Programming Assignment Grading Rubric
Csci4211 Spring 2019

April 9, 2019

This document lays out common criteria used to grade CSCI 4211 programming assignment. Each criterion has a number of different levels of achievement, with a description of how a submission will attain that level and the number of points/percentage assigned for reaching it. Please email us if you have any questions about this rubric.

1 Criteria

1.1 Program Specifications / Correctness

This is the most important criterion. The program meets specifications and functions as clearly described in the programming assignment write-up document found on the course website(Please make sure to download and read the most updated version). This means that it behaves as desired, producing the correct output(Please see output files provided), for a variety of inputs.

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 TAs/Instructor. If you make an assumption about an ambiguous specification, you should clearly mention that in your README file so that the reader/grader knows what you were thinking. Points may be taken off for poor assumptions, however.

NOTE : Successfully completing phase 1 will be a ”poor”.

1.2 Readability

Your 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 baz give the reader no information whatsoever about their purpose or what information they may hold. Names like Metadata Server, S_1 and S_2 are much more useful. Loop variables are a common exception to this idea, and loop variables named i, j, etc. are okay.
- The code should be well organized. Functions should be defined in one section of the program, code should be organized into functions so that blocks of code that need to be reused are contained within functions to enable that, and functions should have meaningful names.

1.3 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. 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.

A clear, concise and well written README file should be provided. This file should be a details description of your code, the logic used, assumptions made and complete data flow through your program. The details in this file should be such that, if you looked at this program in let's say a year from now, you should be able to understand the execution, logic and algorithms used in this program without reading through the code itself. This file should also contain a description of how to run your program and where to find the desired output. i.e, whether the output is displayed on the terminal or dumped in a file and so on.

1.4 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. For example, a certain section of code can be executed ten times by copying and pasting it ten times in a row or by putting it in a simple for loop. The latter is far superior and greatly preferred, not only because it makes it faster to both write the code and read it later, but because it makes it easier for you to change and maintain.

Although efficiency is somewhat beyond the scope of this programming assignment, please ensure that your code does not take more that 1 minutes to process a request from a client.

1.5 Assignment Specifications

Assignments will usually contain specifications and/or requirements outside of the programming problems themselves. For example, the way you name your files to submit them to the course website will be specified in the assignment. Other instructions may be included as well.

Ensure that you follow the programming assignment-write up protocols, submission guidelines and/or datelines. Carefully read the programming assignment write-up page on the course website and follow provided instructions clearly. Please feel free to email the TAs/instructor if you any questions or concerns.

2 Grading Standards

Every criterion will make up an approximate percentage of the grade given to a single programming problem as indicated in the "Approx. % of Grade" column. Points will be assigned for a particular criterion roughly along the lines of the guidelines of the "Excellent," "Adequate," "Poor," and "Not Met" evaluations.

For example, a problem that was marked as "Adequate" in the Program Spec./Correctness criterion, "Poor" for readability, and "Excellent" in all other areas would receive:
$0.8 * 0.6 + 0.6 * 0.15 + 1 * 0.15 + 1 * 0.05 + 1 * 0.05 = 82\%$

Criteria	Appr % grade	Excellent (100%)	Adequate (80%)	Poor (60%)	Not Met 0%
Program Specifications / Correctness	60%	No errors, program always works correctly and produces desired output.	Minor details of the program specification are violated, program functions incorrectly for some inputs.	Significant details of the specification are violated, program often exhibits incorrect behavior.	Program only functions correctly in very limited cases or not at all.
Readability	15%	No errors, code is clean, understandable, and well-organized.	Minor issues with consistent indentation, use of whitespace, variable naming, or general organization.	At least 3 major issue with indentation, whitespace, variable names, or organization.	Major problems with four or more of the readability subcategories.
Documentation	15%	No errors, well described README file and headers including name(you), code description are provided.	README file is not well described and clear.	README file is provided but failed to describe how program is executed and more.	No README file present.
Code Efficiency	5%	No errors, code runs in less than 1 minutes.	N/A	Code takes 2 - 4 mins to to run.	Many things in the code could have been accomplished in an easier, faster, or otherwise better fashion.
Assignment Specifications	5%	No errors	N/A	Minor details of the assignment specification are violated, such as files named incorrectly or extra instructions slightly misunderstood.	Significant details of the specification are violated, such as extra instructions ignored or entirely misunderstood.