Grading Standards

This document provides criteria used to grade the programming assignments. Each criterion has a number of different levels of achievement, with a description of how a submission will accomplish that level and the number of points assigned for reaching it.

Use these criteria while reviewing and providing feedback to your peers during draft review.

Criteria

1. **Correctness**
   This is the most important criterion. A program must meet its specifications and function correctly. This means that it behaves as desired, producing the correct output, for a variety of inputs (test cases). This criterion includes the need to meet specifications by writing a program in a particular way or using a particular language feature, if such a thing is mentioned in the problem.

   If a specification is ambiguous or unclear, you have two choices: You can either make a reasonable assumption about what is required, based on what makes the most sense to you, or you can ask the instructor. If you make an assumption about an ambiguous specification, you should mention that somewhere in a comment so that the reader/grader knows what you were thinking. Points may be taken off for poor assumptions, however.

2. **Code Efficiency**
   There are often many ways to write a program that meets a particular specification, and several of them are often poor choices. They may be poor choices because they take many more lines of code (and thus your effort and time) than needed, or they may take much more of the computer's time to execute than needed.

3. **Documentation**
   Every file containing code should start with a header comment. At the very least, this header should contain the name of the file, a description of what the included code does, and the name of its author (you). Other details you might include are the date it was written, a more detailed description of the approach used in the code if it is complex or may be misunderstood, or references to resources that you used to help you write it. A template file will be provided for each assignment.

   All code should also be well-commented. This requires striking a balance between commenting everything, which adds a great deal of unneeded noise to the code, and commenting nothing, in which case the reader of the code (or you, when you come back to it later) has no assistance in understanding the more complex or less obvious sections of code. In general, aim to put a comment on any line of code that you might not understand yourself if you came back to it in a month without having thought about it in the interim.

4. **Readability**
   Code needs to be readable to both you and a knowledgeable third party. This involves:
   - Using indentation consistently (e.g., every function's body is indented to the same level).
   - Adding whitespace (blank lines, spaces) where appropriate to help separate distinct parts of the code (e.g., space after commas in lists, blank lines between functions or between blocks of related lines within functions, etc.).
   - Giving variables meaningful names. Variables named `A`, `B`, and `C` or `foo`, `bar`, and `bas` give the reader no information whatsoever about their purpose or what information they may hold. Names like `area`, `maximum`, and `counter` are much more useful. Loop variables are a common exception to this idea, and loop variables named `i`, `j`, etc. are fine.

5. **Error Handling**
   Your code should include error handling techniques. Making your programs behave properly when encountering unexpected conditions is where it really gets challenging. Including exception handling can be one technique of error handling.

Criterion	Below Average	Average	Above Average	Excellent
Program Correctness	0-54 Program only functions correctly in very limited cases or not at all.	55-59 Significant details of the specification are violated. Program often exhibits incorrect behavior.	60-64 Minor details of the program specification are violated, program functions incorrectly for some test cases.	65-70 No bugs are evident. Program always works correctly and meets the specifications.
Readability	0-1 Major problems with at three or four of the readability subcategories.	2 At least one major issue with indentation, whitespace, variable names, or organization.	3 Minor issues with consistent indentation, use of whitespace, variable naming, or general organization.	4-5 No bugs are evident. Code is clean, understandable, and well organized.

Documentation	0-1 No file header or comments present.	2 File header is missing, complicated lines or sections of code uncommented or lacking meaningful comments.	3 One or two places that could benefit from comments are missing them or the code is overly commented.	4-5 No bugs are evident. Code is well-commented.
Code Efficiency	0-5 Many things in the code could have been accomplished in an easier, faster, or otherwise better fashion.	6-7 Code uses poorly-chosen approaches in at least one place.		8-10 No bugs are evident. Code uses the best approach in every case.
Error Handling	0-5 No error handling.	6-7 Errors are poorly handled.	8 Some error handling for functions or part of the code.	9-10 Code include error handling techniques.