//Write a program to evaluate infix expression using Stacks.

// header files

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

//Global variables

int numbers[50],tn=-1,to=-1;

char op[50];

//function to push digits

void push\_num(int n)

{

numbers[++tn]=n;

}

//function to push operators

void push\_op(char ch)

{

op[++to]=ch;

}

// function to pop digits

int pop\_num()

{

return numbers[tn--];

}

//function to pop operators

char pop\_op()

{

return op[to--];

}

//evaluating the expression

int infix\_eval(int numbers[50], char op[50])

{

int x, y;

char ope;

//taking first two operands

x=pop\_num();

y=pop\_num();

//taking the operators between them

ope=pop\_op();

//executing the operation

switch(ope)

{

case '+':

return x+y;

case '-':

return y-x;

case '\*':

return x\*y;

case '/':

if(x==0)

{

printf("\n cannot divide by 0");

exit(0);

}

else

return y/x;

}

return 0;

}

//Function to check whether the character is an operator or not

int is\_operator(char ch)

{

return (ch=='+'||ch=='-'||ch=='\*'||ch=='/'||ch=='^');

}

//the precedence of the operators

int precedence (char c)

{

switch (c)

{

case '+':

return 1;

case '-':

return 1;

case '\*':

return 2;

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

//function to evaluate an infix expression

int eval(char exp[20])

{

int i, num, output,r;

char c;

for (i=0;exp[i]!='\0';i++)

{

//taking each character from the expression

c = exp[i];

//checking if it is number

if(isdigit(c)!=0)

{

num = 0;

while (isdigit(c))

{

num = num\*10 + (c-'0');

i++;

if (i<strlen(exp))

c = exp[i];

else

break;

}

i--;

//pushing the number into stack of numbers

push\_num(num);

}

else if(c=='(')

{

//pushing the operators into the stack

push\_op(c);

}

//If we get close bracket, evaluate the entire brackets

else if(c==')')

{

while (op[to]!='(')

{

r = infix\_eval (numbers, op);

//pushing the result back to stack

push\_num(r);

}

pop\_op();

}

//if the current character is operator

else if (is\_operator(c))

{

//evaluating the expression

while (to!=-1 && precedence (c)<= precedence(op[to]))

{

output = infix\_eval(numbers,op);

//pushing the result back to stack

push\_num(output);

}

//pushing the current operator to stack

push\_op(c);

}

}

//if there is any remaining expression, evaluate them

while(to!=-1)

{

output = infix\_eval(numbers,op);

//pushing it back to stack

push\_num(output);

}

return pop\_num();

}

//Main function

int main()

{

char exp[50];

//taking the expression to evaluate

printf("Enter the infix expression to evaluate:");

gets(exp);

//calling the function and printing the result

printf("%d",eval(exp));

return 0;

}