Specification for Homework 3.4

This is the spec for HW3.4 to give you details about how your program is going to be executed, what your program is expected to do, and the further information on input/output format.

How your program will be executed

Your program is expected to read the test data from stdin and output to stdout.

Please provide a Makefile and make sure that TAs can compile your source code into two executable files, 3_4_1.out and 3_4_2.out, by typing 'make'. Since we will directly compile your program through 'make', it's ok to name your .c and .cpp files whatever you want.
(Download all the sample input/output here: http://goo.gl/ckDjbe)

Spec for Homework 3.4.1

Problem Description

Implement a signed integer calculator that supports all the operations listed in hw3.pdf with correct precedence and associativity.

Input Format

The test data file consists of multiple lines, and each line is an arithmetic expression.
Each line contains at most 1,000,000 characters and at least 1 character.
(i.e., $1 \leq MAX\_LENGTH \leq 1{,}000{,}000$)

It is guaranteed that no overflow problem or division-by-zero exists during all arithmetic operations.
The input test data ends with EOF (end-of-file).

Output Format

Please process each line of test data from stdin, and you need to:

1) Show the process of transforming infix expression into postfix one.

There's no strict regulations on how you output the transforming process.
Feel free to express yourself in any "human-understandable" way.

2) Show the corresponding postfix expression.

Separate every element of the postfix expression with a single white-space.

Let it occupy a single line and start with "Postfix Exp: ".

3) Show the evaluated result.

Output the correct result in a single line that starts with "RESULT: ".
Please DO NOT output a white-space between '-' and the number if the result is a negative integer.

*** Note ***
DO NOT output meaningless messages or un-necessary content, e.g., debug messages.
DO NOT have more than one line with “Postfix Exp: ” or “RESULTS: ” in the beginning for each test case.

Sample Input

```
(3 + 2)*(5&7)- 7*7
47/6+123*((!90)&&2)+!(!90)
7+7
```

Sample Output

```
Postfix Exp: 3 2 + 5 7 & * 7 7 * -
RESULT: -24
Postfix Exp: 47 6 / 123 90 ! 2 && * + 90 ! ! +
RESULT: 8
Postfix Exp: 7 7 +
RESULT: 14
```

*** NOTE ***
For unary plus and binary addition, your program should output the same character '+'.
And this applies to unary minus and binary subtraction too.

Spec for Homework 3.4.2

Problem Description

Implement a 8-byte floating point number calculator that supports all the operations listed in hw3.pdf with correct precedence and associativity.

Input Format

The test data file consist of multiple lines, and each line is an arithmetic expression.
Each line contains at most 1,000,000 characters and at least 1 character.
(i.e., 1 ≤ MAX_LENGTH ≤ 1,000,000)

It is guaranteed that no overflow problem or division-by-zero exists during all arithmetic operations.
The input test data ends with EOF (end-of-file).

Output Format

Please process each line of test data from stdin, and you need to:

1) Show the process of transforming infix expression into postfix one.

There's no strict regulations on how you output the transforming process.
Feel free to express yourself in any "human-understandable" way.

2) Show the corresponding postfix expression.

Separate every element of the postfix expression with a single white-space.
Let it occupy a single line and start with "Postfix Exp: ".

3) Show the evaluated result.

Output the correct result in a single line that starts with "RESULT: “.
The number of digits after decimal point to print: 6 digits.
Your may want to use:

```
printf("RESULT: %.6f\n", result);
```

to print the result.
Please DO NOT output a white-space between '-' and the number if the result is a negative floating point number.

*** NOTE ***
DO NOT output meaningless messages or un-necessary content, e.g., debug messages.
DO NOT have more than one line with “Postfix Exp: ” or “RESULTS: ” in the beginning for each test case.

Sample Input

```
-(fabs(sin(37.18/6.76*3.141592)+sqrt(+3.5)))
pow( 5 , exp( log(2) ) )
7 + 7
```

Sample Output

```
Postfix Exp: 37.180000 6.760000 / 3.141592 * sin 3.500000 + sqrt + fabs -
RESULT: -0.870829
Postfix Exp: 5.000000 2.000000 log exp pow
RESULT: 25.000000
Postfix Exp: 7.000000 7.000000 +
RESULT: 14.000000
```

*** NOTE ***
For unary plus and binary addition, your program should output the same character '+'. And this applies to unary minus and binary subtraction too.

Scoring

There are a total number of 50 test cases for 3.4.1, and the other 50 for 3.4.2. However, since this assignment is comparatively much easier than the previous two, we're giving you exponentially decreasing points with respect to the number of test cases your program processed correctly, rather than proportional scores.

Here's the equation for calculating you scores:

```
raw_score_1 = 50 - pow(2, wrong_result_count+wrong_postfix_count)
for case in your_wrong_cases
     if case.transforming is correct
          raw_score_1 += 1
final_score_1 = max(raw_score_1, 0) # => your final score of 3.4.1
```

final_score_2 is also evaluated through this equation with 40 replacing 50 in the first line.

FAQ

1) In hw3.pdf, what does it mean to prepare three test cases?

Ans: Think up 3 test cases for each of the two calculators, and write down (or print out) the test cases as well as the output of your program.

2) Can I use STL stack?

Ans: Yes! Use whatever in STL you like!

3) Does each line of input contains white-space?

Ans: Probably, since "(1+2)*3", "(1 + 2)*3", and "(1+ 2)* 3" are all legal expressions in C/C++. Your program should be able to correctly process the input string with and without white-space separating operands and/or operators.

4) Why isn't there any transforming process in Sample Output section above?

Ans: We skip the transforming part in our sample output so that the other two parts can be more clear to you. You can download the complete sample input/output through the link provided in the first page.

5) Where should I have "\n" (new-line element) in my output?

Ans: Basically you can have new-line elements anywhere you like, but the postfix expression should stay with the same line as "Postfix Exp: ", and so should the result.