ECE 2230 Programming Guide Fall 2018

Completion of a programming assignment involves more than simply turning in code that compiles and runs. In addition to submitting your code, you must also submit a **Test Plan** that shows how your code was tested to verify proper operation. For most assignments, you will also submit a **Test Log** that shows the output of your program and how that output supports your testing plan.

Grading will be based on four components

You must compile with **`gcc`** and use the **`-Wall`** flag. Code that does not compile or compiles but does not run will not be accepted. Points will be deducted for code that compiles but has compiler generated warning messages.

Code design (20%) We will evaluate organization of the interfaces in your program, how your program is broken into pieces, how the pieces communicate, what data structures and algorithms you decide to use for the pieces.

Documentation (20%) Equally important to good code design is ensuring that your code can be read and possibly modified by other people. We will grade on how well you have adapted a clear and meaningful style guide. See comments on a style guide below.

Testing (20%) For the programming assignments in this course it will be impractical to *prove* that the program is correct on all inputs. Instead, you must convince us that it works correctly, by testing the program on various inputs and *documenting* what you have done. It is generally not possible to exhaustively test every possible input and state. However, you should be able to verify that you have tested all the major functions and boundary conditions.

Correctness (40%) We will compile and run your code and test it against our set of test cases. We will also read your code to verify that it follows the requirements specified in the assignment.

Programming style guide

The file with the `main()` function must be named `labX.c` where `X` is the assignment number.

Include a comment block at the top of each file. The header comment must contain your name, the course number, semester, and assignment number. In addition the header comment must contain a section called "Purpose" in which you describe the purpose of the code in that file. *Be brief and informative.* Next, include a section called "Assumptions" and state the assumptions you make regarding the problem to be solved, the kinds of inputs you program will accept, and any deviations from the assignment requirements. Also, list any known bugs. Specific assumptions regarding particular functions should be expressed in comment blocks above the relevant functions.

You must include a brief comment block at the top of each function and structure definition stating its purpose. For a function, you must define the purpose of each input and the values that are valid for the input(s). You must also define the results that are returned, and provide comments about any data structures that are changed by the function.

Use inline comments sparingly but whenever necessary to explain critical portions of the code.

Write self-documenting code. Choose meaningful names for all variables, parameters, and functions. Use complete words instead of abbreviations where practical. If a constant is needed more than one time, use a named constant. The size of an array should **always** be defined using a named constant.

Structure definitions should be kept in the same file (or files) as the ADT they are designed for. For a particular ADT, group all functions for that ADT in the same file, and do not include functions not associated with that ADT in this file.

A `.c` file must never be `#included` into another file. Only a `.h` file can be `#included`.

A `.h` file may only contain type and structure definitions, global variable definitions, macros, function prototypes, and inline functions.

Global variables are to be avoided unless a **very** convincing argument is included as to why the variable must be global. You must obtain instructors approval to use a global variable before you submit a solution.

An ADT must have exactly the interface defined either in the assignment or as discussed in class. Extra methods are allowed, but they must be documented as extensions.

Use common sense. This is just a guide, and your primary concern should be in making sure that others can read and understand the text of your program.

Testing guide

For every programming assignment, you will submit a test plan. Each assignment will also specify if you need to submit a test log or other documentation concerning the performance of your program. A **test plan** describes how you have tested your program, which inputs have been tested, and why the tests are good choices. Often this is a commented test script that also serves as input to your program. When the assignment requires testing results to be submitted, you will often use the results of your test plan. The output should be saved in a **test log,** which shows the results produced by the tests in the test plan. The log should be annotated to show that each test produced the output predicted by the test plan.

Include comments at the top of each file. The header comment should contain your name, the course number, semester, and assignment number. Be sure to give a rationale for each test case.

You must use `valgrind` for final testing for problems related to memory leaks, array boundary errors, uninitialized use of memory, etc. See http://valgrind.org/docs/manual/quick-start.html for a quick introduction. Install `valgrind` using the Ubuntu package manager or software center.