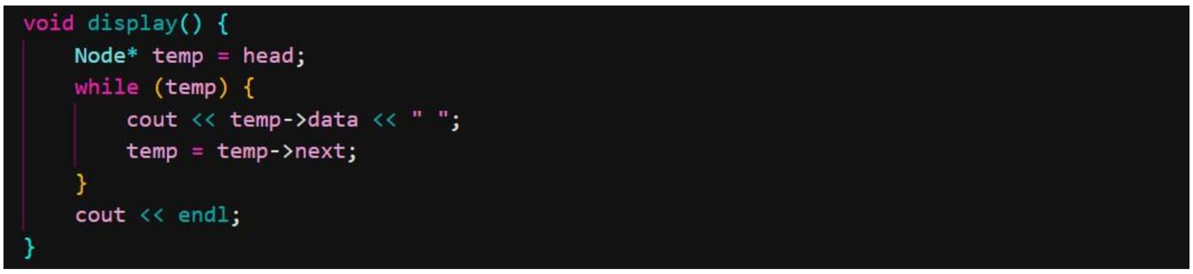
```cpp
    void display() {
        Node* temp = head;
        while (temp) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    LinkedList ll;

    ll.insert(10);
    ll.insert(20);
    ll.insert(30);
    ll.insert(40);

    cout << "Original List: ";
    ll.display();

    ll.deleteByValue(30);

    cout << "After deleting 30: ";
    ll.display();

    return 0;
}
```

Output (Deletion code):



```
Original List: 10 20 30 40
After deleting 30: 10 20 40
```

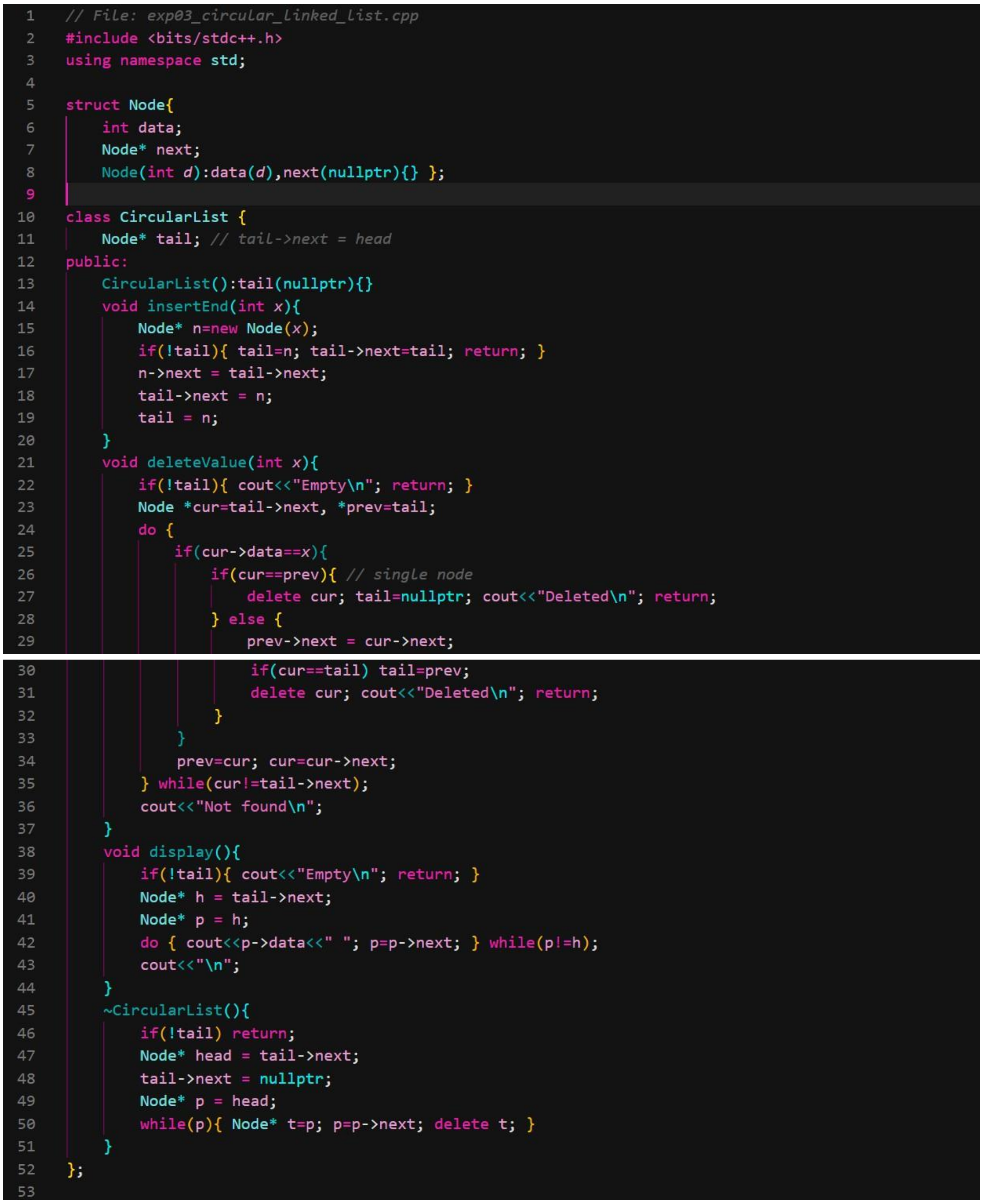
Code (Traversing)



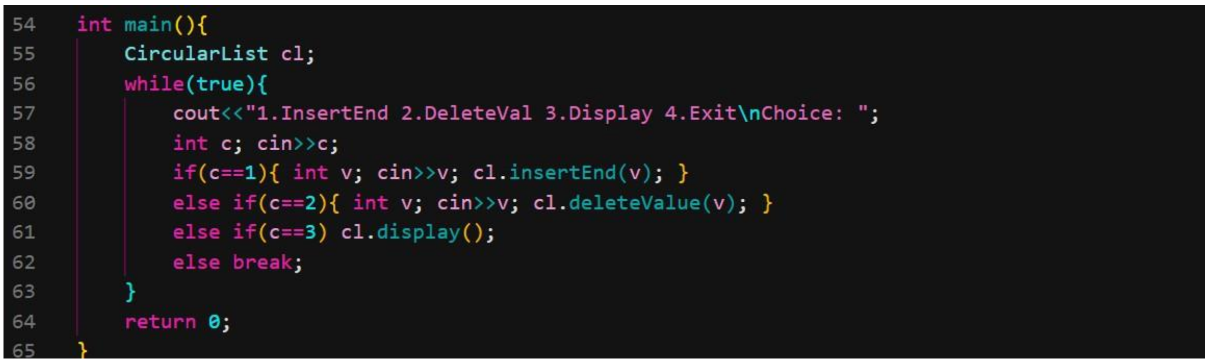
```cpp
void display() {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
```

Problem3: **Write a class to implement a circular linked list with methods to insert an element at the head, insert an element at the tail, delete an element by value, and traverse the list to print all elements.**

Input:

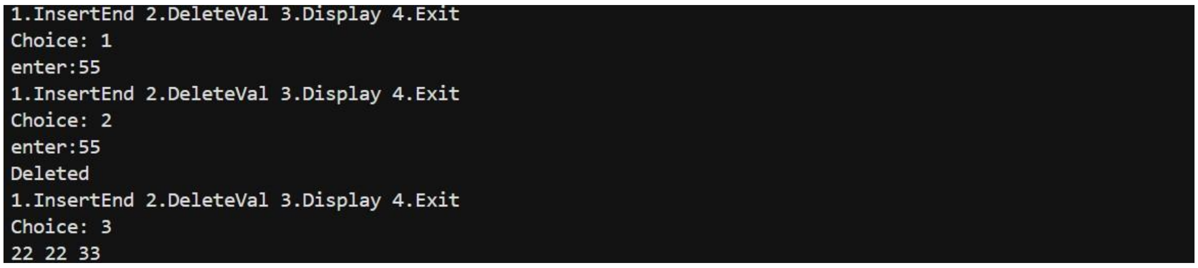


```cpp
// File: exp03_circular_linked_list.cpp
#include <bits/stdc++.h>
using namespace std;

struct Node{
    int data;
    Node* next;
    Node(int d):data(d),next(nullptr){} };

class CircularList {
    Node* tail; // tail->next = head
public:
    CircularList():tail(nullptr){}
    void insertEnd(int x){
        Node* n=new Node(x);
        if(!tail){ tail=n; tail->next=tail; return; }
        n->next = tail->next;
        tail->next = n;
        tail = n;
    }
    void deleteValue(int x){
        if(!tail){ cout<<"Empty\n"; return; }
        Node *cur=tail->next, *prev=tail;
        do {
            if(cur->data==x){
                if(cur==prev){ // single node
                    delete cur; tail=nullptr; cout<<"Deleted\n"; return;
                } else {
                    prev->next = cur->next;
                    if(cur==tail) tail=prev;
                    delete cur; cout<<"Deleted\n"; return;
                }
            }
            prev=cur; cur=cur->next;
        } while(cur!=tail->next);
        cout<<"Not found\n";
    }
    void display(){
        if(!tail){ cout<<"Empty\n"; return; }
        Node* h = tail->next;
        Node* p = h;
        do { cout<<p->data<<" "; p=p->next; } while(p!=h);
        cout<<"\n";
    }
    ~CircularList(){
        if(!tail) return;
        Node* head = tail->next;
        tail->next = nullptr;
        Node* p = head;
        while(p){ Node* t=p; p=p->next; delete t; }
    }
};

```



```cpp
int main(){
    CircularList cl;
    while(true){
        cout<<"1.InsertEnd 2.DeleteVal 3.Display 4.Exit\nChoice: ";
        int c; cin>>c;
        if(c==1){ int v; cin>>v; cl.insertEnd(v); }
        else if(c==2){ int v; cin>>v; cl.deleteValue(v); }
        else if(c==3) cl.display();
        else break;
    }
    return 0;
}
```

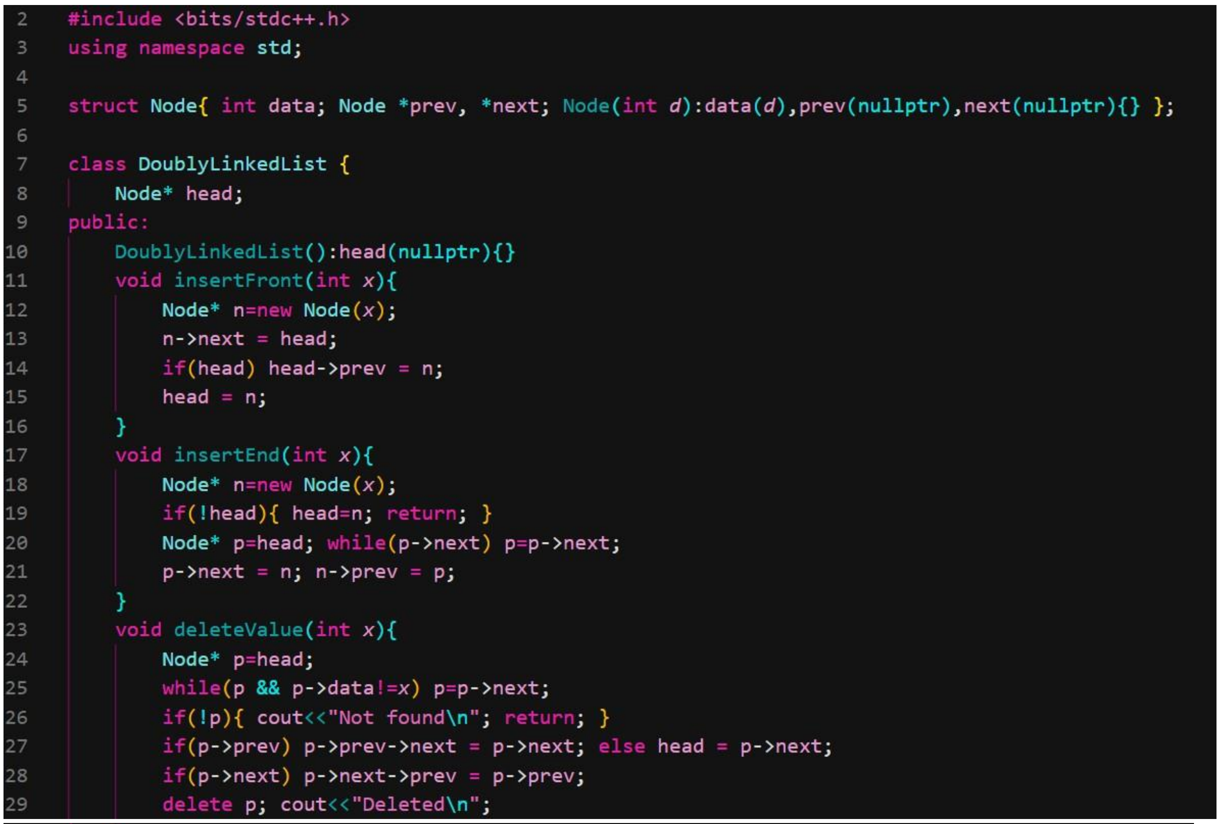
Output:



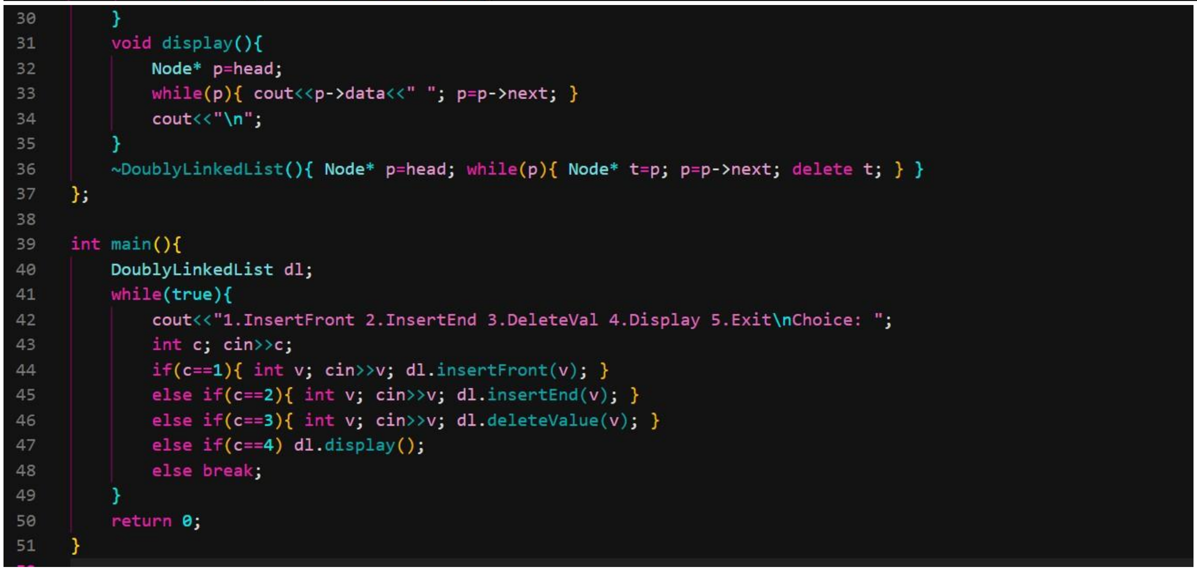
```
1.InsertEnd 2.DeleteVal 3.Display 4.Exit
Choice: 1
enter:55
1.InsertEnd 2.DeleteVal 3.Display 4.Exit
Choice: 2
enter:55
Deleted
1.InsertEnd 2.DeleteVal 3.Display 4.Exit
Choice: 3
22 22 33
```

<u>Problem4:</u> Implement a doubly linked list with methods to insert an element at the head, insert an element at the tail, delete an element by value, and reverse the list. Ensure that all operations handle edge cases appropriately.

Input:

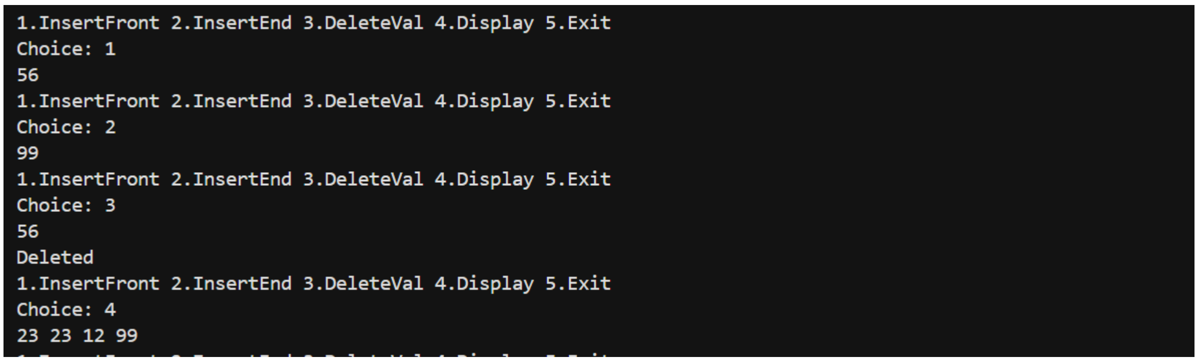


```cpp
#include <bits/stdc++.h>
using namespace std;

struct Node{ int data; Node *prev, *next; Node(int d):data(d),prev(nullptr),next(nullptr){} };

class DoublyLinkedList {
    Node* head;
public:
    DoublyLinkedList():head(nullptr){}
    void insertFront(int x){
        Node* n=new Node(x);
        n->next = head;
        if(head) head->prev = n;
        head = n;
    }
    void insertEnd(int x){
        Node* n=new Node(x);
        if(!head){ head=n; return; }
        Node* p=head; while(p->next) p=p->next;
        p->next = n; n->prev = p;
    }
    void deleteValue(int x){
        Node* p=head;
        while(p && p->data!=x) p=p->next;
        if(!p){ cout<<"Not found\n"; return; }
        if(p->prev) p->prev->next = p->next; else head = p->next;
        if(p->next) p->next->prev = p->prev;
        delete p; cout<<"Deleted\n";
```



```cpp
    }
    void display(){
        Node* p=head;
        while(p){ cout<<p->data<<" "; p=p->next; }
        cout<<"\n";
    }
    ~DoublyLinkedList(){ Node* p=head; while(p){ Node* t=p; p=p->next; delete t; } }
};

int main(){
    DoublyLinkedList dl;
    while(true){
        cout<<"1.InsertFront 2.InsertEnd 3.DeleteVal 4.Display 5.Exit\nChoice: ";
        int c; cin>>c;
        if(c==1){ int v; cin>>v; dl.insertFront(v); }
        else if(c==2){ int v; cin>>v; dl.insertEnd(v); }
        else if(c==3){ int v; cin>>v; dl.deleteValue(v); }
        else if(c==4) dl.display();
        else break;
    }
    return 0;
}
```

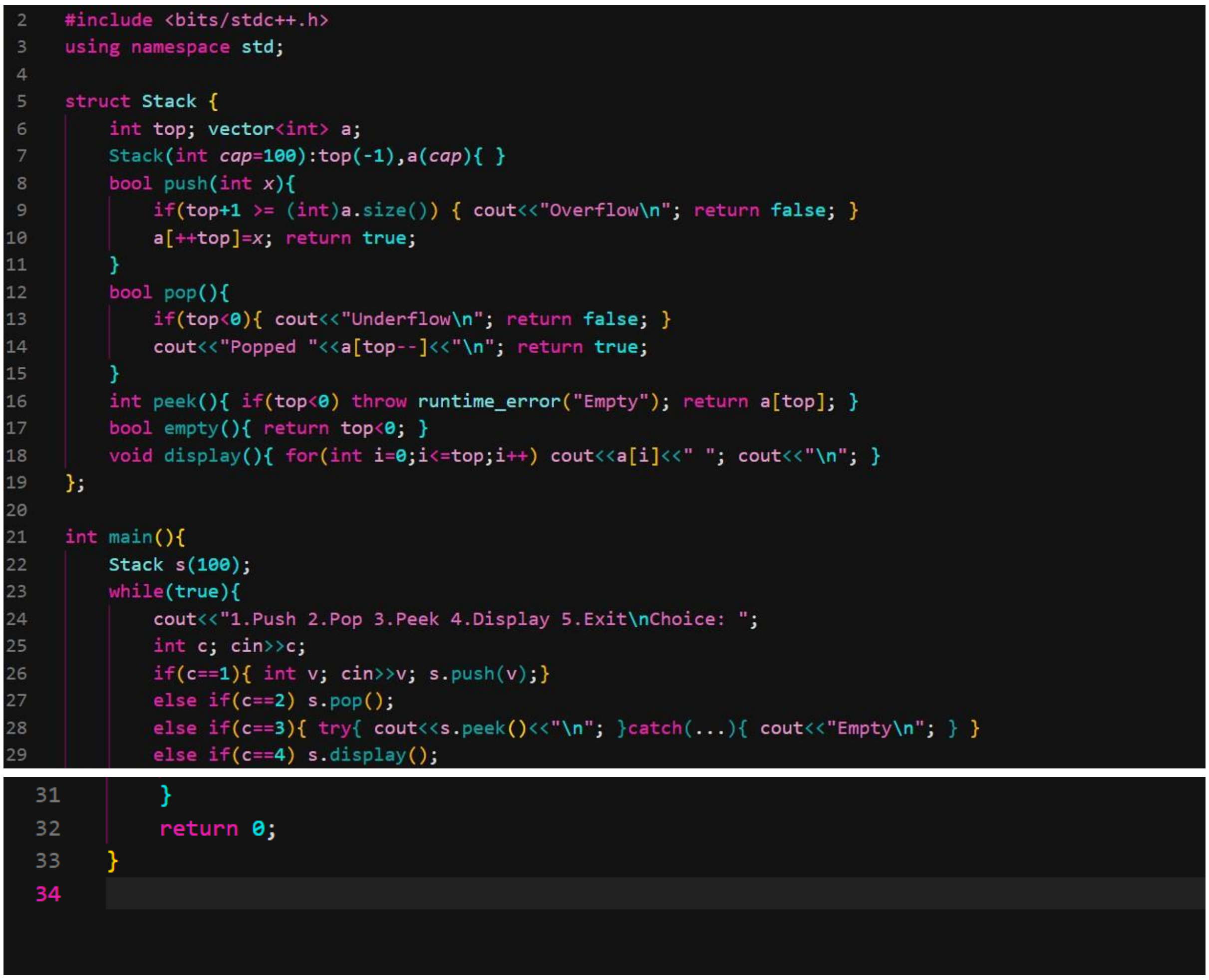
Output:



```
1.InsertFront 2.InsertEnd 3.DeleteVal 4.Display 5.Exit
Choice: 1
56
1.InsertFront 2.InsertEnd 3.DeleteVal 4.Display 5.Exit
Choice: 2
99
1.InsertFront 2.InsertEnd 3.DeleteVal 4.Display 5.Exit
Choice: 3
56
Deleted
1.InsertFront 2.InsertEnd 3.DeleteVal 4.Display 5.Exit
Choice: 4
23 23 12 99
```

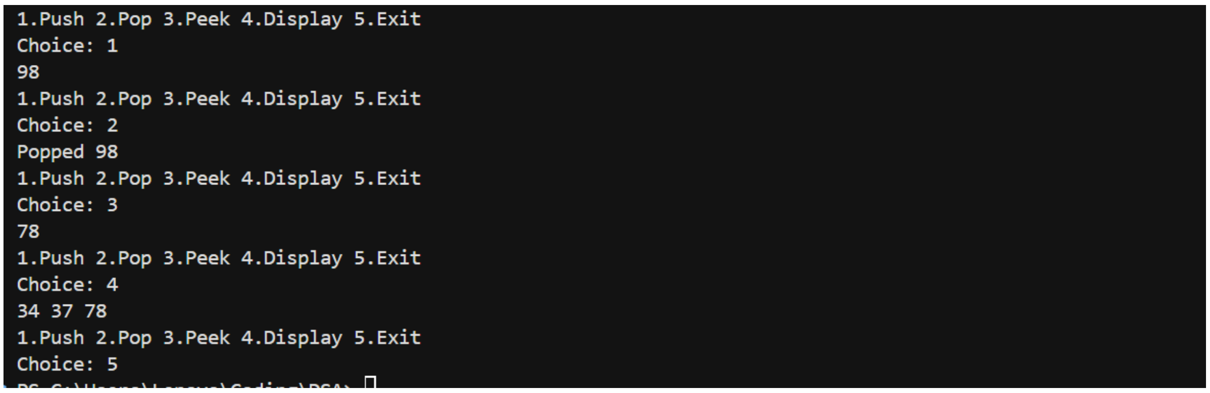
Problem5: Implement a stack using arrays with methods for push, pop, and peek operations.

Input:



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

struct Stack {
    int top; vector<int> a;
    Stack(int cap=100):top(-1),a(cap){ }
    bool push(int x){
        if(top+1 >= (int)a.size()) { cout<<"Overflow\n"; return false; }
        a[++top]=x; return true;
    }
    bool pop(){
        if(top<0){ cout<<"Underflow\n"; return false; }
        cout<<"Popped "<<a[top--]<<"\n"; return true;
    }
    int peek(){ if(top<0) throw runtime_error("Empty"); return a[top]; }
    bool empty(){ return top<0; }
    void display(){ for(int i=0;i<=top;i++) cout<<a[i]<<" "; cout<<"\n"; }
};

int main(){
    Stack s(100);
    while(true){
        cout<<"1.Push 2.Pop 3.Peek 4.Display 5.Exit\nChoice: ";
        int c; cin>>c;
        if(c==1){ int v; cin>>v; s.push(v);}
        else if(c==2) s.pop();
        else if(c==3){ try{ cout<<s.peek()<<"\n"; }catch(...){ cout<<"Empty\n"; } }
        else if(c==4) s.display();
    }
    return 0;
}
```

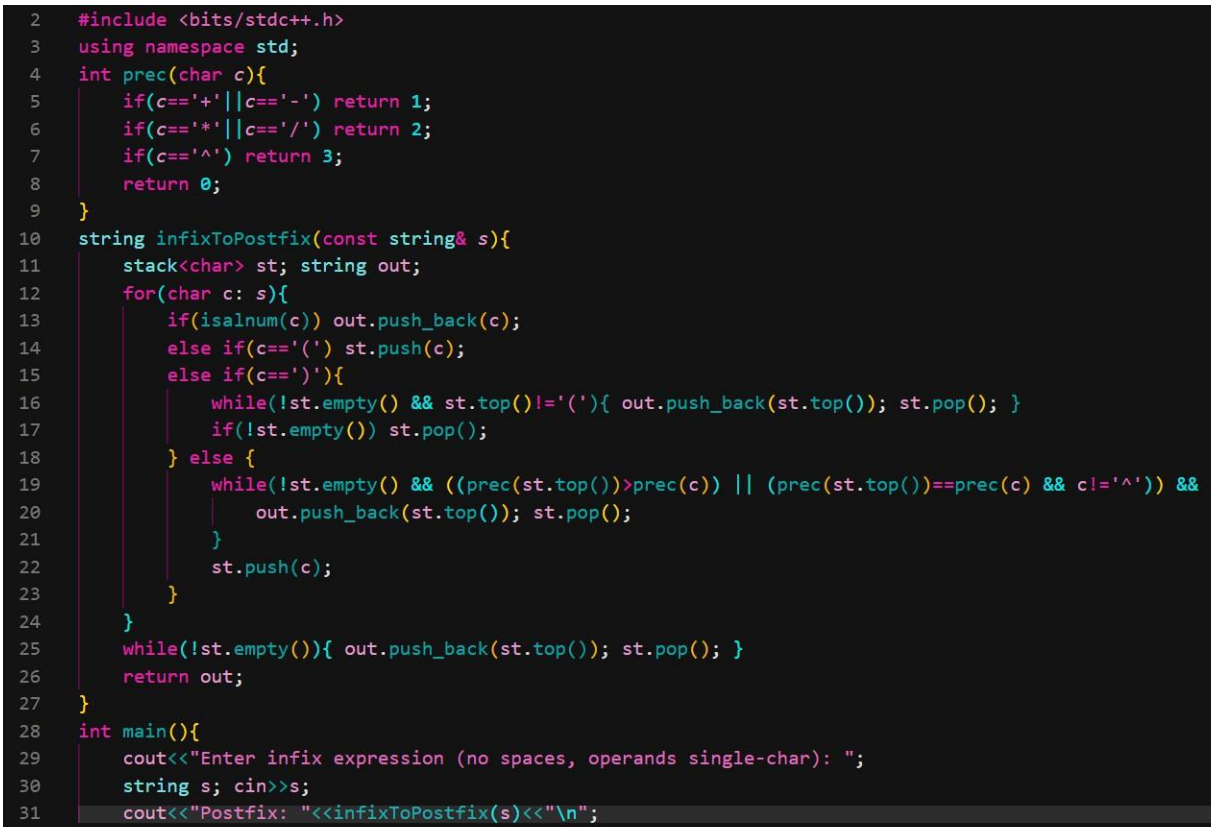
Output:

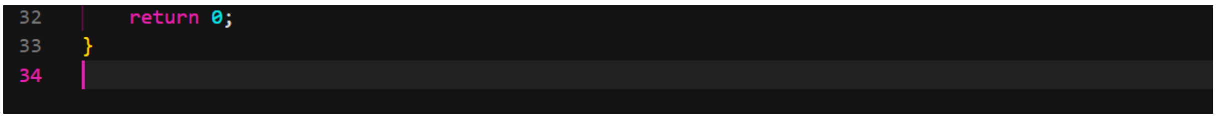


```
1.Push 2.Pop 3.Peek 4.Display 5.Exit
Choice: 1
98
1.Push 2.Pop 3.Peek 4.Display 5.Exit
Choice: 2
Popped 98
1.Push 2.Pop 3.Peek 4.Display 5.Exit
Choice: 3
78
1.Push 2.Pop 3.Peek 4.Display 5.Exit
Choice: 4
34 37 78
1.Push 2.Pop 3.Peek 4.Display 5.Exit
Choice: 5
PS C:\Users\Lenovo\Coding\DSA>
```

Problem6: Write a function to convert an infix expression to a postfix expression using a stack. The function should handle parentheses and operator precedence correctly.

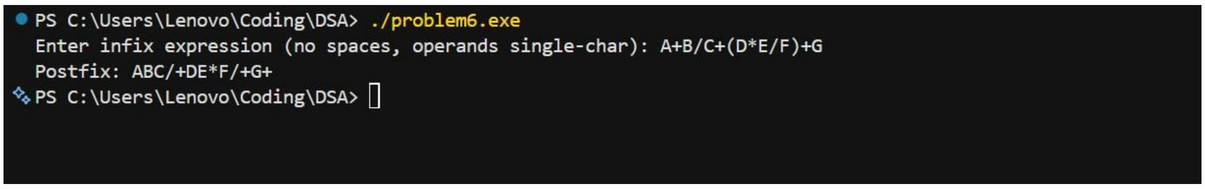
Input:





```cpp
#include <bits/stdc++.h>
using namespace std;
int prec(char c){
    if(c=='+'||c=='-') return 1;
    if(c=='*'||c=='/') return 2;
    if(c=='^') return 3;
    return 0;
}
string infixToPostfix(const string& s){
    stack<char> st; string out;
    for(char c: s){
        if(isalnum(c)) out.push_back(c);
        else if(c=='(') st.push(c);
        else if(c==')'){
            while(!st.empty() && st.top()!='('){ out.push_back(st.top()); st.pop(); }
            if(!st.empty()) st.pop();
        } else {
            while(!st.empty() && ((prec(st.top())>prec(c)) || (prec(st.top())==prec(c) && c!='^')) &&
                out.push_back(st.top()); st.pop();
            }
            st.push(c);
        }
    }
    while(!st.empty()){ out.push_back(st.top()); st.pop(); }
    return out;
}
int main(){
    cout<<"Enter infix expression (no spaces, operands single-char): ";
    string s; cin>>s;
    cout<<"Postfix: "<<infixToPostfix(s)<<"\n";
    return 0;
}
```

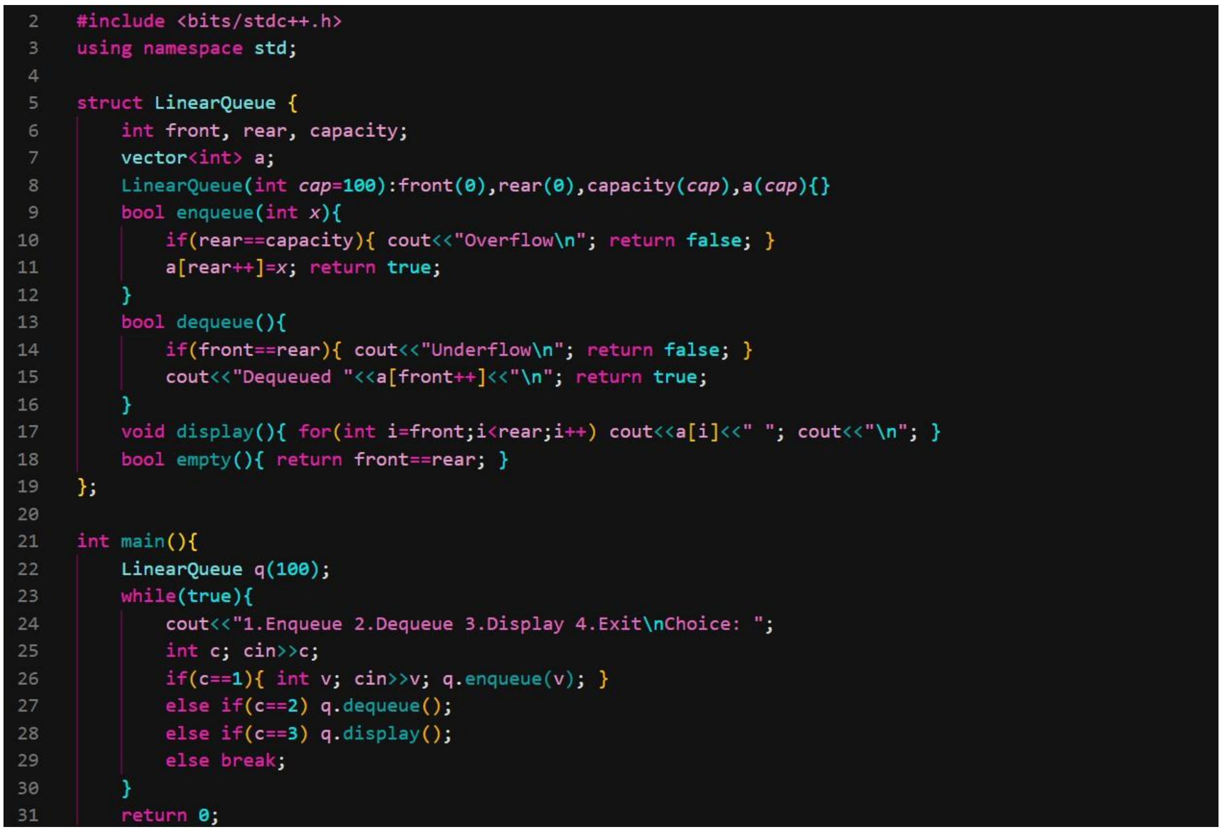
Output:



```
PS C:\Users\Lenovo\Coding\DSA> ./problem6.exe
Enter infix expression (no spaces, operands single-char): A+B/C+(D*E/F)+G
Postfix: ABC/+DE*F/+G+
PS C:\Users\Lenovo\Coding\DSA>
```

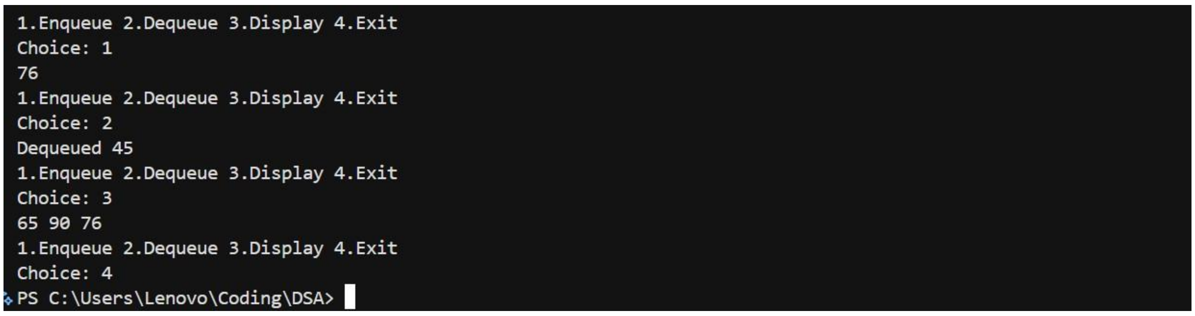
Problem7: Create a linear queue using an array with methods for enqueue, dequeue, and checking if the queue is empty or full.

Input:



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

struct LinearQueue {
    int front, rear, capacity;
    vector<int> a;
    LinearQueue(int cap=100):front(0),rear(0),capacity(cap),a(cap){}
    bool enqueue(int x){
        if(rear==capacity){ cout<<"Overflow\n"; return false; }
        a[rear++]=x; return true;
    }
    bool dequeue(){
        if(front==rear){ cout<<"Underflow\n"; return false; }
        cout<<"Dequeued "<<a[front++]<<"\n"; return true;
    }
    void display(){ for(int i=front;i<rear;i++) cout<<a[i]<<" "; cout<<"\n"; }
    bool empty(){ return front==rear; }
};

int main(){
    LinearQueue q(100);
    while(true){
        cout<<"1.Enqueue 2.Dequeue 3.Display 4.Exit\nChoice: ";
        int c; cin>>c;
        if(c==1){ int v; cin>>v; q.enqueue(v); }
        else if(c==2) q.dequeue();
        else if(c==3) q.display();
        else break;
    }
    return 0;
```

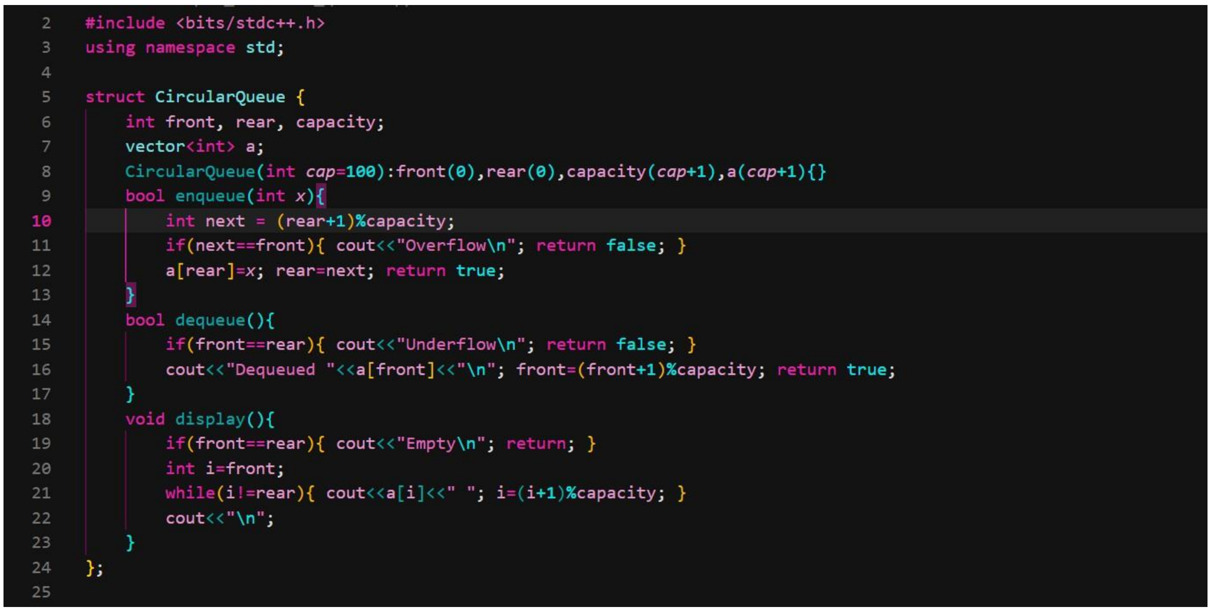
Output:



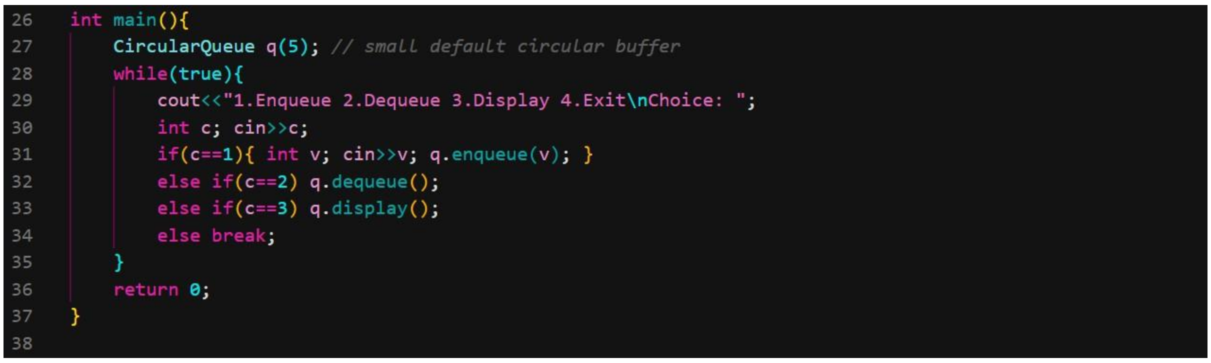
```
1.Enqueue 2.Dequeue 3.Display 4.Exit
Choice: 1
76
1.Enqueue 2.Dequeue 3.Display 4.Exit
Choice: 2
Dequeued 45
1.Enqueue 2.Dequeue 3.Display 4.Exit
Choice: 3
65 90 76
1.Enqueue 2.Dequeue 3.Display 4.Exit
Choice: 4
PS C:\Users\Lenovo\Coding\DSA>
```

Problem8: **Create a circular queue using array with methods for enqueue, dequeue, and checking if the queue is empty or full. Ensure that the circular nature of the queue is maintained after each operation.**

Input:

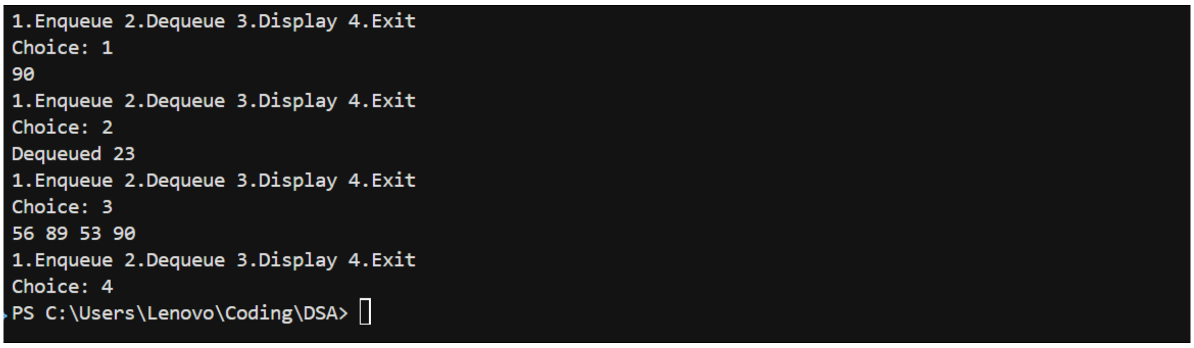


```cpp
#include <bits/stdc++.h>
using namespace std;

struct CircularQueue {
    int front, rear, capacity;
    vector<int> a;
    CircularQueue(int cap=100):front(0),rear(0),capacity(cap+1),a(cap+1){}
    bool enqueue(int x){
        int next = (rear+1)%capacity;
        if(next==front){ cout<<"Overflow\n"; return false; }
        a[rear]=x; rear=next; return true;
    }
    bool dequeue(){
        if(front==rear){ cout<<"Underflow\n"; return false; }
        cout<<"Dequeued "<<a[front]<<"\n"; front=(front+1)%capacity; return true;
    }
    void display(){
        if(front==rear){ cout<<"Empty\n"; return; }
        int i=front;
        while(i!=rear){ cout<<a[i]<<" "; i=(i+1)%capacity; }
        cout<<"\n";
    }
};

```



```cpp
int main(){
    CircularQueue q(5); // small default circular buffer
    while(true){
        cout<<"1.Enqueue 2.Dequeue 3.Display 4.Exit\nChoice: ";
        int c; cin>>c;
        if(c==1){ int v; cin>>v; q.enqueue(v); }
        else if(c==2) q.dequeue();
        else if(c==3) q.display();
        else break;
    }
    return 0;
}

```

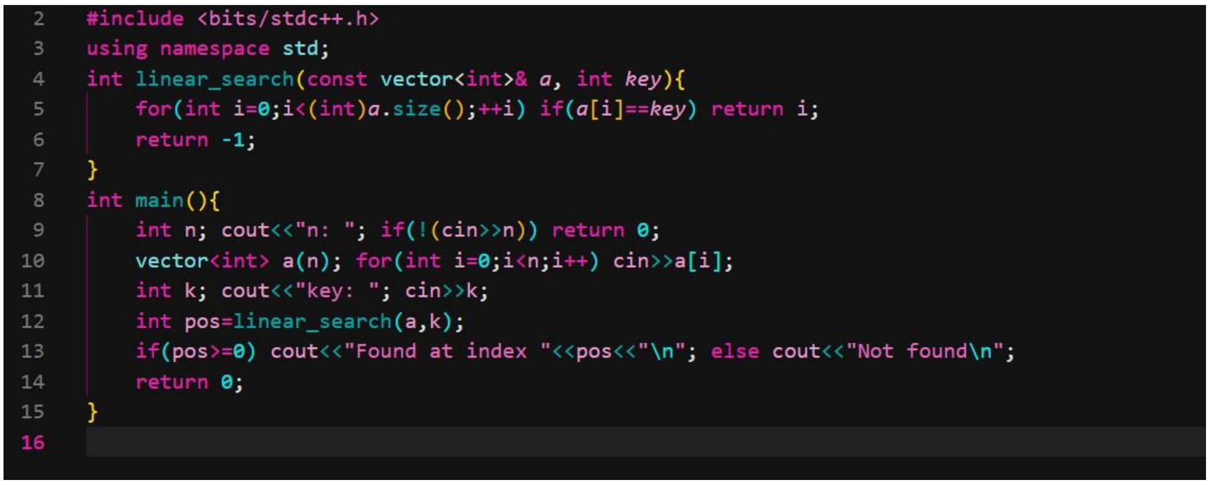
Output:



```
1.Enqueue 2.Dequeue 3.Display 4.Exit
Choice: 1
90
1.Enqueue 2.Dequeue 3.Display 4.Exit
Choice: 2
Dequeued 23
1.Enqueue 2.Dequeue 3.Display 4.Exit
Choice: 3
56 89 53 90
1.Enqueue 2.Dequeue 3.Display 4.Exit
Choice: 4
PS C:\Users\Lenovo\Coding\DSA> 
```

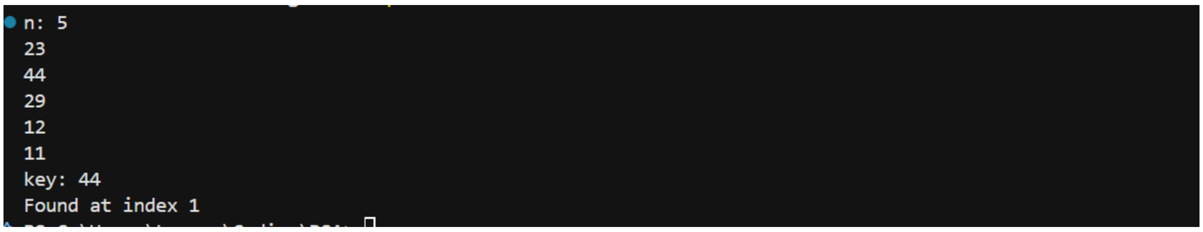
Problem9: Implement linear search

Input:



```cpp
#include <bits/stdc++.h>
using namespace std;
int linear_search(const vector<int>& a, int key){
    for(int i=0;i<(int)a.size();++i) if(a[i]==key) return i;
    return -1;
}
int main(){
    int n; cout<<"n: "; if(!(cin>>n)) return 0;
    vector<int> a(n); for(int i=0;i<n;i++) cin>>a[i];
    int k; cout<<"key: "; cin>>k;
    int pos=linear_search(a,k);
    if(pos>=0) cout<<"Found at index "<<pos<<"\n"; else cout<<"Not found\n";
    return 0;
}
```

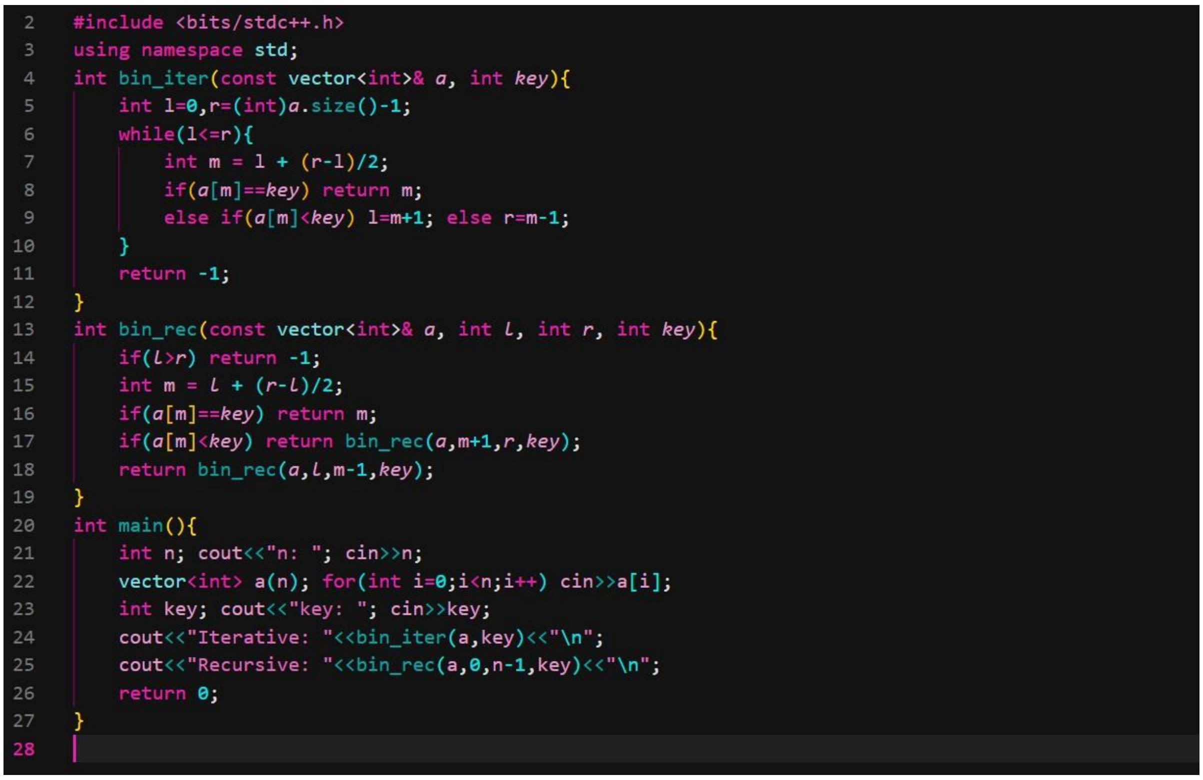
Output:



```
n: 5
23
44
29
12
11
key: 44
Found at index 1
```

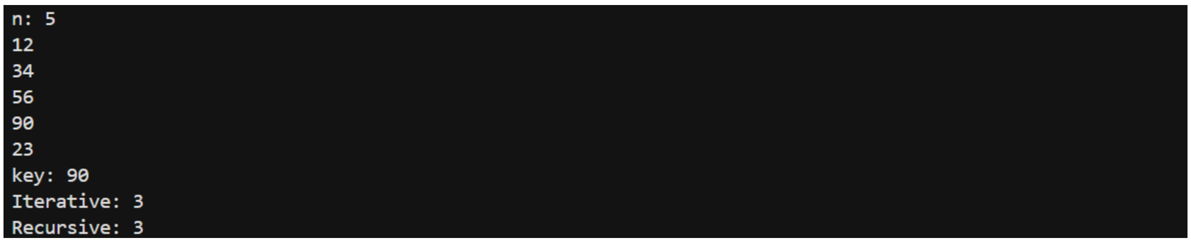
<u>Problem10:</u> Implement binary search (iterative and recursive)

Input:



```cpp
#include <bits/stdc++.h>
using namespace std;
int bin_iter(const vector<int>& a, int key){
    int l=0,r=(int)a.size()-1;
    while(l<=r){
        int m = l + (r-l)/2;
        if(a[m]==key) return m;
        else if(a[m]<key) l=m+1; else r=m-1;
    }
    return -1;
}
int bin_rec(const vector<int>& a, int l, int r, int key){
    if(l>r) return -1;
    int m = l + (r-l)/2;
    if(a[m]==key) return m;
    if(a[m]<key) return bin_rec(a,m+1,r,key);
    return bin_rec(a,l,m-1,key);
}
int main(){
    int n; cout<<"n: "; cin>>n;
    vector<int> a(n); for(int i=0;i<n;i++) cin>>a[i];
    int key; cout<<"key: "; cin>>key;
    cout<<"Iterative: "<<bin_iter(a,key)<<"\n";
    cout<<"Recursive: "<<bin_rec(a,0,n-1,key)<<"\n";
    return 0;
}
```

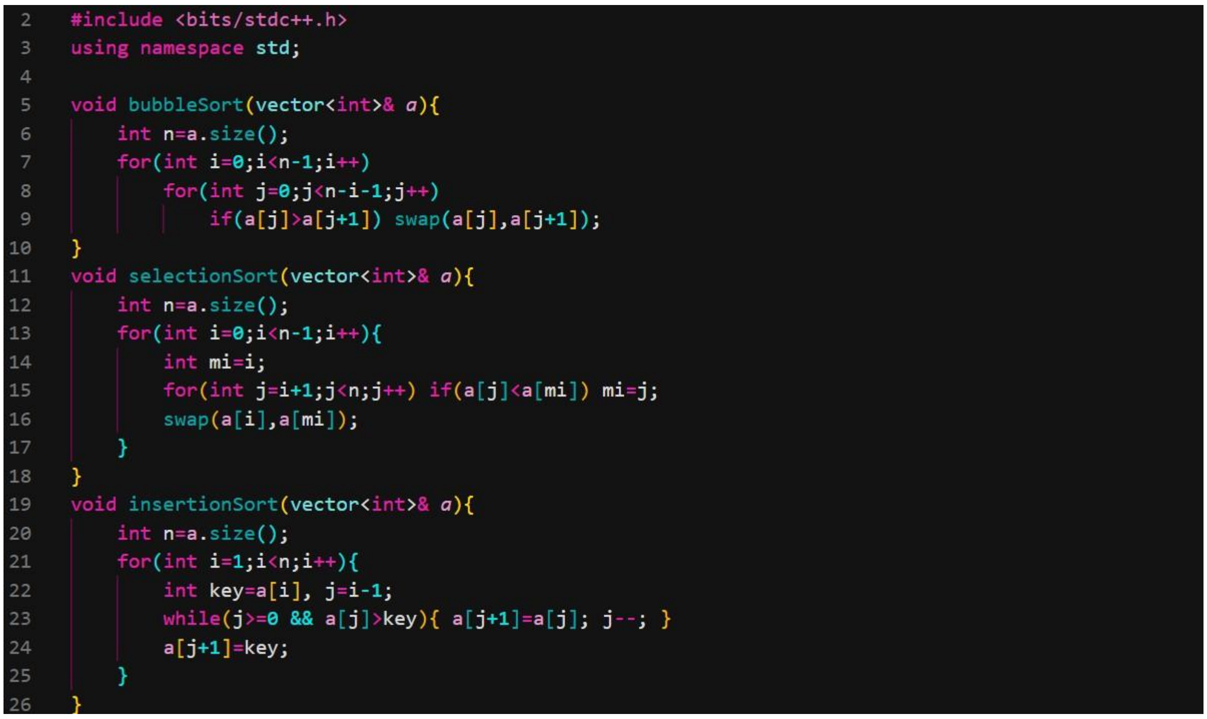
Output:



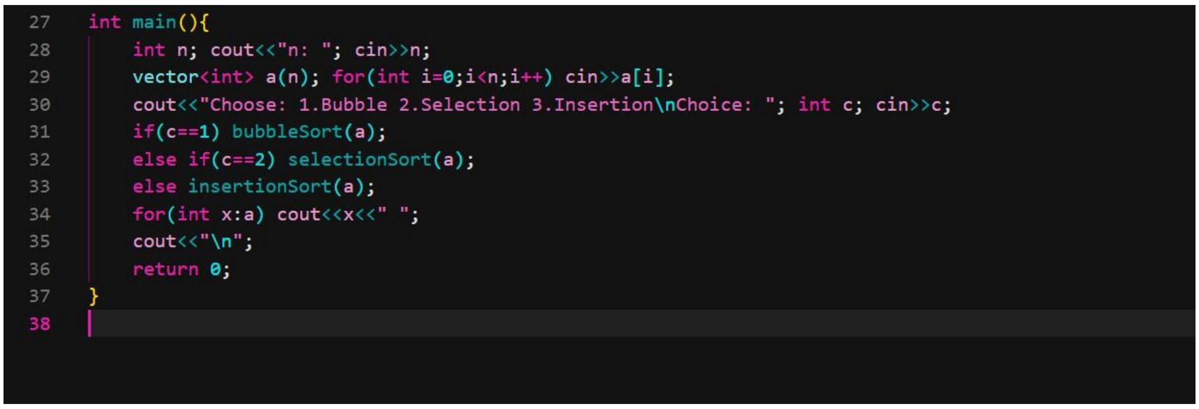
```
n: 5
12
34
56
90
23
key: 90
Iterative: 3
Recursive: 3
```

Problem11: Implement various sorting algorithms including Insertion sort, selection sort, bubble sort, and analyse their performance on different input sizes.

Input:

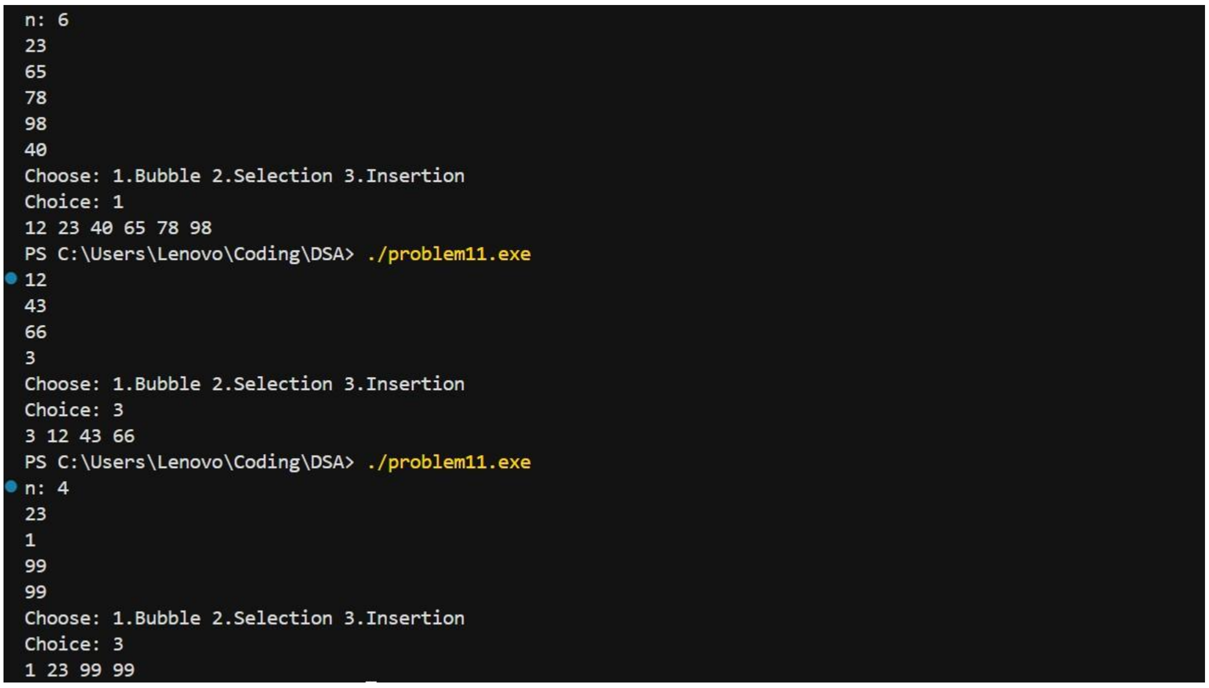


```cpp
#include <bits/stdc++.h>
using namespace std;

void bubbleSort(vector<int>& a){
    int n=a.size();
    for(int i=0;i<n-1;i++)
        for(int j=0;j<n-i-1;j++)
            if(a[j]>a[j+1]) swap(a[j],a[j+1]);
}
void selectionSort(vector<int>& a){
    int n=a.size();
    for(int i=0;i<n-1;i++){
        int mi=i;
        for(int j=i+1;j<n;j++) if(a[j]<a[mi]) mi=j;
        swap(a[i],a[mi]);
    }
}
void insertionSort(vector<int>& a){
    int n=a.size();
    for(int i=1;i<n;i++){
        int key=a[i], j=i-1;
        while(j>=0 && a[j]>key){ a[j+1]=a[j]; j--; }
        a[j+1]=key;
    }
}
```



```cpp
int main(){
    int n; cout<<"n: "; cin>>n;
    vector<int> a(n); for(int i=0;i<n;i++) cin>>a[i];
    cout<<"Choose: 1.Bubble 2.Selection 3.Insertion\nChoice: "; int c; cin>>c;
    if(c==1) bubbleSort(a);
    else if(c==2) selectionSort(a);
    else insertionSort(a);
    for(int x:a) cout<<x<<" ";
    cout<<"\n";
    return 0;
}
```

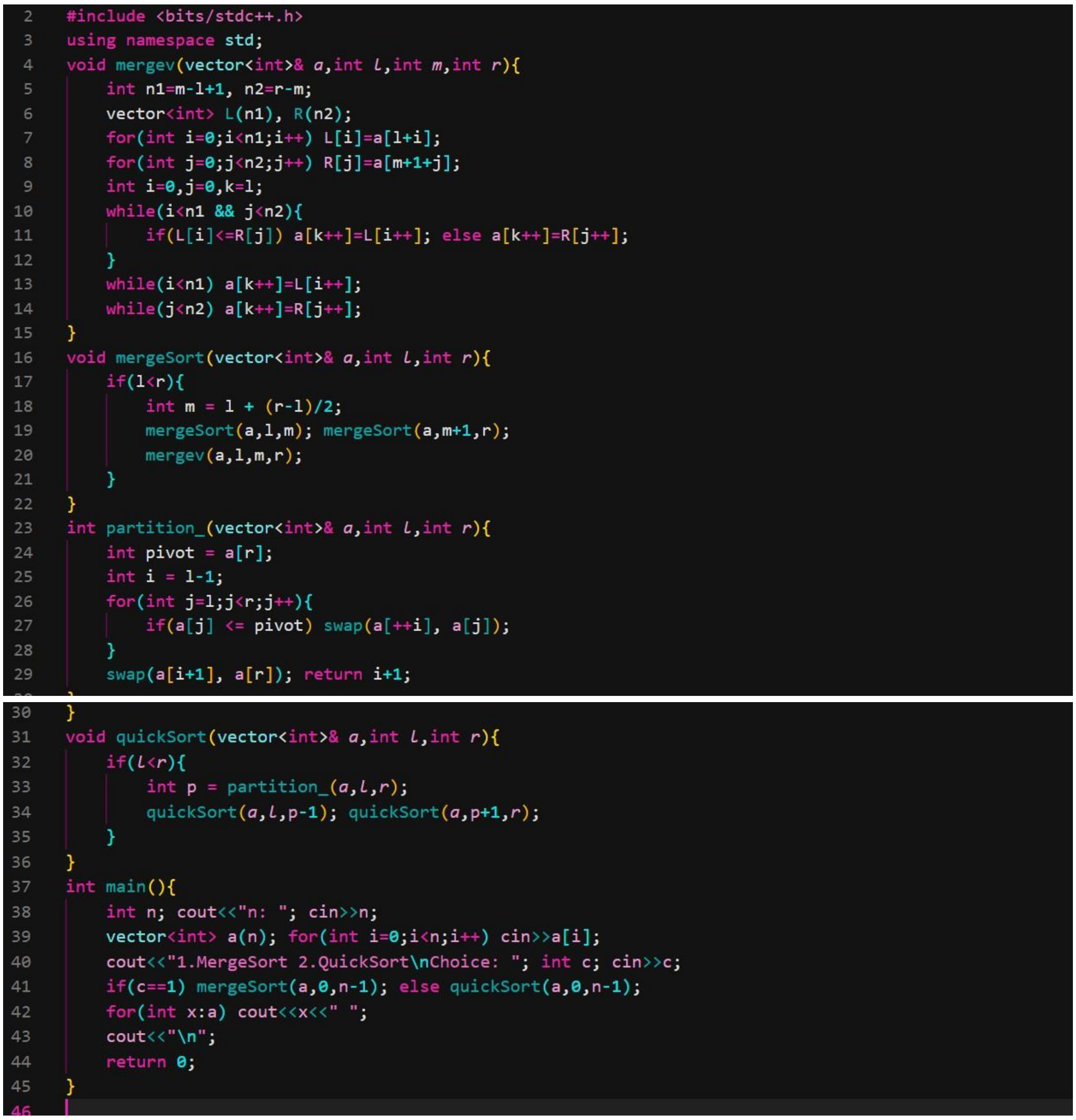
Output:



```
n: 6
23
65
78
98
40
Choose: 1.Bubble 2.Selection 3.Insertion
Choice: 1
12 23 40 65 78 98
PS C:\Users\Lenovo\Coding\DSA> ./problem11.exe
12
43
66
3
Choose: 1.Bubble 2.Selection 3.Insertion
Choice: 3
3 12 43 66
PS C:\Users\Lenovo\Coding\DSA> ./problem11.exe
n: 4
23
1
99
99
Choose: 1.Bubble 2.Selection 3.Insertion
Choice: 3
1 23 99 99
```

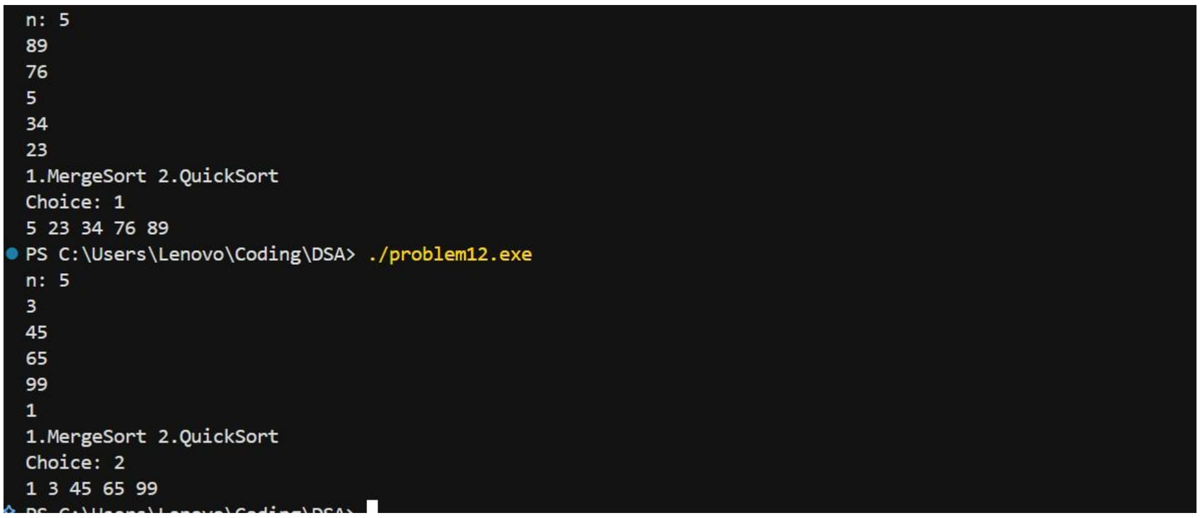
<u>Problem12:</u> Implement various sorting algorithms including Quick Sort, Merge Sort, Heap Sort, and analyse their performance on different input sizes. Ensure the implementation handles edge cases such as duplicate values and nearly sorted arrays.

Input:



```cpp
#include <bits/stdc++.h>
using namespace std;
void mergev(vector<int>& a,int l,int m,int r){
    int n1=m-l+1, n2=r-m;
    vector<int> L(n1), R(n2);
    for(int i=0;i<n1;i++) L[i]=a[l+i];
    for(int j=0;j<n2;j++) R[j]=a[m+1+j];
    int i=0,j=0,k=l;
    while(i<n1 && j<n2){
        if(L[i]<=R[j]) a[k++]=L[i++]; else a[k++]=R[j++];
    }
    while(i<n1) a[k++]=L[i++];
    while(j<n2) a[k++]=R[j++];
}
void mergeSort(vector<int>& a,int l,int r){
    if(l<r){
        int m = l + (r-l)/2;
        mergeSort(a,l,m); mergeSort(a,m+1,r);
        mergev(a,l,m,r);
    }
}
int partition_(vector<int>& a,int l,int r){
    int pivot = a[r];
    int i = l-1;
    for(int j=l;j<r;j++){
        if(a[j] <= pivot) swap(a[++i], a[j]);
    }
    swap(a[i+1], a[r]); return i+1;
}
void quickSort(vector<int>& a,int l,int r){
    if(l<r){
        int p = partition_(a,l,r);
        quickSort(a,l,p-1); quickSort(a,p+1,r);
    }
}
int main(){
    int n; cout<<"n: "; cin>>n;
    vector<int> a(n); for(int i=0;i<n;i++) cin>>a[i];
    cout<<"1.MergeSort 2.QuickSort\nChoice: "; int c; cin>>c;
    if(c==1) mergeSort(a,0,n-1); else quickSort(a,0,n-1);
    for(int x:a) cout<<x<<" ";
    cout<<"\n";
    return 0;
}
```

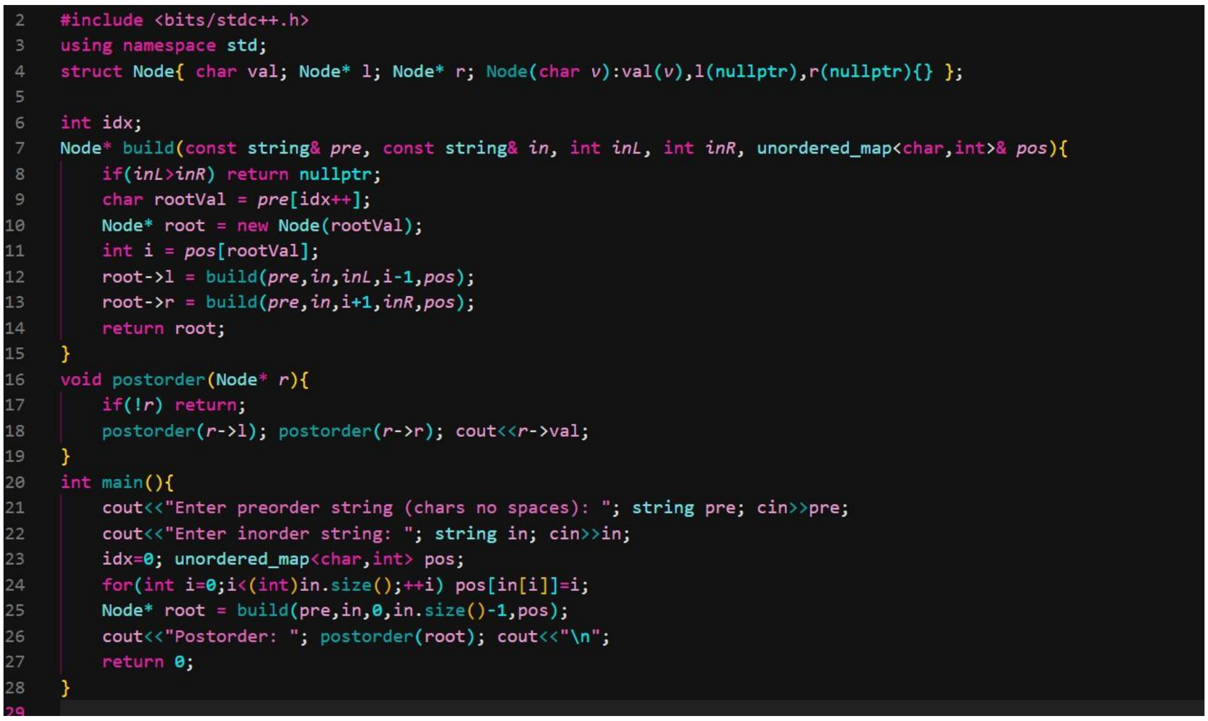
Output:



```
n: 5
89
76
5
34
23
1.MergeSort 2.QuickSort
Choice: 1
5 23 34 76 89
PS C:\Users\Lenovo\Coding\DSA> ./problem12.exe
n: 5
3
45
65
99
1
1.MergeSort 2.QuickSort
Choice: 2
1 3 45 65 99
PS C:\Users\Lenovo\Coding\DSA>
```

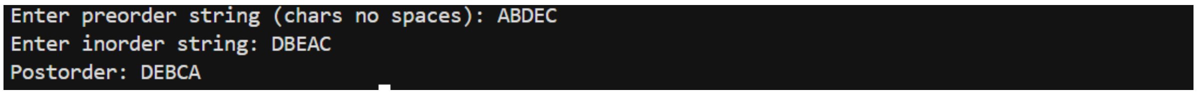
<u>Problem13:</u> Given preorder and in-order traversal of a tree, construct the binary tree.

Input:



```cpp
#include <bits/stdc++.h>
using namespace std;
struct Node{ char val; Node* l; Node* r; Node(char v):val(v),l(nullptr),r(nullptr){} };

int idx;
Node* build(const string& pre, const string& in, int inL, int inR, unordered_map<char,int>& pos){
    if(inL>inR) return nullptr;
    char rootVal = pre[idx++];
    Node* root = new Node(rootVal);
    int i = pos[rootVal];
    root->l = build(pre,in,inL,i-1,pos);
    root->r = build(pre,in,i+1,inR,pos);
    return root;
}
void postorder(Node* r){
    if(!r) return;
    postorder(r->l); postorder(r->r); cout<<r->val;
}
int main(){
    cout<<"Enter preorder string (chars no spaces): "; string pre; cin>>pre;
    cout<<"Enter inorder string: "; string in; cin>>in;
    idx=0; unordered_map<char,int> pos;
    for(int i=0;i<(int)in.size();++i) pos[in[i]]=i;
    Node* root = build(pre,in,0,in.size()-1,pos);
    cout<<"Postorder: "; postorder(root); cout<<"\n";
    return 0;
}
```

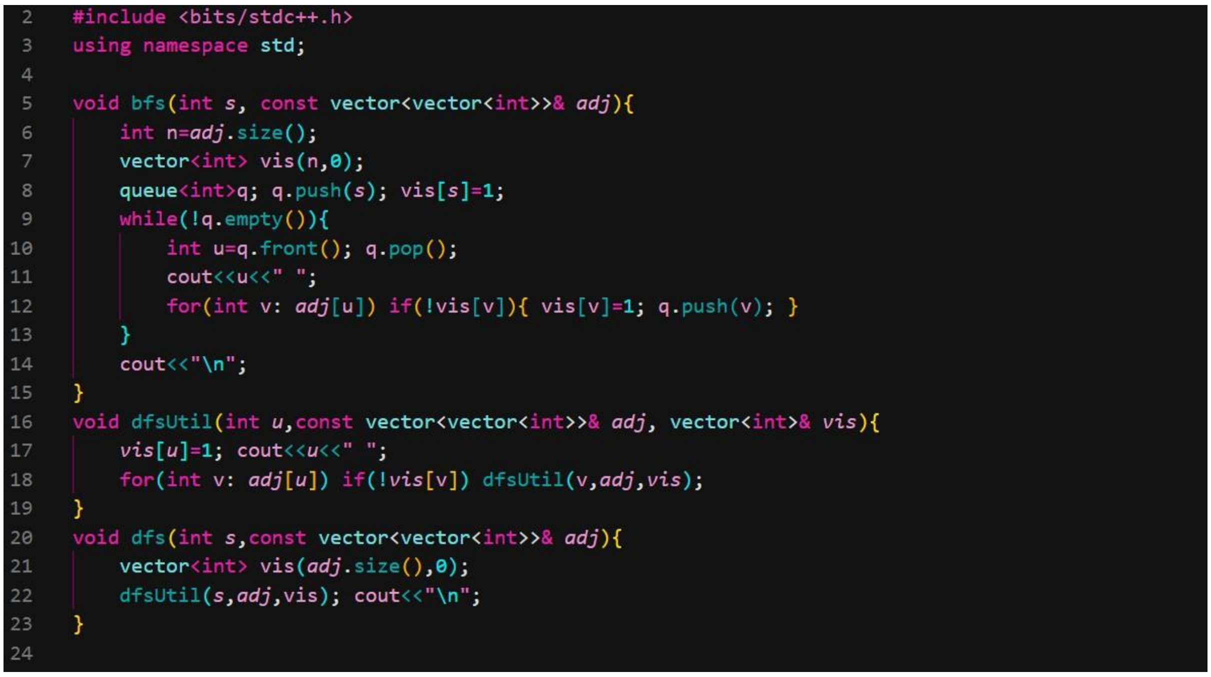
Output:



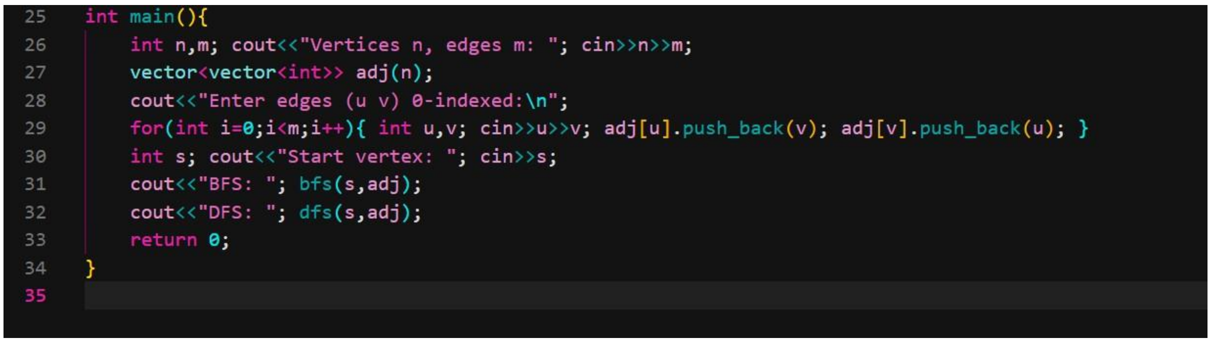
```
Enter preorder string (chars no spaces): ABDEC
Enter inorder string: DBEAC
Postorder: DEBCA
```

Problem14: Perform the traversal of graph(DFS,BFS)

Input:

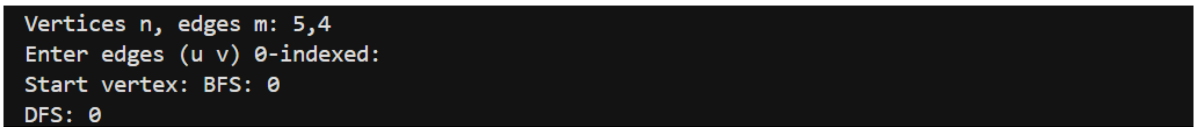


```cpp
#include <bits/stdc++.h>
using namespace std;

void bfs(int s, const vector<vector<int>>& adj){
    int n=adj.size();
    vector<int> vis(n,0);
    queue<int>q; q.push(s); vis[s]=1;
    while(!q.empty()){
        int u=q.front(); q.pop();
        cout<<u<<" ";
        for(int v: adj[u]) if(!vis[v]){ vis[v]=1; q.push(v); }
    }
    cout<<"\n";
}
void dfsUtil(int u,const vector<vector<int>>& adj, vector<int>& vis){
    vis[u]=1; cout<<u<<" ";
    for(int v: adj[u]) if(!vis[v]) dfsUtil(v,adj,vis);
}
void dfs(int s,const vector<vector<int>>& adj){
    vector<int> vis(adj.size(),0);
    dfsUtil(s,adj,vis); cout<<"\n";
}

```



```cpp
int main(){
    int n,m; cout<<"Vertices n, edges m: "; cin>>n>>m;
    vector<vector<int>> adj(n);
    cout<<"Enter edges (u v) 0-indexed:\n";
    for(int i=0;i<m;i++){ int u,v; cin>>u>>v; adj[u].push_back(v); adj[v].push_back(u); }
    int s; cout<<"Start vertex: "; cin>>s;
    cout<<"BFS: "; bfs(s,adj);
    cout<<"DFS: "; dfs(s,adj);
    return 0;
}

```

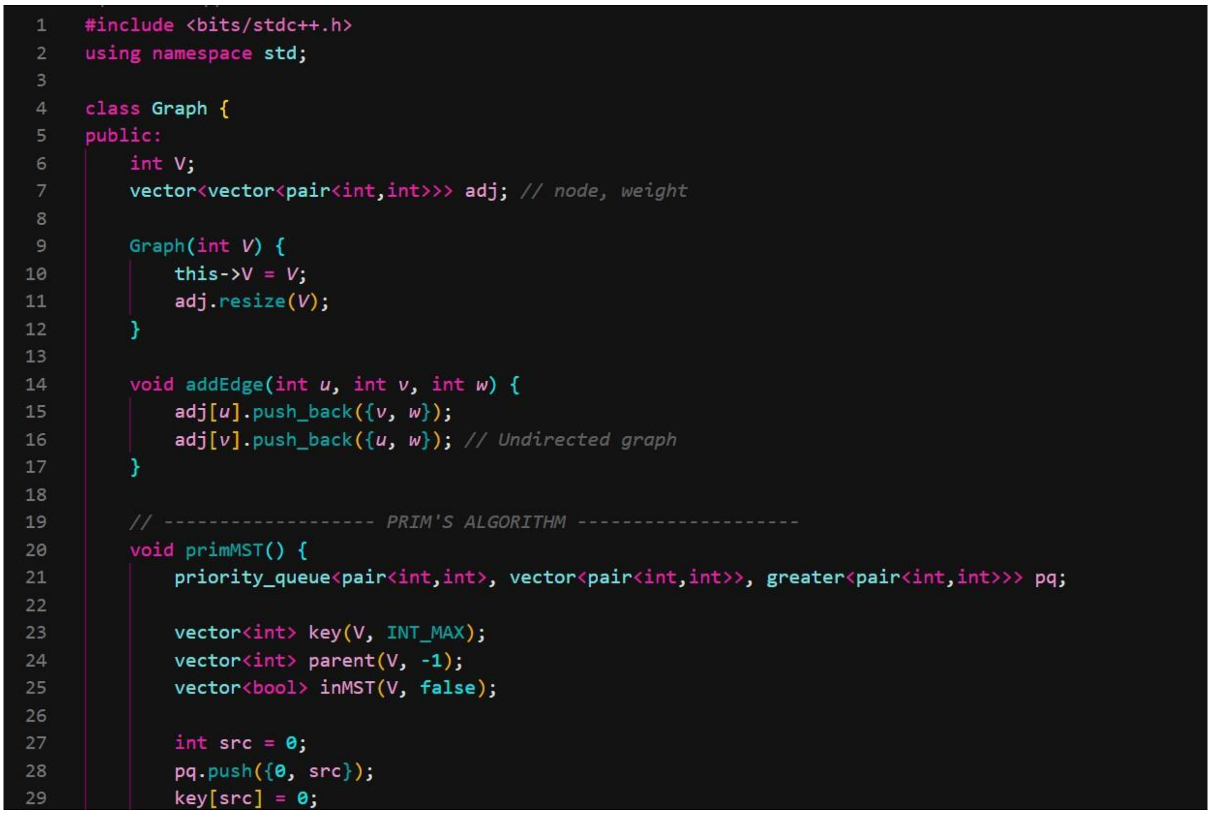
Output:



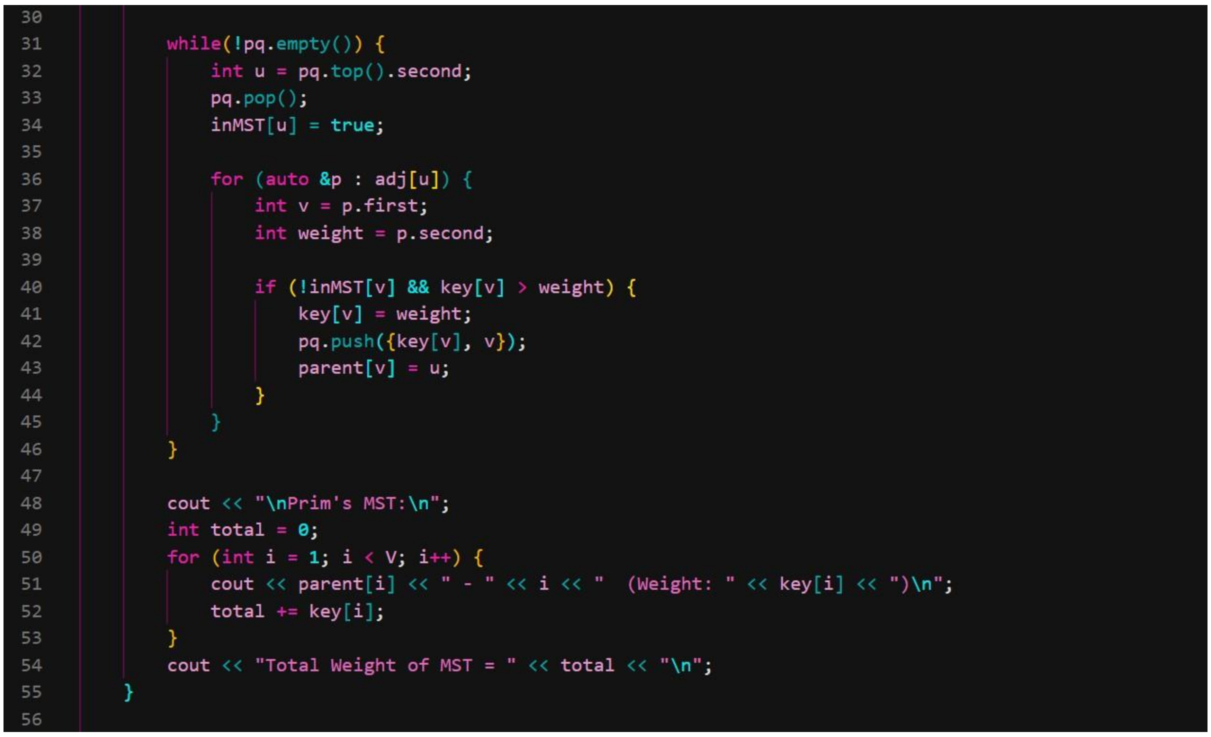
```
Vertices n, edges m: 5,4
Enter edges (u v) 0-indexed:
Start vertex: BFS: 0
DFS: 0
```

<u>Problem15:</u> Implement Prim's and Kruskal's algorithm to find the minimum spanning tree of a graph.

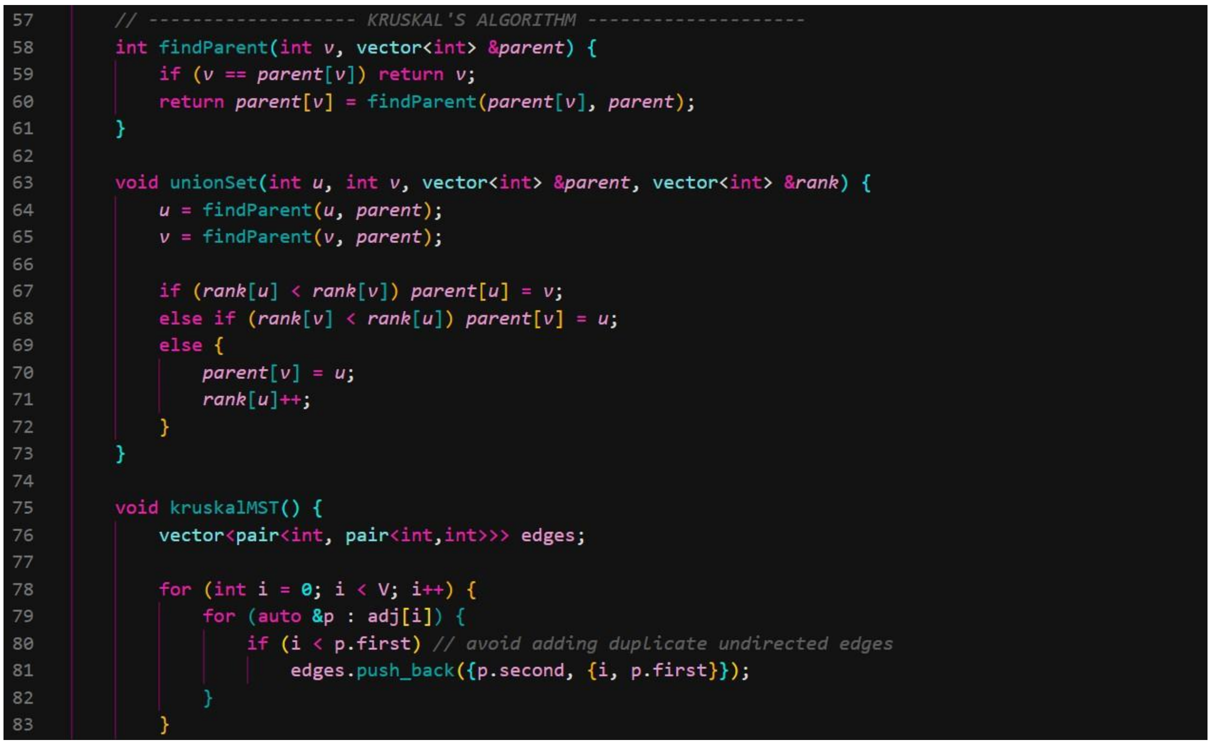
Input:



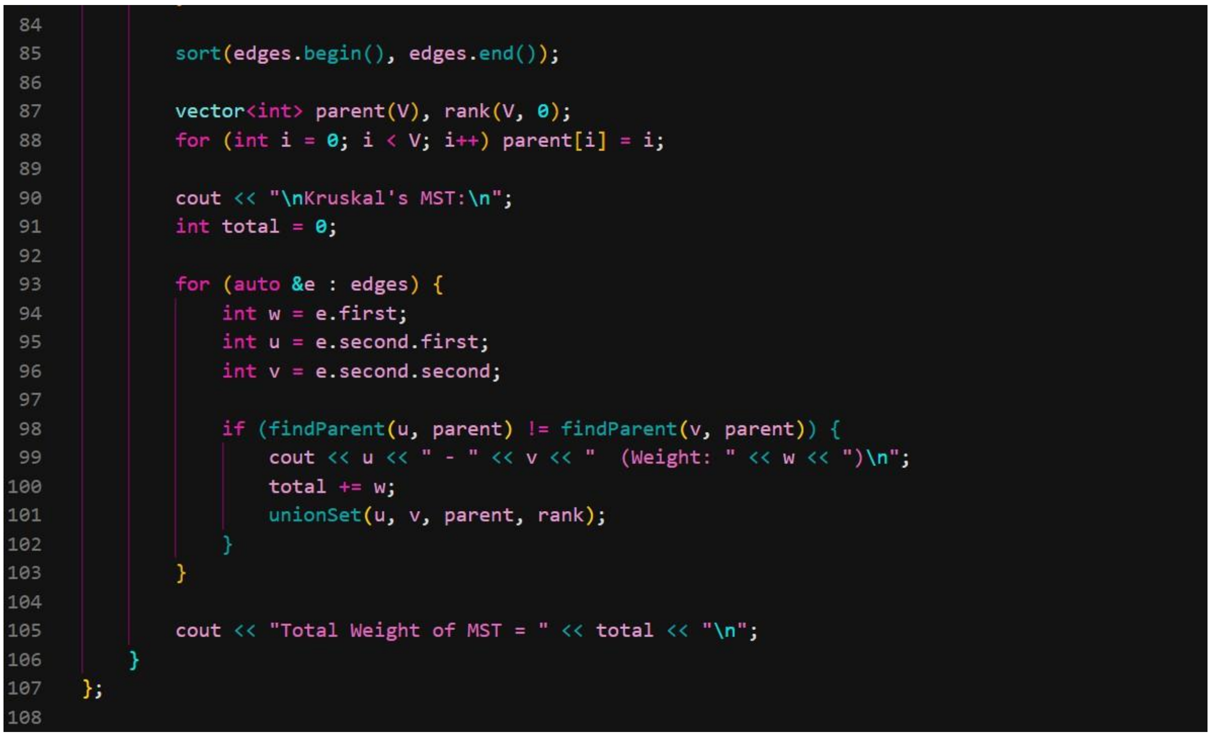
```cpp
#include <bits/stdc++.h>
using namespace std;

class Graph {
public:
    int V;
    vector<vector<pair<int,int>>> adj; // node, weight

    Graph(int V) {
        this->V = V;
        adj.resize(V);
    }

    void addEdge(int u, int v, int w) {
        adj[u].push_back({v, w});
        adj[v].push_back({u, w}); // Undirected graph
    }

    // ------------------- PRIM'S ALGORITHM --------------------
    void primMST() {
        priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;

        vector<int> key(V, INT_MAX);
        vector<int> parent(V, -1);
        vector<bool> inMST(V, false);

        int src = 0;
        pq.push({0, src});
        key[src] = 0;
```



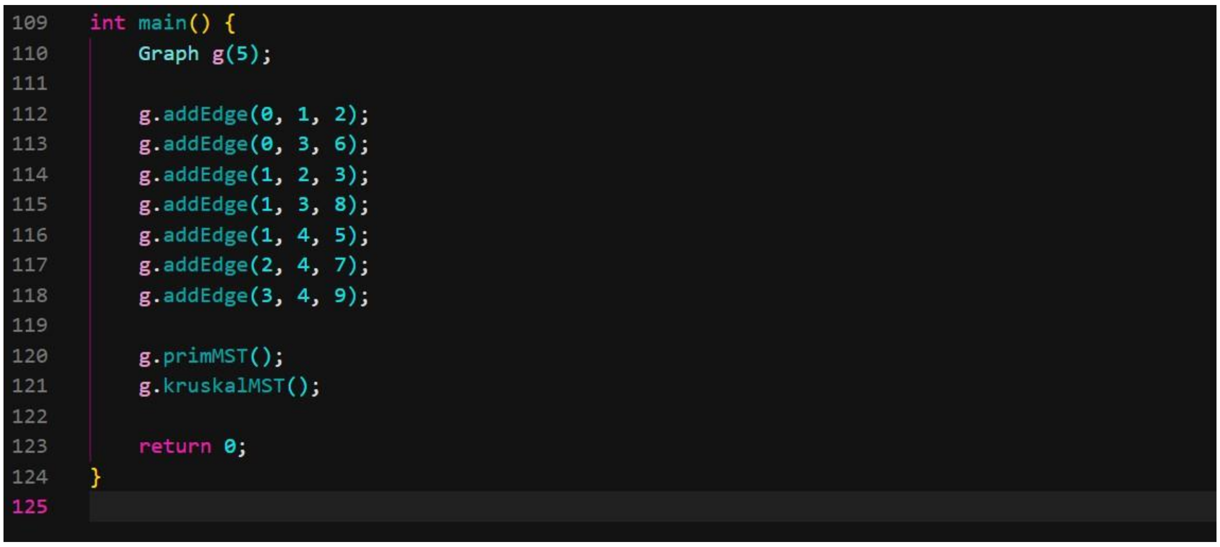
```cpp

        while(!pq.empty()) {
            int u = pq.top().second;
            pq.pop();
            inMST[u] = true;

            for (auto &p : adj[u]) {
                int v = p.first;
                int weight = p.second;

                if (!inMST[v] && key[v] > weight) {
                    key[v] = weight;
                    pq.push({key[v], v});
                    parent[v] = u;
                }
            }
        }

        cout << "\nPrim's MST:\n";
        int total = 0;
        for (int i = 1; i < V; i++) {
            cout << parent[i] << " - " << i << "  (Weight: " << key[i] << ")\n";
            total += key[i];
        }
        cout << "Total Weight of MST = " << total << "\n";
    }

```



```cpp
    // ------------------- KRUSKAL'S ALGORITHM --------------------
    int findParent(int v, vector<int> &parent) {
        if (v == parent[v]) return v;
        return parent[v] = findParent(parent[v], parent);
    }

    void unionSet(int u, int v, vector<int> &parent, vector<int> &rank) {
        u = findParent(u, parent);
        v = findParent(v, parent);

        if (rank[u] < rank[v]) parent[u] = v;
        else if (rank[v] < rank[u]) parent[v] = u;
        else {
            parent[v] = u;
            rank[u]++;
        }
    }

    void kruskalMST() {
        vector<pair<int, pair<int,int>>> edges;

        for (int i = 0; i < V; i++) {
            for (auto &p : adj[i]) {
                if (i < p.first) // avoid adding duplicate undirected edges
                    edges.push_back({p.second, {i, p.first}});
            }
        }
```

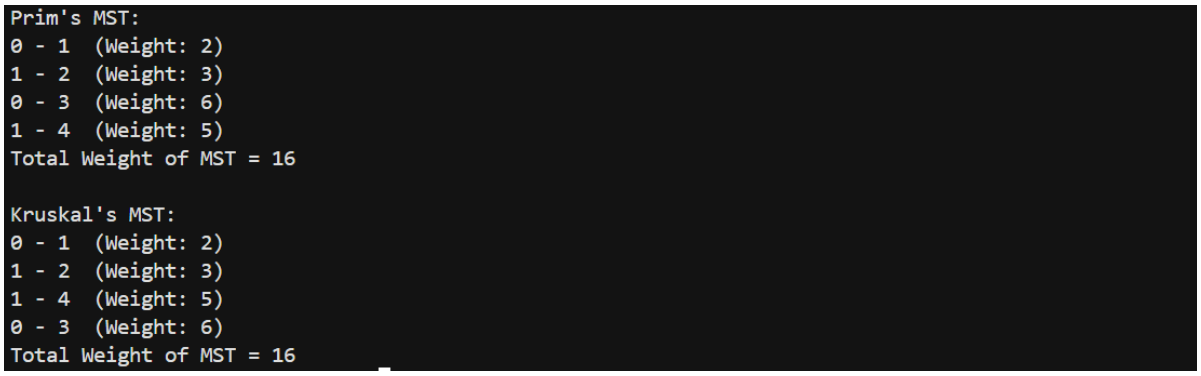


```cpp

        sort(edges.begin(), edges.end());

        vector<int> parent(V), rank(V, 0);
        for (int i = 0; i < V; i++) parent[i] = i;

        cout << "\nKruskal's MST:\n";
        int total = 0;

        for (auto &e : edges) {
            int w = e.first;
            int u = e.second.first;
            int v = e.second.second;

            if (findParent(u, parent) != findParent(v, parent)) {
                cout << u << " - " << v << "  (Weight: " << w << ")\n";
                total += w;
                unionSet(u, v, parent, rank);
            }
        }

        cout << "Total Weight of MST = " << total << "\n";
    }
};
```



```
int main() {
    Graph g(5);

    g.addEdge(0, 1, 2);
    g.addEdge(0, 3, 6);
    g.addEdge(1, 2, 3);
    g.addEdge(1, 3, 8);
    g.addEdge(1, 4, 5);
    g.addEdge(2, 4, 7);
    g.addEdge(3, 4, 9);

    g.primMST();
    g.kruskalMST();

    return 0;
}
```

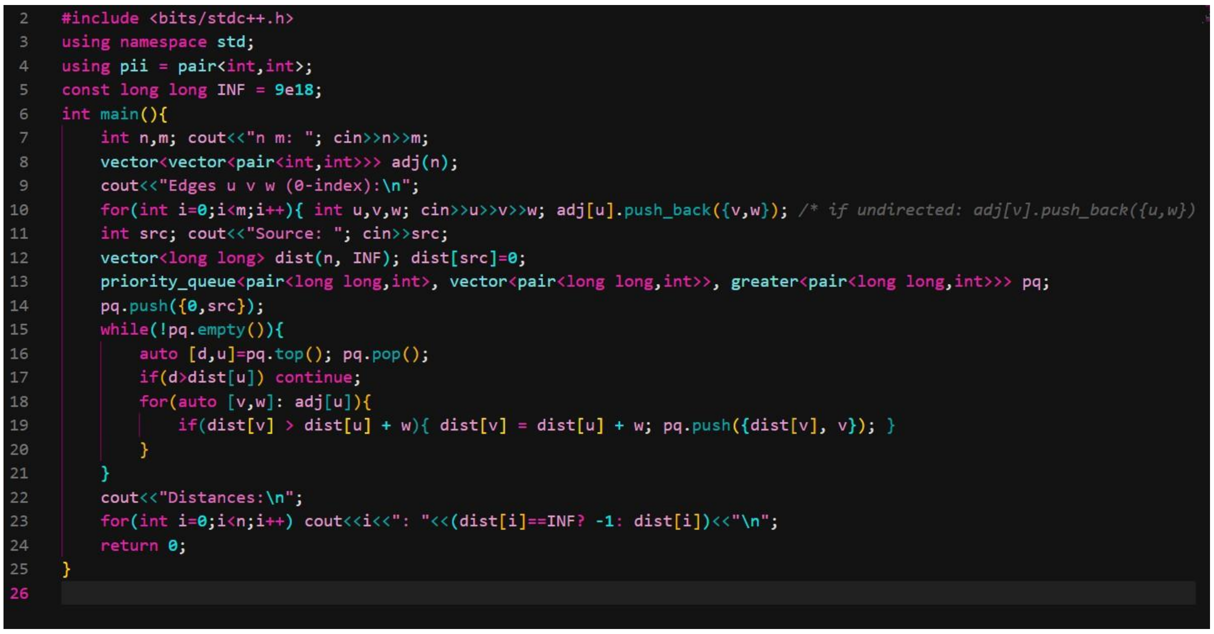
Output:



```
Prim's MST:
0 - 1  (Weight: 2)
1 - 2  (Weight: 3)
0 - 3  (Weight: 6)
1 - 4  (Weight: 5)
Total Weight of MST = 16

Kruskal's MST:
0 - 1  (Weight: 2)
1 - 2  (Weight: 3)
1 - 4  (Weight: 5)
0 - 3  (Weight: 6)
Total Weight of MST = 16
```

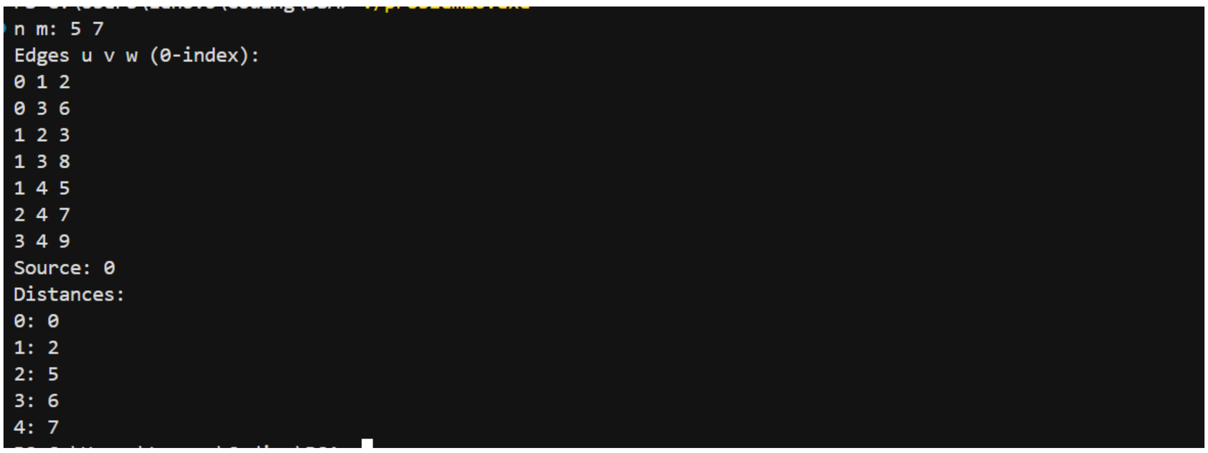
<u>Problem16:</u> Implement Dijkstra’s algorithm to find the shortest path from a source vertex to all other vertices in a weighted graph. Use both adjacency matrix and adjacency list representations for the graph. Ensure the algorithm handles negative weights appropriately.

Input:



```cpp
#include <bits/stdc++.h>
using namespace std;
using pii = pair<int,int>;
const long long INF = 9e18;
int main(){
    int n,m; cout<<"n m: "; cin>>n>>m;
    vector<vector<pair<int,int>>> adj(n);
    cout<<"Edges u v w (0-index):\n";
    for(int i=0;i<m;i++){ int u,v,w; cin>>u>>v>>w; adj[u].push_back({v,w}); /* if undirected: adj[v].push_back({u,w})
    int src; cout<<"Source: "; cin>>src;
    vector<long long> dist(n, INF); dist[src]=0;
    priority_queue<pair<long long,int>, vector<pair<long long,int>>, greater<pair<long long,int>>> pq;
    pq.push({0,src});
    while(!pq.empty()){
        auto [d,u]=pq.top(); pq.pop();
        if(d>dist[u]) continue;
        for(auto [v,w]: adj[u]){
            if(dist[v] > dist[u] + w){ dist[v] = dist[u] + w; pq.push({dist[v], v}); }
        }
    }
    cout<<"Distances:\n";
    for(int i=0;i<n;i++) cout<<i<<": "<<(dist[i]==INF? -1: dist[i])<<"\n";
    return 0;
}
```

Output:



```
n m: 5 7
Edges u v w (0-index):
0 1 2
0 3 6
1 2 3
1 3 8
1 4 5
2 4 7
3 4 9
Source: 0
Distances:
0: 0
1: 2
2: 5
3: 6
4: 7
```