**Aditya Krishna P**

**Registration number: 185001010**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

typedef struct{

int capacity;

int top;

char\* array;

}stack;

void push(char c,stack\* s);

char pop(stack\* s);

char top(stack\* s);

int isEmpty(stack\* s);

int isFull(stack\* s);

int bracketCheck(char exp[]);

int evaluatePostFix(char postfix[]);

void convertInToPost(char infix[],char postfix[]);

int isOperation(char c);

int getPriority(char op);

void push(char c,stack\* s)

{

if(isFull(s)==1)

{

printf("Stack full no elements can be pushed");

}

else

{

s->top=s->top+1;

s->array[s->top]=c;

}

}

char pop(stack \*s)

{

if(isEmpty(s)==1)

{

printf("No elements left to pop");

}

else

{

char pop=s->array[s->top];

s->top=s->top-1;

return pop;

}

}

char top(stack \*s)

{

return s->array[s->top];

}

int isEmpty(stack \*s)

{

if(s->top==-1)

return 1;

else

return 0;

}

int isFull(stack \*s)

{

if(s->top==s->capacity)

return 1;

else

return 0;

}

int bracketCheck(char exp[])

{

stack \*s1;

s1=(stack\*)malloc(sizeof(stack));

s1->top=-1;

s1->capacity=50;

s1->array=malloc(sizeof(char)\*s1->capacity);

char lbrackets[]={'[','{','('};

char rbrackets[]={']','}',')'};

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

{

for(int j=0;j<3;j++)

{

if(exp[i]==lbrackets[j])

{

push(exp[i],s1);

}

else if(exp[i]==rbrackets[j])

{

if(top(s1)==lbrackets[j])

pop(s1);

}

}

}

if(s1->top==-1)

return 1;

else

return 0;

}

int evaluatePostFix(char postfix[])

{

stack \*s1;

s1=(stack\*)malloc(sizeof(stack));

s1->top=-1;

s1->capacity=50;

s1->array=malloc(sizeof(char)\*s1->capacity);

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

{

if(isdigit(postfix[i]))

{

push(postfix[i],s1);

}

else

{

int rho,lho,ans;

rho=(int)pop(s1)-48;

lho=(int)pop(s1)-48;

switch(postfix[i])

{

case '+': ans=lho+rho;

break;

case '-': ans=lho-rho;

break;

case '\*': ans=lho\*rho;

break;

case '/': ans=lho/rho;

break;

case '%': ans=lho%rho;

break;

case '^': ans=lho^rho;

break;

}

ans=ans+48;

push(ans,s1);

}

}

return s1->array[s1->top]-48;

}

void convertInToPost(char infix[],char postfix[])

{

stack \*s1;

s1=(stack\*)malloc(sizeof(stack));

s1->top=-1;

s1->capacity=50;

s1->array=malloc(sizeof(char)\*s1->capacity);

int in=0,post=0;

strcpy(postfix,"");

while(infix[in]!='\0')

{

if(infix[in]=='(')

{

push(infix[in],s1);

in++;

}

else if(infix[in]==')')

{

while(s1->top!=-1 && top(s1)!='(')

{

postfix[post]=pop(s1);

post++;

}

if(s1->top==-1)

{

printf("INCORRECT EXP.\n");

exit(0);

}

char temp;

temp=pop(s1);

in++;

}

else if(isdigit(infix[in]) || isalpha(infix[in]))

{

postfix[post]=infix[in];

post++;

in++;

}

else if(isOperation(infix[in]))

{

while(s1->top!=-1 && top(s1)!='(' && getPriority(top(s1))>getPriority(infix[in]) )

{

postfix[post]=pop(s1);

post++;

}

push(infix[in],s1);

in++;

}

else

{

printf("INCORRECT ELEMENT IN EXP.\n");

exit(0);

}

}

while(s1->top!=-1 && top(s1)!='(')

{

postfix[post]=pop(s1);

post++;

}

postfix[post]='\0';

}

int isOperation(char c)

{

switch(c)

{

case '+':

case '-':

case '/':

case '\*':

case '%': return 1;

default: return 0;

}

}

int getPriority(char op)

{

switch(op)

{

case '/':

case '\*':

case '%':return 1;

case '+':

case '-':return 0;

}

}

#include"stack.h"

void main()

{

char exp[100],ans[100];

printf("Enter bracket expression: ");

scanf("%s",exp);

if(bracketCheck(exp))

{

convertInToPost(exp,ans);

printf("Postfix Expression: %s\n",ans);

printf("The value of expression from postfix: %d\n",evaluatePostFix(ans));

}

else

{

printf("Not properly balanced.\n");

}

}

Sample Output:

**C:\Users\Aditya\Desktop>a**

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

Postfix Expression: 25+36-78\*/\*

The value of expression from postfix: 0

**C:\Users\Aditya\Desktop>a**

Enter bracket expression: (2+3)

Postfix Expression: 23+

The value of expression from postfix: 5

**C:\Users\Aditya\Desktop>a**

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

no proper balancing of brackets.

**C:\Users\Aditya\Desktop>a**

Enter bracket expression: (((3+2)\*(2+5)

no proper balancing of brackets.