

## Assignment 5

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

struct Node
{
    char data;
    Node *left, *right;
    Node(char val): data(val), left(nullptr), right(nullptr){}
};

class Tree
{
public:
    Node *root;
    Tree(): root(nullptr){}
    void buildExpressionTree(const char *prefix)
    {
        Node *stack[50];
        int top = -1;

        for(int i = strlen(prefix)-1;i>=0;i--)
        {
            if(isalpha(prefix[i]))
            {
                stack[++top] = new Node(prefix[i]);
            }
            else
            {
                Node *node = new Node(prefix[i]);
                node-> left = stack[top--];
                node-> right = stack[top--];
                stack[++top] = node;
            }
        }
        root = stack[top];
    }

    void displayPostfix(Node *node)
    {
        if(!node) return;
        displayPostfix(node -> left);
        displayPostfix(node -> right);
        cout<< node -> data;
    }

    void deleteTree(Node *node)
    {
        if(!node) return;
        deleteTree(node -> left);
        deleteTree(node -> right);
        cout<<"Deleting node : "<<node -> data<<endl;
        delete node;
    }
};
```

```

int main()
{
    Tree tree;
    char expression[50];
    int choice;

    do
    {
        cout<<"1 -> Enter prefix expression :\n";
        cout<<"2 -> Display prefix expression :\n";
        cout<<"3 -> Delete Tree :\n";
        cout<<"4 -> Exit\n";
        cout<<"Choose an option (1-4) :";
        cin>>choice;

        switch (choice)
        {
            case 1:
                cout<<"1 -> Enter the prefix expression (e.g., +--a*bc/def) :";
                cin>> expression;
                tree.buildExpressionTree(expression);
                break;
            case 2:
                if(tree.root)
                {
                    tree.displayPostfix(tree.root);
                    cout<< endl;
                }
                else
                {
                    cout<<"Tree is empty..\n";
                }
                break;
            case 3:
                if(tree.root)
                {
                    tree.deleteTree(tree.root);
                    tree.root = nullptr;
                }
                else
                {
                    cout<<"Tree is already empty !!\n";
                }
                break;
            case 4:
                cout<<"\n//END OF CODE\n";
                break;
            default:
                cout<<"Choose a valid option(1-4).\n";
        }
    }
    while(choice!=4);
    return 0;
}

```

```

1 #include<iostream>
2 #include<cstring>
3 #include<cctype>
4 using namespace std;
5
6 struct Node
7 {
8     char data;
9     Node *left, *right;
10    Node(char val): data(val), left(nullptr),
11};
12 class Tree
13 {
14 public:
15     Node *root;
16     Tree(): root(nullptr){}
17     void buildExpressionTree(const char *prefix)
18     {
19         Node *stack[50];
20         int top = -1;
21         for(int i = strlen(prefix)-1; i>=0; i--)
22         {
23             if(isalpha(prefix[i]))
24             {
25                 stack[++top] = new Node(prefix[i]);
26             }
27             else
28             {
29                 Node *node = new Node(prefix[i]);
30                 node->left = stack[top--];
31                 node->right = stack[top--];
32                 stack[++top] = node;
33             }
34         }
35         root = stack[top];
36     }
37
38     void displayPostfix(Node *node)
39     {
40         if(!node) return;
41         displayPostfix(node->left);
42         displayPostfix(node->right);
43         cout<<node->data<<endl;
44     }
45 };
46
47 int main()
48 {
49     Tree t;
50     char prefix[50];
51     cout<<"Enter prefix expression : ";
52     gets(prefix);
53     t.buildExpressionTree(prefix);
54     t.displayPostfix(t.root);
55     return 0;
56 }

```

```

student@student-OptiPlex-3010: ~/Desktop/Nikita
student@student-OptiPlex-3010:~/Desktop/Nikita$ ./a.out
1 -> Enter prefix expression :
2 -> Display prefix expression :
3 -> Delete Tree :
4 -> Exit
Choose an option (1-4) :1
1 -> Enter the prefix expression (e.g., +--a*bc/def) :+--a*bc/def
1 -> Enter prefix expression :
2 -> Display prefix expression :
3 -> Delete Tree :
4 -> Exit
Choose an option (1-4) :2
abc*-de/-f+
1 -> Enter prefix expression :
2 -> Display prefix expression :
3 -> Delete Tree :
4 -> Exit
Choose an option (1-4) :3
Deleting node :a
Deleting node :b
Deleting node :c
Deleting node :*
Deleting node :-
Deleting node :d
Deleting node :e
Deleting node :/
Deleting node :-
Deleting node :f
Deleting node :+
1 -> Enter prefix expression :
2 -> Display prefix expression :
3 -> Delete Tree :
4 -> Exit
Choose an option (1-4) :4

```