**SSN COLLEGE OF ENGINEERING**

**DEPARTMENT OF CSE**

**NAME:S.MADHUMITHA**

**ROLLNO:185001086**

**CODE:**

#define SIZE 50

#include<stdlib.h>

#include <ctype.h>

#include <stdio.h>

char s[SIZE];

int top = -1;

struct sNode

{

char data;

struct sNode \*next;

};

void RemoveSpaces(char\* source) {

char\* i = source;

char\* j = source;

while(\*j != 0) {

\*i = \*j++;

if(\*i != ' ')

i++;

}

\*i = 0;

}

void push(char elem) {

s[++top] = elem;

}

char pop() {

return (s[top--]);

}

int pr(char elem) {

switch (elem) {

case '#':

return 0;

case '(':

return 1;

case '+':

case '-':

return 2;

case '\*':

case '/':

return 3;

}

}

void infix\_to\_postfix(char \*infix, char \*postfix) {

char ch, elem;

int i = 0, k = 0;

RemoveSpaces(infix);

push('#');

while ((ch = infix[i++]) != '\n') {

if (ch == '(')

push(ch);

else if (isalnum(ch))

postfix[k++] = ch;

else if (ch == ')') {

while (s[top] != '(')

postfix[k++] = pop();

elem = pop();

} else {

while (pr(s[top]) >= pr(ch))

postfix[k++] = pop();

push(ch);

}

}

while (s[top] != '#')

postfix[k++] = pop();

postfix[k] = 0;

}

float eval\_postfix(char \*postfix) {

char ch;

float i = 0, op1, op2;

while((ch = postfix[i++]) != 0) {

if(isdigit(ch))

push(ch-'0');

else {

op2 = pop();

op1 = pop();

switch(ch) {

case '+' : push(op1+op2);

break;

case '-' : push(op1-op2);

break;

case '\*' : push(op1\*op2);

break;

case '/' : push(op1/op2);

break;

}

}

}

return s[top];

}

void push1(struct sNode\*\* top\_ref, int new\_data)

{

struct sNode\* new\_node =

(struct sNode\*) malloc(sizeof(struct sNode));

if (new\_node == NULL)

{

printf("Stack overflow n");

getchar();

exit(0);

}

new\_node->data = new\_data;

new\_node->next = (\*top\_ref);

(\*top\_ref) = new\_node;

}

int pop1(struct sNode\*\* top\_ref)

{

char res;

struct sNode \*top;

if (\*top\_ref == NULL)

{

printf("Stack overflow n");

getchar();

exit(0);

}

else

{

top = \*top\_ref;

res = top->data;

\*top\_ref = top->next;

free(top);

return res;

}

}

int isMatchingPair(char character1, char character2)

{

if (character1 == '(' && character2 == ')')

return 1;

else if (character1 == '{' && character2 == '}')

return 1;

else if (character1 == '[' && character2 == ']')

return 1;

else

return 0;

}

/\*Return 1 if expression has balanced Parenthesis \*/

int areParenthesisBalanced(char exp[])

{

int i = 0;

/\* Declare an empty character stack \*/

struct sNode \*stack = NULL;

/\* Traverse the given expression to check matching parenthesis \*/

while (exp[i])

{

/\*If the exp[i] is a starting parenthesis then push it\*/

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

push1(&stack, exp[i]);

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

{

if (stack == NULL)

return 0;

else if ( !isMatchingPair(pop1(&stack), exp[i]) )

return 0;

}

i++;

}

if (stack == NULL)

return 1; /\*balanced\*/

else

return 0; /\*not balanced\*/

}

int main() {

char infx[50], pofx[50];

int c;

printf("enter expression");

fgets(infx, 50, stdin);

if (areParenthesisBalanced(infx)) {

printf("the expression is balanced\n");

infix\_to\_postfix(infx, pofx);

printf("\nGiven Infix Expression: %sPostfix Expression: %s", infx, pofx);

top = -1;

printf("\nResult of evaluation of postfix expression : %f", eval\_postfix(pofx));

}

else

{

printf("the expression is not balanced\n");

exit(0);

}

return 0;

}

**OUTPUT:**

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

the expression is balanced

Given Infix Expression: (2+5)\*(3-6)/(7\*8)

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

Result of evaluation of postfix expression : -0.5357

enter expression7-(((3+2)\*(6+1)/(5+6) )/(5+6))

the expression is balanced

Given Infix Expression: 7-(((3+2)\*(6+1))/(5+6))

Postfix Expression: 732+61+\*56+/-

Result of evaluation of postfix expression : 4.000000

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

the expression is not balanced