Q. Construct an expression tree for postfix expression and perform recursive and non-recursive inorder, preorder and postorder traversals.

Header File

```
#ifndef TREE_H_
#define TREE_H_

template <class T>
class Stack;

template <class T>
class node {
    private:
    T info;
    node <T> *next;
    public:
    node();
    friend class Stack <T>;
};

template <class T>
class Stack{
    private:
    node <T> *top;
    public:
    bool isEmpty();
    void Push(T data);
    void Pop();
    void Display();
    T isTop();
    Stack();
    //void Traversing();
};
class Expression_Tree;

class Tree {
    private:
    char data;
    Tree *left, *right;
    public:
    Tree();
    friend class Expression_Tree;
};

#endif /* STACK_H_ */
```

Stack Implementation using Singly Linked List

```
//============================================================================
=======
// Name        : tree.cpp
// Author      : Gargi
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================================
=======

#include "tree.h"
#include <iostream>
using namespace std;
template <class T>

node <T> :: node(){
    next = NULL;
}

template <class T>
Stack <T> :: Stack(){
    top = NULL;
}

template <class T>
bool Stack<T> :: isEmpty(){
    if (top == NULL){
        return true;
    }
    else {
        return false;
    }
}

template <class T>
T Stack<T> :: isTop(){
    return top->info;
}

template <class T>
void Stack<T> :: Push(T data){
    node <T> *new1 = new node <T>;
    new1 ->info = data;
    if (top == NULL){
        new1 ->next = NULL;
        top = new1;
    }
```

```
    else {
        new1 ->next = top;
        top = new1;
    }
}

template <class T>
void Stack <T> :: Pop(){
    node <T> *new1 = new node <T>;
    isEmpty();
    new1 = top -> next;
    top -> next = NULL;
    delete (top);
    top = new1;
}

Tree :: Tree(){
    left = NULL;
    right = NULL;
}
```

Expression Tree

```
#include <iostream>
#include "tree.h"
#include "tree.cpp"
#include <ctype.h>
#include <stdio.h>
#include <algorithm>

using namespace std;
struct SS {
    Tree* nodes;
    bool bit;
};
class Expression_Tree{
    private:
    string postfix;
    Tree* root;
    public:
    void Input();
    Tree* createTree();
    void inorderRecursive(Tree *cnode);
    void postorderRecursive(Tree *cnode);
    void preorderRecursive(Tree *cnode);
    void inorder();
    void preorder();
    void postorder();
};
void Expression_Tree :: Input(){
    cout <<"Enter a Postfix Expression: ";
    cin >> postfix;
    cout <<"\n";
}

Tree* Expression_Tree ::createTree(){
    Stack <Tree*> treestack;
    for (int i=0; i<postfix.length();i++){
        Tree *tnode = new Tree;
        tnode->data = postfix[i];
        if(isalpha(postfix[i])){
            treestack.Push(tnode);
        }
        else if (postfix[i]=='+' || postfix[i]=='-' ||
postfix[i]=='*'||postfix[i]=='/'||postfix[i]=='^'){
            tnode ->right = treestack.isTop();
            treestack.Pop();
            tnode ->left = treestack.isTop();
            treestack.Pop();
```

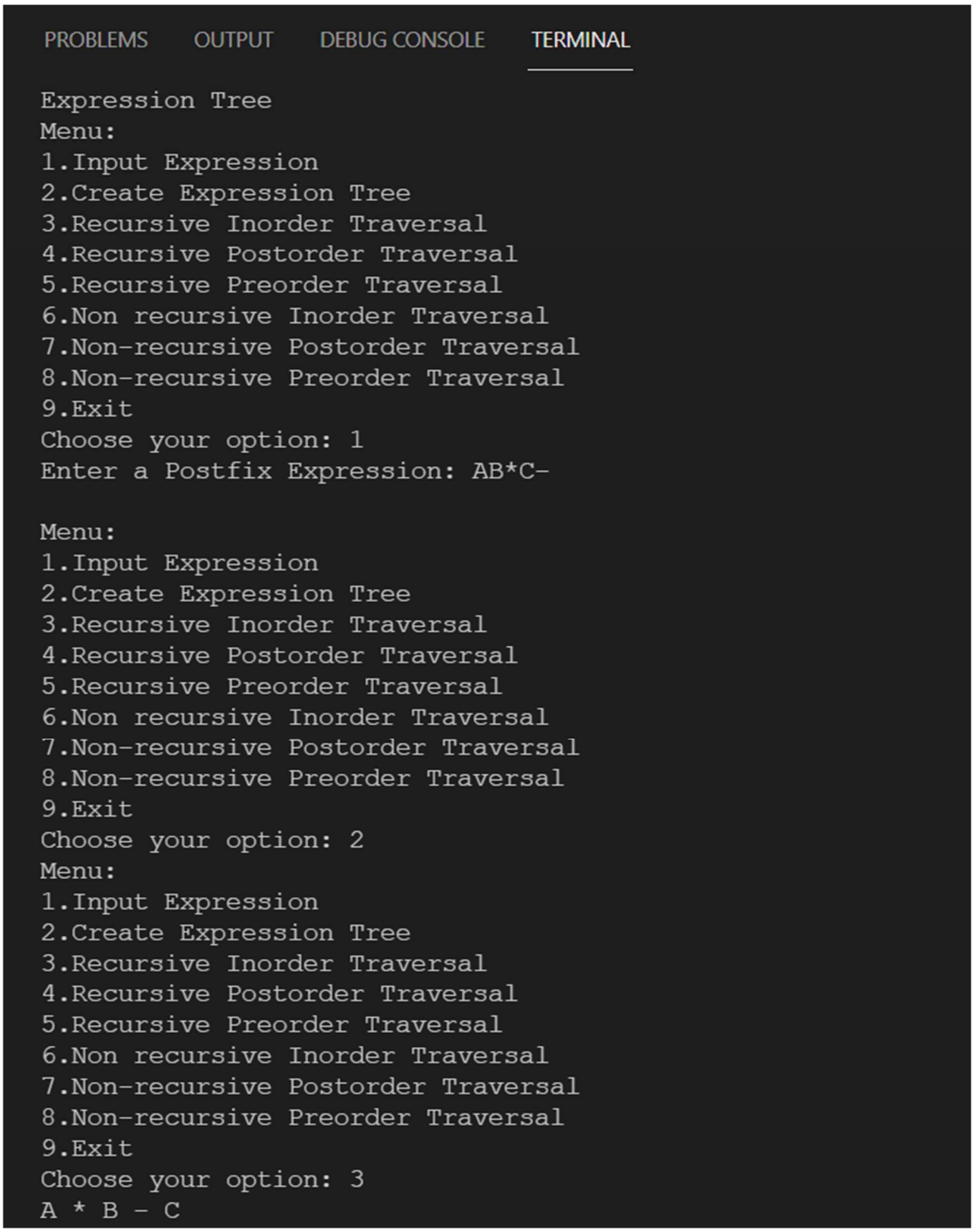
```
            treestack.Push(tnode);
        }
    }
    root = treestack.isTop();
    return treestack.isTop();
}
void Expression_Tree ::inorderRecursive(Tree *cnode){
    if (cnode){
        inorderRecursive(cnode ->left);
        cout << cnode ->data<<" ";
        inorderRecursive(cnode->right);
    }
}
void Expression_Tree ::preorderRecursive(Tree *cnode){
    if (cnode){
        cout << cnode ->data<<" ";
        preorderRecursive(cnode ->left);
        preorderRecursive(cnode->right);
    }
}
void Expression_Tree ::postorderRecursive(Tree *cnode){
    if (cnode){
        postorderRecursive(cnode ->left);
        postorderRecursive(cnode->right);
        cout << cnode ->data<<" ";
    }
}
void Expression_Tree ::inorder(){
    Stack <Tree*> traversal;
    Tree* current = root;
    while(current != NULL || traversal.isEmpty()== false){
        while (current != NULL){
            traversal.Push(current);
            current = current->left;
        }
        current = traversal.isTop();
        traversal.Pop();
        cout << current->data <<" ";
        current = current ->right;
    }
}

void Expression_Tree ::preorder(){
    Stack <Tree*> traversal;
    Tree* current = root;
    while(current != NULL || traversal.isEmpty()== false){
        while (current != NULL){
            cout << current->data <<" ";
```

```
            traversal.Push(current);
            current = current->left;
        }
        current = traversal.isTop();
        traversal.Pop();
        current = current ->right;
    }
}

void Expression_Tree ::postorder(){
    Stack <SS> traversal;
    SS element;
    element.nodes = root;
    element.bit = 0;
    while(element.nodes != NULL || traversal.isEmpty()== false){
        while (element.nodes != NULL){
            traversal.Push(element);
            if (element.nodes->right != NULL){
                SS r_element;
                r_element.bit = 1;
                r_element.nodes = element.nodes ->right;
                traversal.Push((r_element));
            }
            element.nodes = element.nodes->left;
        }
        element = traversal.isTop();
        //traversal.Pop();

        while (element.bit != 1){
            cout << element.nodes->data <<" ";
            //element = traversal.isTop();
            traversal.Pop();
            if (traversal.isEmpty()){
                return;
            }
            element = traversal.isTop();
        }
        if (element.bit == 1){
            element.bit = 0;
            traversal.Pop();
        }
    }
}

int main(){
    int ch;
    Expression_Tree exp;
    cout << "Expression Tree"<<endl;
```

```
    while(1){
        cout<<"Menu:\n1.Input Expression\n2.Create Expression
Tree\n3.Recursive Inorder Traversal\n4.Recursive Postorder
Traversal"<<endl;
        cout<<"5.Recursive Preorder Traversal\n6.Non recursive Inorder
Traversal\n7.Non-recursive Postorder Traversal\n8.Non-recursive
Preorder Traversal\n9.Exit"<<endl;
        cout <<"Choose your option: ";
        cin >> ch;
        switch (ch){
            case 1:
            exp.Input();
            break;

            case 2:
            exp.createTree();
            break;

            case 3:
            exp.inorderRecursive(exp.createTree());
            cout <<"\n";
            break;

            case 4:
            exp.postorderRecursive(exp.createTree());
            cout <<"\n";
            break;

            case 5:
            exp.preorderRecursive(exp.createTree());
            cout <<"\n";
            break;

            case 6:
            exp.inorder();
            cout <<"\n";
            break;

            case 7:
            exp.postorder();
            cout <<"\n";
            break;

            case 8:
            exp.preorder();
            cout <<"\n";
            break;
```

```
            case 9:
            cout <<"Now exiting the program..."<<endl;
            exit(0);
        }
    }

    return 0;
}
```

Sample Output



```
Menu:
1.Input Expression
2.Create Expression Tree
3.Recursive Inorder Traversal
4.Recursive Postorder Traversal
5.Recursive Preorder Traversal
6.Non recursive Inorder Traversal
7.Non-recursive Postorder Traversal
8.Non-recursive Preorder Traversal
9.Exit
Choose your option: 4
A B * C -
Menu:
1.Input Expression
2.Create Expression Tree
3.Recursive Inorder Traversal
4.Recursive Postorder Traversal
5.Recursive Preorder Traversal
6.Non recursive Inorder Traversal
7.Non-recursive Postorder Traversal
8.Non-recursive Preorder Traversal
9.Exit
Choose your option: 5
- * A B C
Menu:
1.Input Expression
2.Create Expression Tree
3.Recursive Inorder Traversal
4.Recursive Postorder Traversal
5.Recursive Preorder Traversal
6.Non recursive Inorder Traversal
7.Non-recursive Postorder Traversal
8.Non-recursive Preorder Traversal
9.Exit
Choose your option: 6
A * B - C
Menu:
```

```
Menu:
1.Input Expression
2.Create Expression Tree
3.Recursive Inorder Traversal
4.Recursive Postorder Traversal
5.Recursive Preorder Traversal
6.Non recursive Inorder Traversal
7.Non-recursive Postorder Traversal
8.Non-recursive Preorder Traversal
9.Exit
Choose your option: 7
A B * C -
Menu:
1.Input Expression
2.Create Expression Tree
3.Recursive Inorder Traversal
4.Recursive Postorder Traversal
5.Recursive Preorder Traversal
6.Non recursive Inorder Traversal
7.Non-recursive Postorder Traversal
8.Non-recursive Preorder Traversal
9.Exit
Choose your option: 8
- * A B C
Menu:
1.Input Expression
2.Create Expression Tree
3.Recursive Inorder Traversal
4.Recursive Postorder Traversal
5.Recursive Preorder Traversal
6.Non recursive Inorder Traversal
7.Non-recursive Postorder Traversal
8.Non-recursive Preorder Traversal
9.Exit
Choose your option: 1
Enter a Postfix Expression: ABCE^*+
```

```
Menu:
1.Input Expression
2.Create Expression Tree
3.Recursive Inorder Traversal
4.Recursive Postorder Traversal
5.Recursive Preorder Traversal
6.Non recursive Inorder Traversal
7.Non-recursive Postorder Traversal
8.Non-recursive Preorder Traversal
9.Exit
Choose your option: 2
Menu:
1.Input Expression
2.Create Expression Tree
3.Recursive Inorder Traversal
4.Recursive Postorder Traversal
5.Recursive Preorder Traversal
6.Non recursive Inorder Traversal
7.Non-recursive Postorder Traversal
8.Non-recursive Preorder Traversal
9.Exit
Choose your option: 3
A + B * C ^ E
Menu:
1.Input Expression
2.Create Expression Tree
3.Recursive Inorder Traversal
4.Recursive Postorder Traversal
5.Recursive Preorder Traversal
6.Non recursive Inorder Traversal
7.Non-recursive Postorder Traversal
8.Non-recursive Preorder Traversal
9.Exit
Choose your option: 4
A B C E ^ * +
Menu:
1.Input Expression
```

```
A B C E ^ * +
Menu:
1.Input Expression
2.Create Expression Tree
3.Recursive Inorder Traversal
4.Recursive Postorder Traversal
5.Recursive Preorder Traversal
6.Non recursive Inorder Traversal
7.Non-recursive Postorder Traversal
8.Non-recursive Preorder Traversal
9.Exit
Choose your option: 5
+ A * B ^ C E
Menu:
1.Input Expression
2.Create Expression Tree
3.Recursive Inorder Traversal
4.Recursive Postorder Traversal
5.Recursive Preorder Traversal
6.Non recursive Inorder Traversal
7.Non-recursive Postorder Traversal
8.Non-recursive Preorder Traversal
9.Exit
Choose your option: 6
A + B * C ^ E
Menu:
1.Input Expression
2.Create Expression Tree
3.Recursive Inorder Traversal
4.Recursive Postorder Traversal
5.Recursive Preorder Traversal
6.Non recursive Inorder Traversal
7.Non-recursive Postorder Traversal
8.Non-recursive Preorder Traversal
9.Exit
Choose your option: 7
A B C E ^ * +
```

```
Menu:
1.Input Expression
2.Create Expression Tree
3.Recursive Inorder Traversal
4.Recursive Postorder Traversal
5.Recursive Preorder Traversal
6.Non recursive Inorder Traversal
7.Non-recursive Postorder Traversal
8.Non-recursive Preorder Traversal
9.Exit
Choose your option: 8
+ A * B ^ C E
Menu:
1.Input Expression
2.Create Expression Tree
3.Recursive Inorder Traversal
4.Recursive Postorder Traversal
5.Recursive Preorder Traversal
6.Non recursive Inorder Traversal
7.Non-recursive Postorder Traversal
8.Non-recursive Preorder Traversal
9.Exit
Choose your option: 9
Now exiting the program...
PS C:\Users\suhas\OneDrive\Desktop\DSAL\.vscode\expression tree>
```