//============================================================================

// Name        : ExpressionTree.cpp

// Author      : Yash Sonar

// Version     :

// Copyright   :

// Description : Hello World in C++, Ansi-style

//============================================================================

#include<string>

#include <iostream>

using namespace std;

class node;

class tree;

const int MAX=50;

//===========Stack ==========================

class Stack

{

    int top;

    node \*info[MAX];

public:

    Stack()

{

        top=-1;

}

    void push(node \*cnode)

    {

        //  cout<<"here\n";

        //      if(!isFull())

        //      {

        //          cout<<"in push\n";

        top=top+1;

        info[top]=cnode;

        //      else

        //      {

        //          cout<<"\nStack Overflow\n";

        //      }

    }

    node \*Top()

    {

        return info[top];

    }

    node \* pop()

    {

        if(!empty())

        {

            return info[top--];

        }

        return NULL;

    }

    bool empty()

    {

        if(top==-1)

            return true;

        else

            return false;

    }

    bool isFull()

    {

        if(top==MAX-1)

            return true;

        else

            return false;

    }

};

//==================================================

class node

{

    node \*left,\*right;

    char data;

public:

    node()

{

        left=right=NULL;

}

    node(char ch)

    {

        left=right=NULL;

        data=ch;

    }

    friend class tree;

};

class tree

{

    node \*root;

public:

    tree()

{

        root=NULL;

}

    void create(string str);

    void inorder\_rec(node \*rnode);

    void preorder\_rec(node \*rnode);

    void postorder\_rec(node \*rnode);

    void inorderNonRec();

    void preorderNonRec();

    void postorderNon();

    void postorder()

    {

        postorder\_rec(root);

    }

    void inorder()

    {

        inorder\_rec(root);

    }

    void preorder()

    {

        preorder\_rec(root);

    }

    int priority(char ch);

};

int tree::priority(char ch)

{

    switch(ch)

    {

    case '+':

    case '-':

        return 0;

        break;

    case '\*':

    case '/':

        return 1;

        break;

    case '^':

        return 2;

        break;

    }

    return -1;

}

void tree::postorderNon()

{

    Stack s1;

    node \*ptr=root;

    int arr[MAX],i=-1,flag;

while(ptr!=NULL)

    {

        s1.push(ptr);

        arr[++i]=0;

        if(ptr->right!=NULL)

        {

            s1.push(ptr->right);

            arr[++i]=1;

        }

        ptr=ptr->left;

    }

while(!s1.empty())

{

    ptr=s1.pop();

    flag=arr[i--];

    while(flag==0&& i>=0)

    {

        cout<<" "<<ptr->data;

        ptr=s1.pop();

        flag=arr[i--];

    }

    if(flag==1 && i>=0)

    {

        while(ptr!=NULL)

    {

        s1.push(ptr);

        arr[++i]=0;

        if(ptr->right!=NULL)

        {

            s1.push(ptr->right);

            arr[++i]=1;

        }

        ptr=ptr->left;

    }

    }

}

cout<<" "<<ptr->data;

}

void tree::preorderNonRec()

{

    Stack s1;

    node \*ptr=root;

    while(ptr!=NULL)

    {

        cout<<" "<<ptr->data;

        if(ptr->right!=NULL)

            s1.push(ptr->right);

        if(ptr->left!=NULL)

            ptr=ptr->left;

        else

            ptr=s1.pop();

    }

}

void tree::inorderNonRec()

{

    node \*ptr=root;

    Stack s1;

    while(ptr!=NULL)

    {

        s1.push(ptr);

        ptr=ptr->left;

    }

    while(!s1.empty())

    {

    ptr=s1.pop();

    cout<<" "<<ptr->data;

    ptr=ptr->right;

    while(ptr!=NULL)

    {

        s1.push(ptr);

        ptr=ptr->left;

    }

}

}

void tree::inorder\_rec(node \*rnode)

{

    if(rnode)

    {

        inorder\_rec(rnode->left);

        cout<<" "<<rnode->data;

        inorder\_rec(rnode->right);

    }

}

void tree::preorder\_rec(node \*rnode)

{

    if(rnode)

    {

        cout<<" "<<rnode->data;

        preorder\_rec(rnode->left);

        preorder\_rec(rnode->right);

    }

}

void tree::postorder\_rec(node \*rnode)

{

    if(rnode)

    {

        postorder\_rec(rnode->left);

        postorder\_rec(rnode->right);

        cout<<" "<<rnode->data;

    }

}

void tree::create(string str)

{

    Stack s1,s2;

    int i=0;

    char ch;

    while(str[i]!='\0')

    {

        //cout<<"in create()";

        ch=str[i];

        if(isalpha(ch)) //s1 operand stack and s2===operator

        {

            node \*temp=new node(ch);

            s1.push(temp);

        }

        else //operator

        {

            //cout<<"in operator block()";

            if(s2.empty())

            {

                node \*temp=new node(ch);

                s2.push(temp);

            }

            else if(priority(ch)>priority(s2.Top()->data))

            {

                node \*temp=new node(ch);

                s2.push(temp);

            }

            else

            {

                while(!s2.empty()&&priority(ch)<=priority(s2.Top()->data) )

                {

                    node \*op=s2.pop();

                    node \*rchild=s1.pop();

                    node \*lchild=s1.pop();

                    op->right=rchild;

                    op->left=lchild;

                    s1.push(op); //push on operand stack

                }

                s2.push(new node(ch)); //push operand at last

            }

        }

        i++;

        //cout<<" i "<<i;

    }

    while(!s2.empty()) //pop() until operator stack is not empty

    {

        node \*op=s2.pop();

        node \*rchild=s1.pop();

        node \*lchild=s1.pop();

        op->right=rchild;

        op->left=lchild;

        s1.push(op);

    }

    //set the root element to s1->top()

    root=s1.pop();

}

int main() {

    cout << "" << endl; // prints

    tree t1;

    string exp="a-b\*c-d/e+f";

    cout<<"\nOriginal Expression: "<<exp;

    t1.create(exp);

    cout<<"\nInorder Traversal Recursive: ";

    t1.inorder();

    cout<<"\nInorder Non-Recursive: ";

    t1.inorderNonRec();

    cout<<"\nPreorder Traversal Recursive: ";

    t1.preorder();

    cout<<"\nPreorder traversal Non-Recursive: ";

    t1.preorderNonRec();

    cout<<"\nPostorder Traversal recursive: ";

    t1.postorder();

    cout<<"\nPostorder Non-Recursive: ";

    t1.postorderNon();

    return 0;

}