16.216: ECE Application Programming

Program Grading Guidelines

1. Acknowledgements

This grading rubric has, for all intents and purposes, been copied directly from the UMass Dartmouth ECE 160 rubric, with some minor modifications. Thanks to Prof. Phil Viall and Ben Viall for allowing me to use their work.

2. Grading breakdown

This rubric will be used unless explicitly specified otherwise in the program handout. Although some of these items may seem “picky,” they are included to ensure that programs can be graded in a reasonable amount of time.

Correct submission name.....**10%**
Each program has a specified name that must be used when submitting your files. This name will be listed in the assignment, typically in the form "`prog#_progname.c`". If a program requires multiple files (for example, header and source files), multiple filenames will be listed. The name(s) you use must match the given name(s) exactly (pay attention to upper/lower case, etc). The quote marks are not part of the name.

Appropriate documentation.....**10%**
At a minimum, the main program file must have a header comment which includes your name, the date, and a description of the program. Each variable used in the main function must be described. Each and every function must have a header which includes the name of the function, a brief description, and a description of the “preconditions” (On Entry), and “post-conditions” (On Exit). Each variable used in a function must be described.

Appropriate looking code.....**20%**
Appropriate looking code is code which appears (to me) related to the given problem. In addition, all code must be consistently indented, all variables and constants should have descriptive names, and variables and constant names should use a consistent capitalization scheme.

Successful compilation **10%**
The code compiles without any fatal errors/warnings that prevent it from working.

Correct output **50%**
Output is as specified in the problem statement. Do not simply try the sample inputs to see if your program produces the specified output. I expect you to try additional possibilities; I assure you I will try several data sets which are not in the handout. Your program should function as specified in the problem statement. Incorrect output results in lowering of score.

Examples (not an exhaustive list, and assuming a max of 100 points):

- You write a bunch of code which looks like you've been thinking about the problem and hand in a correctly named program, with OK documentation...you would earn a score of 40 points.
- You write a bunch of code which looks like you've been thinking about the problem (and it compiles) and hand in a correctly named program, with OK documentation and no output...you would earn a score of 50 points.
- You hand in a perfect program with the wrong name...80 points
- You hand in a perfectly running program with no documentation...90 points

3. Deductions

Incorrect name for associated data file ……..... **-10%**
Some programs will require your program to read and/or write a file. The name of the file will be specified in the handout. Your program must use this exact file name.

Submission after due date .....**-(2^{n-1}) points**
Where n is the number of days late. i.e. a program that is one day late loses 1 point; a program which is 4 days late loses 8 points. A program that is more than seven days late will receive 0 points. Programs are due on the date specified by 11:59:59 PM. A "day" is defined as a 24 hour period beginning at midnight, and includes Saturdays, Sundays, and holidays.

Incorrect command structure ..... **up to -10%**
In some projects there will be some type of command interface. This command interface is part of the specification, and must be followed to avoid loss of credit. Making the program more "user friendly" will not result in a higher score (it will likely lower your score). Following the specification will earn you the maximum number of points.

Unaligned output..... **up to -10%**
Assignments may require that your output be in tabular (table) format, with right justified values, and/or with decimal points aligned.

Examples (again, not an exhaustive list):

- You hand in a perfect program...
  - 1 day late.....99 points (-2^{1-1}= -1)
  - 2 days late...98 points (-2^{2-1}= -2)
  - 3 days late...96 points (-2^{3-1}= -4)
  - 4 days late...92 points (-2^{4-1}= -8)
  - 5 days late...84 points (-2^{5-1}= -16)
  - 6 days late...68 points (-2^{6-1}= -32)
  - 7 days late...36 points (-2^{7-1}= -64)
  - 8+ days late...0 points (-2^{8-1}= -128)
- You hand in a program 3 days late, and the decimal points in your output do not line up as the problem specifies...86 points (-4 late, -10 output)