C Code Style Guidelines

- Use good **modular design**. Think carefully about the functions and data structures that you are creating before you start writing code.
- Use good **error detection** and handling. Always check return values from functions, and handle errors appropriately (SEGFAULT is not appropriate error handling).
- Every time you **dynamically allocate memory** (e.g. call malloc), you should have code to free this memory at some other point in your program.
- As a general guide, **no function should be longer than a page long**. Of course there are exceptions, but these should truly be the exception.
- **Use descriptive names** for variables, functions, and constants You don't want to make function and variable names too long, but they should be descriptive (e.g. use "getRadius" or "get_radius" rather than "foo" for a function that returns the value of the radius of a circle). Also, stick with C-style naming conventions (e.g. i and j for loop counter variables).
- **Pick a capitalization style** for function names, local variable names, global variable names, and stick with it. For example, for function name style you could do something like "square_the_biggest" or "squareTheBiggest" or "SquareTheBiggest".
- **Define Constants and use them** in your program rather than using numerical values. Constants make your code more readable, and easier to change (if you decide MAX should be 100 instead of 50, just change the constant definition rather than finding all uses of 50 in your program and trying to decide which 50's correspond to max value and changing just those 50's). For example:

```
Do this:                              Not this:
--------                              --------
#define MAX   50

int  buf[MAX];                        int buf[50];
if( i< MAX) { ... }                   if( i < 50) { ... }
```

- **Avoid using global variables**; instead, pass variables by reference to functions that change their value.
- The **main function should not contain low-level details**. It should be a high-level overview of your solution (remember **top-down design**).
- **Use good indentation**. Bodies of functions, loops, if-else stmts, etc. should be indented, and statements within the same body-level should be indented the same amount:

```
int blah(int x, int y) {
  stmt1;
  stmt2;
  while (...) {
    stmt3;
    stmt4;
    if(...) {
      stmt5;
      stmt6;
      stmt7;
```

```
    } else {
      stmt8;
    } stmt9;
  }
  stmt10;
}
```

vim will auto-indent C code for you. You can also re-format C source in vim by puting the cursor at the start of the line you want to select to start reformating and scroll down (or shift-G to select to the end) and then hit = (SHIFT-V, SHIFT-G, =) (visual select, go to last line, re-format)

- **Line Length**: your source code should **not contain lines that are longer than 80 characters long**. If you have a line that is longer than 80 character, break it up into multiple lines. Here is an example of how to break up a long boolean expression into three lines:

```
if ( ((blah[i] < 0 ) && (grr[j] > 234))
     || ((blah[j] == 3456) && (grr[i] <= 4444))
     || ((blah[i] > 10) && (grr[j] == 3333))
   )
{
        stmt1;
        stmt2;
} else { ...
```

Here is an example of breaking up a long comment into multiple lines:

```
x = foo(x);     /* compute the value of the next prime number  */
                /* that is larger than x  (foo is a really bad */
                /* choice for this function's name)            */
```

Here are two ways to break up a long string constant:

```
printf("here is a really long string with an int value %d", x);
printf(" that I want to print to stdout\n");
```

OR

```
printf("here is a really long string with an int value %d"
       " that I want to print to stdout\n", x);
```

The easiest way to make sure you are not adding lines longer than 80 characters wide is to always work inside a window that is exactly 80 characters wide. If your line starts wrapping around to the next line, its too long.

- **Comment your code!**

  File Comments: Every .h and .c file should have a high-level comment at the top describing the file's contents, and should include your name(s) and the date.

  Function Comments: Every function (in both the .h and the .c files) should have a comment describing:

  1. what function does;
  2. what its parameter values are
  3. what values it returns (if a function returns one type of value usually, and another value to indicate an error, your comment should describe both of these types of return values).

  In header files, function comments are for the user of the interface. In a source file, function comments are for readers of the implementation of that function. Because of this, function comments in C source files often additionally include a description of how the function is implemented. In particular, if a function

implements a complicated algorithm, its comment may describe the main steps of the algorithm.

My advice on writing function comments: write the function's comment first, then write the function code. For complicated functions, having a comment that lists the steps of the algorithm, will help you

When commenting stick to a particular style. For example:

```
/*
 * Function:  approx_pi
 * --------------------
 * computes an approximation of pi using:
 *    pi/6 = 1/2 + (1/2 x 3/4) 1/5 (1/2)^3  + (1/2 x 3/4 x 5/6) 1/7 (1/2)^5 +
 *
 *  n: number of terms in the series to sum
 *
 *  returns: the approximate value of pi obtained by suming the first n terms
 *           in the above series
 *           returns zero on error (if n is non-positive)
 */

double approx_pi(int n) {
  ...

     (note: for this function, I'd likely have in-line comments describing
            how I'm computing each part of the next term in the series)


/*
 * Function: square_the_biggest
 * ----------------------------
 *   Returns the square of the largest of its two input values
 *
 *   n1: one real value
 *   n2: the other real value
 *
 *   returns: the square of the larger of n1 and n2
 */

double square_the_biggest(float n1, float n2) {
 ...
```

In-line Comments: Any complicated, tricky, or ugly code sequences in the function body should contain in-line comments describing what it does (here is where using good function and variable names can save you from having to add comments).

Inline comments are important around complicated parts of your code, but it is important to not go nuts here; over-commenting your code can be as bad as under-commenting it. Avoid commenting the obvious. Your choice of good function and variable names should make much of your code readable. For example, a comment like the following is unnecessary as it adds no information that is not already obvious from the C code itself, and it can obscure the truley important comments in your code:

```
        x = x + 1;  /* increment the value of x */
```