

CS2606 Grading Rubric for Documentation and Style

Programming project grading is based on three criteria:

1. Documentation and Style
2. Design
3. Correctness and Efficiency

Typically, 20-25% is allocated to documentation and style, 25-35% is allocated to design, and the rest to correctness and efficiency. This document provides guidance to students and GTAs on how grading for documentation and style should be conducted. Grading for design and correctness and efficiency depend on the specifics of the project.

1 Naming

Excellent All names are well-chosen. Method names and parameter names have self-explanatory names, so that their role is clear without requiring reference to comments or documentation. Methods are named after verbs or verb phrases, while data members are named after nouns or noun phrases. Getters and setters follow an accepted naming convention. Method names are chosen to improve the “naturalness” of reading method calls, and so that the role of actuals is more apparent to the reader of a call to the method. Operator overloading is used judiciously in a way that increases both readability and writeability. Variable names (except for loop indices, which may be single-letter) are also appropriately named, so that the need for commenting is minimized. Names are spelled correctly, and abbreviations are avoided (except for those that are most common, such as pos, ptr, etc.). Appropriate (noun phrase) class names are used, and appropriate capitalization conventions are followed. Appropriate choices are made to minimize the scope and visibility of names to an appropriate region, particularly loop indices, helper classes, and temporary variables.

Good Most names are well-chosen, but a small percentage of the user-defined names do not meet the requirements for excellence and there is some room for improvement. Appropriate choices are still made to minimize the scope and visibility of names to an appropriate region.

Satisfactory A solid attempt to choose meaningful names has been made, but there is significant room for improvement. Many names could be refined. Some methods may be given misleading or uninformative names, some data members may give little or no indication of the role the data plays, or some variable names may give no practical information about how the variable is used. Some variables or data members may be declared in a scope that is too large, or with an inappropriate level of visibility.

Unsatisfactory Although some names are acceptable, many names violate expectations, are poorly chosen, or hamper readability.

Not Acceptable No attempt has been made to use well-chosen names or to follow acceptable naming conventions.

2 Commenting

Excellent There is a comment preceding main() containing the Honor Code pledge, the compiler and operating system used, the date completed and a description of the program’s function. This description is more than just language borrowed from the specification. All public and protected classes and methods have appropriate comments in the corresponding header file. All such comments include a meaningful “summary” (the first sentence), and provide an appropriate level of supplementary explanation for the reader. All parameters, return values, and thrown exceptions are included. All classes include

author and version information. Implementation files need not repeat these comments, and should instead include any internal commenting necessary for understanding the logic of the class' methods. Any implicit relationships between data members in the class is clearly documented. The preconditions (if any) and postconditions in each public method's contract are clearly defined in that method's documentation.

Good Most required comments are present, but there is still room for some improvement. Implicit relationships between data members may be missing, or some protected items may be missing appropriate documentation. Some internal portions of the code may be in need of additional explanatory commenting. The preconditions (if any) and postconditions for nearly every public method's contract are clearly defined in that method's documentation.

Satisfactory A solid attempt at providing appropriate documentation has been made, but there is significant room for improvement. The overall quality of documentation on public and protected features is lacking. Some parameters, return values, or entire methods, may be missing from the comments. The implementation file may be lacking appropriate commenting in some areas. Comments may be poorly worded or difficult to read.

Unsatisfactory Although some attempt has been made to document classes and methods, comments are sporadic. Many public and/or protected classes or methods have little or no appropriate documentation, and there is no coherent attempt to follow the course commenting guidelines.

Not Acceptable No comments, or comments so poorly written that they provide no useful value to the reader.

3 Style

Excellent The code is clean and professional looking. Significant attention has been paid to readability, visual layout, presentation, and organization. Use of whitespace, "dividers" (comments that break a file into visual "chunks" that are semantically meaningful), and an extremely consistent file organization scheme make it nearly effortless to navigate the code and find exactly what you are looking for. All spelling is correct, no questionable or error-prone coding practices are present, indentation is flawless, and no lines are over 80 characters. Code is clearly and appropriately divided into files, all header files have appropriate guards, and no circular inclusion is present. All contracts appearing in documentation are reinforced with assertions in the code. Named constants and enumerated types are used in place of literal constants where appropriate.

Good The code is clean and readable, and attention has been paid to white space and visual layout, but there is still some room for improvement. There may be some small indentation problems, a few lines may be a bit too long, or there may be less judicious use of whitespace overall. Some minor coding issues may be present that either decrease readability or increase the likelihood of errors (using assignment within logical tests, missing braces on some control constructs, empty blocks with no explanation, non-virtual destructors, mismatched new/delete, minor memory leaks, missing copy constructors or assignment operators, overly complex method logic, inappropriate use of multiple returns, unsafe exception practices, unsafe pointer practices, etc.). Code is clearly and appropriately divided into files, all header files have appropriate guards, and no circular inclusion is present. Nearly all contracts appearing in documentation are reinforced with assertions in the code.

Satisfactory A solid attempt at providing a clean, readable code, but there is significant room for improvement. The overall quality of the visual layout could stand improvement. Visually navigating the code and locating/identifying a particular point of interest is not facilitated by the layout used. A larger number of coding issues may be present. Significant mistakes in memory management or destructor code may be present. Some decisions about source file/header file structure or header file guards may

be inappropriate. While the overall structure of the code is legible, there are multiple instances of hard-to-read code.

Unsatisfactory Although some attempt has been made to produce clean, readable code, the effort is very sporadic. Code is difficult to navigate and layout is inconsistent. Numerous coding issues are present.

Not Acceptable No honest attempt at producing readable, clean code has been made.