#include <iostream>

#include <stack>

#include <algorithm>

using namespace std;

bool isOperator(char c){

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

return true;

else

return false;

}

int precedence(char c){

if (c == '^' || c == '$')

return 3;

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

return 2;

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

return 1;

else

return -1;

}

// INFIX TO POSTFIX CONVERSION FUNCTION

string InfixToPostfix(stack<char> s, string infix ){

string postfix;

for (int i = 0; i < infix.length(); i++)

{

if ((infix[i] >= 'a' && infix[i] <= 'z') ||(infix[i] >= 'A' && infix[i] <='Z'))

{

postfix += infix[i];

}

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

s.push(infix[i]);

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

{

while ((s.top() != '(') && (!s.empty()))

{

postfix += s.top();

s.pop();

}

if(s.top() == '(')

s.pop();

}

else if(isOperator(infix[i]))

{

if(s.empty())

{

s.push(infix[i]);

}

else

{

if (precedence(infix[i]) == precedence(s.top()) && infix[i] =='^')

{

s.push(infix[i]);

}

else

{

while ((!s.empty()) && (precedence(infix[i]) <=

precedence(s.top())))

{

postfix+=s.top();

s.pop();

}

s.push(infix[i]);

}

}

}

}

while (!s.empty())

{

// cout << s.top() << endl;

postfix += s.top();

s.pop();

}

return postfix;

}

int main()

{

string infix, postfix;

cout << "Enter a Infix EXPRESSION :";

cin >> infix;

stack <char> stack;

postfix = InfixToPostfix(stack, infix);

cout << "\nPOSTFIX EXPRESSION :" << postfix << endl;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Infix to Prefix \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <stack>

#include <algorithm>

using namespace std;

int precedence(char c)

{

if (c == '^' || c == '$')

return 3;

if (c == '\*' || c == '-')

return 2;

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

return 1;

else

return -1;

}

// INFIX TO PREFIX CONVERSION FUNCTION

string InfixToPrefix(stack<char> s, string infix)

{

string prefix;

reverse(infix.begin(), infix.end());

for (int i = 0; i < infix.length(); i++)

{

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

infix[i] = ')';

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

infix[i] = '(';

}

for (int i = 0; i < infix.length(); i++)

{

if ((infix[i] >= 'a' && infix[i] <= 'z') || (infix[i] >= 'A' && infix[i]<= 'Z'))

prefix += infix[i];

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

s.push(infix[i]);

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

{

while (s.top() != '(' && (!s.empty()))

{

prefix += s.top();

s.pop(); //remove top of the stack

}

if (s.top() == '(')

{

s.pop();

}

}

else if (infix[i] == '+' || infix[i] == '-' || infix[i] == '\*' || infix[i] =='/' || infix[i] == '^')

{

if (s.empty())

s.push(infix[i]);

else

{

if (precedence(infix[i]) > precedence(s.top()))

s.push(infix[i]);

else if ((precedence(infix[i]) == precedence(s.top())) && (infix[i] == '^'))

{

while ((s.empty() != true) && (precedence(infix[i]) ==

precedence(s.top())) && (infix[i] == '^'))

{

if (s.empty() != true)

{

prefix += s.top();

s.pop();

}

}

s.push(infix[i]);

}

else if (precedence(infix[i]) == precedence(s.top()))

{

s.push(infix[i]);

}

else

{

while ((!s.empty()) && (precedence(infix[i]) < precedence(s.top())))

{

prefix += s.top();

s.pop();

}

s.push(infix[i]);

}

}

}

}

while (!s.empty())

{

prefix += s.top();

s.pop();

}

reverse(prefix.begin(), prefix.end());

return prefix;

}

int main()

{

string infix, prefix;

cout << "Enter a Infix EXPRESSION :" ;

cin >> infix;

stack<char> stack; //creation of stack

prefix = InfixToPrefix(stack, infix);

cout << "\nPREFIX EXPRESSION :" << prefix << endl;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Evaluation of Postfix Expression\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <stack>

#include <algorithm>

#include<math.h>

using namespace std;

bool isOperator(char c)

{

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

return true;

return false;

}

int postfixToInfix(string postfix)

{

stack <int> s;

int ans = 0;

for (int i = 0; i < postfix.length(); i++)

{

if (((postfix[i] - '0')>= 0 && (postfix[i] - '0') <= 9))

s.push(postfix[i] - '0');

else if (isOperator(postfix[i]))

{

int op1= s.top();

s.pop();

int op2 = s.top();

s.pop();

switch (postfix[i])

{

case '+' :

ans = op2 + op1;

break;

case '-':

ans = op2 - op1;

break;

case '\*':

ans = op2 \* op1;

break;

case '/':

ans = op2 / op1 ;

break;

case '^':

ans = int(pow(op2,op1));

break;

default:

break;

}

s.push(ans);

}

}

return s.top();

}

int main()

{

string postfix;

int infix;

cout << "Enter a POSTFIX Expression : ";

cin >> postfix;

infix = postfixToInfix(postfix);

cout << "\n Your Evaluated POSTFIX EXPRESSION is : " << infix << endl;

return 0;

}

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Evaluation of Prefix Expression\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <stack>

#include <algorithm>

#include<math.h>

using namespace std;

bool isOperator(char c)

{

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

return true;

return false;

}

int postfixToInfix(string prefix)

{

stack <int> s;

int ans = 0;

for (int i = prefix.length(); i >= 0; i--)

{

if (((prefix[i] - '0')>= 0 && (prefix[i] - '0') <= 9))

s.push(prefix[i] - '0');

else if (isOperator(prefix[i]))

{

int op1= s.top();

s.pop();

int op2 = s.top();

s.pop();

switch (prefix[i])

{

case '+' :

ans = op1 + op2;

break;

case '-':

ans = op1 - op2;

break;

case '\*':

ans = op1 \* op2;

break;

case '/':

ans = op1 / op2 ;

break;

case '^':

ans = int(pow(op1,op2));

break;

default:

break;

}

s.push(ans);

}

}

return s.top();

}

int main()

{

string prefix;

int infix;

cout << "Enter a PREFIX Expression : ";

cin >> prefix;

infix = postfixToInfix(prefix);

cout << "Your Evaluated PREFIX EXPRESSION is : " << infix << endl;

return 0;

}

PREFIX EXPRESSION :+AB

--------------------------------

Process exited after 3.252 seconds with return value 0

Press any key to continue . . .

Enter a POSTFIX Expression : 56+

Your Evaluated POSTFIX EXPRESSION is : 11

--------------------------------

Process exited after 4.477 seconds with return value 0

Press any key to continue . . .

Enter a PREFIX Expression : +56

Your Evaluated PREFIX EXPRESSION is : 11

--------------------------------

Process exited after 4.396 seconds with return value 0

Press any key to continue . . .

\*/