#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <math.h>

#define max 100

void strrev(char \*str)

{

int i, j;

char temp;

for (i = 0, j = strlen(str) - 1; i < j; i++, j--)

{

temp = str[i];

str[i] = str[j];

str[j] = temp;

}

}

int whitespace(char ch)

{

if (ch == ' ' || ch == '\t' || ch == '\n')

return 1;

else

return 0;

}

int isOperator(char c)

{

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

return 1;

else

return 0;

}

int precedence(char c)

{

if (c == '^')

return 3;

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

return 2;

else if (c == '+' || c == '-')

return 1;

else

return 0;

}

int instack\_priority(char c)

{

if (c == '^')

return 4;

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

return 2;

else if (c == '+' || c == '-')

return 1;

else

return 0;

}

void push(char \*stack, int \*top, char c)

{

if (\*top == max - 1)

printf("Stack Overflow");

else

{

\*top = \*top + 1;

stack[\*top] = c;

}

}

char pop(char \*stack, int \*top)

{

char c;

if (\*top == -1)

printf("Stack Underflow");

else

{

c = stack[\*top];

\*top = \*top - 1;

}

return c;

}

char peek(char \*stack, int top)

{

if (top == -1)

return ' ';

else

return stack[top];

}

void infixToPostfix(char \*infix, char \*postfix)

{

char stack[max];

int top = -1, i = 0, j = 0;

char c;

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

{

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

{

push(stack, &top, infix[i]);

}

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

{

while ((c = pop(stack, &top)) != '(')

{

postfix[j] = c;

j++;

}

}

else if (isOperator(infix[i]) == 1)

{

while (precedence(infix[i]) <= precedence(peek(stack, top)) && top != -1)

{

postfix[j] = pop(stack, &top);

j++;

}

push(stack, &top, infix[i]);

}

else if (whitespace(infix[i]) == 0)

{

postfix[j] = infix[i];

j++;

}

}

while (top != -1)

{

postfix[j] = pop(stack, &top);

j++;

}

postfix[j] = '\0';

}

void infixToPrefix(char \*infix, char \*prefix)

{

char stack[max];

int top = -1, i = 0, j = 0;

char c;

for(i=strlen(infix)-1; i>=0; i--)

{

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

{

push(stack, &top, infix[i]);

}

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

{

while ((c = pop(stack, &top)) != ')')

{

prefix[j] = c;

j++;

}

}

else if (isOperator(infix[i]) == 1)

{

while (precedence(infix[i]) < instack\_priority(peek(stack, top)) && top != -1)

{

prefix[j] = pop(stack, &top);

j++;

}

push(stack, &top, infix[i]);

}

else if (whitespace(infix[i]) == 0)

{

prefix[j] = infix[i];

j++;

}

}

while (top != -1)

{

prefix[j] = pop(stack, &top);

j++;

}

prefix[j] = '\0';

strrev(prefix);

}

int operate(int a, int b, char c)

{

switch (c)

{

case '+':

return a + b;

case '-':

return b - a;

case '\*':

return a \* b;

case '/':

return b/a;

case '%':

return b%a;

case '^':

return pow(b,a);

default:

return 0;

}

}

int evaluatePostfix(char \*postfix)

{

long int a,b,temp,result;

int top = -1;

char stack[max];

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

{

if (isdigit(postfix[i]))

{

push(stack, &top, postfix[i] - '0');

}

else if (isOperator(postfix[i]) == 1)

{

a = pop(stack, &top);

b = pop(stack, &top);

temp = operate(a, b, postfix[i]);

push(stack, &top, temp);

}

}

result = pop(stack, &top);

return result;

}

int evaluatePrefix(char \*prefix)

{

long int a,b,temp,result;

int top = -1;

char stack[max];

for(int i=strlen(prefix)-1; i>=0; i--)

{

if(isdigit(prefix[i]))

{

push(stack, &top, prefix[i]-'0');

}

else if(isOperator(prefix[i])==1)

{

a=pop(stack, &top);

b=pop(stack, &top);

temp=operate(b, a, prefix[i]);

push(stack, &top, temp);

}

}

result=pop(stack, &top);

return result;

}

int main()

{

char infix[max], postfix[max], prefix[max];

int result;

//menu

while(1)

{

printf("\n1.Infix to Postfix");

printf("\n2.Infix to Prefix");

printf("\n3.Evaluate Postfix");

printf("\n4.Evaluate Prefix");

printf("\n5.Exit");

printf("\nEnter your choice: ");

int choice;

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("\nEnter infix expression: ");

scanf("%s", infix);

infixToPostfix(infix, postfix);

printf("\nPostfix expression: %s", postfix);

break;

case 2:

printf("\nEnter infix expression: ");

scanf("%s", infix);

infixToPrefix(infix, prefix);

printf("\nPrefix expression: %s", prefix);

break;

case 3:

printf("\nEnter postfix expression: ");

scanf("%s", postfix);

result = evaluatePostfix(postfix);

printf("\nResult: %d", result);

break;

case 4:

printf("\nEnter prefix expression: ");

scanf("%s", prefix);

result = evaluatePrefix(prefix);

printf("\nResult: %d", result);

break;

case 5:

exit(0);

default:

printf("\nInvalid choice");

}

}

return 0;

}