#include<iostream>

#include<ctype.h>

using namespace std;

struct node

{

char data;

node \*left;

node \*right;

};

class ExpressionTree

{

int top;

public:

node \*root,\*stack[20];

ExpressionTree()

{

top=-1;

root=NULL;

}

void construct(char exp[30]);

void inorder(node\*);

void preorder(node\*);

void postorder(node\*);

void non\_rec\_inorder(node\*);

void non\_rec\_preorder(node\*);

void non\_rec\_postorder(node\*);

};

void ExpressionTree::construct(char exp[30])

{

int i;

node \*newnode,\*l,\*r;

for(i=0;exp[i]!='\0';i++)

{

newnode=new node;

newnode->left=NULL;

newnode->right=NULL;

newnode->data=exp[i];

if(isalnum(exp[i]))

{

top++;

stack[top]=newnode;

}

else

{

r=stack[top];

top--;

l=stack[top];

top--;

newnode->left=l;

newnode->right=r;

top++;

stack[top]=newnode;

}

}

root=stack[top];

top--;

}

void ExpressionTree::inorder(node \*ptr)

{

if(ptr!=NULL)

{

inorder(ptr->left);

cout<<ptr->data<<" ";

inorder(ptr->right);

}

}

void ExpressionTree::preorder(node \*ptr)

{

if(ptr!=NULL)

{

cout<<ptr->data<<" ";

preorder(ptr->left);

preorder(ptr->right);

}

}

void ExpressionTree::postorder(node \*ptr)

{

if(ptr!=NULL)

{

postorder(ptr->left);

postorder(ptr->right);

cout<<ptr->data<<" ";

}

}

void ExpressionTree::non\_rec\_inorder(node \*ptr)

{

loop:

while(ptr!=NULL)

{

top++;

stack[top]=ptr;

ptr=ptr->left;

}

if(ptr==NULL && top!=-1)

{

cout<<stack[top]->data<<" ";

ptr=stack[top]->right;

top--;

goto loop;

}

}

void ExpressionTree::non\_rec\_preorder(node \*ptr)

{

node \*r,\*l,\*temp;

top++;

stack[top]=ptr;

while(top!=-1)

{

cout<<stack[top]->data<<" ";

temp=stack[top];

top--;

if(temp->right!=NULL)

{

r=temp->right;

top++;

stack[top]=r;

}

if(temp->left!=NULL)

{

l=temp->left;

top++;

stack[top]=l;

}

}

}

void ExpressionTree::non\_rec\_postorder(node \*ptr)

{

node \*s1[20],\*s2[20],\*temp;

int top1=-1;

int top2=-2;

top1++;

s1[top1]=ptr;

while(top1!=-1)

{

top2++;

s2[top2]=s1[top1];

temp=s1[top1];

top1--;

if(temp->left!=NULL)

{

top1++;

s1[top1]=temp->left;

}

if(temp->right!=NULL)

{

top1++;

s1[top1]=temp->right;

}

}

for(int i=top2;i>=0;i--)

{

cout<<s2[i]->data<<" ";

}

}

int main()

{

ExpressionTree e;

char exp[30];

int ch;

do

{

cout<<"\n 1. Construct an expression tree";

cout<<"\n 2. Recursive Traversal";

cout<<"\n 3. Non-recursive Traversal";

cout<<"\n 4. Exit";

cout<<"\n Enter your choice";

cin>>ch;

switch(ch)

{

case 1:

cout<<"\n enter postfix expression:";

cin>>exp;

e.construct(exp);

break;

case 2:

cout<<"\n Recursive Traversals:";

cout<<"\n inorder";

e.inorder(e.root);

cout<<"\n preorder";

e.preorder(e.root);

cout<<"\n postorder";

e.postorder(e.root);

break;

case 3:

cout<<"\n Non-Recursive Traversals:";

cout<<"\n inorder";

e.non\_rec\_inorder(e.root);

cout<<"\n preorder";

e.non\_rec\_preorder(e.root);

cout<<"\n postorder";

e.non\_rec\_postorder(e.root);

break;

}

}while(ch!=4);

return 0;

}

OUTPUT:

1. Construct an expression tree

2. Recursive Traversal

3. Non-recursive Traversal

4. Exit

Enter your choice1

enter postfix expression:a+b\*

1. Construct an expression tree

2. Recursive Traversal

3. Non-recursive Traversal

4. Exit

Enter your choice2

Recursive Traversals:

inorder+ a \* b

preorder\* + a b

postordera + b \*

1. Construct an expression tree

2. Recursive Traversal

3. Non-recursive Traversal

4. Exit

Enter your choice3

Non-Recursive Traversals:

inorder+ a

preorder\* + a b

postordera + b

1. Construct an expression tree

2. Recursive Traversal

3. Non-recursive Traversal

4. Exit

Enter your choice4