Homework Assignment #5: Programming Project #1

Due Time/Date

2:10PM Monday, November 14, 2016. Late submission will be penalized by 20% for each working day overdue.

Task Description

Develop a C++ application (called "myCalculator" perhaps) that reads a C/C++ arithmetic expression, evaluates it, and prints out the result. The simplest kind of arithmetic expression that you must handle is a C/C++ constant integral expression built up from integers and the six arithmetic operators binary `+` (addition), binary `-` (subtraction), `*` (multiplication), `/` (division), `%` (remainder/modulo), and unary `-` (negation), with possible parentheses `(` and `)` to group subexpressions. Below are two examples:

- `1 + 2 - 3 * 4 / 5`
- `(1 + 23 - 456) * (78 / 9)`

Be careful with illegal inputs. When the input is illegal, your program should be able to report an error and stop (or ask for another input).

You may go beyond this basic requirement, but please follow the C/C++ syntax and semantics for integral arithmetic expressions.

Submission Guidelines

- Pack everything, excluding compiler-generated files, in a .zip file, named with the pattern "`b047050xx-ds2016-hw5.zip`".
- Upload the .zip file to the Ceiba course site for Data Structures 2016:

  `https://ceiba.ntu.edu.tw/1051ds2016`.
- If you use a Makefile, make sure that it outputs "`hw5`". Otherwise, make sure that the whole application can be compiled by a single command like "`gcc hw5.c`", "`g++ hw5.cpp`", or "`javac hw5.java`".

Grading

This assignment constitutes 5% of your grade (of this course). Your work will be graded according to its completeness, correctness, and presentation. You should provide evidences (such as tests) showing that your program is correct. You should also organize and document (by adding comments to) your program in such a way that other programmers, for example your classmates, can understand it. Below is a more specific grading policy:

Criteria	Score
far from complete, or doesn't compile	≤ 20
nearly complete, compiles, but with major errors	≤ 40
nearly complete, with minor errors	≤ 60
complete, but handles only single-digit integers	≤ 80
complete, except unary $-$	≤ 90
complete and correct	≤ 100
allows variables and assignments	$+10$
can detect an overflow/underflow	$+10$