DSC – Praticals

DATA STRUCTURES

Submitted By –

Gaurav

Roll no -24570019

Course – BSc(H) Computer Science

Sem – 3rd

Submitted to –

Mr. Sahil Pathak

(Assistant Professor)

Department of Computer Science

Ramanujan College, DU

1 - Write a program to implement singly linked list as an ADT that supports the following oper ations:

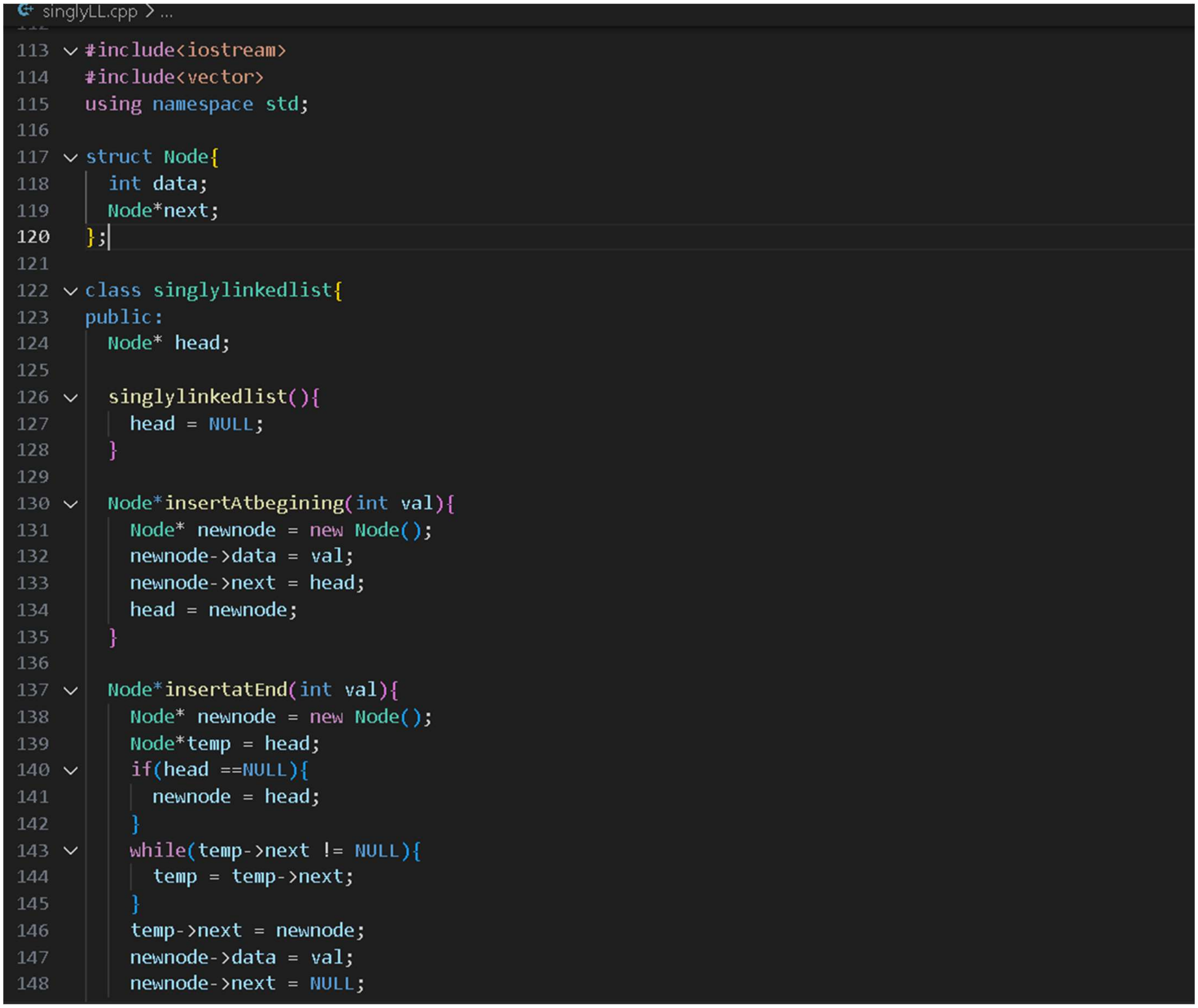
i. Insert an element x at the beginning of the singly linked list

ii. Insert an element x at i th position in the singly linked list

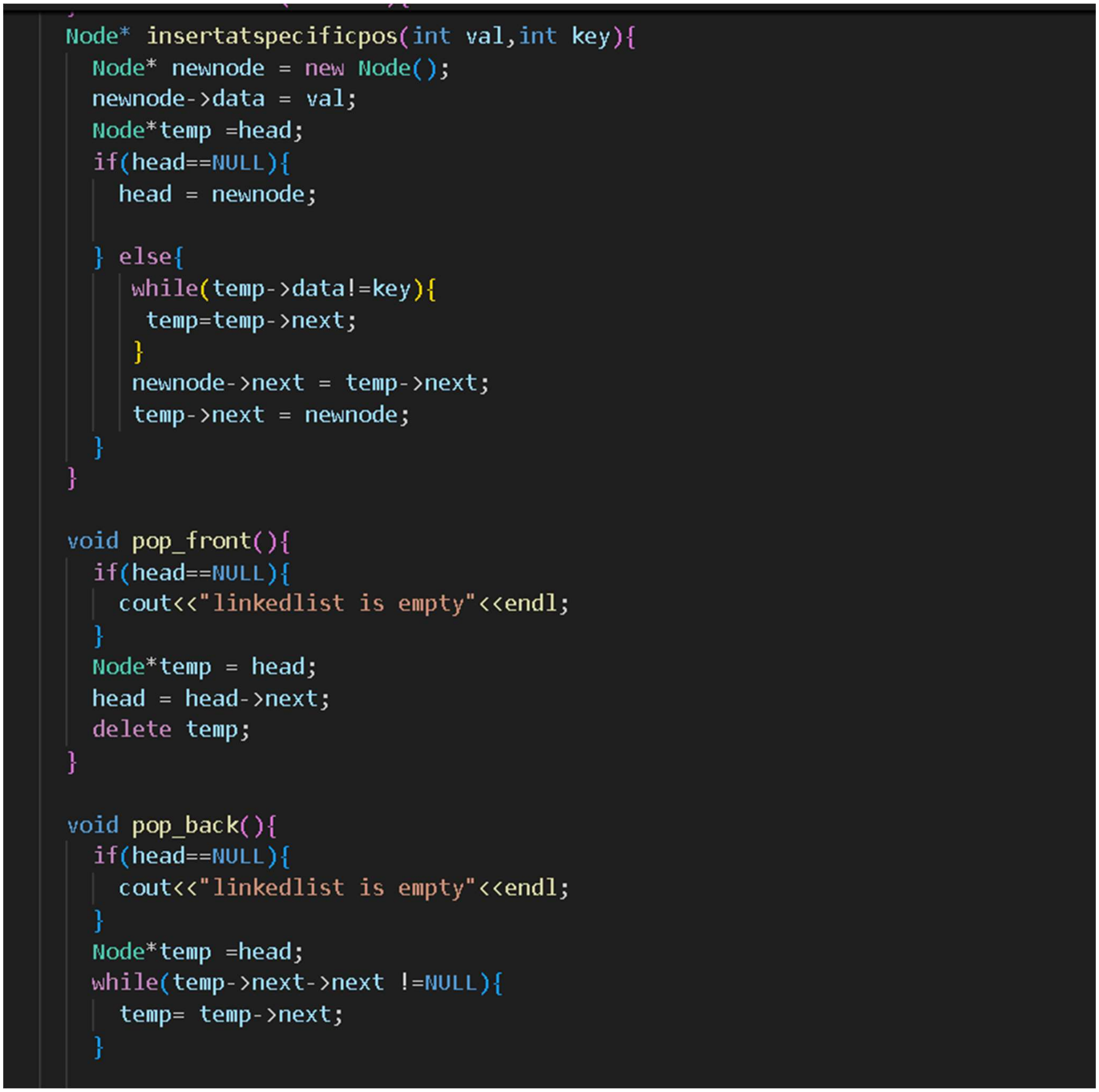
iii. Remove an element from the beginning of the doubly linked list

iv. Remove an element from i th position in the singly linked list.

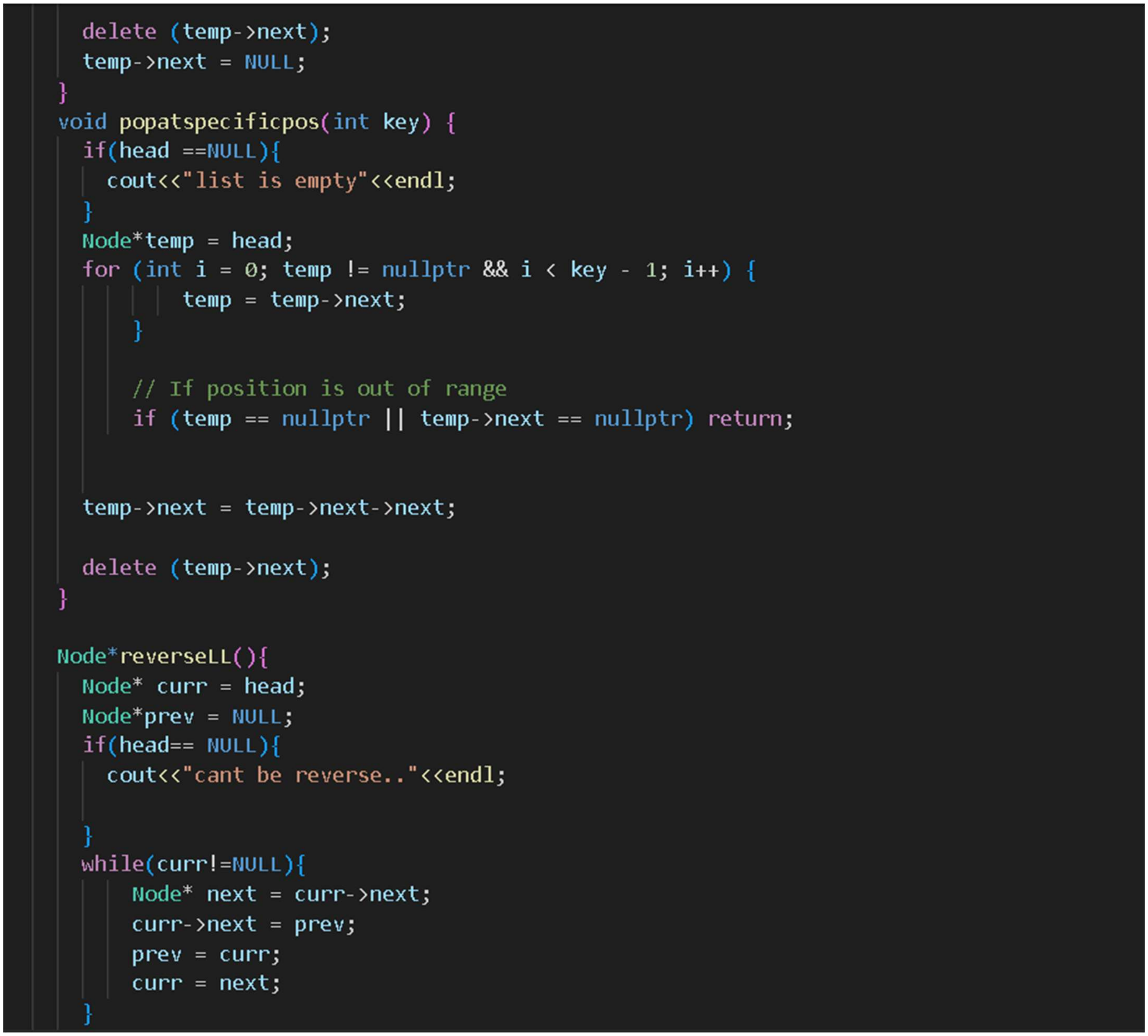
v. Search for an element x in the singly linked list and return its pointer



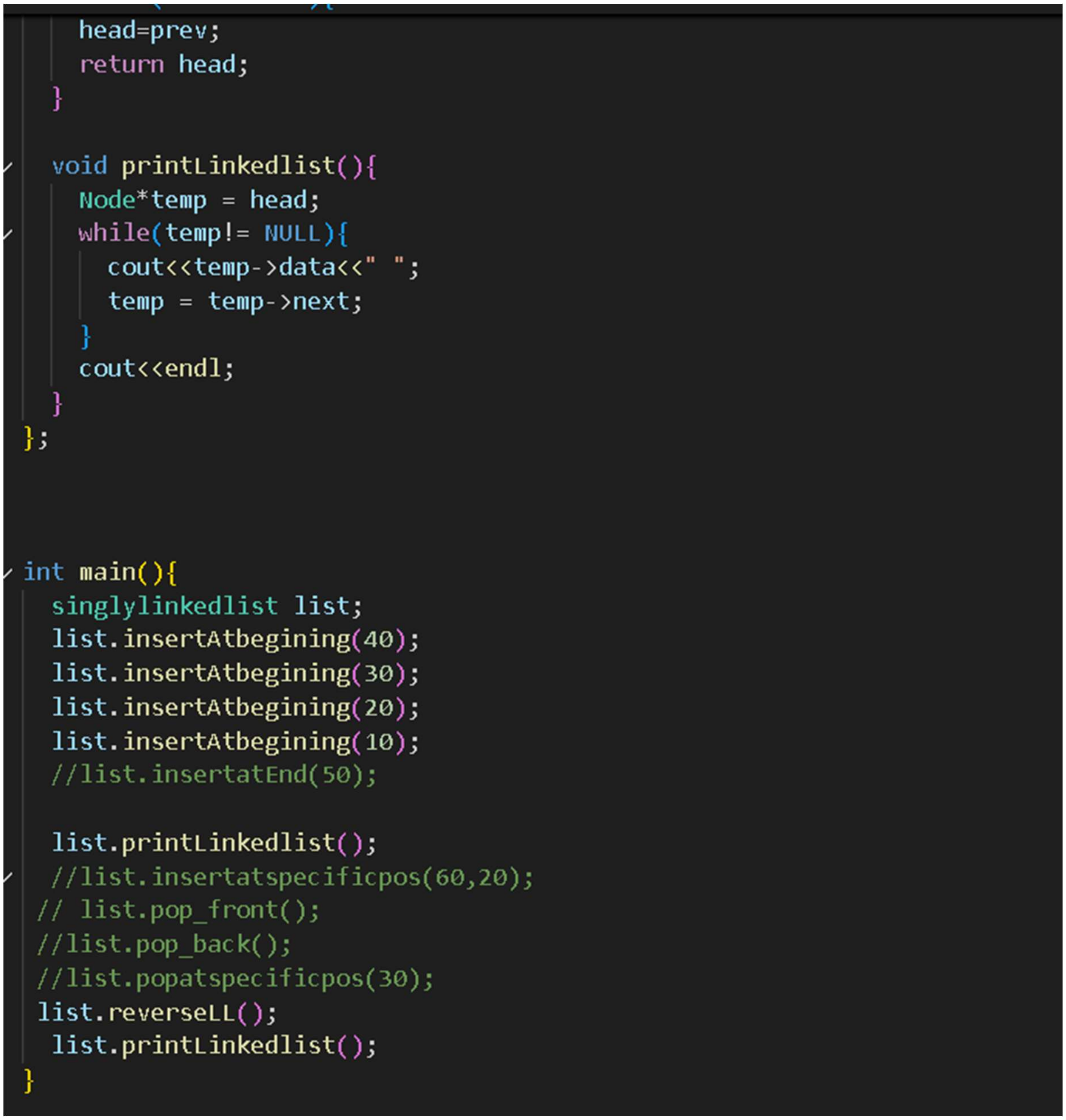
```cpp
#include<iostream>
#include<vector>
using namespace std;

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

class singlylinkedlist{
public:
    Node* head;

    singlylinkedlist(){
        head = NULL;
    }

    Node*insertAtbegining(int val){
        Node* newnode = new Node();
        newnode->data = val;
        newnode->next = head;
        head = newnode;
    }

    Node*insertatEnd(int val){
        Node* newnode = new Node();
        Node*temp = head;
        if(head ==NULL){
            newnode = head;
        }
        while(temp->next != NULL){
            temp = temp->next;
        }
        temp->next = newnode;
        newnode->data = val;
        newnode->next = NULL;
```



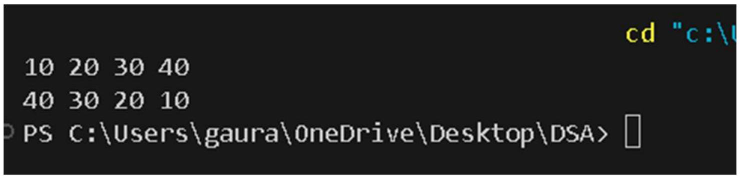
```
Node* insertatspecificpos(int val,int key){
  Node* newnode = new Node();
  newnode->data = val;
  Node*temp =head;
  if(head==NULL){
    head = newnode;

  } else{
     while(temp->data!=key){
      temp=temp->next;
     }
     newnode->next = temp->next;
     temp->next = newnode;
  }
}

void pop_front(){
  if(head==NULL){
    cout<<"linkedlist is empty"<<endl;
  }
  Node*temp = head;
  head = head->next;
  delete temp;
}

void pop_back(){
  if(head==NULL){
    cout<<"linkedlist is empty"<<endl;
  }
  Node*temp =head;
  while(temp->next->next !=NULL){
    temp= temp->next;
  }
```



```
  delete (temp->next);
  temp->next = NULL;
}
void popatspecificpos(int key) {
  if(head ==NULL){
    cout<<"list is empty"<<endl;
  }
  Node*temp = head;
  for (int i = 0; temp != nullptr && i < key - 1; i++) {
          temp = temp->next;
      }

      // If position is out of range
      if (temp == nullptr || temp->next == nullptr) return;


  temp->next = temp->next->next;

  delete (temp->next);
}

Node*reverseLL(){
  Node* curr = head;
  Node*prev = NULL;
  if(head== NULL){
    cout<<"cant be reverse.."<<endl;

  }
  while(curr!=NULL){
      Node* next = curr->next;
      curr->next = prev;
      prev = curr;
      curr = next;
  }
```



```
    head=prev;
    return head;
  }

  void printLinkedlist(){
    Node*temp = head;
    while(temp!= NULL){
      cout<<temp->data<<" ";
      temp = temp->next;
    }
    cout<<endl;
  }
};



int main(){
  singlylinkedlist list;
  list.insertAtbegining(40);
  list.insertAtbegining(30);
  list.insertAtbegining(20);
  list.insertAtbegining(10);
  //list.insertatEnd(50);

  list.printLinkedlist();
  //list.insertatspecificpos(60,20);
 // list.pop_front();
 //list.pop_back();
 //list.popatspecificpos(30);
 list.reverseLL();
  list.printLinkedlist();
}
```



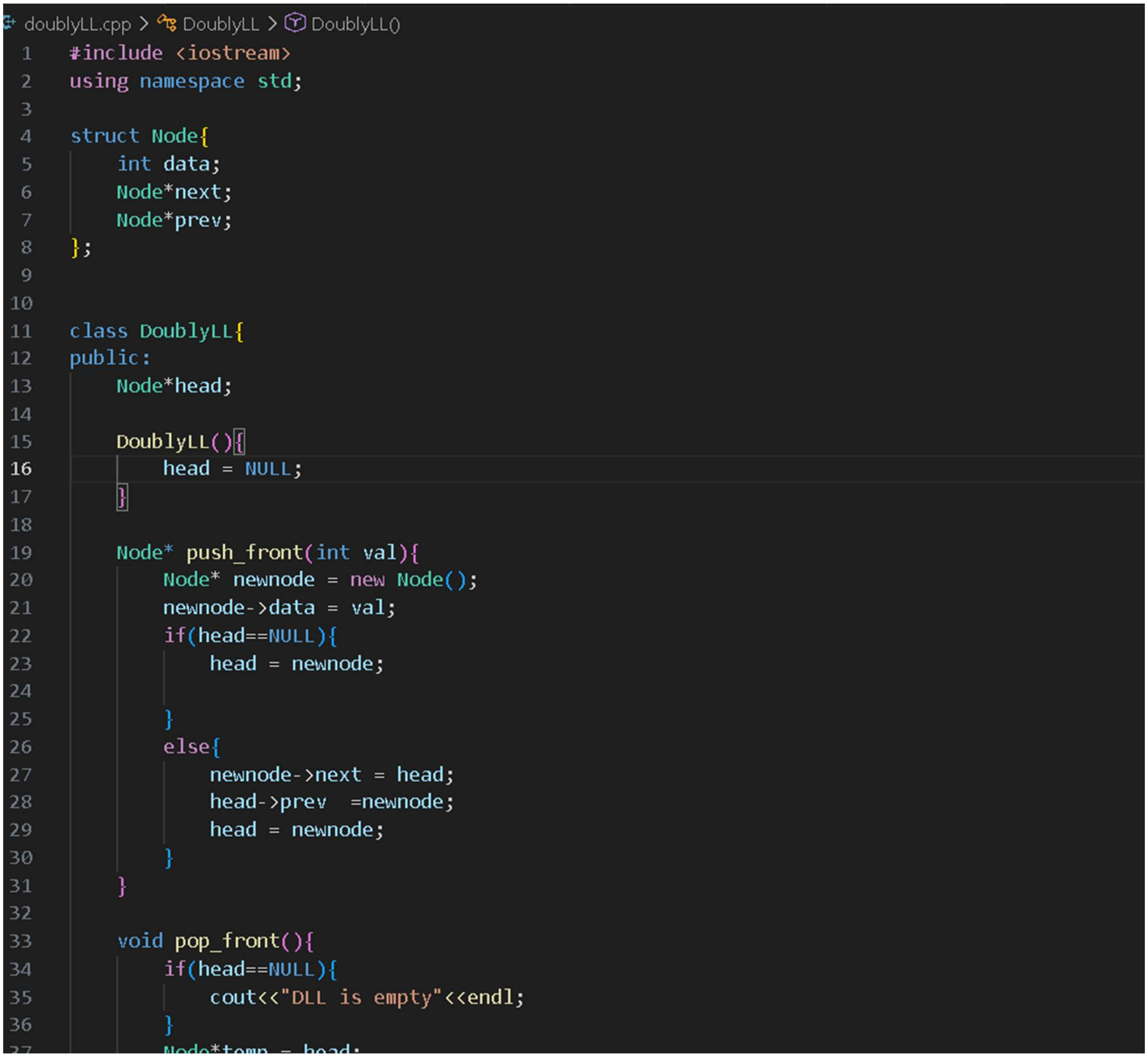
```
                                        cd "c:\
10 20 30 40
40 30 20 10
PS C:\Users\gaura\OneDrive\Desktop\DSA> 
```

2 - Write a program to implement doubly linked list as an ADT that supports the following op rations: ii. (i)insert an element x at the beginning of the doubly linked list

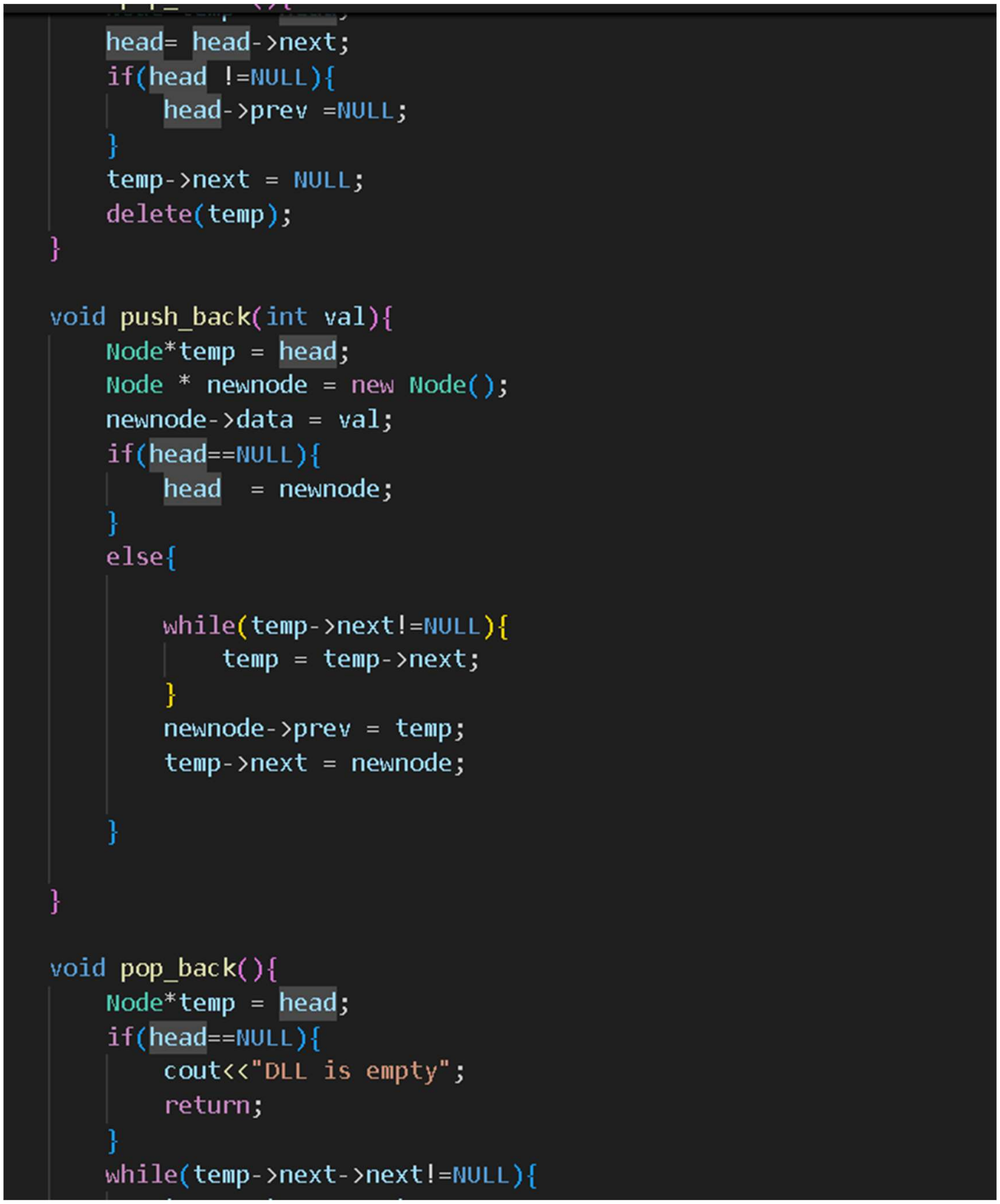
(ii) Insert an element x at the end of the doubly linked list

(iii)Remove an element from the beginning of the doubly linked list

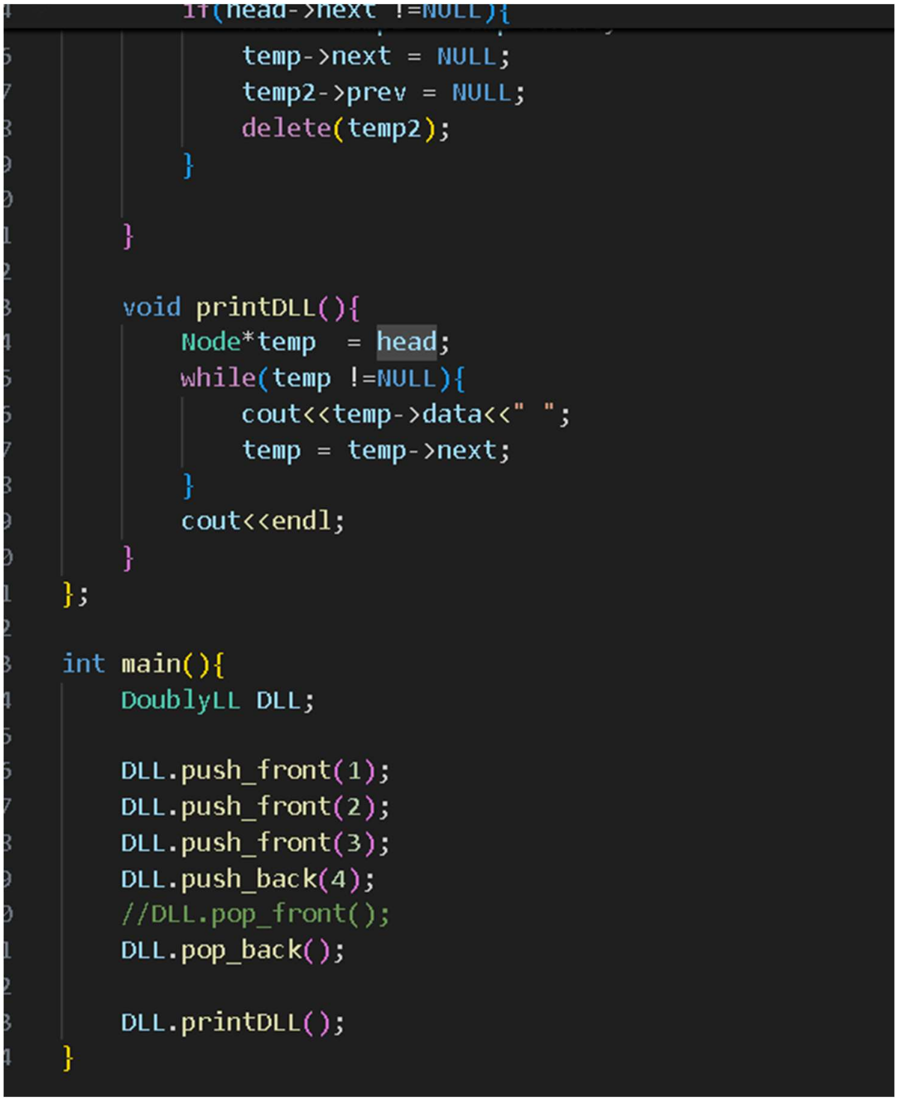
(iv)Remove an element from the end of the doubly linked list



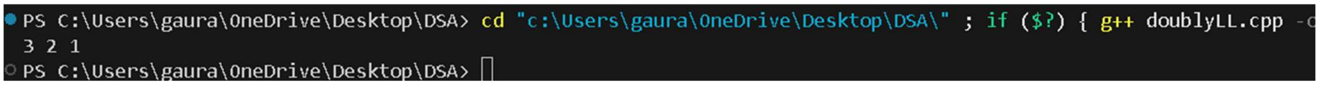
```cpp
#include <iostream>
using namespace std;

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


class DoublyLL{
public:
    Node*head;

    DoublyLL(){
        head = NULL;
    }

    Node* push_front(int val){
        Node* newnode = new Node();
        newnode->data = val;
        if(head==NULL){
            head = newnode;

        }
        else{
            newnode->next = head;
            head->prev  =newnode;
            head = newnode;
        }
    }

    void pop_front(){
        if(head==NULL){
            cout<<"DLL is empty"<<endl;
        }
        Node*temp = head;
```



```
    head= head->next;
    if(head !=NULL){
        head->prev =NULL;
    }
    temp->next = NULL;
    delete(temp);
}

void push_back(int val){
    Node*temp = head;
    Node * newnode = new Node();
    newnode->data = val;
    if(head==NULL){
        head  = newnode;
    }
    else{

        while(temp->next!=NULL){
            temp = temp->next;
        }
        newnode->prev = temp;
        temp->next = newnode;

    }

}

void pop_back(){
    Node*temp = head;
    if(head==NULL){
        cout<<"DLL is empty";
        return;
    }
    while(temp->next->next!=NULL){
```

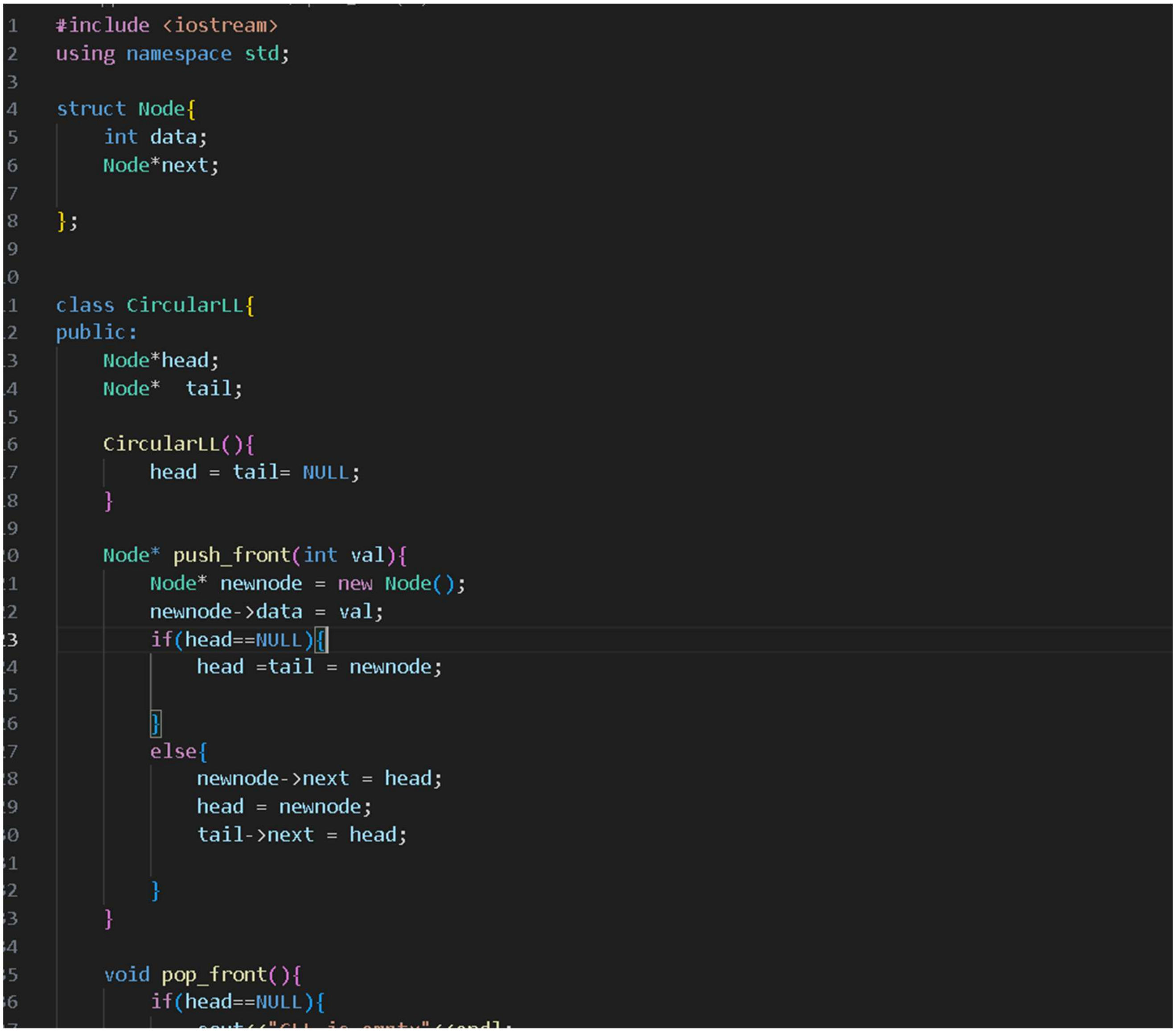


```
        if(head->next !=NULL){
            temp->next = NULL;
            temp2->prev = NULL;
            delete(temp2);
        }

    }

    void printDLL(){
        Node*temp  = head;
        while(temp !=NULL){
            cout<<temp->data<<" ";
            temp = temp->next;
        }
        cout<<endl;
    }
};

int main(){
    DoublyLL DLL;

    DLL.push_front(1);
    DLL.push_front(2);
    DLL.push_front(3);
    DLL.push_back(4);
    //DLL.pop_front();
    DLL.pop_back();

    DLL.printDLL();
}
```

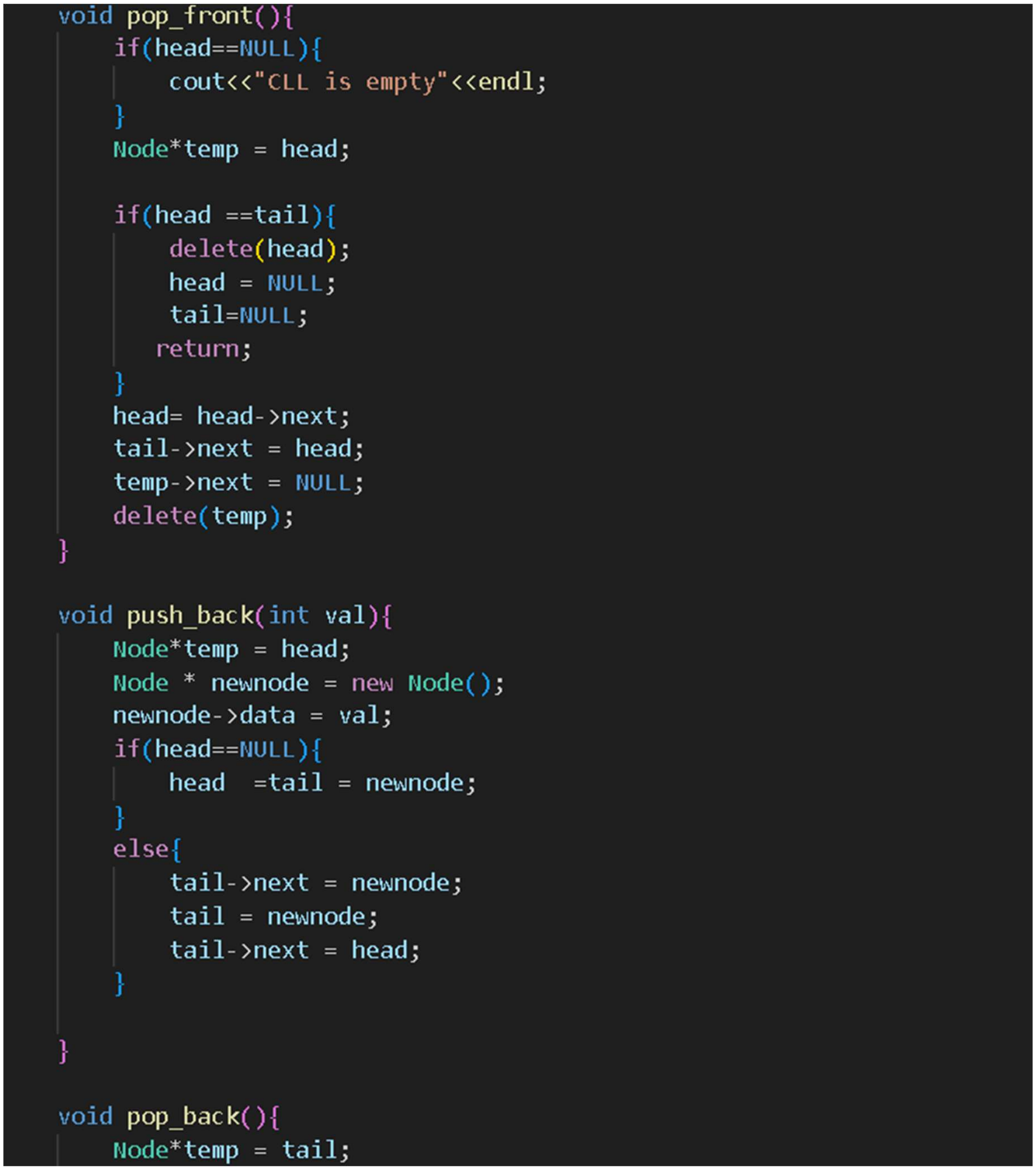


```
PS C:\Users\gaura\OneDrive\Desktop\DSA> cd "c:\Users\gaura\OneDrive\Desktop\DSA\" ; if ($?) { g++ doublyLL.cpp -c
 3 2 1
PS C:\Users\gaura\OneDrive\Desktop\DSA> 
```

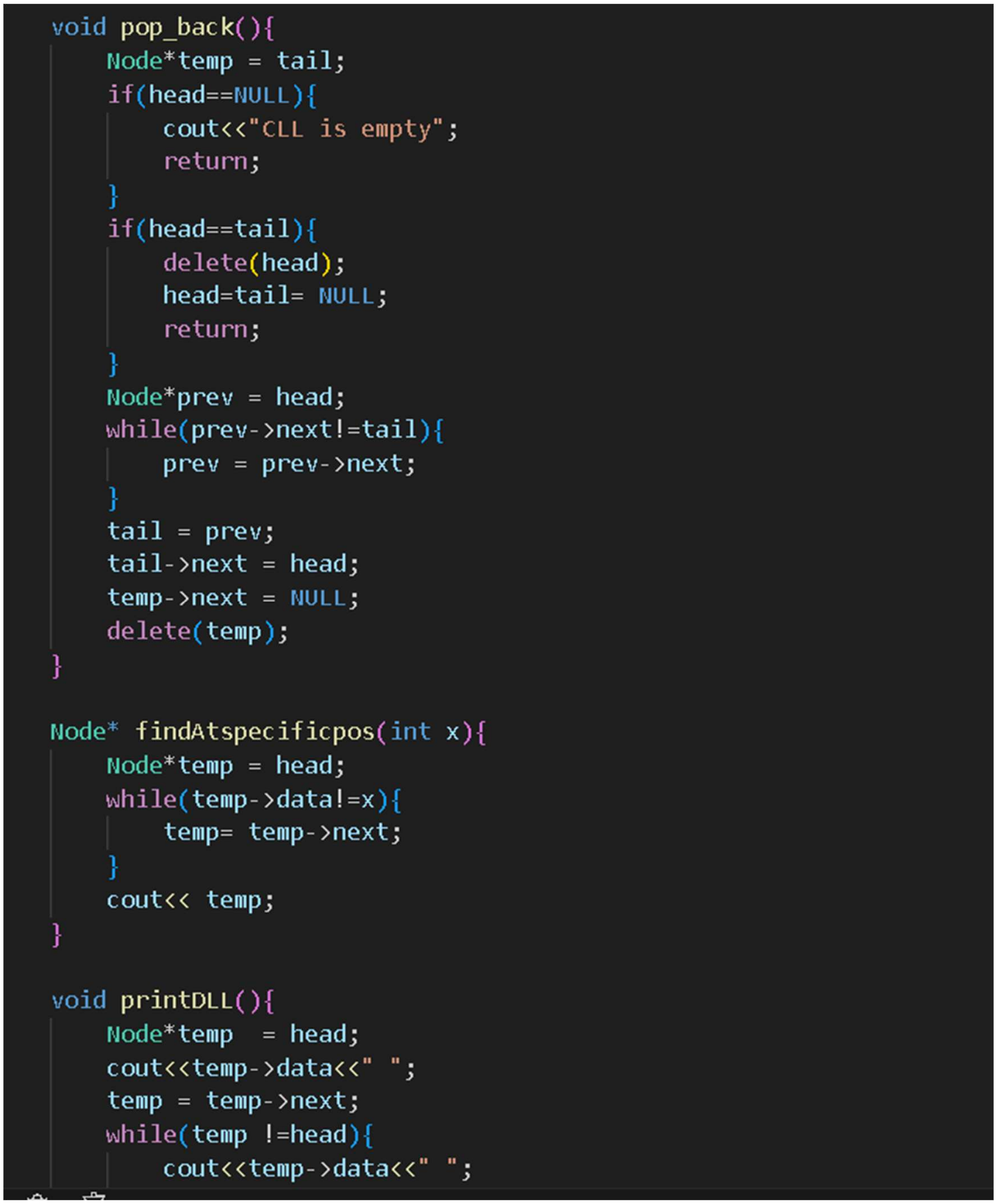
3 - Write a program to implement circular linked list as an ADT which supports the following operations: i. ii. iii. Insert an element x in the list Remove an element from the list Search for an element x in the list and return its pointer



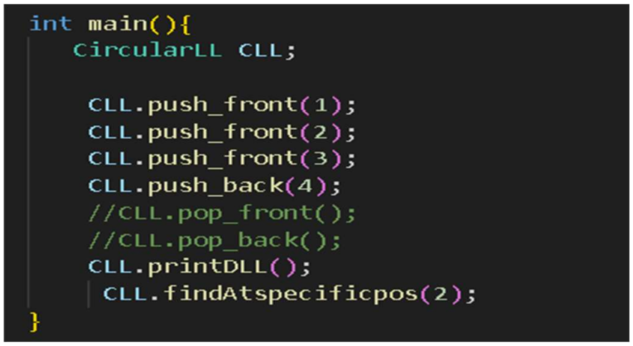
```
#include <iostream>
using namespace std;

struct Node{
    int data;
    Node*next;

};


class CircularLL{
public:
    Node*head;
    Node*  tail;

    CircularLL(){
        head = tail= NULL;
    }

    Node* push_front(int val){
        Node* newnode = new Node();
        newnode->data = val;
        if(head==NULL){
            head =tail = newnode;

        }
        else{
            newnode->next = head;
            head = newnode;
            tail->next = head;

        }
    }

    void pop_front(){
        if(head==NULL){
            cout<<"CLL is empty"<<endl;
```



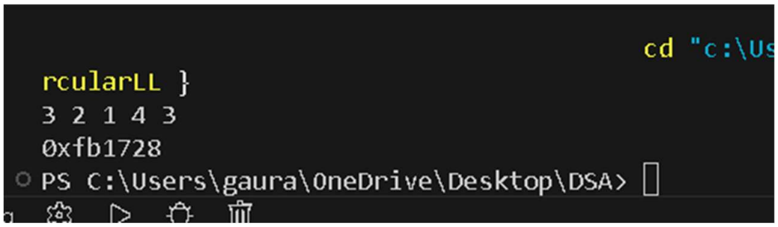
```
    void pop_front(){
        if(head==NULL){
            cout<<"CLL is empty"<<endl;
        }
        Node*temp = head;

        if(head ==tail){
            delete(head);
            head = NULL;
            tail=NULL;
           return;
        }
        head= head->next;
        tail->next = head;
        temp->next = NULL;
        delete(temp);
    }

    void push_back(int val){
        Node*temp = head;
        Node * newnode = new Node();
        newnode->data = val;
        if(head==NULL){
            head  =tail = newnode;
        }
        else{
            tail->next = newnode;
            tail = newnode;
            tail->next = head;
        }

    }

    void pop_back(){
        Node*temp = tail;
```



```
void pop_back(){
    Node*temp = tail;
    if(head==NULL){
        cout<<"CLL is empty";
        return;
    }
    if(head==tail){
        delete(head);
        head=tail= NULL;
        return;
    }
    Node*prev = head;
    while(prev->next!=tail){
        prev = prev->next;
    }
    tail = prev;
    tail->next = head;
    temp->next = NULL;
    delete(temp);
}

Node* findAtspecificpos(int x){
    Node*temp = head;
    while(temp->data!=x){
        temp= temp->next;
    }
    cout<< temp;
}

void printDLL(){
    Node*temp  = head;
    cout<<temp->data<<" ";
    temp = temp->next;
    while(temp !=head){
        cout<<temp->data<<" ";
```

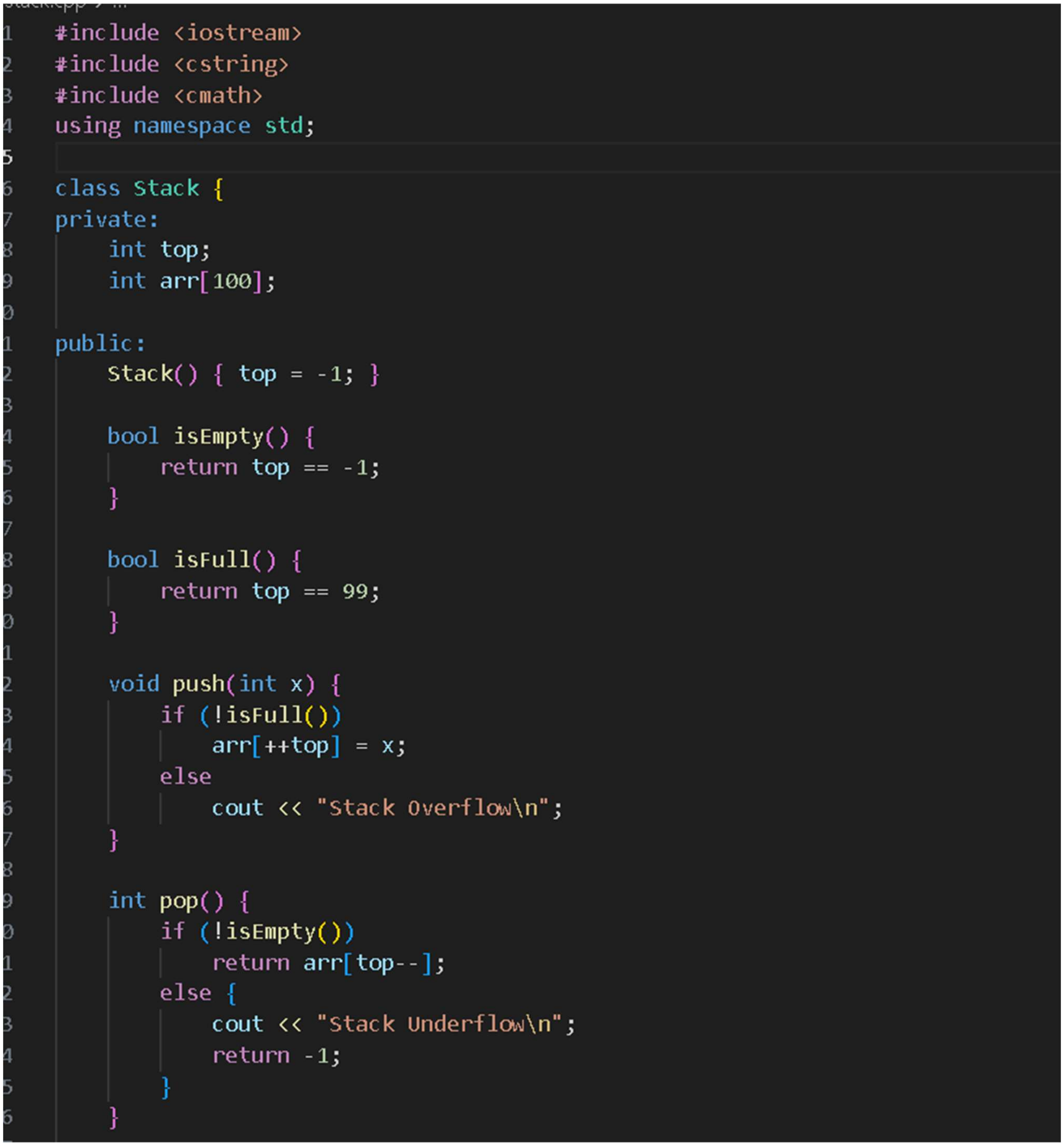


```
int main(){
   CircularLL CLL;

    CLL.push_front(1);
    CLL.push_front(2);
    CLL.push_front(3);
    CLL.push_back(4);
    //CLL.pop_front();
    //CLL.pop_back();
    CLL.printDLL();
     CLL.findAtspecificpos(2);
}
```

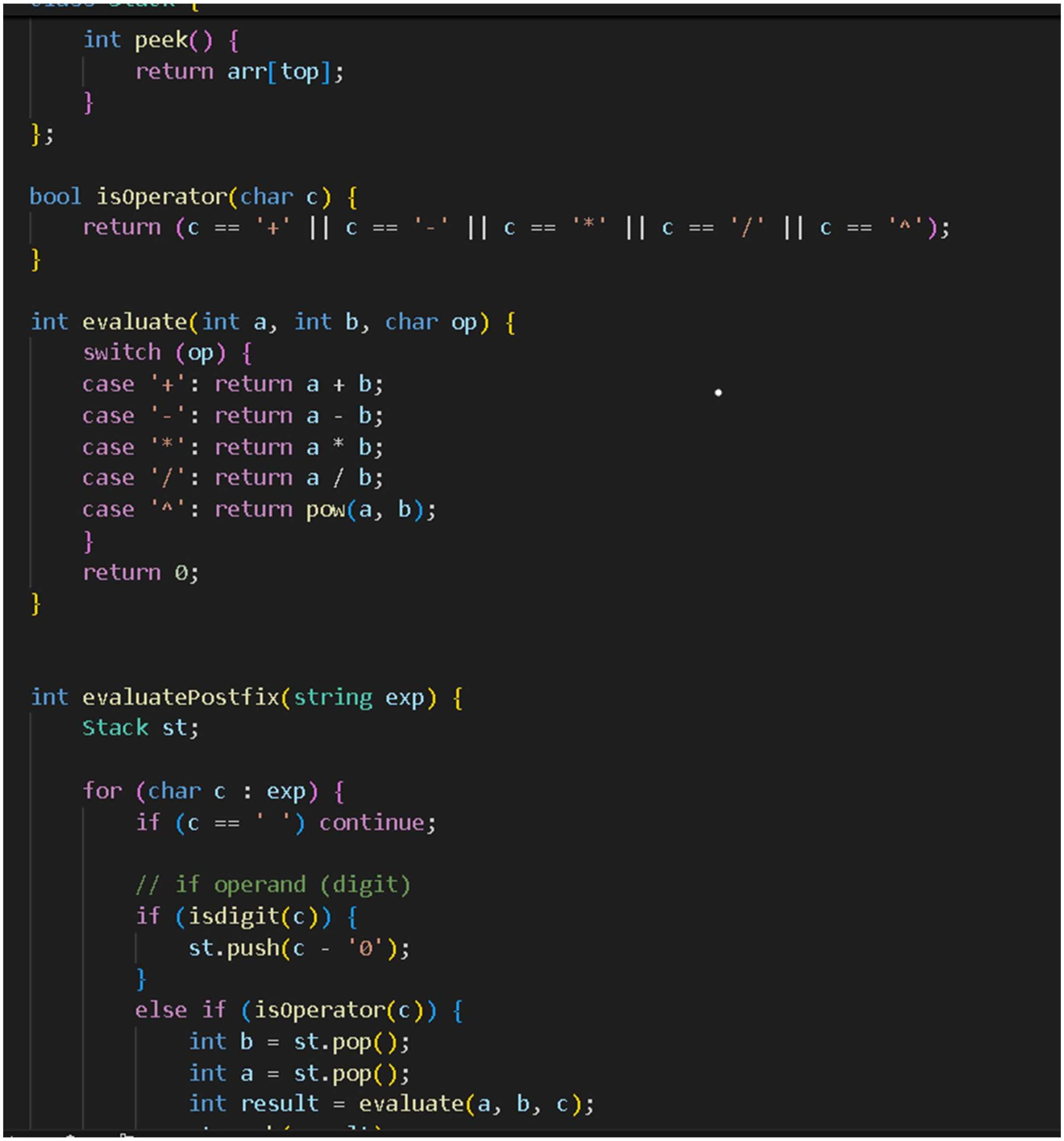


```
                                          cd "c:\Us
rcularLL }
3 2 1 4 3
0xfb1728
PS C:\Users\gaura\OneDrive\Desktop\DSA> 
```

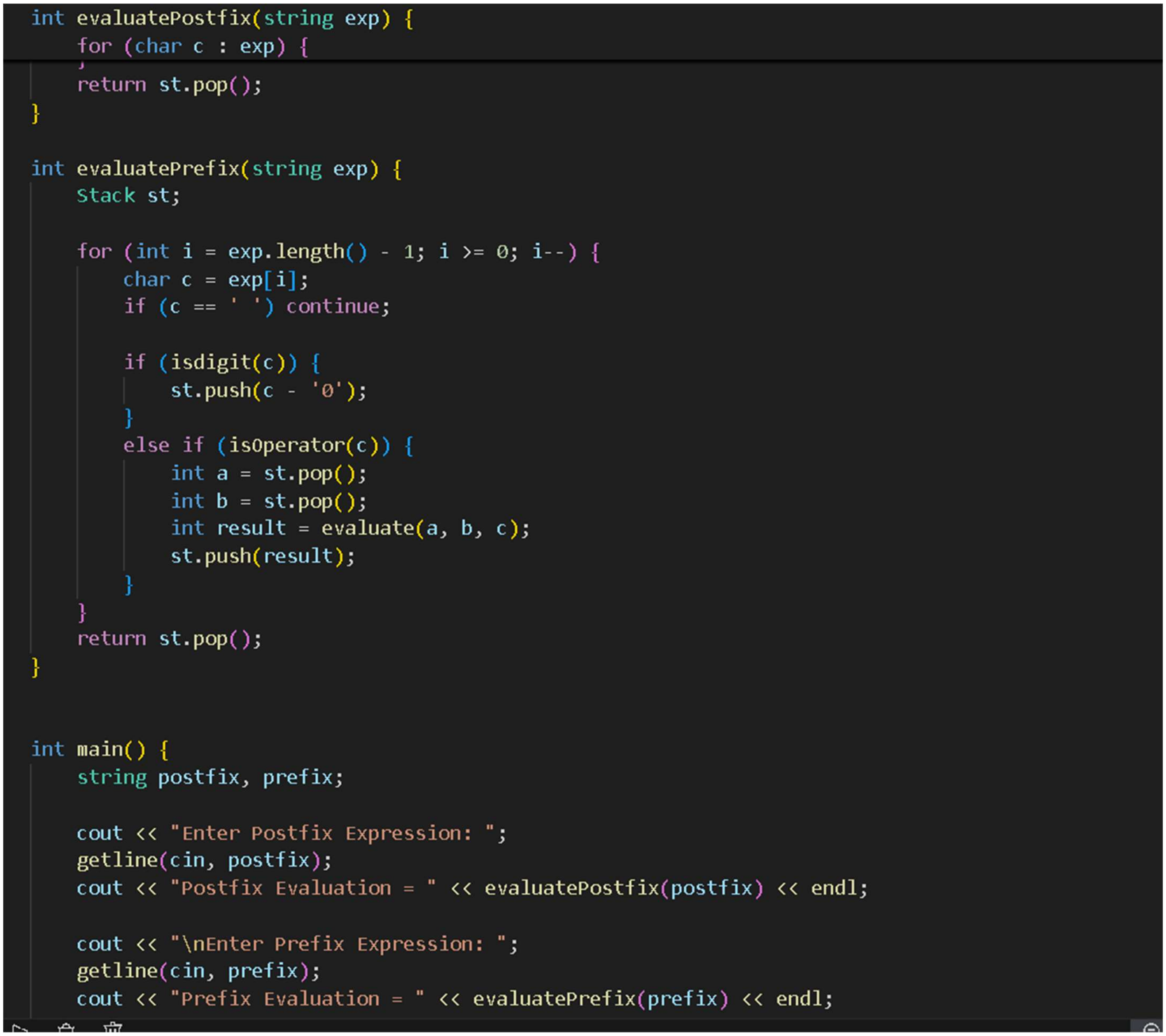
4- Implement Stack as an ADT and use it to evaluate a prefix/postfix expression.



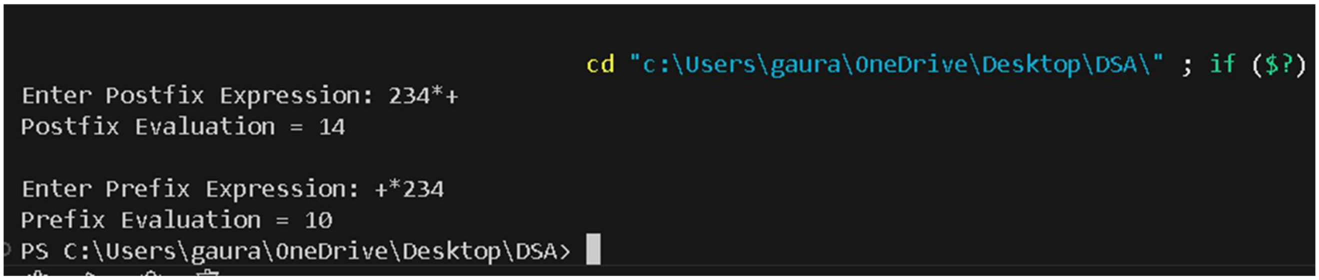
```cpp
#include <iostream>
#include <cstring>
#include <cmath>
using namespace std;

class Stack {
private:
    int top;
    int arr[100];

public:
    Stack() { top = -1; }

    bool isEmpty() {
        return top == -1;
    }

    bool isFull() {
        return top == 99;
    }

    void push(int x) {
        if (!isFull())
            arr[++top] = x;
        else
            cout << "Stack Overflow\n";
    }

    int pop() {
        if (!isEmpty())
            return arr[top--];
        else {
            cout << "Stack Underflow\n";
            return -1;
        }
    }
```



```cpp
class Stack {
    int peek() {
        return arr[top];
    }
};

bool isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
}

int evaluate(int a, int b, char op) {
    switch (op) {
    case '+': return a + b;
    case '-': return a - b;
    case '*': return a * b;
    case '/': return a / b;
    case '^': return pow(a, b);
    }
    return 0;
}


int evaluatePostfix(string exp) {
    Stack st;

    for (char c : exp) {
        if (c == ' ') continue;

        // if operand (digit)
        if (isdigit(c)) {
            st.push(c - '0');
        }
        else if (isOperator(c)) {
            int b = st.pop();
            int a = st.pop();
            int result = evaluate(a, b, c);
            st.push(result);
```

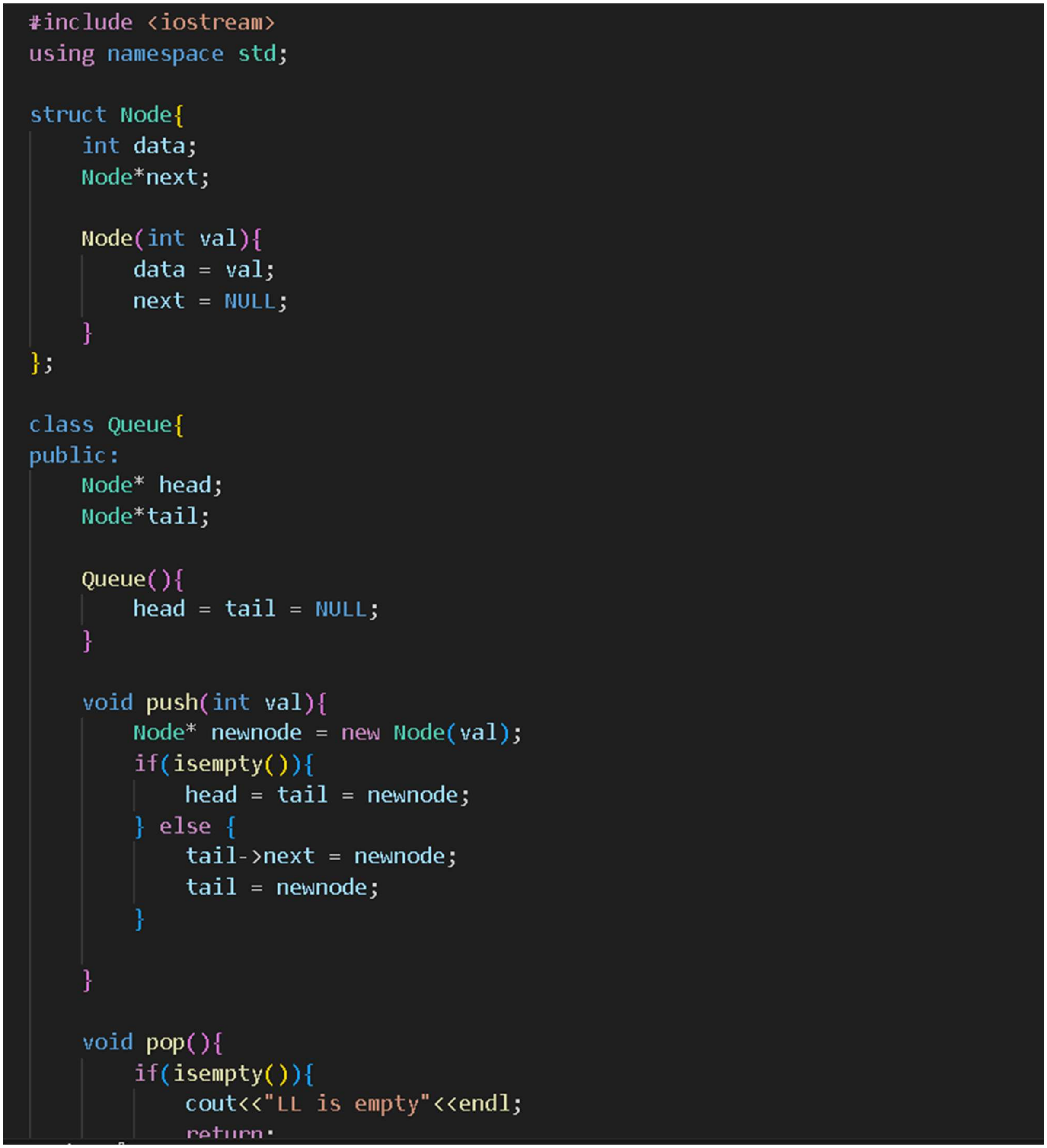


```
int evaluatePostfix(string exp) {
    for (char c : exp) {
    }
    return st.pop();
}

int evaluatePrefix(string exp) {
    Stack st;

    for (int i = exp.length() - 1; i >= 0; i--) {
        char c = exp[i];
        if (c == ' ') continue;

        if (isdigit(c)) {
            st.push(c - '0');
        }
        else if (isOperator(c)) {
            int a = st.pop();
            int b = st.pop();
            int result = evaluate(a, b, c);
            st.push(result);
        }
    }
    return st.pop();
}


int main() {
    string postfix, prefix;

    cout << "Enter Postfix Expression: ";
    getline(cin, postfix);
    cout << "Postfix Evaluation = " << evaluatePostfix(postfix) << endl;

    cout << "\nEnter Prefix Expression: ";
    getline(cin, prefix);
    cout << "Prefix Evaluation = " << evaluatePrefix(prefix) << endl;
```

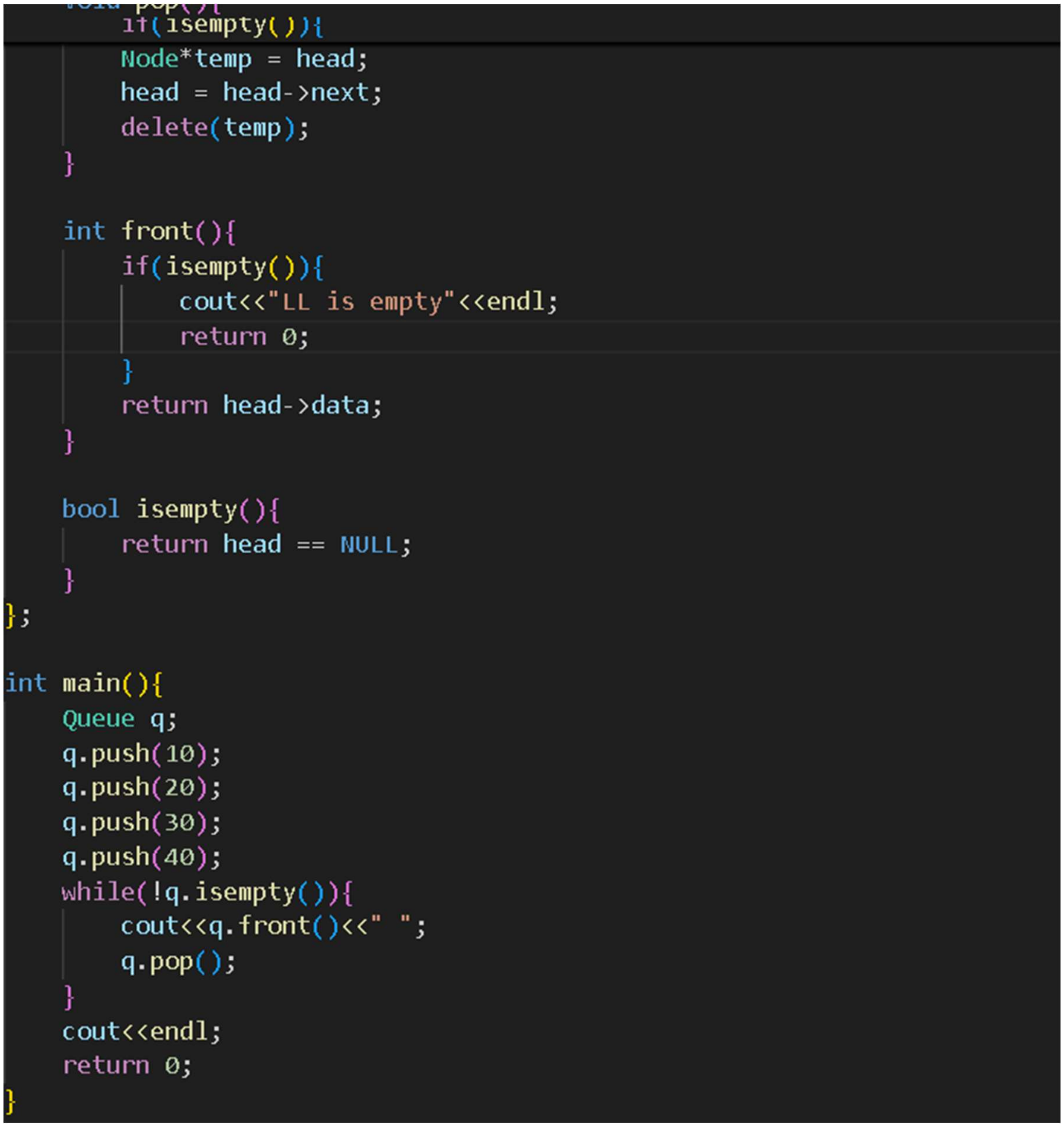


```
                                        cd "c:\Users\gaura\OneDrive\Desktop\DSA\" ; if ($?)
Enter Postfix Expression: 234*+
Postfix Evaluation = 14

Enter Prefix Expression: +*234
Prefix Evaluation = 10
PS C:\Users\gaura\OneDrive\Desktop\DSA>
```

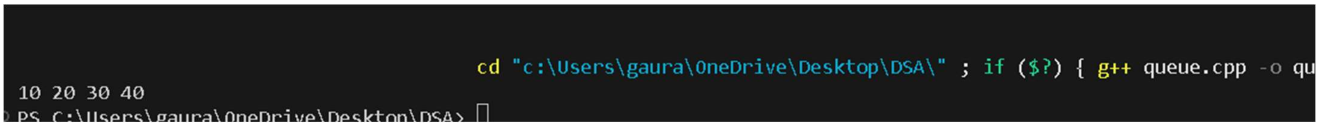
5 - Implement Queue as an ADT.



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

struct Node{
    int data;
    Node*next;

    Node(int val){
        data = val;
        next = NULL;
    }
};

class Queue{
public:
    Node* head;
    Node*tail;

    Queue(){
        head = tail = NULL;
    }

    void push(int val){
        Node* newnode = new Node(val);
        if(isempty()){
            head = tail = newnode;
        } else {
            tail->next = newnode;
            tail = newnode;
        }

    }

    void pop(){
        if(isempty()){
            cout<<"LL is empty"<<endl;
            return;
```



```
    void pop(){
        if(isempty()){
        Node*temp = head;
        head = head->next;
        delete(temp);
    }

    int front(){
        if(isempty()){
            cout<<"LL is empty"<<endl;
            return 0;
        }
        return head->data;
    }

    bool isempty(){
        return head == NULL;
    }
};

int main(){
    Queue q;
    q.push(10);
    q.push(20);
    q.push(30);
    q.push(40);
    while(!q.isempty()){
        cout<<q.front()<<" ";
        q.pop();
    }
    cout<<endl;
    return 0;
}
```



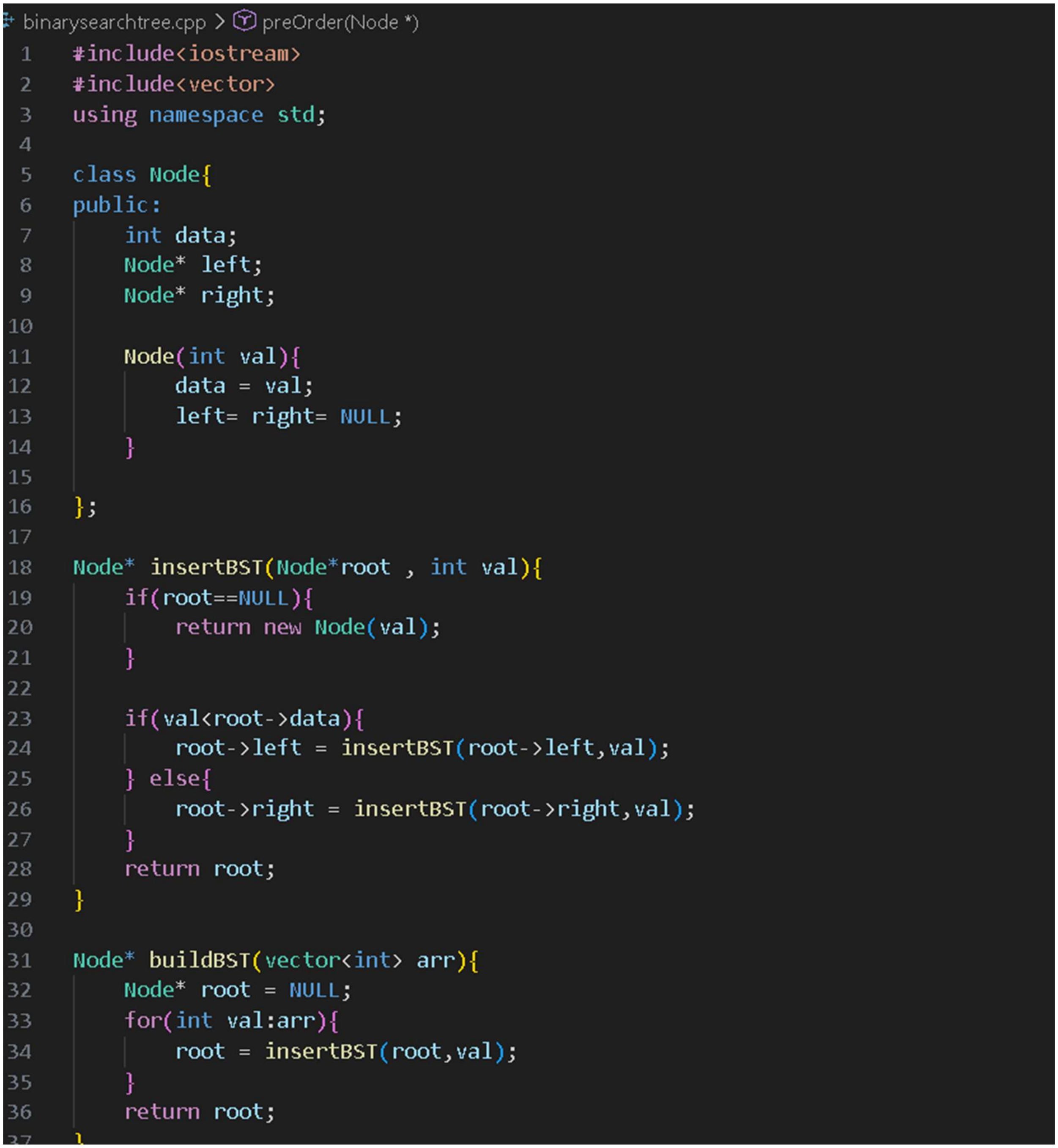
```
                                                cd "c:\Users\gaura\OneDrive\Desktop\DSA\" ; if ($?) { g++ queue.cpp -o qu
 10 20 30 40
 PS C:\Users\gaura\OneDrive\Desktop\DSA> 
```

6 - Write a program to implement Binary Search Tree as an ADT which supports the following operations: i. Insert an element x

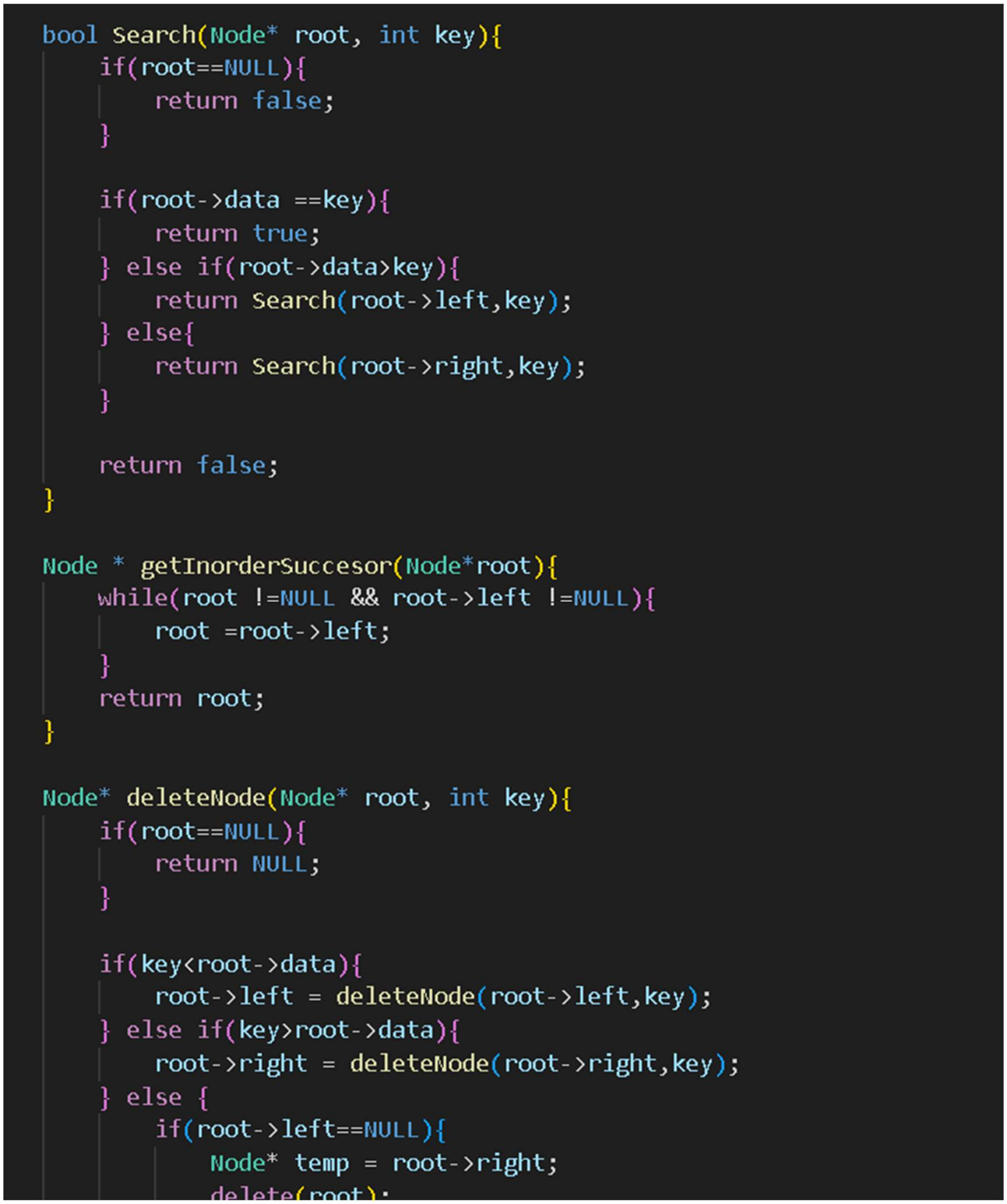
(ii)Delete an element x

(iii)Search for an element x in the BST

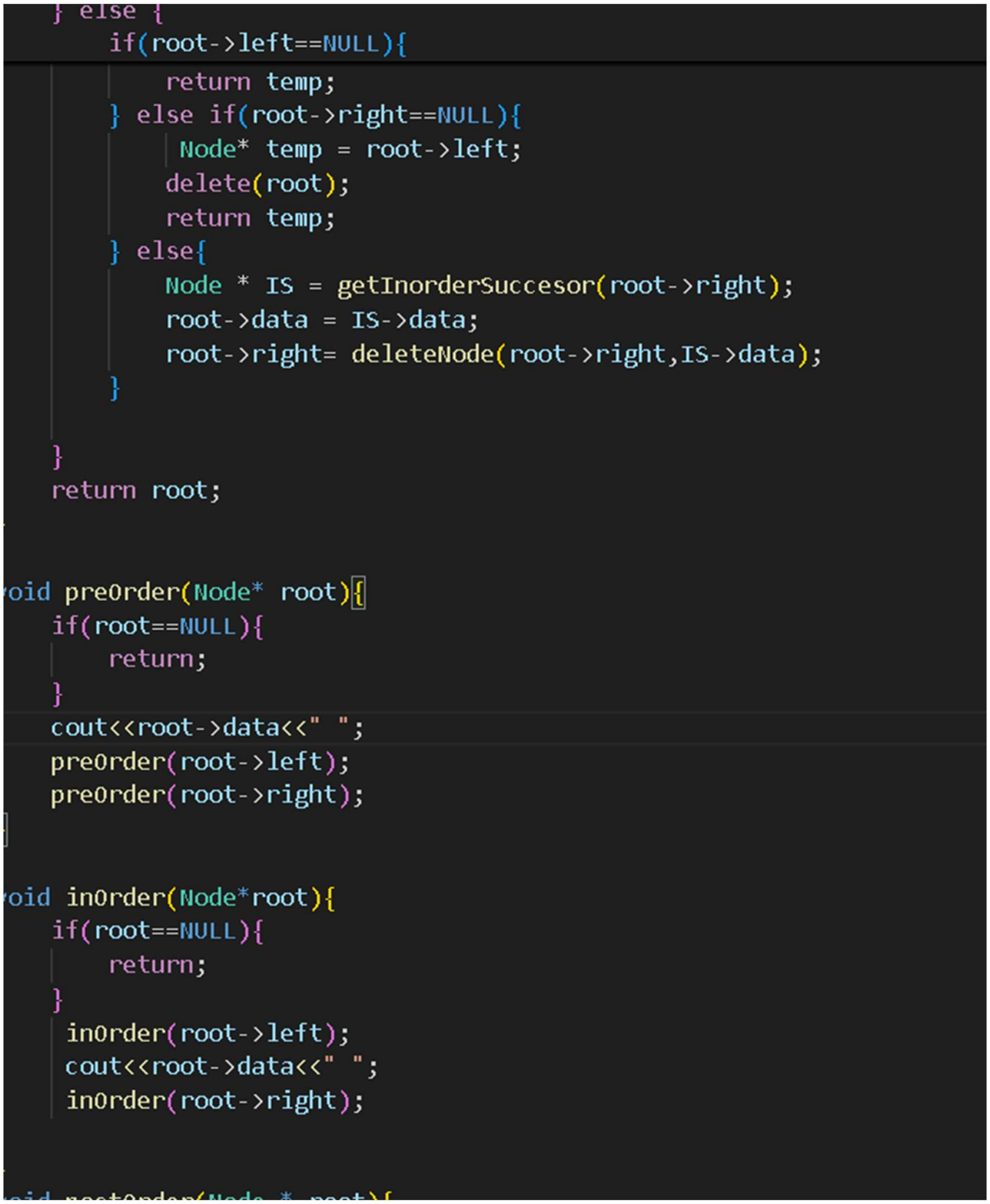
(iv)Display the elements of the BST in preorder, inorder, and postorder traversal



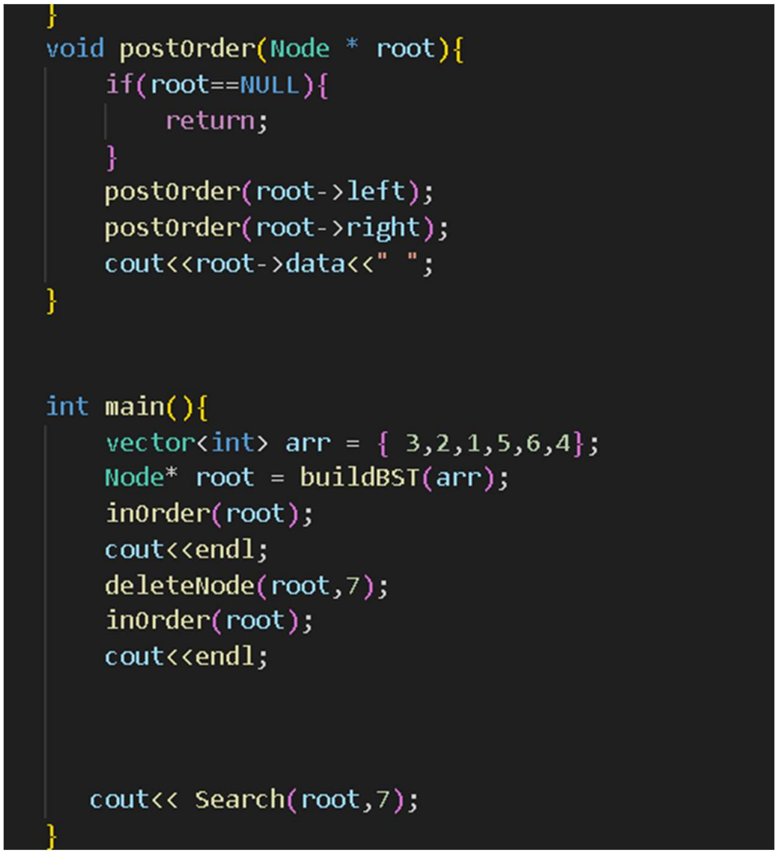
```cpp
#include<iostream>
#include<vector>
using namespace std;

class Node{
public:
    int data;
    Node* left;
    Node* right;

    Node(int val){
        data = val;
        left= right= NULL;
    }

};

Node* insertBST(Node*root , int val){
    if(root==NULL){
        return new Node(val);
    }

    if(val<root->data){
        root->left = insertBST(root->left,val);
    } else{
        root->right = insertBST(root->right,val);
    }
    return root;
}

Node* buildBST(vector<int> arr){
    Node* root = NULL;
    for(int val:arr){
        root = insertBST(root,val);
    }
    return root;
}
```



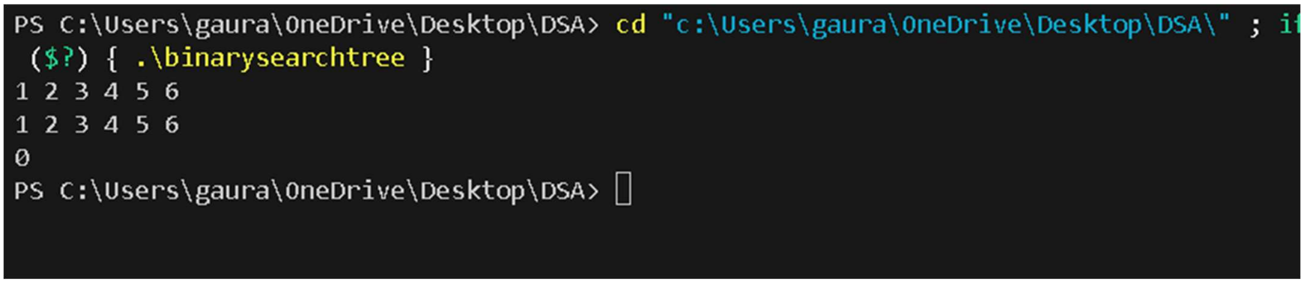
```
bool Search(Node* root, int key){
    if(root==NULL){
        return false;
    }

    if(root->data ==key){
        return true;
    } else if(root->data>key){
        return Search(root->left,key);
    } else{
        return Search(root->right,key);
    }

    return false;
}

Node * getInorderSuccesor(Node*root){
    while(root !=NULL && root->left !=NULL){
        root =root->left;
    }
    return root;
}

Node* deleteNode(Node* root, int key){
    if(root==NULL){
        return NULL;
    }

    if(key<root->data){
        root->left = deleteNode(root->left,key);
    } else if(key>root->data){
        root->right = deleteNode(root->right,key);
    } else {
        if(root->left==NULL){
            Node* temp = root->right;
            delete(root);
```



```
    } else {
        if(root->left==NULL){
            return temp;
        } else if(root->right==NULL){
            Node* temp = root->left;
            delete(root);
            return temp;
        } else{
            Node * IS = getInorderSuccesor(root->right);
            root->data = IS->data;
            root->right= deleteNode(root->right,IS->data);
        }

    }
    return root;


void preOrder(Node* root){
    if(root==NULL){
        return;
    }
    cout<<root->data<<" ";
    preOrder(root->left);
    preOrder(root->right);


void inOrder(Node*root){
    if(root==NULL){
        return;
    }
     inOrder(root->left);
     cout<<root->data<<" ";
     inOrder(root->right);


void postOrder(Node * root){
```

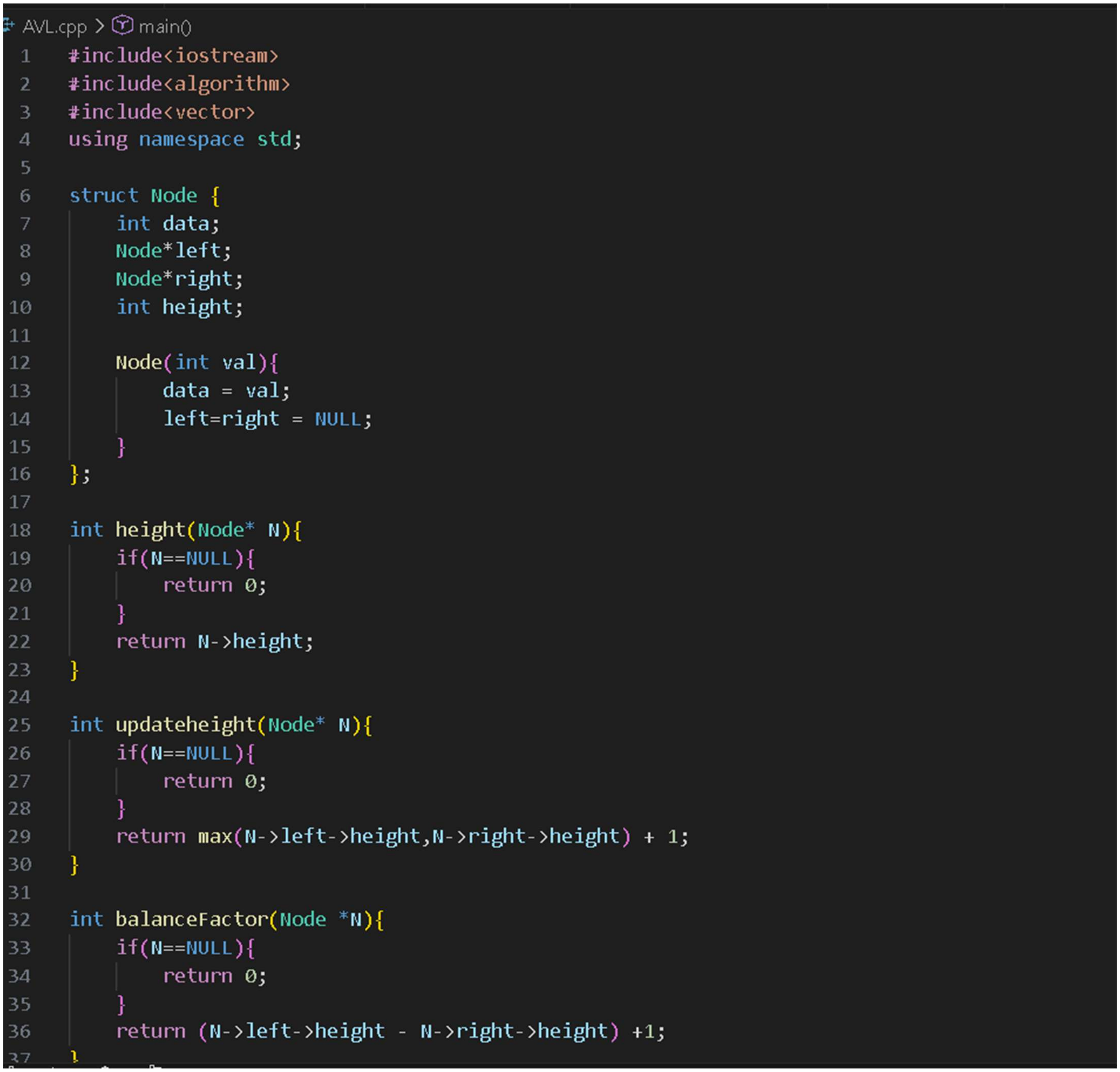


```
}
void postOrder(Node * root){
    if(root==NULL){
        return;
    }
    postOrder(root->left);
    postOrder(root->right);
    cout<<root->data<<" ";
}


int main(){
    vector<int> arr = { 3,2,1,5,6,4};
    Node* root = buildBST(arr);
    inOrder(root);
    cout<<endl;
    deleteNode(root,7);
    inOrder(root);
    cout<<endl;



   cout<< Search(root,7);
}
```

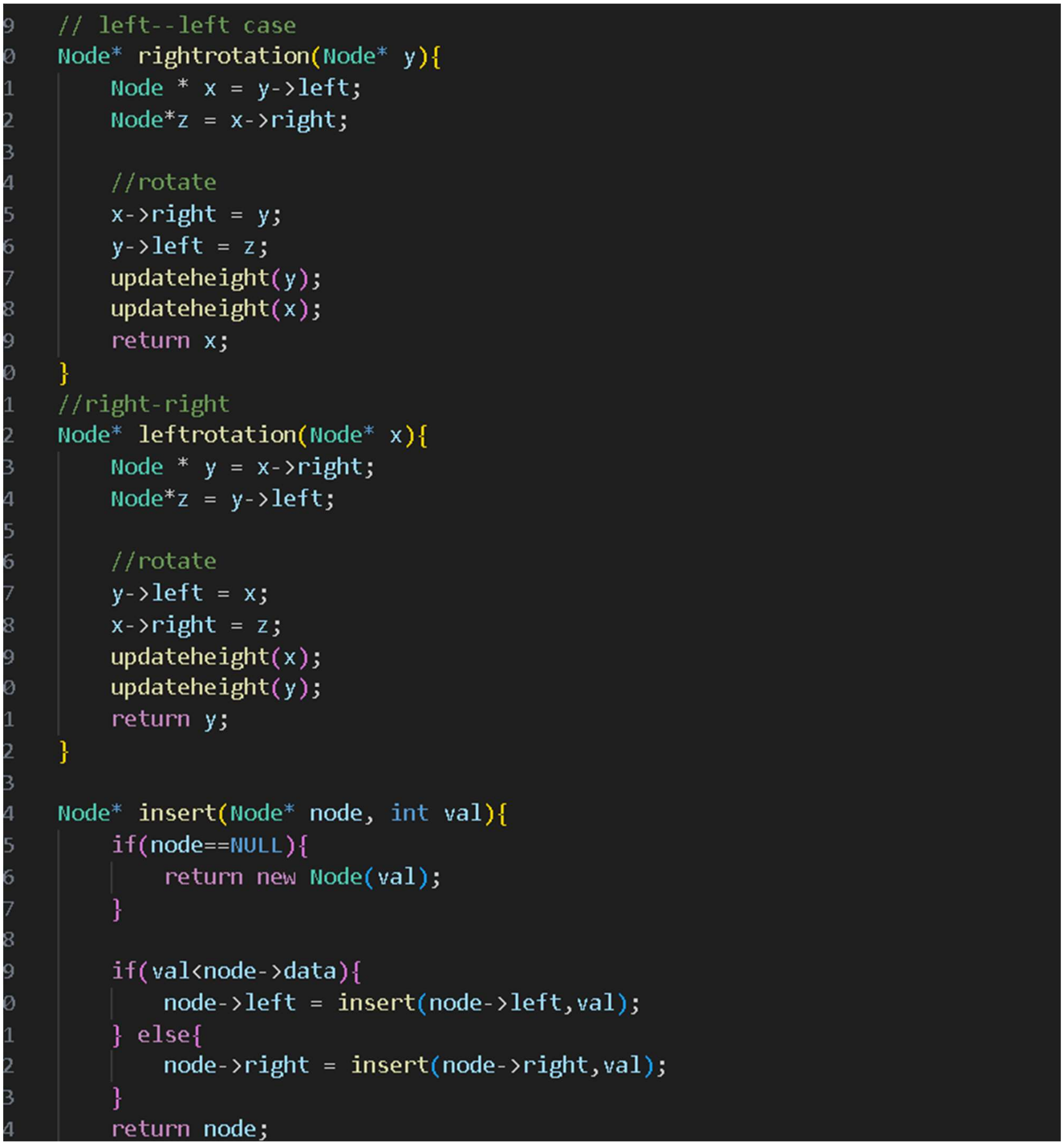


```
PS C:\Users\gaura\OneDrive\Desktop\DSA> cd "c:\Users\gaura\OneDrive\Desktop\DSA\" ; if
 ($?) { .\binarysearchtree }
1 2 3 4 5 6
1 2 3 4 5 6
0
PS C:\Users\gaura\OneDrive\Desktop\DSA>
```

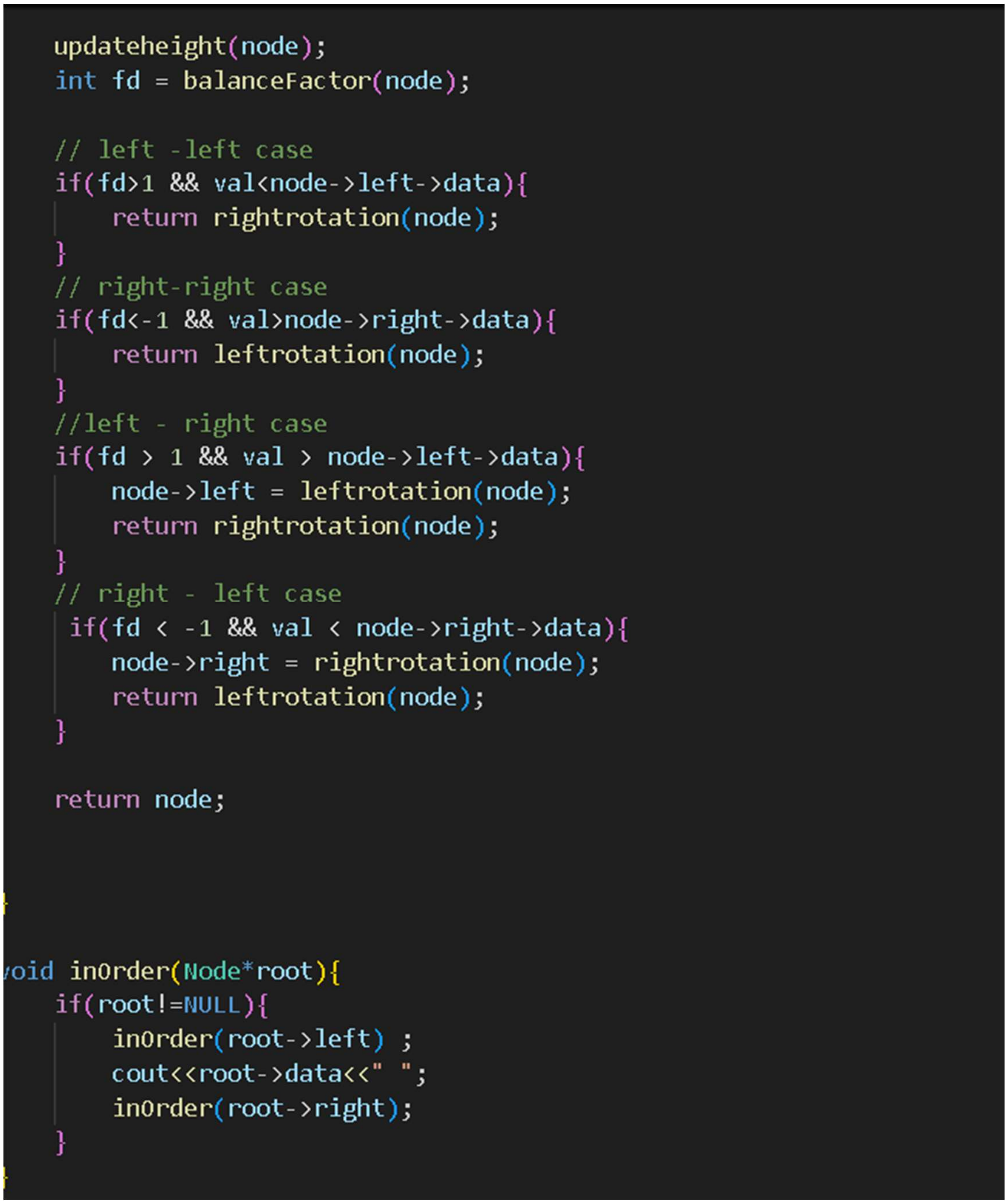
7 - Write a program to implement insert and search operation in AVL trees.



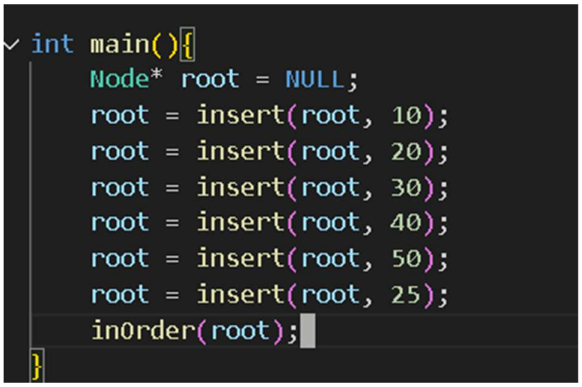
```cpp
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;

struct Node {
    int data;
    Node*left;
    Node*right;
    int height;

    Node(int val){
        data = val;
        left=right = NULL;
    }
};

int height(Node* N){
    if(N==NULL){
        return 0;
    }
    return N->height;
}

int updateheight(Node* N){
    if(N==NULL){
        return 0;
    }
    return max(N->left->height,N->right->height) + 1;
}

int balanceFactor(Node *N){
    if(N==NULL){
        return 0;
    }
    return (N->left->height - N->right->height) +1;
}
```



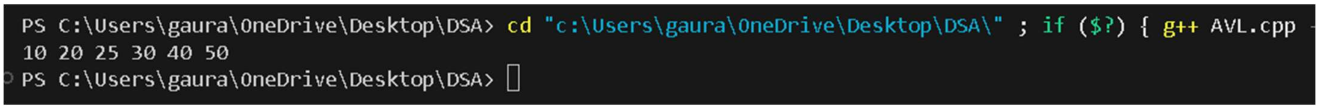
```
// left--left case
Node* rightrotation(Node* y){
    Node * x = y->left;
    Node*z = x->right;

    //rotate
    x->right = y;
    y->left = z;
    updateheight(y);
    updateheight(x);
    return x;
}
//right-right
Node* leftrotation(Node* x){
    Node * y = x->right;
    Node*z = y->left;

    //rotate
    y->left = x;
    x->right = z;
    updateheight(x);
    updateheight(y);
    return y;
}

Node* insert(Node* node, int val){
    if(node==NULL){
        return new Node(val);
    }

    if(val<node->data){
        node->left = insert(node->left,val);
    } else{
        node->right = insert(node->right,val);
    }
    return node;
```



```
    updateheight(node);
    int fd = balanceFactor(node);

    // left -left case
    if(fd>1 && val<node->left->data){
        return rightrotation(node);
    }
    // right-right case
    if(fd<-1 && val>node->right->data){
        return leftrotation(node);
    }
    //left - right case
    if(fd > 1 && val > node->left->data){
        node->left = leftrotation(node);
        return rightrotation(node);
    }
    // right - left case
     if(fd < -1 && val < node->right->data){
        node->right = rightrotation(node);
        return leftrotation(node);
    }

    return node;


}


void inOrder(Node*root){
    if(root!=NULL){
        inOrder(root->left) ;
        cout<<root->data<<" ";
        inOrder(root->right);
    }
}
```



```
int main(){
    Node* root = NULL;
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    root = insert(root, 40);
    root = insert(root, 50);
    root = insert(root, 25);
    inOrder(root);
}
```



```
PS C:\Users\gaura\OneDrive\Desktop\DSA> cd "c:\Users\gaura\OneDrive\Desktop\DSA\" ; if ($?) { g++ AVL.cpp -
10 20 25 30 40 50
PS C:\Users\gaura\OneDrive\Desktop\DSA> 
```