Programming Grading Rubric

Program is complete and correct

This aspect of grading looks for code that compiles correctly and meets all of the program specifications. The highest grade is received if the program is complete and performs **all** operations correctly under all situations (all test cases). The grade is reduced a small amount when the program runs, but makes a minor error handling one case. The grade is increasingly reduced for

- Major errors produced during testing
- Failing to meet one or more specifications
- Memory Leaks anywhere
- Allowing access to invalid data, e.g., an array element outside the bounds of the array or de-referenced memory of an unknown address (pointer)
- Implementations which are inefficient (in terms of complexity and memory)
- Programs which compile, but crash during testing
- Programs which are implemented, but do not compile

Code that is easy to understand and uses good programming practices; directions are followed

This aspect of grading looks for code that is easy to understand, uses good programming practices, and is well documented. The following describes these practices.

- Describing the contents of each file at the beginning of the file. Describe the purpose of each class (or other code), the functionality, and any assumptions made. Give the author and a brief description of code use.
- Documenting the purpose including appropriate conditions for each function (including main). Document input and output as appropriate. Complete documentation should be included *at both definition and implementation,* i.e., in both the .h and .cpp files.
- Documenting each logical code block within each function when performing non-obvious operations.
- Using indentation appropriately and consistently to delineate code blocks.
- Using meaningful identifier names, e.g., function and variables names.
- Using appropriate white space between logical code blocks. Using a line to delineate functions.
- Using white space within lines of code. Operators such as = and + and << should have a space on either side.
- Writing lines of code with reasonable length (limit these to 80 characters, using a return when appropriate so lines do not wrap when printing on paper in portrait mode). Break long lines into shorter lines when possible.
- Using a non-proportional font, so indentation is meaningful (each character takes up same amount of space).

- Object-oriented: (Objects should be self-contained with private data hidden. Methods should be public only when necessary. Do not circumvent information hiding by allowing operations that return internal, implementation-dependent data, e.g., pointers or reference to data members, unless appropriate.)
- Modular: Each function should perform a single well-defined operation. If a function performs two tasks, break it into two functions. If a function (including main) performs a series of steps, each step should be a function.
- Miscellaneous: Use const member functions whenever possible. When passing by reference, use const whenever possible. Do not use global variables. Global constants are acceptable. Eliminate compiler warnings when possible.

- Following directions (e.g., naming files as specified, coding data files as specified, turn-in of only files desired, hardcopy of only files desired in the order specified).