DFCS - C Programming Standards

Compilers don't need comments, descriptive variable names or structured code to compile a program. On the other hand, programmers ***must*** have well written code in order to understand and effectively maintain programs. Therefore, adhering to the programming standards described in this handout is important for developing good C code. In addition, a portion of your grade for programming exercises (PEX's) will be based on good coding style.

1. **Global Program File (*.c) Structure**

   - Lay out the elements of your programs in the following order:
     a. program file heading
     b. `#include` preprocessor directives
     c. main function (if applicable)
     d. other function definitions

2. **Global Header File (*.h) Structure**

   - Lay out the elements of your header files in the following order:
     a. program file heading
     b. `#include` preprocessor directives
     c. `#define` preprocessor directives
     d. type definitions
     e. function prototypes

3. **Program File Heading**

   - Each code file (both .c and .h files) must have a "comment header block" at the top of the file.
   - Here is a template:

```
/** pex1.c  Put your file name first
 * ============================================================
 * Name: <your name here, followed by a date>
 * Section: <your section here!>
 * Project: <put the assignment information here>
 * Purpose: <high level description of purpose of the program
 * ============================================================
 */
```

4. **Function Header**

   - Each function within a C source code file must have descriptive comments directly preceding the function definition.
   - Here is a template:

```
/** ------------------------------------------------------------
 * <high level description of purpose of the function
 * including what the parameters are used for>
 * @param <explanation of each of the function parameters>
 * @return <explanation of what the function returns>
 */
```

5. Variable/Constant Declarations

- Each variable or constant declaration must be on a separate line.
- If the name of an identifier is not explicitly clear, it must be clarified using a comment.
  Here are some examples of clear identifiers:

```
int widthInPixels;
float milesFromCenter;
```

  Here are some examples that need a comment for clarification:

```
int with;       // in pixels
float distance; // from the center in miles
```

6. Variable, Constant, and Function Names

- Each variable, constant, and function name should be descriptive of its use or purpose.
- Using single letter variable names, such as `j`, `k`, or `n` for loop counters is OK.
- All identifier names must use a consistent "`camelCase`" style.
  - All identifiers and function names must use "camel Case" notation that start with a lower case letter with the first letter of each new "word" in uppercase.
    For example, "`computeInterestRate`".
  - All typedefs that create a new data type name must use "`camelCase`" notation that starts with an upper case letter.
    For example, "`MyNewType`"
- File names will use "camel Case" notation as well.
  - If a file contains the definition of a new data type, capitalize the first letter of the file name. All other file names should start with a lower case letter.
- Make constant names all uppercase, and use an underscore to separate words:

```
const int SCREEN_HEIGHT = 25; // height of screen in rows
```

7. Formatting

- Use indentation/spacing to indicate logical structure (selection, iteration, sequence) of code. In CLion, `Code → Reformat Code` will do this for you automatically.
- Separate related code blocks with a single blank line.
- Include white-space between operators and operands to improve the "readability" of your code.
- Each function should be logically cohesive; all of the code in a given function should work together toward a common, narrowly focused purpose.
- No line in your program should exceed 100 characters (indentation included).
- Be consistent with the braces used to mark compound statements (i.e., the bodies of functions, if statements, and for loops). Place the beginning brace on the line that starts the statement block, and the last brace in the starting column of the initial statement. Here is an example

```
for (j=0; j < n; j++) {
   printf("%d\n", j);
}
```

- Minimize the use of magic numbers – always use *named constants* instead.