Style Guidelines

This page is intended to summarize general style guidelines, which should help you build good habits in writing and documenting code. It is not intended to describe every possible situation, exception, or variation in different coding/documenting styles. This is a set of general guidelines you should use for this course. If some points are not clear to you, please ask an instructor.

General Coding Guidelines

1. Identifiers should be meaningful, but not overly lengthy
   - For variables and constants, it's good to use nouns that describe what is stored. Examples:

     ```
     numStudents          // good
     num_students         // also fine
     lastName             // good
     GPA                  // good, since its a pretty standard abbreviation

     Make sure not to make them too short or too long...

     ns                   // not so good
     the_number_of_students_in_the_class_im_taking        // god help us...
     ```

   - A general exception to this would be in the case of things like loop counters -- since they are in very localized scopes, variables like `i`, `j`, etc are easily understood

     ```
     for (int i = 0; i < 10; i++)
     ```

   - For function names, it's good to use descriptive verbs, nouns, or adjectives, that describe the function's purpose:

     ```
     CalculateGrade
     PrintReport
     Maximum
     Factorial
     ```

2. Individual functions (including `main()`) should not be excessively long. (For example, if it takes more than 1 or 2 pages when printed out, it's pretty long).

3. Avoid the use of global variables (meaning variables that are declared outside the scope of any function). Things that are only used within a given function should be declared local to that function. The only general exception should be constants, though only if used througout the program.

4. Get into the habit of declaring variables at the top of main(), and using them later on.

Commenting Guidelines

Comments should be used to *clarify* and make your code more understandable to the reader -- which might be somebody else, but also might be YOU, a day/week/month/year later. There are many varied styles out there, but overall I'm looking for documentation that improves readbility, but does not clutter up the code and detract from readability.

1. Include a main header comment at the top of each file. This comment block should include:
   - Your name
   - Your course and section number
   - Project number
   - Brief summary of the file contents / structure (i.e. this would include the purpose of the program, as well as how your file and code are arranged

   Note: While for this course, I want to make sure your header includes your name, section, etc -- it is ALWAYS a good idea to have a header on any code file that summarizes the file's purpose. This is especially true when you one day work on multiple-file programs, and each file contains different elements or modules used in a given program

2. Avoid "well, duh" comments. Don't just repeat what the code says directly in a comment. Example:

```
x = temp;            // set x to temp    <---- well, DUH!
```

3. Comments on variable declarations should be meaningful, to clarify what the variable's purpose in the program is. Examples:

```
int test1, test2, final;      // for user entry of test scores
double average;               // for storing the calculated test average
int i;                        // loop counter
```

4. Line comments in nearby sections of code are more clear if they line up to the same starting point. Good:

```
int num1, num2, num3;                 // user entered numbers
double average;                       // calculated average of the numbers
int score1,                           // score on exam 1
    score2;                           // score on exam 2
```

   Bad:

```
int num1, num2, num3;          // user entered numbers
double average;    // calculated average of the numbers
int score1,    // score on exam 1
    score2;                       // score on exam 2
```

   Note that this second one looks junky and cluttered. Harder to read.

5. Comments that are longer (i.e. take more than one line) are often best done as block comments, *before*

the item or section being commented on. Line comment style is okay, too -- but make sure it's clearly one section of comments and doesn't obscure actual code. Example:

```
/* This function takes in a fahrenheit temperature (fahr)
 *  and returns the equivalent temperature converted
 *  to the celsius scale
 */
double ConvertToCelsius(double fahr)
{
  ...
}
```

6. Good places to USE comments in a program:
    - Variable declarations, to clarify the purpose of the variables you are using in various sections of a program
    - At the start of any function, to state the purpose of a given function, as well as any assumptions, pre-conditions, or post-conditions of the function
    - In between sections of a program where some new "task" (or subtask) is happening.
        - For example, a program might (1) read student records from a file, (2) processes the information by calculating the GPA for each student, (3) sort the students by GPA (highest to lowest), (4) print the results in this order to an output file. In this case, I would suggest a comment before each of these major "task" sections, to clarify what step you are on.
    - Anywhere in your code where the code itself needs clarification for a reader's understanding

Formatting Programs

1. It's best to write your code so that you don't use more than 80 characters per "line".
    - Note that if you use *longer* lines, your lines will often not appear in full in many text editors, and printouts of code (using standard fonts and sizes) will often cause lines to wrap around -- leading to bad readaibility

2. Separate different "sections" of code with blank lines (like a section of variable declarations, then a section out output statements). See examples on the course web site

3. You should always indent the contents of a block by a few spaces, to make it easy to spot the contents of a given block:

```
{
   int length, width;
   int area;

   // now another block, just for fun
   {
       cout << "Woo hoo!";
       area = length * width;
   }
```

```
   cout << "Area = " << area;
}
```

4. main() should be the first function in your program -- meaning, that anytime you write other functions, they need to be defined below main(), with their declarations above main() (But below the header/library includes). Example:

```
#include
using namespace std;

/*Function DECLARATIONS*/

int main()
{
        /* main() code*/
}

/*Function DEFINITIONS*/
```

5. As we learn new control structures and language elements, you will see examples of good style and indentation for these. More to be added when we get to them.