Self-Evaluation Report for Programming Assignments

Section I: Basic Information

1. **Programming assignment #: 1A**
2. **Name of the author: Esther Choi**
3. **Name of the peer reviewer, if any: Priscilla Son**
4. **Due date of the assignment: February 8, 2017**
5. **Date when the assignment was finished: February 6, 2017**
6. **Number of hours spent in programming: roughly 4 hours**

Section II: Integrity Review

Integrity rules for regular programming assignments

- **Peer discussion**: Peer discussion of code shown on a screen or board is acceptable for explanation of ideas and for debugging purpose. Such discussion may help to cultivate an open learning environment in the class, but you should carefully read the guidelines below to avoid any dishonest behavior and never step over the guidelines explicitly described in the following.
- **Never use any code (i.e. C++ statements, segments of a program or an entire program) written by others (except for examples in our textbooks or reading)**: Any copy-and-paste of code from other people's programs or from websites is viewed as cheating and you will get 0 points for the assignment.
- **Never circulate your code to others**: You should never pass around your code (electronically or on paper) to others except for the TA and the instructor. Violating this rule is viewed as cheating in the class and the provider will receive 0 points for the assignment.
- **Never provide false or exaggerated results of test cases**: You need to report results of test cases in the self-evaluation report together with all your source code files for each assignment**.** Providing false or exaggerated results of test cases in the report is viewed as cheating and you will receive 0 points for the assignment.
- **Demonstrate the credibility of your authorship of the work**: When you submit your code as your own work for points, you should make sure that you are able to explain your code and reconstruct your code from scratch without any outside help when requested. If you are not able to do that on your own when requested, you will get 0 points for the assignment and there will be an investigation.
- **Consequence of cheating in the class**: Cheatings end in 0 points for the assignments followed by discipline actions described in the student handbook.

1. **Have you ever received any code written by others? No**
2. **Have you ever passed any code you wrote to others? No**
3. **Have you ever used any code written by others? No**

Section III: Test cases and peer review

Note: To get all the points, you should have a peer reviewer watch the behavior of your program before you submit the work. You should prepare your own test cases and have your reviewer see the results when you run your program over the test cases. Optionally, you may also have the reviewer run your program through the reviewer's own test cases to see whether your program works correctly.

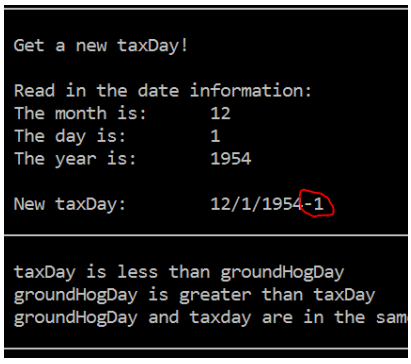
1. **Compile and run your code using Visual C++ 2013 as the testing environment and describe the test cases used and the results you and the reviewer have observed**:

Test Case (date1, date2)	**Testing For**	**Results/Comments**
1/1/2000, 1/2/2000	Testing for correct day precedence	- taxDay is greater than groundHogDay - groundHogDay is less than taxDay - groundHogDay and taxDay are in the same century
1/1/2000, 6/1/2000	Testing for correct month precedence	- taxDay is greater than groundHogDay - groundHogDay is less than taxDay - groundHogDay and taxDay are in the same century
1/1/2000, 1/1/2000	Testing to see that it would correctly read equal dates (testing for equivalency)	- taxDay is equal to groundHogDay - groundHogDay is equal to taxDay - groundHogDay and taxDay are in the same century
5/1/2000, 1/18/2000	Later month/earlier day and earlier month/later day testing	- taxDay is less than groundHogDay - groundHogDay is greater than taxDay - groundHogDay and taxDay are in the same century
1/18/1996, 5/1/1996	Earlier month/later day and later month/earlier day testing	- taxDay is greater than groundHogDay - groundHogDay is less than taxDay - groundHogDay and taxDay are in the same century
Century Tests		
1/1/2000, 6/5/1969	Testing for behavior of program when the final year of a century is the first date	- taxDay is less than groundHogDay - groundHogDay is greater than taxDay - groundHogDay and taxDay are in the same century
1/2/1969, 2/2/2000	Testing for behavior of program when the tail end (final year) of a given century is the second date	- taxDay is greater than groundHogDay - groundHogDay is less than taxDay - groundHogDay and taxDay are in the same century
3/24/2000, 2/16/1900	Testing for behavior of program when both years occur at the turn of the century	- taxDay is less than groundHogDay - groundHogDay is greater than taxDay - groundHogDay and taxDay are in the same century My peer reviewer and I found that the *if/else if* is not sufficient to cover for such a case (the program would only read the first if statement and ignore the other). Thus, the second year

		would pass through unaltered, causing the wrong output to be displayed. **Fix**: Discussed in #2 below.
3/25/1776, 1/20/1990	Testing earlier century date followed by later century date	- taxDay is greater than groundHogDay - groundHogDay is less than taxDay - groundHogDay is in an earlier century
1/20/1990, 3/25/1776	Testing later century date followed by earlier century date	- taxDay is less than groundHogDay - groundHogDay is greater than taxDay - groundHogDay is in a later century
Invalid Data Entry Tests		
0/20/1776, 13/1/2000	Testing for months that fall outside the 1-12 range	Error message displays and re-entry is required until data within 1-12 range is received
1/0/2000, 1/32/2000	Testing for days that fall outside the 1-31 range	Error message displays and re-entry is required until data within 1-31 range is received
1/1/0, 2/2/-832	Testing for years less than 1	Error message displays and re-entry is required until data within the stated bound is received

2. **Description of bugs or other problems discovered by you or the peer reviewer, if any:**

I don't know if this would count as a bug exactly, but the line of code on line 113 of DateTest.cpp really had me running around in circles. I had no idea that it was there and that it was put in place for testing, and so when I started mindlessly uncommenting the code for 'R', I started seeing these strange numbers (-1, 0, 1) just after getting the two dates (see screenshot below).



I emailed the professor and she told me to comment that line back out and all was well. I felt silly, because I was so sure the problem was isolated to a different part of the code that I hadn't even bothered to consider that it must be taking place in the main code. I'd focused all of my efforts on the wrong part of the code, in other words. As I said, I'm not sure if this would count as a bug, but I am listing it here for posterity!

For future 106 students, I feel it would be nice if there was a comment next to that line, indicating what it is for, and why it was commented out twice. Like I said, it was so obvious to me after the professor pointed it out and I did a total facepalm. *'Wow, Esther, you're so duuuuuumb…'* I thought to myself. However, it wasn't all that obvious when I was heavily immersed in the program and rushing against the clock while thinking about all the other assignments in my queue, so a comment there would be nice in my opinion.

The other thing is, the demo program we were given doesn't seem to account for turn-of-the-century years (1900, 2000, 2100, etc.). For example, 1969 and 2000 are in the same century, but the demo program considers 2000 to be in a later century. I found a

workaround for this by subtracting a century for those years that occur at the turn-of-the-century (see DateType.cpp, ComparedCentury(), lines 174-7).

Problems in my own program: Unlike with ComparedTo(), my peer reviewer and I saw that the *if/else if* implementation is insufficient for ComparedCentury(), because while it can account for years like (2000, 1969) or (1969, 2000), it cannot correctly account for two turn-of-the-century years (i.e. 1900, 2000).

How I solved said problem: I separated the *if-else if* statements into two individual if statements. This way, the program would consider the two years separately, and thus, both if statements could be executed if needed (see DateType.cpp, ComparedCentury(), lines 174-7).

3. **Have you implemented everything required by the programming assignment? If not, describe what are missing.**

   Yes, I have implemented everything that was required of us for part 1A, according to the instructions on the class website.

Section IV Self-evaluation: Points you think you deserve 6

- **Deduct one point if you submit the work after the due date but before it's closed.**
- **Grading scale**:

0. Nothing done **or missing the self-evaluation report or missing the integrity review** in the report
1. Source code is completed but the code fails to compile successfully
2. Source code can compile and do something required, but has serious bugs or miss a couple of key features.
3. Source code can compile and do most of the features required, but has many minor bugs or miss a key required feature.
4. Source code can compile and do all the features required, nearly fully functional, only a couple of minor bugs.
5. Source code can compile and do all the features required, fully functional, no bugs.
6. **In addition to the points received according to the rubrics above, get one more point** if
   a. **the self-evaluation report contains sufficient descriptions of test cases used (0.25 point)**, and
   b. **the self-evaluation report indicates the results of the test cases were verified by a peer reviewer (0.25 point)**, and
   c. **the source code is well indented and commented to make it visually very readable (0.5 point)**.