# SSN College of Engineering, Kalavakkam

# Department of Computer Science and Engineering

# III Semester - CSE 'A ',’B’ & ‘C’

# UCS 1312 Data Structures Lab

# Academic Year: 2019-2020 Batch: 2018-2022

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Name : Rahul Ram M**

**Class : CSE – B**

**Register Number : 185001121**

**Exercise 7: Expression Tree**

**• Input : infix expression, Output: expression tree**

**• Use CharStack ADT to convert the infix expression into postfix expression**

**• Create TreeStack ADT to push and pop tree nodes (element, left ptr, right ptr)**

**• Use TreeStack ADT to convert the postfix expression into expression tree**

**• Display the tree using preorder, inorder and postorder traversals.**

**• Test for the following expressions**

**a. (2+5) \* (3-6) / (7\*8)**

**b. 7 - (((3+2) \* (6+1)) / (5+6))**

**c. ((3+2) \* (2+5))**

**//CharStack.h**

typedef struct mystack

{

char data;

struct mystack \*next;

}stack;

int testexp(char exp[]);

stack\* push(char ele,stack \*top);

char pop(stack\*\* top);

int ismatching(char c1,char c2);

char \*postfix(char exp[]);

int testexp(char exp[])

{

stack \*top;

top=NULL;

int i=0,s=1;

while(exp[i])

{

if(exp[i]=='('||exp[i]=='{'||exp[i]=='[')

{

top=push(exp[i],top);

}

else if(exp[i]=='}'||exp[i]==')'||exp[i]==']')

{

if(top!=NULL)

{

s=ismatching(pop(&top),exp[i]);

}

else

{

return 0;

}

if(s==0)

{

return 0;

}

}

i++;

}

if(top==NULL)

{

return 1;

}

else

{

return 0;

}

}

char \*postfix(char exp[])

{

stack \*top1;

top1=NULL;

char c;

char \*nexp=malloc(sizeof(char)\*50);

int i=0,n=0;

while(exp[i])

{

if(exp[i]=='('||exp[i]=='{'||exp[i]=='[')

{

top1=push(exp[i],top1);

}

else if(exp[i]=='+'||exp[i]=='-')

{

if(top1!=NULL)

{

if((top1->data)=='+'|| (top1->data)=='-' ||(top1->data)=='\*' ||(top1->data)=='/')

{

c=pop(&top1);

nexp[n]=c;

n++;

nexp[n]='\0';

top1=push(exp[i],top1);

}

else

{

top1=push(exp[i],top1);

}

}

else

{

top1=push(exp[i],top1);

}

}

else if(exp[i]=='\*' || exp[i]== '/')

{

if(top1!=NULL)

{

if(top1->data=='\*' || top1->data=='/')

{

c=pop(&top1);

nexp[n]=c;

n++;

nexp[n]='\0';

top1=push(exp[i],top1);

}

else

{

top1=push(exp[i],top1);

}

}

else

{

top1=push(exp[i],top1);

}

}

else if(exp[i]=='}'||exp[i]==')'||exp[i]==']')

{

c=pop(&top1);

nexp[n]=c;

n++;

nexp[n]='\0';

c=pop(&top1);

while(c!='{' && c!='(' && c!='[')

{

nexp[n]=c;

n++;

nexp[n]='\0';

c=pop(&top1);

}

}

else

{

nexp[n]=exp[i];

n++;

nexp[n]='\0';

}

i++;

}

while(top1!=NULL)

{

c=pop(&top1);

nexp[n]=c;

n++;

nexp[n]='\0';

}

return nexp;

}

stack\* push(char ele,stack \*top)

{

stack \*new;

new=(stack \*)malloc(sizeof(stack));

new->data=ele;

new->next=top;

top=new;

return top;

}

char pop(stack\*\* top)

{

char temp;

stack \*new;

new=(stack \*)malloc(sizeof(stack));

new=(\*top);

temp=new->data;

(\*top)=(\*top)->next;

free(new);

return temp;

}

int ismatching(char c1,char c2)

{

if(c1=='{'&&c2=='}')

return 1;

else if(c1=='('&&c2==')')

return 1;

else if(c1=='['&&c2==']')

return 1;

else

return 0;

}

**//TestStack.h**

typedef struct exptree

{

char value;

struct exptree \*left, \*right;

}et;

typedef struct Tree

{

et \*node;

struct tree \*next;

}tree;

tree \*top=NULL;

et\* constructTree(char postfix[]);

et\* newNode(char v);

int isOperator(char c);

et\* poptop();

void pushtop(et \*t);

void preorder(et\* t);

void inorder(et\* t);

void postorder(et\* t);

int isOperator(char c)

{

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

return 1;

return 0;

}

et\* newNode(char v)

{

et\* temp = (et\*)malloc(sizeof(et));

temp->left = temp->right = NULL;

temp->value = v;

return temp;

}

et\* poptop()

{

et \*ret=NULL;

if(top==NULL)

{

printf("Stack is Empty!");

}

else

{

tree \*temp=top;

top=temp->next;

ret=temp->node;

free(temp);

}

return ret;

}

void pushtop(et \*t)

{

tree \*temp=(tree \*)malloc(sizeof(tree));

temp->node=t;

if(top==NULL)

temp->next=NULL;

else

temp->next=top;

top=temp;

}

et\* constructTree(char postfix[])

{

et \*t, \*t1, \*t2;

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

{

if (!isOperator(postfix[i]))

{

t = newNode(postfix[i]);

pushtop(t);

}

else

{

t = newNode(postfix[i]);

t1 = poptop();

t2 = poptop();

t->right = t1;

t->left = t2;

pushtop(t);

}

}

t = poptop();

return t;

}

void preorder(et\* t)

{

if(t!=NULL)

{

printf("%c",t->value);

preorder(t->left);

preorder(t->right);

}

}

void inorder(et\* t)

{

if(t!=NULL)

{

inorder(t->left);

printf("%c",t->value);

inorder(t->right);

}

}

void postorder(et\* t)

{

if(t!=NULL)

{

postorder(t->left);

postorder(t->right);

printf("%c",t->value);

}

}

**//main**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include"CharStack.h"

#include"TestStack.h"

int main()

{

int ch=1,t;

char exp[50],postexp[50];

while(ch==1)

{

t=1;

printf("Enter the infix expression:");

scanf("%s",exp);

t=testexp(exp);

if(t==0)

{

printf("Unbalanced expression!\n");

}

else

{

printf("Balanced expression!\n");

strcpy(postexp,postfix(exp));

int j=0;

printf("Required postfix expression!\n");

while(postexp[j]!='\0')

{

printf("%c" ,postexp[j]);

j++;

}

printf("\n");

et\* t = constructTree(postexp);

printf("Preorder : ");

preorder(t);

printf("\nInorder : ");

inorder(t);

printf("\nPostorder: ");

postorder(t);

printf("\n");

}

printf("Continue(1.Yes 2.No):");

scanf("%d",&ch);

}

return 0;

}

**Sample I/P o/P:**

Enter the infix expression:(2+5)\*(3-6)/(7\*8)

Balanced expression!

Required postfix expression!

25+36-\*78\*/

Preorder : /\*+25-36\*78

Inorder : 2+5\*3-6/7\*8

Postorder: 25+36-\*78\*/

Continue(1.Yes 2.No):1

Enter the infix expression:7-(((3+2)\*(6+1))/(5+6))

Balanced expression!

Required postfix expression!

732+61+\*56+/-

Preorder : -7/\*+32+61+56

Inorder : 7-3+2\*6+1/5+6

Postorder: 732+61+\*56+/-

Continue(1.Yes 2.No):1

Enter the infix expression:((3+2)\*(2+5))

Balanced expression!

Required postfix expression!

32+25+\*

Preorder : \*+32+25

Inorder : 3+2\*2+5

Postorder: 32+25+\*

Continue(1.Yes 2.No):2