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

const int MAX = 50;

class node_cls
{
public:
    char data;
    node_cls *left, *right;

    node_cls()
    {
        left = right = NULL;
    }

    node_cls(char ch)
    {
        data = ch;
        left = right = NULL;
    }

    friend class tree_cls;
};

class Data_stek
{
private:
    int top;
    node_cls *info[MAX];

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

    void push(node_cls *cnode_cls)
    {
        top++;
        info[top] = cnode_cls;
    }

    node_cls *Top()
    {
        return info[top];
    }

    node_cls *pop()
    {
        if (!empty())
            return info[top--];
        return NULL;
    }

    bool empty()
    {
        return (top == -1);
    }
```

```
    bool isFull()
    {
        return (top == MAX - 1);
    }
};

class tree_cls
{
public:
    node_cls *root;

    tree_cls()
    {
        root = NULL;
    }

    void create(string str);
    void inorder_rec(node_cls *rnode_cls);
    void postorder_rec(node_cls *rnode_cls);
    void inorderNonRec();
    void postorder();
    void inorder();
    void deleteTree(node_cls *node);
    int priority(char ch);
};

void tree_cls::create(string str)
{
    Data_stek s1, s2;
    int i = 0;
    char ch;

    while (str[i] != '\0')
    {
        ch = str[i];
        if (isalpha(ch))
        { // is operand
            node_cls *temp = new node_cls(ch);
            s1.push(temp);
        }
        else
        { // operator
            if (s2.empty())
            {
                node_cls *temp = new node_cls(ch);
                s2.push(temp);
            }
            else if (priority(ch) > priority(s2.Top()->data))
            {
                node_cls *temp = new node_cls(ch);
                s2.push(temp);
            }
            else
            {
                while (!s2.empty() && priority(ch) <= priority(s2.Top()->data))
                {
                    node_cls *op = s2.pop();
                    node_cls *rchild = s1.pop();
                    node_cls *lchild = s1.pop();
                    op->right = rchild;
```

```
                    op->left = lchild;
                    sl.push(op);
                }
                s2.push(new node_cls(ch));
            }
        }
        i++;
    }

    while (!s2.empty())
    {
        node_cls *op = s2.pop();
        node_cls *rchild = sl.pop();
        node_cls *lchild = sl.pop();
        op->right = rchild;
        op->left = lchild;
        sl.push(op);
    }

    root = sl.pop();
}

int tree_cls::priority(char ch)
{
    switch (ch)
    {
    case '+':
    case '-':
        return 0;
    case '*':
    case '/':
        return 1;
    case '^':
        return 2;
    default:
        return -1;
    }
}

void tree_cls::inorderNonRec()
{
    node_cls *ptr = root;
    Data_stek sl;

    while (ptr != NULL || !sl.empty())
    {
        while (ptr != NULL)
        {
            sl.push(ptr);
            ptr = ptr->left;
        }
        ptr = sl.pop();
        cout << ptr->data << " ";
        ptr = ptr->right;
    }
}

void tree_cls::inorder_rec(node_cls *rnode_cls)
{
    if (rnode_cls)
    {
```

```
        inorder_rec(rnode_cls->left);
        cout << rnode_cls->data << " ";
        inorder_rec(rnode_cls->right);
    }
}

void tree_cls::postorder_rec(node_cls *rnode_cls)
{
    if (rnode_cls)
    {
        postorder_rec(rnode_cls->left);
        postorder_rec(rnode_cls->right);
        cout << rnode_cls->data << " ";
    }
}

void tree_cls::postorder()
{
    postorder_rec(root);
}

void tree_cls::inorder()
{
    inorder_rec(root);
}

void tree_cls::deleteTree(node_cls *node)
{
    if (node == NULL)
        return;
    deleteTree(node->left);
    deleteTree(node->right);
    cout << "\n Deleting node: " << node->data;
    delete node;
}

int main()
{
    tree_cls tl;
    string exp = "+--a*bc/def";
    cout << "Original Expression: " << exp << endl;

    tl.create(exp);

    cout << "\nInorder Traversal Recursive: ";
    tl.inorder();

    cout << "\nInorder Non-Recursive: ";
    tl.inorderNonRec();

    cout << "\nPostorder Traversal recursive: ";
    tl.postorder();

    cout << "\nDeleting Tree: ";
    tl.deleteTree(tl.root);

    cout << "\nEntire tree is deleted..";
    return 0;
}
```