C Programming Style Guide

For all of your programming assignments be sure to copy-and-paste your program output to the bottom of the source file using block-style (/* … */) comments. The following style guide is to be followed to create programs that are easier to read, understand and maintain.

Comment Header

Take credit for your work. Every program that you write should have a leading comment block that identifies the author and the purpose of the program. This is called the Header. Use C style comments (//). Reserve the use of /*…*/ for commenting out large chunks of code.

```
// Author:                 <Your name here>
// Assignment Number:      Lab <number>
// File Name:              L<lab number>_<lastname>.zip
// Course/Section:         CS 2123 Section <your section number here>
// Date:                   <date goes here>
// Instructor:             <your instructor's name here>
```

Identifier Naming

Choosing good identifier names is more of an art than a science, but there are some guidelines that may help.

Abbreviate with care. Don't think you're saving the reader by removing all the vowels. Some abbreviations (min , max , num , ptr -- for "pointer") are so common that they're considered OK. When combining several words, capitalize the first letter of each word after the first word for variables (camelCase). Constants should be all capitals with an underscore separating the combination of more than one word. Most declarations should have only one identifier per line with a comment explaining the purpose of the identifier. **No Global variables are allowed!**

```
char selection;                    // menu selection from the user
int i, j, k;                       // loop counters
double average;                    // average grade
const int TEN = 10;                // multiplier
const double SALES_TAX =  0.0825;  // amount of state sales tax
```

Avoid one letter variable names. Identifiers should be assigned meaningful names. Names like X, Y or Z, give no insight into how the identifier or the types of values that can be assigned. Although it may take a little longer to type out a descriptive variable name, doing so can save hours of frustration when trying to correct a program that doesn't work or when trying to modify a program that you haven't worked on for some period of time. One exception to the descriptive naming rule is that variable used solely for iteration purposes (an index in a for loop), may be named with single letters. Preferred letters for this purpose are i,j and k.

Boolean variables and boolean function names should state a fact that can be true or false. That way, they will read smoothly in an **if()** or **while()** statement. Use of the verb "is" can be quite helpful. For example:

```
if (stringIsMissing)
```

Line Wrapping
Try to avoid this when at all possible, but sometimes there's nothing that can be done to avoid this. When it happens, there are two options:

1. Write helper functions or use temporary variables to hold intermediate results.
2. Break the expression up over several lines. Do this at the operator boundary.

```
tax = (income * TAX_ RATE)
    + stateTax + localTax
    - (numberOfDependents * deduction);
```

(Note that the operators are on the left and that the continued expression is indented.)

When **scanf** and **printf** statements get too long they can also be broken up. (Note where the semicolon falls.)

Expressions
All operators should be surrounded by spaces, with the exception of unary operators (operators that work on just one argument.) This includes ++, --, !, etc.

```
sum = a + b;
sum += 12;
```

Functions
You should always document your functions by writing comments that describe what they do. These comments should appear just before the function definition.

Indentation
To improve the readability of programs, and to aid you in correcting your programs, there should be indentation that indicates different blocks of statements. *Comments after closing } are optional.*

```
#include <stdio.h>         /* printf, scanf definitions */
#define KMS_PER_MILE 1.609 /* conversion constant */

int main(void) {
    float miles,   // input - distance in miles
          kms;     // output - distance in kilometres

/* Get the distance in miles */
    printf("Enter distance in miles: ");
    scanf("%f", &miles);

// Convert the distance to kilometres
```

```
    kms = KMS_PER_MILE * miles;

// Display the distance in kilometres

    printf("That equals %9.2f km.\n", kms);
    return 0;
}
```