HW5_P3 - Convert infix to postfix

Student name :	Carter Hawks
Student email :	ckh170000@utdallas.edu
Class name :	2336.001_F18
Submitted on :	Nov 05, 2018 02:17 pm

solution.cpp

```
#include <iostream>
#include <stack>
#include <string>
#include <sstream>
#include <vector>
#include <iterator>
using namespace std;
/*
Analysis
Given an input string expression, convert it from mathematical infix format to postfix using a stack.

Design
We start by dividing the input string expression up using a space delimiter and placing it into a vector.
This allows us to separate out the pieces of the expression into usable pieces.
Then, we iterate through each element of the expression vector, doing something different based on what type it
If the element is an operand, we append it it to the output string.
If the element is a left parenthesis, we push it to the operator stack.
If the element is a right parenthesis, we pop the stack and append to outputuntil we reach the corresponding le
It can be assumed that there is a left parenthesis somewhere on the stack because we are currently working on t
Once we have popped all elements and pushed to stack, we pop the left parenthesis, but do not add to the stack
If the element is an operator, we pop all operators from the operator stack with a lower precedence to the outp
push the current operator to the stack.
Once we have iterated through every element, we finish by popping and appending all of the remaining operators
At this point, we are done.

*/
class InfixToPostfix{
    public:
       string infixToPostfix(string expression);
};

int precedence(string s){
        if(s == "*" || s == "/"){
        return 3;
    } else if(s == "+" || s == "-"){
        return 1;
    } else {
        return -1;
    }
}

bool isOperator(string s){
    return s == "+" || s == "-" || s == "*" || s == "/" || s == "(" || s == ")";
}

string InfixToPostfix :: infixToPostfix(string expression){
    //write your code here

    string output = "";

    // divide string expression by space delimiter
    std::istringstream iss(expression);
    vector<string> results((istream_iterator<string>(iss)), istream_iterator<string>());

    stack<string> opStack;
    for(int i = 0; i < results.size(); i++){
        //cout << results[i] << endl;
        if(isOperator(results[i])){
            if(results[i] == "("){
                // left parenthesis
                opStack.push("(");
            } else if(results[i] == ")"){

                while(opStack.top() != "("){
                        string s = opStack.top();
                                opStack.pop();
                                output = output + " " + s;
                }
                opStack.pop();

            } else {
                // operator * / + -
                int opPrec = precedence(results[i]);
                while(!opStack.empty()){
                        if(precedence(opStack.top()) > opPrec && opStack.top() != "("){
                                string top = opStack.top();
                                        output = output + " " + top;
                                        opStack.pop();
                        } else {
                                break;
                        }
                }
                opStack.push(results[i]);
            }
            //cout << output << endl;
        } else {
            // don't add trailing space at the end of the output string
            if(i != 0){
                output = output + " ";
            }
            output = output + results[i];
        }
    }

    // at end of output, pop remaining elements to output
    while(opStack.size() > 0){
        string s = opStack.top();
        output = output + " " + s;
        opStack.pop();
    }

    return output;

}
//Your program will be evaluated by this main method and several test cases.

int main(){
    string input;
        getline(cin, input);
        InfixToPostfix postfix;
        try {
                cout << postfix.infixToPostfix(input) << endl;
        } catch (exception ex) {
                cout << "Wrong expression" << endl;
        }
}
```

Name

Custom test case

Input

20 * (10 + 30)

Output (Lines:1)

20 10 30 + *

Expected Output (Lines:0)

Status
NA

Name
Custom test case

Input
(13 + 25) * 34

Output (Lines:1)
```
 13 25 + 34 *
```

Expected Output (Lines:0)

Status
NA

Name
Custom test case

Input
55 / (5 + 3) - 4

Output (Lines:1)
```
55 5 3 + / 4 -
```

Expected Output (Lines:0)

Status
NA

Name
Default1

Input
(1 + 2) * 3

Output (Lines:1)
```
 1 2 + 3 *
```

Expected Output (Lines:1)
```
1 2 + 3 *
```

Status
Pass
