**CSCI-130 (Wayne Wall): C++ Program Requirements**

The following are requirements for all C++ programs. Points will be deducted if the program is lacking a requirement and/or the requirement is poorly implemented. The number of points deducted is at my sole discretion.

* Programs must start with a comment line with the programmer’s name, immediately followed by a few comment lines briefly (but concisely) stating the problem that’s being solved. For example:  
    
  **// Programmer: Wayne Wall  
  // Description: Use gravitation law to calculate the force exerted by the earth on the moon.**
* Important constants should be placed in named storage and the names (not the numeric values) should be used in subsequent calculations. Meaningful descriptive names should be used. Any measurement units associated with a name should be indicated in an associated comment. For example:  
    
  **const double UNIV\_GRAV\_CONSTANT = 6.673e-11; // newton meters^2 / kg^2  
  const double EARTH\_MASS = 6.0e24; // kg  
  const double MOON\_MASS = 7.4e22; // kg**
* Use intermediate named variables to break down long and/or complex calculations into simpler to understand steps. Meaningful descriptive names should be used. If measurement units are associated with a variable, they should be indicated in an associated comment. For example:  
    
  **double mass = 10.0; // kg  
  double acceleration = 9.8; // meters/sec^2  
  double force; // newtons  
    
  force = mass \* acceleration;**
* Always store the final result in a variable with a meaningful descriptive name.  
    
  **pounds\_CO2\_per\_year = pounds\_CO2\_per\_gallon \* gallons\_per\_year;**  
  [No comment indicating the units is needed here as the units are self-evident from the variable name.]
* In general, when a program is run, there should be minimal intermediate output. Always display the variable holding the final answer last. Only display the results of an intermediate calculation if it will enhance the reader’s understanding of the solution process. For example:  
    
  **// Distance between the earth and the moon.  
    
  double distance = 3.8e8; // meters  
    
  force = UNIV\_GRAV\_CONSTANT \* EARTH\_MASS \* MOON\_MASS / (distance \* distance); // newtons  
    
  cout << “When the earth and moon are << distance << “ meters apart,” << endl;  
  cout << “The force of attraction is “ << force << “ newtons.” << endl;**
* Use horizontal and vertical “whitespace” (i.e., spaces and blank lines) to separate and delineate the structure of your program so as to make it easier for the human eye-brain combination to identify and comprehend. Don’t scrunch things close together just for the sake of reducing the amount of typing. Doing so may (or may not) save a little time when you’re coding the initial program, but the important goal is to make it easy for someone (perhaps your boss or even yourself) to understand later on. (See any of my homework solutions as examples.)
* Curly braces (“**{**“, “**}**”) should be placed on a line by themselves. This is consistent with the “whitespace” rule above.
* Statements contained within a code block (i.e., a pair of curly braces) should be indented (at least 2 and not more than 4 spaces) and left-aligned. For example:  
    
  **int main()  
  {  
   // Statement #1  
   // Statement #2   
  }**
* All words should be spelled correctly! Spelling errors reflect poorly on both programmer and employer.
* To the extent possible, each line of console output should be generated by a single “**cout << ... << endl;**” statement.
* Your program should show your understanding and correct use of C++ concepts and techniques. For example, don’t replicate the same statements multiple times when a looping operation could be used instead.
* In addition to the above, the program, when run, must NOT result in any compile or execution errors AND it must produce a correct final result.

In the future, in order to reduce the volume of comments I have to put in your graded programs, I’ve assigned three letter codes to each requirement. Here are the codes and their meaning:

**NPR** – Programmer name and concise problem statement as comments at beginning of program.  
**UNC** – Use named constants.  
**MSN** – Meaningful symbolic names.  
**SMU** – Specify measurement units.  
**IVU** – Intermediate variable usage to break complex operations into simpler pieces.  
**SFR** – Store final result in a meaningful name.  
**DFR** – Display final result.  
**FRC** – Final result is correct.  
**MCW** – Minimal console output.  
**WSU** – White space usage to enhance readability.  
**PUC** – Proper use of C++ concepts.  
**NCE** – No compile or runtime errors.  
**SLC** – Single line of console output generated by single **cout** statement.  
**SPL** – Spelling error.

From now on, when I deduct points for a missing or poorly implemented requirement, I’ll simply list the code followed by the number of points deducted. For instance, if you fail to use one or more meaningful symbolic names, I’ll put this comment:

**// (WW) MSN (-3)**

I’ll always preface any comments I add to your programs with “**(WW)**” to differentiate them from your own.