CSS 341 Grading Rubric[1] Autumn 2012

The general grading categories below will vary in weight depending on the nature and extent of the individual assignments.

Program is complete and correct (~40 - ~60 % of the grade)

This aspect of grading looks for code that runs 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
- Failure for any test cases
- Lack of user friendliness

Code that is efficient and does not have potential bugs (15-25%)

- Unreachable code
- Useless loops/conditions/variables
- Redundant computation
- Uninitialized variables
- Implementations which include unnecessary or unnecessarily complicated code

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.

Code that is easy to understand and uses good programming practices; directions are followed (15-30%)

- Describing the contents of each file and module 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 or subroutine. Document input and output as appropriate.
- Documenting each logical code block within each function or subroutine when performing non-obvious operations.
- Using indentation appropriately and consistently to delineate code blocks.
- Using meaningful identifier names, e.g., function, subroutine, and variables names.
- Using appropriate white space between logical code blocks. Using a line to delineate functions and subroutines.
- Using white space within lines of code. Operators such as = or + or * should have a space on either side.

[1] Modified from Professor Carol Zander, with permission.

- Writing lines of code with reasonable length (limit these to 80 characters, using a return and continuing the code on the next line 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).
- Modular: Each function or subroutine should perform a single well-defined operation. If a function or subroutine performs two tasks, break it into two routines. If a routine performs a series of steps where each involves some work, then each step should be performed in another routine. Use either a function or a subroutine as appropriate.
- Miscellaneous: <u>Use constants always instead of numbers in programs.</u> In JavaScript, some versions do not provide for *constant* data types. You are to define the constants as an ordinary variable with a name that will remind you it is a constant and not to be changed. Use pass by reference and pass by value correctly. Do not use global variables more than necessary, but use parameter passing when routines need information. All variables are to be defined before usage. Write code as if "use strict"; were in effect. In fact, our versions of JavaScript do not seem to implement it. 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).

Software development Process and analysis (5-15%)

This section is graded for evidence provided of a well-organized, thoughtful software development process, including analysis of the plan and actual effort required.