*SSN College of Engineering*  
*Department of Computer Science and Engineering*  
 *III Semester - CSE B*  
 *UCS 1312 Data Structures Lab Laboratory*

*Exercise 7: Expression Tree*

*Name : Prathyush .S*

*Roll no: 185001112*

*Class: CSE-B*

***PROGRAM:-***

#include"tree.h"

#include"stack.h"

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<ctype.h>

int main()

{

    char exp[20],\*post,fo[5];

    int r,x,i,j,f=0;

    float b,c;

    char ch;

    float o1,o2,s=0.0;

    j=0;

    post=(char \*)malloc(sizeof(char)\*20);

    printf("\nEnter the infix expression: \n");

    scanf("%s",exp);

    if(balanced(exp)==1)

    {

        for(i=0;i<strlen(exp);i++)

        {

            r=isop(exp[i]);

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

            {

                push(exp[i]);

            }

            else if(isalnum(exp[i]))

            {

                post[j]=exp[i];

                j++;

            }

            else if(r>0)

            {

                if(TOP!=NULL&&r==1&&(TOP->data=='%'||TOP->data=='\*'||TOP->data=='/'))

                {

                    b=pop();

                    post[j]=b;

                    j++;

                    push(exp[i]);

                }

                else if(TOP!=NULL&&r==2&&(TOP->data=='%'||TOP->data=='\*'||TOP->data=='/'))

                {

                    b=pop();

                    post[j]=b;

                    j++;

                    push(exp[i]);

                }

                else

                {

                    push(exp[i]);

                }

            }

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

            {

                while((exp[i]==')'&&TOP->data!='(')||(exp[i]==']'&&TOP->data!='[')||(exp[i]=='}'&&TOP->data!='{'))

                {

                    b=pop();

                    post[j]=b;

                    j++;

                }

                b=pop();

            }

            else

            {

                printf("Invalid symbol");

                f=1;

                break;

            }

        }

        while(f==0&&TOP!=NULL)

        {

            b=pop();

            post[j]=b;

            j++;

        }

        post[j]='\0';

    }

    printf("\nThe postfix expression is %s",post);

    et \*t=constructTree(post);

    printf("\npreorder display....");

    preorder(t);

    printf("\ninorder display....");

    inorder(t);

    printf("postorder display....");

    postorder(t);

    return 0;

}

File: stack.h

//stack functions

struct Node

{

float data;

struct Node \*next;

}\*TOP = NULL;

void push(float value)

{

struct Node \*newNode;

newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = value;

if(TOP == NULL)

newNode->next = NULL;

else

newNode->next = TOP;

TOP = newNode;

}

float pop()

{

if(TOP == NULL)

printf("\nStack is Empty!!!\n");

else

{

struct Node \*temp = TOP;

float r=temp->data;

TOP = temp->next;

free(temp);

return r;

}

}

int isop(char x)

{

if(x=='+'||x=='-')

return 1;

else if(x=='/'||x=='\*'||x=='%')

return 2;

else

return 0;

}

int balanced(char exp[20])

{

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

{

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

{

push(exp[i]);

}

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

{

if(TOP==NULL)

{

push(exp[i]);

}

else if((exp[i]==')'&&TOP->data=='(')||(exp[i]=='}'&&TOP->data=='{')||(exp[i]==']'&&TOP->data=='['))

{

pop();

}

else

continue;

}

}

if(TOP==NULL)

{

return 1;

}

else

{

TOP=NULL;

return 0;

}

}

File: tree.h

//tree funtions

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

//tree

typedef struct et

{

char data;

struct et \*left, \*right;

}et;

 //

typedef struct stack{

    et \*t;

    struct stack \*next;

}stack;

stack \*top=NULL;

void poptree(){

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

    s=top;

    top=s->next;

    free(s);

}

void pushtree(et \* p){

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

    s->t=p;

    s->next=top;

    top=s;

}

et \* givetop(){

    return top->t;

}

et\* newNode(int v)

{

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

    temp->left = temp->right = NULL;

    temp->data = v;

    return temp;

}

int isop(char x);

et\* constructTree(char postfix[])

{

    stack \* st;

    et \*t, \*t1, \*t2;

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

    {

        if (!isop(postfix[i]))

        {

            t = newNode(postfix[i]);

            pushtree(t);

        }

        else // operator

        {

            t = newNode(postfix[i]);

            t1 = givetop(); // Store top

            poptree(); // Remove top

            t2 = givetop();

            poptree();

            t->right = t1; // make them children

            t->left = t2;

            pushtree(t); // Add this subexpression to stack

        }

    }

    t = givetop();

    poptree();

    return t;

}

void preorder(et\* t)

{

    if(t!=NULL)

    {

    printf("%c",t->data);

    preorder(t->left);

    preorder(t->right);

    }

}

void inorder(et\* t)

{

    if(t!=NULL)

    {

        inorder(t->left);

        printf("%c",t->data);

        inorder(t->right);

    }

}

void postorder(et\* t)

{

    if(t!=NULL)

    {

        postorder(t->left);

        postorder(t->right);

        printf("%c",t->data);

    }

}

OUTPUT:

Enter the infix expression:

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

The postfix expression is 25+36-\*78\*/

preorder display..../\*+25-36\*78

inorder display....2+5\*3-6/7\*8postorder display....25+36-\*78\*/

Enter the infix expression:

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

The postfix expression is 32+25+\*

preorder display....\*+32+25

inorder display....3+2\*2+5postorder display....32+25+\*