Name : AARYAN BAIARGI

Roll no: 27002

Class : SE IT-B

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

\*/

#include <iostream>

#include <stack>

#include <string.h>

using namespace std;

struct node

{

    char data;

    node \*leftchild;

    node \*rightchild;

}\*root;

    node \*store\_root\_address;

    node \*construct\_prefix\_tree(char prefix[]);

    node \*construct\_postfix\_tree(char posfix[]);

    void recursive\_nonrecursive\_choice();

    void expression\_choice();

    void recursive\_inorder(node\*root);

    void recursive\_preorder(node\*root);

    void recursive\_postorder(node\*root);

    void nonrecursive\_inorder(node \*root);

    void nonrecursive\_preorder(node \*root);

    void nonrecursive\_postorder(node \*root);

bool isOperator(char c)

{

    if (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^')

    {

        return true;

    }

    return false;

}

 node\* newNode(char v)

{

    node \*temp = new node;

    temp->leftchild = NULL;

    temp->rightchild = NULL;

    temp->data = v;

    return temp;

};

node\* construct\_postfix\_tree(char postfix[])

{

    stack<node \*> s;

    node \*root, \*t1, \*t2;

    for (int i=0; i<strlen(postfix); i++)

    {

        if (!isOperator(postfix[i]))

        {

            root=newNode(postfix[i]);

            s.push(root);

        }

        else

        {

            root= newNode(postfix[i]);

            t1 = s.top();

            s.pop();

            t2 = s.top();

            s.pop();

            root->rightchild = t1;

            root->leftchild = t2;

            s.push(root);

        }

    }

    root= s.top();

    s.pop();

    return root;

}

node\* construct\_prefix\_tree(char prefix[])

{

    stack<node \*> s;

    node \*root, \*t1, \*t2;

    for (int i=0; i<strlen(prefix); i++)

    {

        if (!isOperator(prefix[i]))

        {

            root= newNode(prefix[i]);

            s.push(root);

        }

        else

        {

            root= newNode(prefix[i]);

            t1 = s.top();

            s.pop();

            t2 = s.top();

            s.pop();

            root->leftchild = t1;

            root->rightchild = t2;

            s.push(root);

        }

    }

    root = s.top();

    s.pop();

    return root;

}

void recursive\_inorder(node\*root)

{

    node\*temp;

    temp=root;

    if(temp!=NULL)

    {

        recursive\_inorder(temp->leftchild);

        cout<<" "<<temp->data;

        recursive\_inorder(temp->rightchild);

    }

}

void recursive\_preorder(node\*root)

{

    node\*temp;

    temp=root;

    if(temp!=NULL)

    {

        cout<<" "<<temp->data;

        recursive\_preorder(temp->leftchild);

        recursive\_preorder(temp->rightchild);

    }

}

void recursive\_postorder(node\*root)

{

    node\*temp;

    temp=root;

    if(temp!=NULL)

    {

        recursive\_postorder(temp->leftchild);

        recursive\_postorder(temp->rightchild);

        cout<<" "<<temp->data;

    }

}

void nonrecursive\_inorder(node \*root)

{

    stack<node \*> s;

    node \*current\_node = root;

    while (current\_node != NULL || s.empty() == false)

    {

        while (current\_node!= NULL)

        {

            s.push(current\_node);

            current\_node=current\_node->leftchild;

        }

        current\_node= s.top();

        s.pop();

        cout<<current\_node->data;

        current\_node=current\_node->rightchild;

    }

}

void nonrecursive\_preorder(node \*root)

{

    stack<node \*>s;

    s.push(root);

    if (root == NULL)

    {

        return;

    }

    while (s.empty()==false)

    {

        node \*temp=s.top();

        s.pop();

        cout<<" "<<temp->data;

        if (temp->rightchild)

        {

            s.push(temp->rightchild);

        }

        if (temp->leftchild)

        {

            s.push(temp->leftchild);

        }

    }

}

void nonrecursive\_postorder(node \*root)

{

    if(root==NULL)

    {

        return;

    }

    stack<node \*>s1;

    stack<node \*> s2;

    s1.push(root);

    node \*temp;

    while(s1.empty()==false)

    {

        temp=s1.top();

        s1.pop();

        s2.push(temp);

        if(temp->leftchild)

        {

            s1.push(temp->leftchild);

        }

        if(temp->rightchild)

        {

            s1.push(temp->rightchild);

        }

    }

    while(!s2.empty())

    {

        temp=s2.top();

        s2.pop();

        cout<<" "<<temp->data;

    }

}

int main()

{

    int k =0;

    int choice, choice\_R, choice\_NR;

    node \*root\_address;

    char reverse[100],prefix[100],postfix[100];

    node \*construct\_prefix\_tree(char prefix[]);

    node \*construct\_postfix\_tree(char posfix[]);

    void recursive\_non\_recursive\_choice();

    void expression\_choice();

    void recursive\_inorder(node\*root);

    void recursive\_preorder(node\*root);

    void recursive\_postorder(node\*root);

    void nonrecursive\_inorder(node \*root);

    void nonrecursive\_preorder(node \*root);

    void nonrecursive\_postorder(node \*root);

        do

        {

        cout<<"\n";

        cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

        cout<<"\n From which expression you want to construct binary expression tree?\n";

        cout<<"\n 1) Prefix Expression. \n 2) Postfix Expression. \n 3) Exit.";

        cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

        cout<<"\n Enter Your Choice: ";

        cin>>choice;

        cout<<"...............................................";

            switch(choice)

            {

                case 1:

                   //ACCEPTING PREFIX EXPRESSION

                    cout<<"\nEnter Prefix Expression: ";

                    cin>>prefix;

                    for(int j=strlen(prefix)-1;j>=0;j--)

                    reverse[k++]=prefix[j];

                    reverse[k]='\0';

                    //cout<<reverse;

                    root\_address=construct\_prefix\_tree(reverse);

                    cout<<"\nIn which manner you want to traverse the binary expression tree?";

                    cout<<"1) Recursively. \n2) Non-Recursively. \n3) Exit.";

                    cout<<"Enter Your Choice:  ";

                    cin>>choice\_R;

                    switch(choice\_R)

                    {

                        case 1:

                                //RECURSIVE FUNCTIONS ON PREFIX

                                cout<<"\nRecursive preorder is: ";

                                recursive\_preorder(root\_address);

                                cout<<"\nRecursive inorder is: ";

                                recursive\_inorder(root\_address);

                                cout<<"\nRecursive postorder is: ";

                                recursive\_postorder(root\_address);

                                break;

                        case 2:

                                //NON RECURSIVE FUNCTIONS ON PREFIX

                                cout<<"\n\nNon-Recursive preorder is: ";

                                nonrecursive\_preorder(root\_address);

                                cout<<"\nNon-Recursive inorder is: ";

                                nonrecursive\_inorder(root\_address);

                                cout<<"\nNon-Recursive postorder is: ";

                                nonrecursive\_postorder(root\_address);

                                break;

                        case 3:

                                cout<<"You Have Successfully Exitted.....";

                                break;

                        default:

                                cout<<"INVALID CHOICE.....";

                    }

                        break;

                case 2:

                    cout<<"\nEnter Postfix Expression: ";

                    cin>>postfix;

                    root\_address=construct\_postfix\_tree(postfix);

                                       cout<<"\nIn which manner you want to traverse the binary expression tree?";

                    cout<<"1) Recursively. \n2) Non-Recursively. \n3) Exit.";

                    cout<<"\nEnter Your Choice:  ";

                    cin>>choice\_R;

                    switch(choice\_R)

                    {

                        case 1:

                                //RECURSIVE FUNCTIONS ON PREFIX

                                cout<<"\nRecursive preorder is: ";

                                recursive\_preorder(root\_address);

                                cout<<"\nRecursive inorder is: ";

                                recursive\_inorder(root\_address);

                                cout<<"\nRecursive postorder is: ";

                                recursive\_postorder(root\_address);

                                break;

                        case 2:

                                //NON RECURSIVE FUNCTIONS ON PREFIX

                                cout<<"\n\nNon-Recursive preorder is: ";

                                nonrecursive\_preorder(root\_address);

                                cout<<"\nNon-Recursive inorder is: ";

                                nonrecursive\_inorder(root\_address);

                                cout<<"\nNon-Recursive postorder is: ";

                                nonrecursive\_postorder(root\_address);

                                break;

                        case 3:

                                cout<<"You Have Successfully Exitted.....";

                                break;

                        default:

                                cout<<"INVALID CHOICE.....";

                    }

            }

        }

        while(choice!=3);

    return 0;

}