COMSC-165 Lab 1, Console Programming Basics

In this lab assignment you will develop a seemingly simple "console program". The purpose of this assignment is to understand how console input works, and how to manage programs with multiple, mixed inputs of text and numbers from the console.

As you complete this assignment, post the required file(s) to the COMSC server. For assignments that involve more than one file, you may post them all at once, or one-at-a-time as you complete them, as you prefer. For assignments that involve more than one separate "lab" (like lab 1a, 1b, and 1c), the individual labs are graded in order, starting with "a", which must be fully correct before "b" is graded, and so on. This assignment is worth 50-points, to be awarded after all labs are successfully completed. Use the "Submit your FA2014 work" link on the class website to post your file for this lab, using the **lab1** folder provided.

LAB 1: Console Programming Basics `[ TheBasics.cpp ]`
Following the programming conventions for this course, write a C++ console program named **TheBasics.cpp**, to read and print two sets of inputs, with two calculated values. Be sure to do things in the order specified.

The first set is the user's age (in whole number years) and name (first and last), in that order, each on its own separate line, following its prompt. The second set is the current outside temperature (in floating point degrees F), and the user's location (city), in that order, each on its own separate line, following its prompt. For example:

```
Enter your age: 21
Enter your name: Joe Student
Enter the temperature outside right now (degrees F): 45
What city are you in right now? Walnut Creek
```

After both sets have been read in, calculate two values: (1) the user's age one year from now, and (2) the temperature in degrees C. Finally, print the 4 input values and the two calculated values in the following 2-line format, spaced and punctuated per the example:

```
Joe Student is 21 years old now, and will be 22 a year from now.
It's 45 degrees F in Walnut Creek -- that's 7.2 degrees C.
```

Here are the specifications, in addition to the universal requirements and programming conventions explained in lab 1b above:

1. If the user enters a non-numeric for age or temperature, the program should not fail -- non-numeric entries should default to zero.
2. Allow for user name and city name to have more than one word in them, like Joe Student and Walnut Creek.
3. Print the degrees F in the same precision as entered (that is, if the input is 45.61, print it as 45.61).
4. Print the degrees C with one digit after the decimal point, rounded to the closest tenth of a degree.
5. Do NOT use the manipulator "fixed" -- use cout.setf instead, as explained in lecture.
6. Use ONLY int and double for numeric variables.
7. For text variables, use C strings, or C++ strings, or both, as you prefer.

8. Be sure to include your identifying information in both comment form and cout form.

Refer to your notes from lecture, that should explain how to do these. Test it carefully and thoroughly before submitting it.

HINTS: If you have to use if-statements in this one, you are not doing it right. Also, the "fixed" in `cout << fixed;` IS a manipulator -- do NOT use it. The use of "fixed" in `cout.setf(ios::fixed|ios::showpoint);` is NOT a manipulator -- you may use it.

*Post **TheBasics.cpp** to the COMSC server.*

GRADING POLICY

Lab work will be returned for you to fix and resubmit for any of the following reasons, or for not following stated specifications, unless directed otherwise by your instructor, Point penalties will be assessed per the syllabus for each time a lab is returned for redo for any listed reason below or specification above. Points will be awarded for an assignment upon final, successful completion of all of its labs. Note that you will NOT receive a list of corrections to make, because to do so would remove any incentive for students to do quality work. Instead, grading of your work will stop upon discovery of the first mistake. Grading will not continue until the mistake is corrected. So to avoid multiple cycles of correction and resubmission, and the cumulative point penalties, make sure that your work is as correct as you can make it before submitting it.

Basic checklist, even if your compiler "forgives" doing otherwise:

1. Spell and case filenames *exactly* as specified. To avoid errors, be sure to configure your Windows computer to NOT hide file name extensions -- refer to the Burns Intro to Programming text, chapter 2.
2. Submit files to the specified folder. Do NOT submit archive files, like ZIP, JAR, or RAR.
3. Include identifying information as the first lines in the file AND as the first lines of your output in main.
4. Do NOT use the statement `system("PAUSE");` or any other system-specific code.
5. Do NOT try to access the value stored in an uninitialized variable.
6. CPP and H files must be free of careless typing mistakes that would prevent compilation.
7. Put an `endl` as the last item in the last-executed `cout` statement before the closing curly brace of `main`.
8. Do NOT forget to use the `cin.ignore` statements for non-string console input, unless specifically directed otherwise.
9. Do NOT forget to use the `fin.ignore` statements for non-string file input, unless specifically directed otherwise.
10. For file I/O, NEVER include a path designation. ALWAYS use the "working folder" for files.
11. Use the proper `#includes` and `usings` as listed in the C++ Library PDF.
12. Do NOT #include C-libraries such as `math.h` -- include their C++ equivalents instead, like `cmath`.
13. Do NOT #include libraries that you do not use.
14. Do NOT #include "**stdafx.h**"; do NOT use `pragma`.
15. Do NOT use C/C++ constructs or library items that are not specifically taught in this class, such as `goto`, `scanf` or `fscanf`, `fixed`, `strcmp_c`, `stricmp`, etc.
16. Do NOT use `void main()` -- use `int main()` always.
17. Do NOT use `return 0;` as the last statement in `int main()`, because it is not required by ANSI Standard C++.
18. If you use output statements in your program for debugging purposes, be sure to delete them before submitting your work.

19. Do *not* use "global variables".

Advanced checklist, even if your compiler "forgives" doing otherwise:

1. `NULL` is ONLY a pointer value -- do NOT use it for integer or other numeric values, even if your compiler allows it.
2. Do NOT declare a static array with a variable size specification, such as `int n =...; int a[n];`, even if your compiler allows it.
3. NEVER #include a CPP file, even if the textbook includes examples of such.
4. For console input and text file input with `>>`, use `.ignore` as explained in the Burns Intro to Programming text, chapter 5, unless specifically directed otherwise.
5. NEVER use `cin >>` to read directly into a numeric variable, like `int` or `double` or `float`. Read as a string (C or C++) and convert to a number using `atoi` or `atof`. Include `cstdlib`!
6. Do NOT use the `inline` keyword. To make class member functions inline, write their definitions inside the class definition.
7. Once the `const` keyword is introduced in this course, use it where applicable. Note that class "getter" functions are not really getter functions unless they have a trailing `const` keyword to flag them as such.
8. Make sure that all value-returning functions return a valid value under any logical circumstance.
9. Exit functions with `return` statements, and NOT `exit` -- do NOT use `exit` under any circumstances.
10. In every H and CPP file, ALWAYS #include libraries and H files for ALL functions, classes, and identifiers used in that file. Exception: use a class "forward declaration" instead of #including the class' H file when only pointers and reference variables appear.
11. Include the `cstring` library for char-array-based string ("C string") *functions*, and the `string` library for STL-based-strings ("C++ strings").
12. When using the random number generator seed function `srand`, call it ONCE ONLY in your program, and "sacrifice" the first value, like this: `srand(time(0)); rand();` at the top of main. Be sure to #include both `ctime` and `cstdlib`.
13. In COMSC 210, use `using namespace std;`. In 165 and 200, use `using std::` statements, properly placed under the #includes of the libraries to which they pertain.
14. `std::string` and `std::getline` belong to the `string` library. In 165 and 200, use `using std::` statements for these. Note that `cin.getline` is a different thing, and only requires `using std::cin;`.
15. Do NOT use `using std::` for anything in the C libraries. Use it for items from C++ libraries, only.
16. Test drivers MUST at least include ifndef testing, object copy testing with assignment after declaration, and const object testing with assignment upon declaration.
17. Test drivers must NOT include console or file input, unless directed otherwise.
18. Do NOT allow memory leaks, unless specifically instructed otherwise.