Coding standards – CS237

1. **Style:** Good programmers write beautiful code. Pay attention to readability. You want me to be able to read and understand your code, and you want to be able to read and understand it, too.

   1.1. If you are modifying existing code, you should follow the style used in that code.

   1.2. Choose meaningful names for quantities such as variables, constants, structs and functions. The names should be succinct but should be suggestive of the purpose of what they name.
      1.2.1. Naming conventions often follow one of two styles. Either you capitalize each new word in a name (e.g. `gradePointAverage`) or you put underscores between each new word in a name (e.g. `grade_point_average`). Choose (or match if modifying existing code) one approach and use it consistently.
      1.2.2. Non-constant variable names should begin with a lowercase letter.

   1.3. Choose an indentation style and stick with it. Editors and IDEs (e.g. emacs and NetBeans) understand how to indent a program. Take advantage of this feature. Add a level of indentation for each level of nesting of control structures in your program. Never put code after an opening bracket ('{') or code before a closing bracket ('}').

   1.4. Constant variable names (i.e. `#define`) should be all uppercase letters.

   1.5. Use blank lines to visually set off sections of your program. Adding vertical space at appropriate places makes programs look less crowded and makes them easier to read. Good candidates for additional vertical space are
      1.5.1. between methods, after blocks of related declarations,
      1.5.2. before and after control structures like if, while or for statements,
      1.5.3. and after any sequence of statements that are part of some recognizable part or phase of the computation your program performs.
      1.5.4. before every line that contains only a comment.

   1.6. Use horizontal whitespace within a line to make it easier to read your code. Good style places whitespace around
      1.6.1. operators,
      1.6.2. variables,
      1.6.3. arguments, and
      1.6.4. function calls.

   1.7. Horizontal lines should be no more than 80 characters to prevent lines wrapping to the next line.

2. **More Style:** Good programmers write understandable code. Documentation is vital.

   2.1. If the name of a struct data field or a local variable is not completely self-explanatory, write a comment that describes what it will be used for. (This is not necessary for loop control variables.)

   2.2. Every function you write should begin with a comment that describes the following:
      2.2.1. the purpose of the function,
      2.2.2. what the function expects when it is called (i.e. preconditions),
      2.2.3. what the programmer can expect when the function terminates (i.e. postconditions),

2.2.4. what each parameter is used for,
2.2.5. and what the function returns, if anything.
When documenting functions, these comments should appear both in the header file where the prototypes are declared and in the implementation file where they are defined. (The header file is usually available for inspection by the authors of client programs; the implementation file may not be, but you want to be able to easily refer to the description of the function while you are writing or modifying its definition.)

2.3. Complex functions should contain internal documentation. Any portion of your code whose purpose isn't completely obvious (not to you, but rather to someone else reading your code) should have an explanatory comment. Things like incrementing a variable or reading a value from the keyboard don't require comments; mathematical formulas, if/else or for/while statements, and any reasonably complicated sequence of steps should have a comment. Logical groups of code should be **preceded** with a line (or several lines) of comments explaining what the logical group of code does at a high level. Design decisions (why one algorithm was chosen over another, why a certain data structure is appropriate, why a file format or input format was chosen, etc.) should also be explained in the code. Please note that **not** every line will need a comment and over-commenting can reduce code readability.

2.4. Every struct you write should begin with a comment that describes what the struct represents.

2.5. Comments on function and struct declarations should NOT reveal implementation details. They should provide a general description of what the functions and structs do or represent.

3. **Design**: Good programmers write well-designed, modular, maintainable code.
   3.1. Functions should perform a single action. If more than one action needs to be performed, several simple functions should be created, where each function performs a single action, and then all of those functions should be called sequentially. For example, if you want to sort an array and then print it, two functions should be created (one for sorting and one for printing).

   3.2. Use defined constants for widely used values that may be subject to change, or where using a meaningful name would improve the understandability of code when used in place of a cryptic literal value.

   3.3. Parallel arrays are seldom the easiest or safest way to deal with several pieces of data of different types that are logically related. Learn how to construct simple structs for data aggregation purposes.

   3.4. Global variables should **not** generally be used. Static variables should be used sparingly. Non-constant static variables should be rare and provide a very specific and necessary functionality that could not be obtained without using a static non-constant variable.

   3.5. Do not declare subfunctions (functions declared inside other function definitions) in C.

   3.6. Do not include .c files using the `#include` directive.