**Summary of CSCI 3342 Style Rules**

1. **Naming** Name things well in your code, and *remember especially to start the name of any void free function or void class member function (method) with a verb*.
2. **Formatting** Be *consistent* when formatting code, and be especially careful to ensure the following:
   1. All statements at the same indentation level must be vertically aligned. Indentation levels are 4 spaces. (Statements that continue beyond a single line may continue to use this indentation level or use a modified level to enhance readability.)
   2. All matching braces **must** align vertically and each brace **must** be on a line by itself.
   3. Appropriate vertical and horizontal spacing must be inserted and used consistently for maximum readability. At least one blank line should separate “logical chunks” of code. Operators will generally have a blank space on either side.
   4. Your code must contain **no** TAB characters and **no** wrapped lines (80 or more characters).
   5. You must use the proper style of capitalization for each kind of program entity. (Class names and names of void free functions start with a capital, while value-returning free functions and *all* member functions start with a lowercase letter. Variables start with a lowercase letter. C*amel notation* is used throughout. The one exception to all of this is named constants, which use all caps and underscores to separate words in the name.)
   6. If a function has more than one parameter, in the function header the parameters must appear one per line.
3. **Commenting** Each submitted source code file must have // comments at the beginning for the ID information, a /\* … \*/ comment for the self-assessment, and short descriptive comments preceding classes, methods and free functions.
   1. If short, comments can be at the end of the same line as the thing being commented. Otherwise, comments must be indented to the same level as the thing being commented.
   2. When in doubt, check the commenting style used in the utilities.h file.
   3. It's a good idea to provide an in/out/inout comment to indicate the direction of information flow for any function parameter, using a // comment following the parameter itself in the case of multiple parameters or /\* \*/ in the case of a single parameter..
4. **Error Checking** Your programs are not required to check user input for potential errors and deal with them unless this is specifically requested in the specifications for a submission. The default is to assume that users will enter what they are asked to enter, and if they don’t your program is not responsible for what happens next.
5. **Submitting** Copy any files required for a submission from your working directory. Do not use any submission directory as a working directory.
6. **Miscellaneous Do’s and Don’ts**
   1. Never use unauthorized global variables.
   2. using namespace std; is OK for the standard namespace but  
      using Scobey::whatever; illustrates the style to be used for *any* non-standard namespace
   3. If getting and/or setting class data members is required, always use getter and setter member functions (methods) to get and set those values.
   4. Make sure any submitted executable displays any required identification when it runs.
   5. Several short functions are preferred to one long one (a long switch or nested if..else *may* be permitted).

The following is not an exhaustive list, since it is impossible to make a list of all the things that might be wrong with a given program. However, the list does give you some particular transgressions for which you will be penalized if you commit them.

In each case the initial penalty will be at least 1 point, and sometimes 2 or more points out of 30, depending on the severity of the transgression, and at the discretion of the marker. In addition, if you commit one or more of these transgressions and the marker points out that you should not do so, and you then continue to do so, you may expect to lose an increasing number of points for so doing, until you correct the problem. It is worth remembering that we are as serious about style as we are about substance.

1. Missing self-assessment, enclosed in /\* … \*/ comment delimiters.
2. Comments that do not align with the thing being commented (except for inline comments).
3. Wrapped lines (all lines must be < 80 characters in length and contain **no** TAB characters).
4. Improper indentation levels (our indentation level is four spaces).
5. Improper statement alignment (which is likely to happen if you have TABs in your code).
6. Matching braces that enclose a code block and are not aligned vertically, with each brace on a line by itself.
7. Improperly formatted function parameter lists, which don’t have vertically aligned parentheses or don’t have one parameter per line.
8. Use of one or more unauthorized global variables.
9. A free void function that does not begin with a verb or does not start with a capital letter.
10. A free value-returning function that does not begin with a lowercase letter.
11. A class member function that does not start with a lowercase letter.
12. A void class member function that does not start with a verb.
13. Poor variable naming or misuse of “camel case” spelling in program identifiers.
14. Named constants that are not all caps, with underscores as word separators.

Let's be clear that the marker must be able to display your code and run your executable to fully evaluate your submission. Since both the running and the displaying of your files are performed by scripts, it is critical that your submitted files are in the correct location and have the correct names.

Our **fcc** command will generally do a lot of the required formatting for you, but the ultimate responsibility for having well-formatted code is still yours.