Programming Assignments Guidelines

General Guidelines

There will be four assigned programming assignments (programming labs), each constituting 10% of the total grade. All programming assignments will consist of JAVA programs coded and executed by the student. The use of JAVA is required. Programming assignments are due on Tuesday midnight of the specified week and must be submitted to the course website. Assignments received after Tuesday midnight of the specified week are considered late and the grade will be reduced by 5 points. The grade will be reduced by an additional 5 points for each subsequent day that the assignment is late. Assignments more than 1 week late cannot be accepted, except by prior arrangement with the instructor. It is your responsibility to get your assignments in on time. Problems with your system do not constitute a legitimate excuse for late assignments, so make plans to deal with the unavailability of your system. All programs are to be turned in electronically. **Programs which do not produce some form of legitimate output cannot be accepted for grading.**

Please limit your JAVA to Standard JAVA, in particular your code should be Java 7 or 8 (JDK 1.7 or 1.8). The use of nonstandard language may be penalized, except for variations for file handling and timing. It is the student's responsibility to insure language compliance. Please check with the instructor, if you have a question.

Style

Your code must incorporate a consistent, well-documented style. Close attention will be given during grading to the comments. They should be clear, concise, and adequate. The use of substantive, descriptive blocks for each module as well as an introductory descriptive block for the entire **program IS REQUIRED**. Comments should explain the purpose of a function, its inputs, and outputs. They should also explain the algorithm being applied, a particular approach to a problem, or restrictions in using the module. Comments should NOT simply restate the code, e.g. comments along the lines of stating that something is being printed are not worthwhile.

Please note that the use of global variables is strongly discouraged and will be penalized if not well justified. Some other points of style include:

- Write one driver that executes your entire code.
- Consistent, well-delineated use of both upper and lower case is encouraged.
- Using indentation to show nesting of control statements is encouraged. Set tabs to only 4 spaces to reduce line wrap around.
- Keep line lengths to about 78 columns to reduce wrap around.
- Avoid confusing programming "tricks".
- Keep code modular. One page is a good rough guide to module size.
- Comments should describe the function performed, not restate the code in English. Use white space liberally.
- Do not use GOTOs.

Proper testing of your program should include checking extreme cases as well as any reasonable errors that might occur, due to typos or lack of user experience. When testing, approach your code as though you are a total novice, then on a second pass, as though you are an experienced end-user.

Input

Sometimes a handout for a lab will specify particular input cases. Your code must, of course, handle these correctly and you must turn in output to show that it does so. However, you are also expected to generate your own test cases during development. Proper testing of the program should include checking extreme cases as well as any errors that might reasonably occur.

Programming assignments must use named files to handle I/O and NOT use GUI or regular keyboard input. You will be penalized for GUI or keyboard input. However, you may, if properly specified in a README file, use redirection for output. There will be a small penalty if file names are hard coded in your program.

You will lose points for not providing adequate additional input. If you want credit for something your code does then it is important to demonstrate it with an appropriate I/O set.

A Complete Assignment

A complete lab assignment consists of the following turned in to the assignment item in the corresponding module, in a single zip file attachment, with your name as part of the file name: JSmithLab1.zip. Be sure to include your name inside each file, also. Do not paste content into the boxes in the assignment item.

1. **A written, analysis of the project, submitted as a PDF**. The analysis should include comments about what you learned, what you might do differently next time, justification of your design decision, issues of efficiency. etc. The handout for a specific assignment will usually indicate some more specific requirements. It is not necessary to reiterate the requirements of the assignment. This commentary will constitute 20% of your grade for each lab assignment. Please give this some thought. As a guideline, this should be 1-2 pages or longer. **Use Times New Roman Size 12 font, single spaced, with .75-inch margins on all sides.** There will be a deduction if the formatting requirements are not met.
2. **The source code**. Please upload all your source files to the web site. It is important to include a copy of all your files (any include file) except standard libraries. **You are expected to write your own code** except as provided in a specific assignment. Do NOT download code from the internet or other sources to be part of your assignment. Use of standard libraries is restricted to uses such as standard I/O calls and standard math functions, etc. In other words, you can't use the library stack code.
3. **The compiled code.** You should upload your compiled Java code, in case we have problems compiling your source code.
4. **Copies of all your input data sets** (as text files/s): this should be included although your output echoes the input. You should include the required input sets, as well as those you create beyond the minimum expected set. In most cases, the handouts will specify some required input, but you are expected to generate additional input sets to properly exercise your code. It is your responsibility to show that your code does all the things it is supposed to do and that all reasonable error cases are handled. You will be penalized for insufficient input.
5. **Copies of your output,** annotated as needed for clarity, in case we have problems compiling your source code. Be sure to include output for the required input cases. Output is separate and not to be incorporated into the analysis. Please be sure to include your name inside the file.

Grading on Programming Assignments

The grade for each lab assignment is broken down as follows:

- **40% - Correctness** - In problem solution and results.
- **20% - Style/Proper Coding** - Following a reasonable, consistent style with STRONG documentation, with appropriate use of structures, modularity, error checking, etc.
- **10% - Input/Output** - Labeled, formatted, correct use of prompts, correctly handles specified inputs and outputs as well as additional cases provided by the student, is user friendly.
- **20% - Analysis** - See item 1 above.
- **10% - Enhancements** - Recognition of superior work on one of the required aspects of the assignments or work above and beyond the requirements. If you do something extra, please make sure it is reflected in the I/O set so you get proper credit. Discuss it in your analysis. We can't give you credit for something unless we are aware of it. If you add an extra feature to your code, make sure it is "in addition to", not "in place of" a required component of the problem.

This grading policy is a reflection of the expectation that you can already write minimal, working code. If the unexpected comes up, please let me know. I will be happy to discuss your grade with you anytime.

Examples of Programming Assignment Grades:

94-100% - This is a very strong lab that correctly implements all the required elements, and includes corresponding example I/O cases, and uses a reasonable and consistent style. Each module has an introductory comment block giving an extensive, high level description of the module, detailed descriptive comments in the declarations sections and occasional detailed comments throughout the code. Error checking is strong and covers boundary conditions and additional cases beyond the minimum specified input. Output contain all the required elements, is user friendly, formatted in a visually pleasing manner, and contains useful descriptive statistics about the results. Extra features may have been added. The analysis addresses the design decisions incorporated into the code development and explicitly justifies the specific data structures used and the specific implementation used. It considers both the theoretical and observed efficiency and explains any discrepancy that may exist, using Big Oh and theta notation appropriately. It summarizes what the user learned in the lab, from language and structure specific experiences through big-picture concepts. It displays the writer's command of the topic in the assignment by identifying additional work that addresses known problems and/or continues to improve and expand the scope. It contains graphs or tables enabling the reader to more readily assimilate the results of the lab. It is formatted as required.

86-94% - This is good solid lab. Correctness errors, if any, are minimal. Documentation, style and error checking are good. Error checking may have some small omissions. There is some extra input, but less than might reasonably be expected for the problem as coded. Output looks fine and can be followed without any problem. There are probably no extra features, or just small, easy ones. The analysis is good, mostly formatted as required, but is less comprehensive. Some areas may be treated only superficially, or omitted altogether.

80-86% - This is a good lab, with room for improvement. There may be small correctness errors, sometimes due to a misunderstanding of the requirements. Error checking, style, and documentation are standard. Output minimally covers what it needs to. There are few, if any, extra input cases. The analysis does not do a very good job of justifying the design decisions and structure choices and may not be formatted as required. Other aspects of the analysis, especially algorithmic efficiency, may be omitted.

70-80% - This is a weak lab. It may do everything correctly but minimally, with no extras thrown in, minimal documentation, and a very sketchy, superficial analysis, probably not formatted as required. Alternatively, this may be a well done lab aimed for the categories above, with a moderately serious correctness error or a significant omission of a required component.

<70% - This is a lab with major errors in correctness and minimal efforts on style, and error checking. Documentation may be non-existent or extremely minimal. There are probably no additional I/O cases, no extra features. The analysis is probably under a page in length, not formatted as required, and addresses only one or two required topic areas.

At the discretion of the instructor, students may be permitted to resubmit a lab with problems in the area of correctness. Resubmissions on the analysis and enhancements section are not permitted.

Academic Integrity

IMPORTANT: You are expected to do your own work. Help from other sources must be acknowledged. It is okay to discuss the problem with others for perspective or to make sure you understand it correctly, but the code you write must be your own. Downloading code from other sources for the programming assignments while strongly discouraged should absolutely be properly accredited. It is prohibited except for the Lab on Sorting.

Practical Points

- Reread this before turning in lab.
- Reread the handout of the assignment before turning in assignment.
- Check that your analysis is correctly formatted.
- Turn in a README file, noting any special requirements.
- **Compile and execute your code to verify that it works properly.**
- Exercise the features of your code with an extensive I/O set.

- Fully document your code.
- Provide input files with the required input and your supplemental input.
- DO NOT turn in other's work.
- Provide output to all the required input cases and your supplemental cases.
- **Do NOT turn in an assignment that does not compile.**
- Consolidate everything in a single zip file, with your name as part of the file name: JSmithLab1.zip. I/O files should be text. The Analysis should be a PDF.
- If you find you need to resubmit, just go ahead and resubmit to the course website. It should be accepted, even after the due date.

Advice on Programming

- The single most important thing you can do to smooth the code development process and reduce the overall time required is to work out the entire problem on paper, including debugging. Resist the temptation to jump into the editor and type just to make yourself feel productive.
- Once you actually start to implement on the computer, adopt either a top-down or bottom up approach, and stick with it. Implement a single module and fully debug it before you go onto the next module. Then when you add a new module, if there are problems you can more easily locate them (in other words, if you combine two working pieces of code, any problems occurring have to be in the interface between them).
- Remember that a debugger is a piece of software and is subject to bugs. If you cannot find a problem, then consider debugging the old fashioned way with write statements.
- Back it up. Disk space is cheap and you cannot back up your code too often. It is very easy to lose track of all the changes and iterations, so consider a version numbering system like V2.1 where changing the 1st number represents a big change and changing the second number represents a minor change.
- The final piece of advice: "If it ain't broke, don't fix it". Less experienced programmers sometimes start changing things willy-nilly in a frustrated effort to address code problems they can't figure out. Get help, or at least save a copy, first.