Construct an Expression Tree from postfix and prefix expression. Perform recursive and non- recursive In-order, pre-order and post-order traversals.

Code

```
#include<iostream>
using namespace std;
class node {
public:
	char value;
	node* left;
	node* right;
	node* next ;
	node(char c)
	{
		this->value = c;
		left = NULL;
		right = NULL;
	}
	node()
	{
		left = NULL;
		right = NULL;
	}
/*	friend class Stack;
	friend class expression_tree;
*/
};

// Class stack to hold
```

```
// tree nodes
class Stack {
	node* head ;

public:
	void push(node*);
	node* pop();
//	friend class expression_tree;
};

// Class to implement
// inorder traversal
class expression_tree {
public:
	// Function to implement
	// inorder traversal
	void inorder(node* x)
	{
		if (x == NULL)
			return;
		else {
			inorder(x->left);
			cout << x->value << " ";
			inorder(x->right);
		}
	}
	void postorder(node* x)
	{
		if (x == NULL)
			return;
```

```
            else {
                postorder(x->left);
                postorder(x->right);
                cout << x->value << " ";
            }
        }
        void preorder(node* x)
        {
            if (x == NULL)
                return;
            else {
                cout << x->value << " ";
                preorder(x->left);
                preorder(x->right);
            }
        }
};

// Function to push values
// onto the stack
void Stack::push(node* x)
{
        if (head == NULL) {
            head = x;
        }

        // We are inserting nodes at
        // the top of the stack [
        // following LIFO principle]
        else {
```

```
                x->next = head;
                head = x;
        }
}

// Function to implement pop
// operation in the stack
node* Stack::pop()
{
        // Popping out the top most[
        // pointed with head] element
        node* p = head;
        head = head->next;
        return p;
}
int  post()
{
        string s = "ABC*+D/";

        Stack e;
        expression_tree a;
        node *x, *y, *z;
        int l = s.length();

        for (int i = 0; i < l; i++) {
                // if read character is operator
                // then popping two other elements
                // from stack and making a binary
                // tree
                if (s[i] == '+' || s[i] == '-'
```

```
                || s[i] == '*' || s[i] == '/'
                || s[i] == '^') {
                z = new node(s[i]);
                x = e.pop();
                y = e.pop();
                z->left = y;
                z->right = x;
                e.push(z);
            }
            else {
                z = new node(s[i]);
                e.push(z);
            }
        }

        // RECURSIVE traversal
        cout << " The Inorder Traversal of Expression
Tree(POSTFIX-INORDER): ";
        a.inorder(z);
        cout<<endl<<endl;
        cout << " The postorder Traversal of Expression
Tree(POSTFIX-POSTORDER): ";
        a.postorder(z);
        cout<<endl<<endl;
        cout << " The preorder Traversal of Expression
Tree(POSTFIX-PREORDER): ";
        a.preorder(z);
        cout<<endl<<endl;
        return 0;
}
```

```
 int pre()
{
        // Prefix expression
    string s = "*+ABC";

    Stack e;
    expression_tree a;
    node *x, *y, *z;
    int l = s.length();

    for(int i = l-1; i>=0; i--) {
        // if read character is operator
        // then popping two other elements
        // from stack and making a binary
        // tree
        if(s[i] == '+' || s[i] == '-'|| s[i] == '*' || s[i] == '/'|| s[i] == '^') {
            z = new node(s[i]);
            x = e.pop();
            y = e.pop();
            z->left = y;
            z->right = x;
            e.push(z);
        }
        else {
            z = new node(s[i]);
            e.push(z);
        }
    }

    // RECURSIVE traversal
```

```
        cout << " The Inorder Traversal of Expression
Tree(PREFIX-INORDER): ";
        a.inorder(z);
        cout<<endl<<endl;
        cout << " The postorder Traversal of Expression
Tree(PREFIX-POSTORDER): ";
        a.postorder(z);
        cout<<endl<<endl;
        cout << " The preorder Traversal of Expression
Tree(PREFIX-PREORDER): ";
        a.preorder(z);
        cout<<endl<<endl;
        return 0;
}
int main()
{
post();
pre();
}
```

Output

```
The Inorder Traversal of Expression Tree(POSTFIX-INORDER): A + B * C / D


The postorder Traversal of Expression Tree(POSTFIX-POSTORDER): A B C * +
    D /

The preorder Traversal of Expression Tree(POSTFIX-PREORDER): / + A * B C
    D

The Inorder Traversal of Expression Tree(PREFIX-INORDER): C * B + A

The postorder Traversal of Expression Tree(PREFIX-POSTORDER): C B A + *

The preorder Traversal of Expression Tree(PREFIX-PREORDER): * C + B A
```