C Coding Guidelines for CS240 (from Dr. Cheung)

1. Each file should start with a comment which includes your name, the date you wrote the file, the name of the file.
2. All functions, constants, and types that are intended to be used outside the file should be declared in a header file. Static functions, file-local constants and types should be declared at the top of the .c file.
3. Unless the name of the function makes it obvious what the function does (and this should be your goal when choosing a name), you should add a comment describing the use and purpose of the function.
4. Choose identifier names with care. Proper identifiers naming is an important way to help readers understand your code. In particular, except for indices (where generic *i*, *j*, etc. may be appropriate) identifiers should rarely have one-letter names. Very long names can also be a nuisance as they can make programs hard to format. Name selection is non-trivial. By K&R convention, variable names start with a lowercase letter, type names (defined by typedef) with an uppercase letter.
5. Except for trivial constants (say *0*, *1*), it is much better to use *#define* and set an identifier (all capital letters) equal to the constant than to use the constant in code directly (a practice known as “wired-in numbers”). This allows you to aid the reader, as the name you choose for the constant should help him/her figure out what the constant represents. Also (in many cases) when you need to

change the constant you can do so by changing its value in one place, rather than in many places in the code. You can use *const* instead of *#define* to declare constants (though this is more the custom in *C++* than in *C*).

6. Minimize use of global variables. When you do use one, define it in a .c file, not in a header file, since header files are meant to be included in multiple .c files and each of them will try to define the variable. Not good. You can put an extern declaration for it in a header if necessary. Try to keep to static file-global variables, the C equivalent of Java class variables, plus local variables.
7. Avoid long functions (over about 60 lines). Use helper functions.
8. Avoid repeated code. Use functions to 'factor' out the repeated code. Repeated code is a serious error.
9. Use proper indentation. Poorly indented code is difficult to understand. Emacs knows how to do this.
10. No *goto*'s.
11. Comment anything that is not obvious. Or better, rewrite the code so that it becomes obvious.
12. Use *for* and *while* appropriately. Generally use *for* when you have an index that changes each iteration. Use *while* when you don't.
13. When possible use *switch* instead of complex if-then-else's.
14. Use standard C library functions (strcpy, sprintf, sscanf, etc.) when you are permitted to use them.