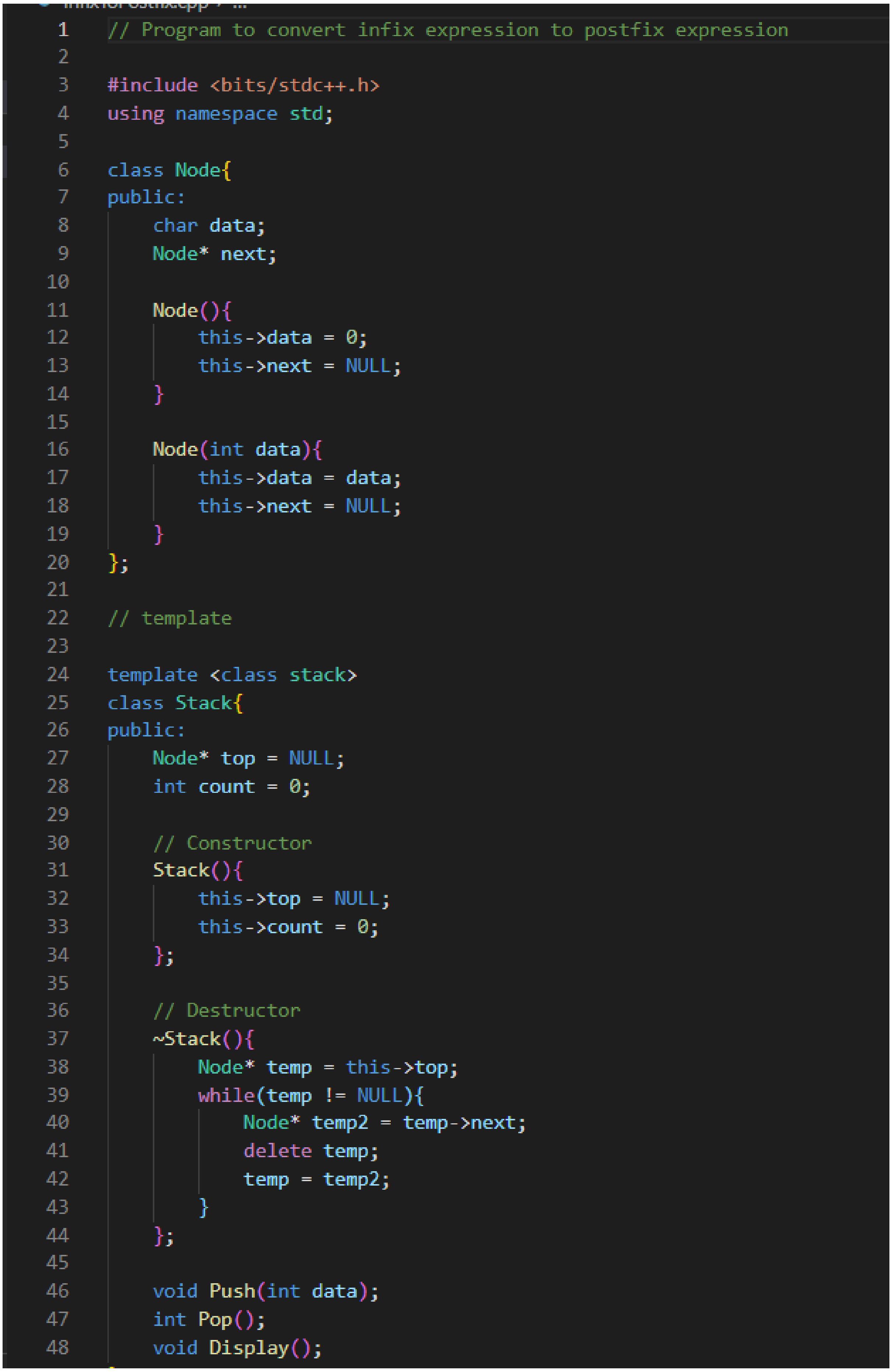
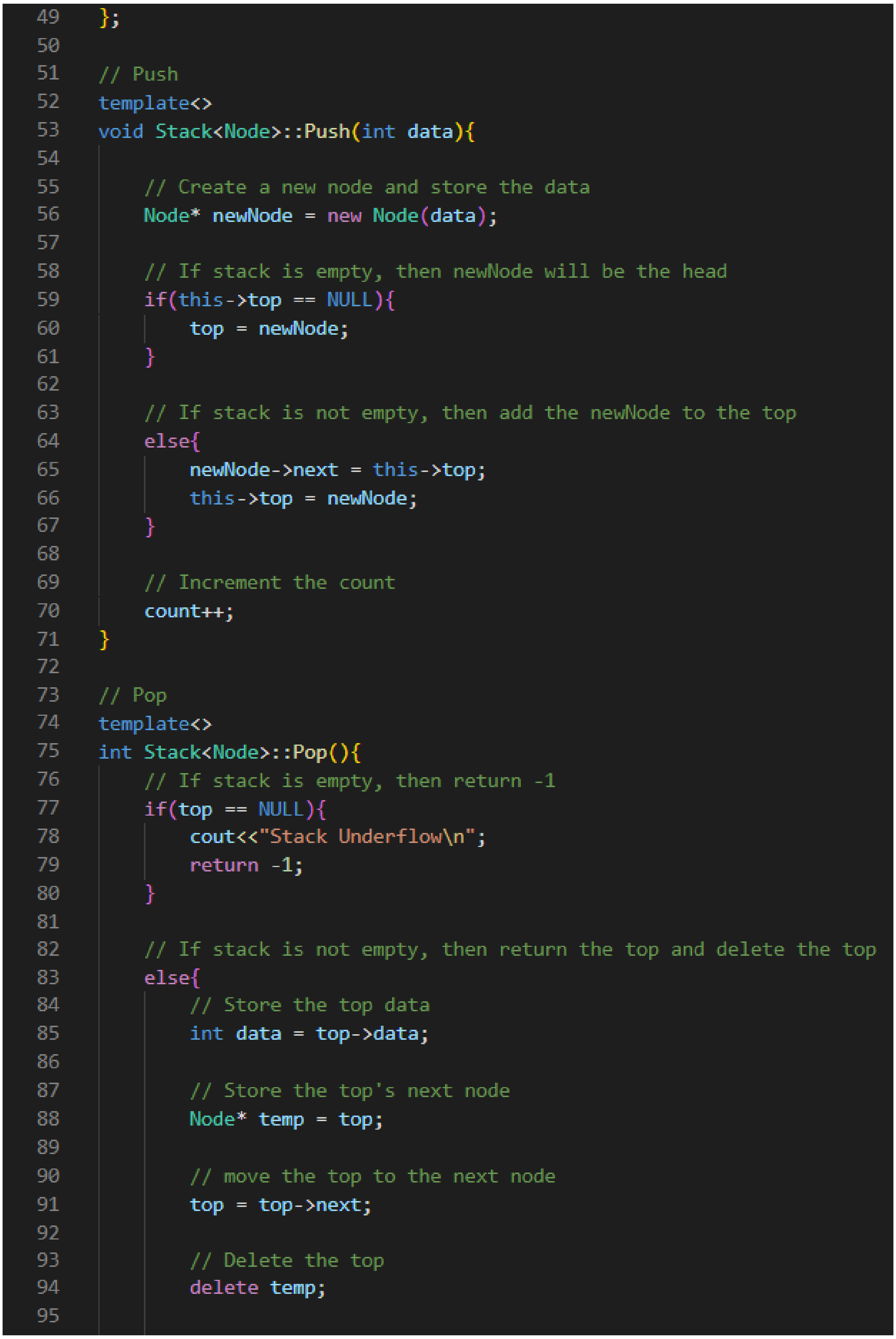
PRN: 2020BTEIT00041

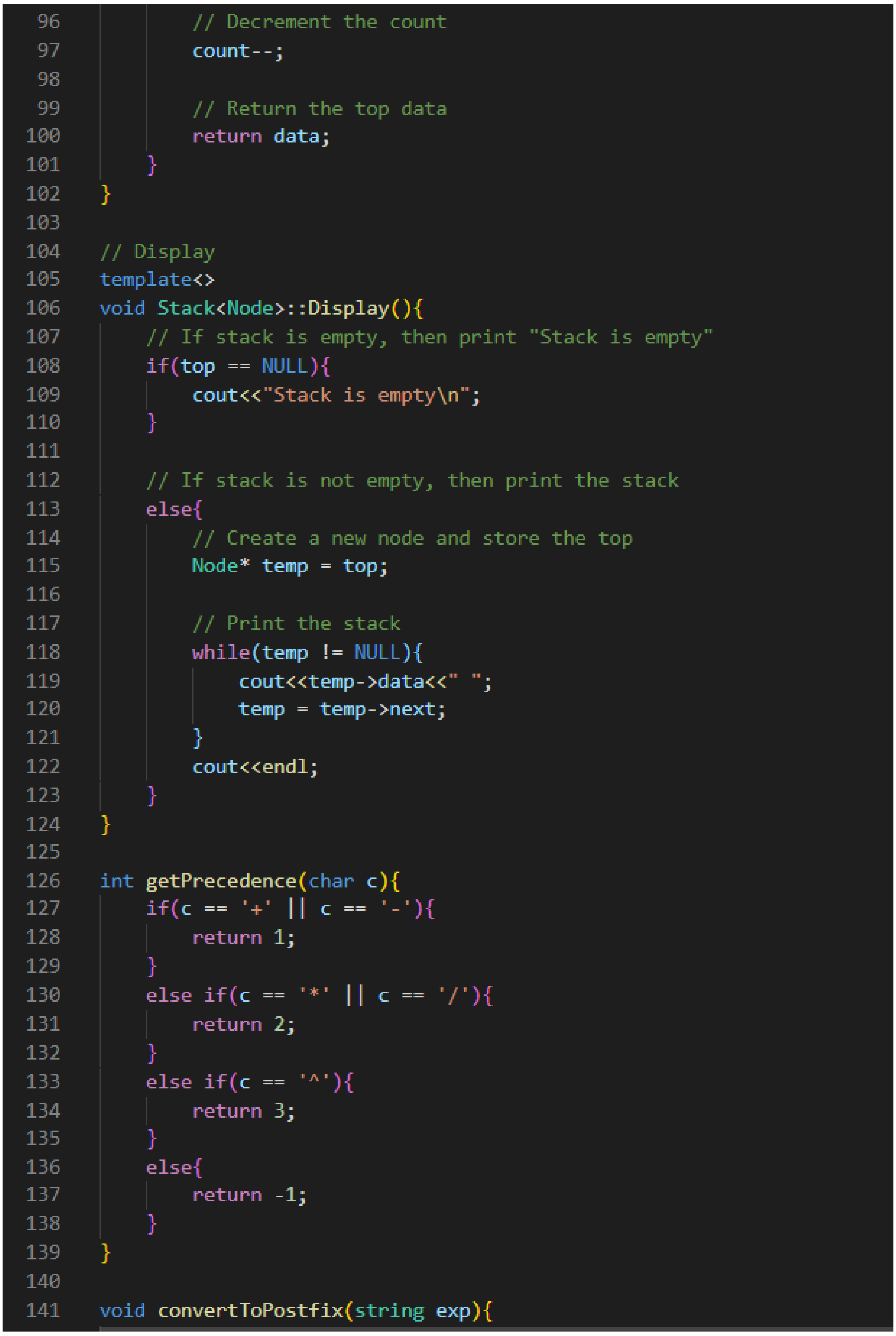
Infix to Postfix:



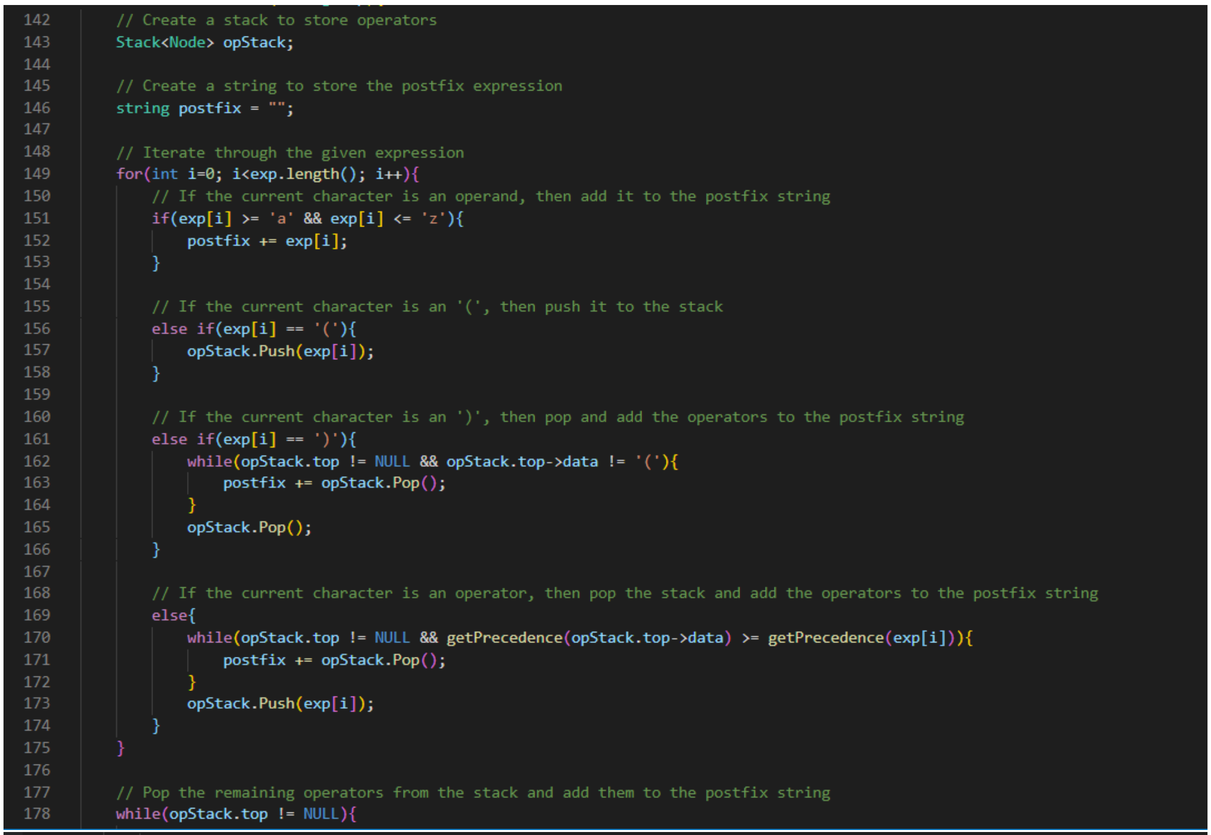
```cpp
// Program to convert infix expression to postfix expression

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

class Node{
public:
    char data;
    Node* next;

    Node(){
        this->data = 0;
        this->next = NULL;
    }

    Node(int data){
        this->data = data;
        this->next = NULL;
    }
};

// template

template <class stack>
class Stack{
public:
    Node* top = NULL;
    int count = 0;

    // Constructor
    Stack(){
        this->top = NULL;
        this->count = 0;
    };

    // Destructor
    ~Stack(){
        Node* temp = this->top;
        while(temp != NULL){
            Node* temp2 = temp->next;
            delete temp;
            temp = temp2;
        }
    };

    void Push(int data);
    int Pop();
    void Display();
```



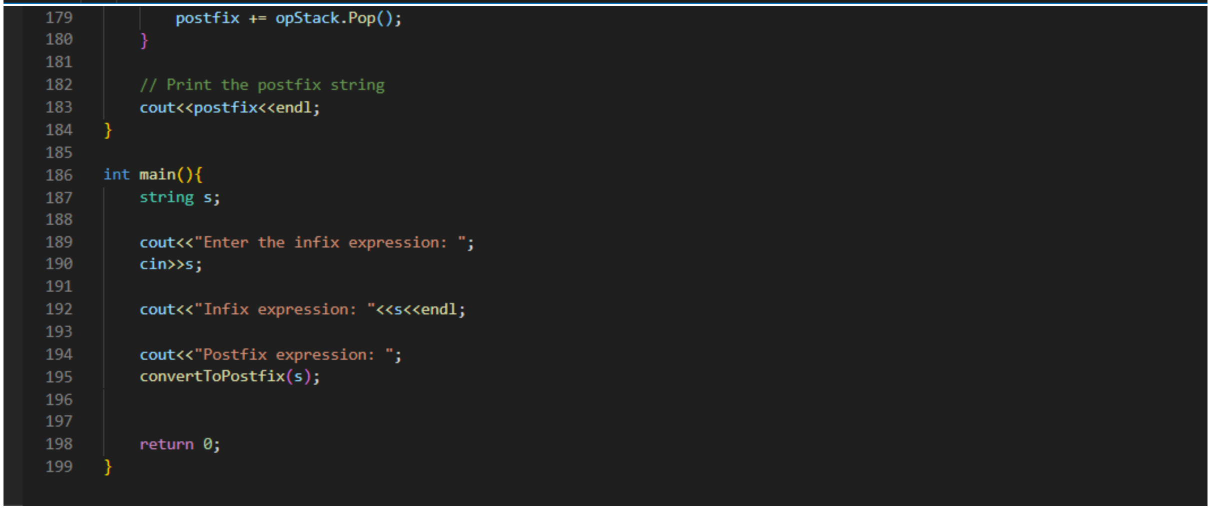
```cpp
};

// Push
template<>
void Stack<Node>::Push(int data){

    // Create a new node and store the data
    Node* newNode = new Node(data);

    // If stack is empty, then newNode will be the head
    if(this->top == NULL){
        top = newNode;
    }

    // If stack is not empty, then add the newNode to the top
    else{
        newNode->next = this->top;
        this->top = newNode;
    }

    // Increment the count
    count++;
}

// Pop
template<>
int Stack<Node>::Pop(){
    // If stack is empty, then return -1
    if(top == NULL){
        cout<<"Stack Underflow\n";
        return -1;
    }

    // If stack is not empty, then return the top and delete the top
    else{
        // Store the top data
        int data = top->data;

        // Store the top's next node
        Node* temp = top;

        // move the top to the next node
        top = top->next;

        // Delete the top
        delete temp;

```



```cpp
        // Decrement the count
        count--;

        // Return the top data
        return data;
    }
}

// Display
template<>
void Stack<Node>::Display(){
    // If stack is empty, then print "Stack is empty"
    if(top == NULL){
        cout<<"Stack is empty\n";
    }

    // If stack is not empty, then print the stack
    else{
        // Create a new node and store the top
        Node* temp = top;

        // Print the stack
        while(temp != NULL){
            cout<<temp->data<<" ";
            temp = temp->next;
        }
        cout<<endl;
    }
}

int getPrecedence(char c){
    if(c == '+' || c == '-'){
        return 1;
    }
    else if(c == '*' || c == '/'){
        return 2;
    }
    else if(c == '^'){
        return 3;
    }
    else{
        return -1;
    }
}

void convertToPostfix(string exp){
```

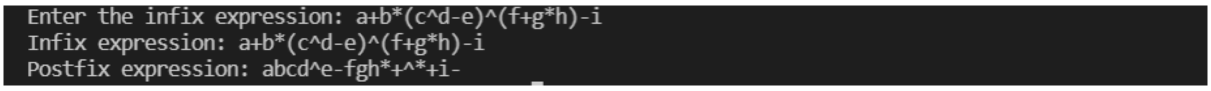


```cpp
    // Create a stack to store operators
    Stack<Node> opStack;

    // Create a string to store the postfix expression
    string postfix = "";

    // Iterate through the given expression
    for(int i=0; i<exp.length(); i++){
        // If the current character is an operand, then add it to the postfix string
        if(exp[i] >= 'a' && exp[i] <= 'z'){
            postfix += exp[i];
        }

        // If the current character is an '(', then push it to the stack
        else if(exp[i] == '('){
            opStack.Push(exp[i]);
        }

        // If the current character is an ')', then pop and add the operators to the postfix string
        else if(exp[i] == ')'){
            while(opStack.top != NULL && opStack.top->data != '('){
                postfix += opStack.Pop();
            }
            opStack.Pop();
        }

        // If the current character is an operator, then pop the stack and add the operators to the postfix string
        else{
            while(opStack.top != NULL && getPrecedence(opStack.top->data) >= getPrecedence(exp[i])){
                postfix += opStack.Pop();
            }
            opStack.Push(exp[i]);
        }
    }

    // Pop the remaining operators from the stack and add them to the postfix string
    while(opStack.top != NULL){
```



```cpp
        postfix += opStack.Pop();
    }

    // Print the postfix string
    cout<<postfix<<endl;
}

int main(){
    string s;

    cout<<"Enter the infix expression: ";
    cin>>s;

    cout<<"Infix expression: "<<s<<endl;

    cout<<"Postfix expression: ";
    convertToPostfix(s);


    return 0;
}
```

OUTPUT:



```
Enter the infix expression: a+b*(c^d-e)^(f+g*h)-i
Infix expression: a+b*(c^d-e)^(f+g*h)-i
Postfix expression: abcd^e-fgh*+^*+i-
```