Advanced Data Structures Assignment

Name : Shrirang R Mhalgi

Roll number : 222006

Class : SE B

Batch : B1

--------------------------------------------------------------------------------------------------------------------------------------

**Problem Statement:**

For given expression eg. a-b\*c-d/e+f construct inorder sequence and traverse it using postorder traversal(non recursive).

**Code:**

#include<iostream>

#include<stack>

#include<string.h>

using namespace std;

class node

{

char data;

node \*left;

node \*right;

friend class expr;

};

class expr

{

stack<char>s;

stack<node \*>s1;

stack<char>s2;

node \*New,\*root,\*t;

int i,j,v;

char infix[50];

char postfix[50];

int size;

char \*type1,\*type2,op1,op2,result;

public:

void infixtopostfix();

int getweight(char ch);

void accept();

void display();

void create\_tree();

void postorder(node \*);

node \* ret\_r(){

return root;

}

};

void expr :: infixtopostfix()

{

int weight;

int i=0,k=0;

char ch;

while(i<size)

{

ch=infix[i];

if(ch=='(')

{

s.push(ch);

//top++;

i++;

continue;

}

if(ch==')')

{

while(!s.empty() && s.top()!='(')

{

postfix[k++]=s.top();

s.pop();

// top--;

}

if(!s.empty())

{

s.pop();

//top--;

}

i++;

continue;

}

weight=getweight(ch);

if(weight==0)

postfix[k++]=ch;

else

{

if(s.empty())

{

s.push(ch);

//top++;

}

else

{

while(!s.empty() && s.top()!='(' && weight<=getweight(s.top()))

{

postfix[k++]=s.top();

s.pop();

//top--;

}

s.push(ch);

//top++;

}

}

i++;

}

while(!s.empty())

{

postfix[k++]=s.top();

s.pop();

//top--;

}

postfix[k]=0;

}

int expr :: getweight(char ch)

{

switch(ch)

{

case '^' : return 3; break;

case '/' :

case '\*' : return 2;

break;

case '+' :

case '-' : return 1;

break;

default : return 0;

}

}

void expr :: accept()

{

cout<<"\nEnter the expression to be converted into postfix\n";

cin.getline(infix,50);

size=strlen(infix);

}

void expr :: display()

{

cout<<"\n\nPostfix expression is\n";

for(i=0;i<size;i++)

{

cout<<postfix[i];

}

cout<<endl<<endl;

}

void expr :: create\_tree()

{

node \*t1,\*t2;

int d;

for(i=0;i<size;i++)

{

d=postfix[i];

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

{

t1 = s1.top();

s1.pop();

t2 = s1.top();

s1.pop();

New = new node;

New->data=d;

New->left = t1;

New->right = t2;

if(i==size-1)

{

root = New;

}

s1.push(New);

}

else

{

New = new node;

New->right = NULL;

New->left = NULL;

New->data = postfix[i];

s1.push(New);

}

}

s1.pop();

//cout<<root->data;

}

void expr :: postorder(node \*root)

{

while(!s1.empty())

s1.pop();

s1.push(root);

while(!s1.empty()){

node \*curr=s1.top();

s1.pop();

s2.push(curr->data);

if(curr->left!=NULL)

s1.push(curr->left);

if(curr->right!=NULL)

s1.push(curr->right);

}

while(!s2.empty()){

cout<<s2.top();

s2.pop();

}

}

int main()

{

expr ob1;

ob1.accept();

ob1.infixtopostfix();

ob1.display();

ob1.create\_tree();

cout<<"Postorder traversal of tree\n";

ob1.postorder(ob1.ret\_r());

cout<<endl;

}

**Output:**

Enter the expression to be converted into postfix

a-b\*c-d/e+f

Postfix expression is

abc\*-de/-f+

Postorder traversal of tree

fed/cb\*a--+