Homework #3

RELEASE DATE: 04/15/2020

DUE DATE: 05/05/2020, **noon** on Gradescope/CEIBA

As directed below, you need to submit your code to the designated place on the course website.

Any form of cheating, lying or plagiarism will not be tolerated. Students can get zero scores and/or get negative scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.

Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.

Since everyone needs to write the final solutions alone, there is absolutely no need to lend your homework solutions and/or source codes to your classmates at any time. In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

Both English and Traditional Chinese are allowed for writing any part of your homework (if the compiler recognizes Traditional Chinese, of course). We do not accept any other languages. As for coding, either C or C++ or a mixture of them is allowed.

This homework set comes with 200 points and 20 bonus points. In general, every homework set of ours would come with a full credit of 200 points.

3.1 Analysis Tools

In this problem, you can use any theorems in the textbook as the foundation of your proof. You **cannot** use any other theorems unless you prove them first.

(1) (10%) Do Exercise R-4.15 of the textbook.

(2) (10%) Do Exercise R-4.28 of the textbook.

(3) (10%) Do Exercise R-4.34 of the textbook.

(4) (10%) Do Exercise R-4.39 of the textbook.

(5) (10%) Do Exercise C-4.16(b) of the textbook. Note that Horner's method is an important tool to learn when you want to evaluate a polynomial somewhere.

(6) (Bonus 10%) Given any function $f(n)$ from $\mathbf{N}$ to $\mathbb{R}^+ \cup \{0\}$ (i.e.a non-negative function), prove or disprove that $f(n) = O((f(n))^2)$.

3.2 Stack, Queue, Deque

(1) (10%) Do Exercise C-5.9 of the textbook.

(2) (10%) Use any pseudocode to write down an algorithm that uses two stacks (with push, pop and isempty operations but no others) to simulate one deque (for push/pop front and push/pop back operations). What is the total running time after N operations?

(3) (Bonus 10%) Do Exercise C-5.9 of the textbook, but with three stacks instead of two stacks and one deque.

3.3 List, Iterator

(1) (10%) Do Exercise C-6.7 of the textbook.

(2) (10%) Do Exercise C-6.13 of the textbook by Googling the *Knuth Shuffle* (yes, Knuth again!).

3.4 Calculators

In this problem, you will be asked to implement two calculators: an "integer calculator" that works on 4-byte integers and supports the arithmetic and bitwise operations in C; a "scientific calculator" that works on 8-byte floating point numbers and supports some scientific functions.

(1) (60%) Implement the (signed) integer calculator (`hw3_1.{c, cpp}`). You can also assume that the input will contain characters only from numbers, the needed operators, and ignore all other characters (including space). You need to implement the following operations with the correct precedence and associativity:

- multiplicative `*`, division `/` or modulo `%`
- binary add `+` or subtract `-`
- bitwise and `&`, exclusive or `^`, or `|`, left shift `<<` or right shift `>>`
- parentheses `()`
- unary minus `-` or plus `+`
- bitwise not ∼
- logical and `&&` (for simplicity, assume that it outputs 1 when both operands are non-zero, and 0 otherwise), logical or `||` (assume that it outputs 1 when either operands are non-zero, and 0 otherwise)
- logical not `!` (assume that it changes non-zero to 0 and 0 to 1)
- equal `==` and not equal `!=` (for simplicity, assume that it outputs 1 when the result is true, and 0 otherwise)

Your program should satisfy the following requirements.

- keep allowing the user to input a line of (i.e. read a line of length $\leq 1000,000$ from `stdin`) infix expression that supports all the operations above on integers, until no more lines can be read (EOF)
- show your stack operations on how to transform the infix expression to a postfix one
- show the corresponding postfix expression
- show the evaluated result, which should be exactly the same as the result computed by a usual C statement on the same expression

You can assume that the expressions provided will be valid, non-ambiguous and non-conflicting to the usual C expression. For instance, something like `i++-j` will not be used to test the program.

Please prepare THREE test cases that help the TA verify all the requirements above. Then, print out the output of your program on those three cases in the written part.

(2) (50%) Implement the scientific calculator (`hw3_2.{c, cpp}`). Now you may have floating point numbers of the form 2.354. You need to implement the following operations with the correct precedence and associativity:

- parentheses and function calls, as the highest precedence
- unary minus or plus
- multiplicative
- binary add or substract

You also need to implement the following function calls (as defined in `math.h`)

- sin, cos
- exp, log
- pow, sqrt
- fabs

Your program should satisfy the following requirements.

- allow the user to input a line of infix expression that supports all the operations above on doubles (Note: pow$(2, 3)$ is of postfix notation 2 3 pow)
- show your stack operations on how to transform the infix expression to a postfix one
- show the corresponding postfix expression
- show the evaluated result, which should be exactly the same as the result computed by a usual C statement on the same expression (if the inputs are all doubles)

Please prepare THREE test cases that help the TA verify all the requirements above. Then, print out the output of your program on those three cases in the written part.

For all problems, you need to "show the evaluated result" by printing out a single line of the form for the integer calculator

```
RESULT: 1126
```

or for the scientific calculator, show the evaluated results rounded to exactly 6 digits after the decimal point.

```
RESULT: 11.262110
```

You can freely decide how you want to show your other parts (stacks, postfix) by printing out other lines.

Submission File

Please submit your written part of the homework to Gradescope before the deadline. Also, you need to upload your coding part as a single ZIP compressed file to CEIBA before the deadline. The zip file should contain no directories and only the following items:

- all the source code
- a Makefile to compile your code and run your program. The TAs will type `make` to compile your source files to two programs. Please follow the specs of what you need to do in the Makefile.
- an optional `README` or `README.md`, anything you want the TAs to read before grading your code

Please make sure that your code can be compiled and run with the Makefile on CSIE R217 linux machines. Otherwise your program "fails" its most basic test and can result in ZERO!