C Programming Style Guide

1. Modularization.

Separate your program into independent, self-complete modules. We will talk more about this later in the semester.

2. Consistency

Make your style as consistent as possible. The more consistent your style is, the easier it is to read and understand. Moreover, the less likely mistakes will be made. The more consistent your style is, the more automatic it will become. As a result, you'll spend less time thinking about program style and more about the problem at hand.

3. Make the indenting of your code reflect the structure of the program.

For example,

```
 if (count == 0) printf(‘‘No data.\n’’);
```

fails to match the structure of the program. To do this it's better to write

```
if (count == 0)
  printf(‘‘No data.\n’’);
```

Here's another example. Does the indentation of the code fragment below reflect its structure?

```
if (count != 0)
  printf(‘‘%d\n’’, count);
  average = total/count;
```

4. Make the reading of the program flow vertically.

This is the natural flow of reading. Things that get in the way of or detract from this tend to make a program harder to read.

5. Don't put more than one variable in a variable declaration.

DO:
```
int i,
    j;
```

DON'T:
```
int i, j;
```

6. Line up names in variable declarations.

```
int    i;
float  f;
char   c;
```

If you do, the variables will be easier to reference by a reader.

7. Line up the braces of a compound statement.

Two acceptable styles are shown below. I tend to prefer the second one.

```
                   /* Style 1 */
if (...)                            while (...)
{                                   {
     .                                   .
     .                                   .
     .                                   .
}                                    }
else
{
     .
     .
     .
}

                     /* Style 2 */

if (...) {                          while (...) {
     .                                   .
     .                                   .
     .                                   .
}                                   }
else {
     .
     .
     .
}
```

8. Localize code.

The things that the proper operation of a function depend upon should, as much as possible, be together. A worthy goal is for each function to be independently understandable, in which case the reader of such a function will not have to memorize large and separated quantities of code to understand what is going on. Another way to say this: Modularize your code.

9. No global variables.

If you think you need a global variable, pass it as a parameter instead.

10. Properly comment your code.

Comments are meant to help the reader of a program. They should add something that is not immediately evident from the code, or collect into one place information that is spread from the source. Comments do not help by saying things the code already plainly says, or by contradicting the code. Here are some of my favorite *useless* comments.

```
/* return 0 */
return 0;

i++; /* increment i by 1 */

/* initialize i to 0 */
i = 0;

/* Add 5 to x */
x = 6;
```

An outline for your programming assignments:

```
/*********************************************************
 *
 * COP 3502 – Spring 2003
 * LAB SECTION:
 * ASSIGNMENT NO:
 * NAME :
 * DATE :
 *
 *********************************************************/

/*********************************************************
 *  Description of the program
 *
 *********************************************************/

/* List all of the header files used here. */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* List the function prototypes */
```

```
void f1 (int x, int y);
int f2 ();

/* Describe the purpose of f1 */
void f1 (int x, int y)
{
     .
     .
     .
}

/* Describe the purpose of f2 */
int f2 ()
{
     .
     .
     .
}

int main (void)
{
   /* Indent the declarations and the body of the functions
      by at least 3 spaces. */

     int x; /* If necessary, give a short description of a
               variable’s purpose.*/
     int y; /* Give meaningful, but short, variable names. */
          .
          .
          .
    /* if, while, for, do-while, and switch statements must
     * use the style shown in the class notes or
     * programming style guide. Here’s an example.
     */

     /* Leave blank lines before and after the loops */

     while (...) {
          .
          .
          .
     }    /* denote the end of loop with a comment */

     if (...) {
          .
          .
          .
```

```
        }
        else {
                .
                .
                .
        }

        return 0;
}
```