Code Format Notes

My style developed over the years, influenced by a few different languages. Most of my formatting style comes from Pascal. It was my first truly modular language. Before then I had written code in FORTRAN, BASIC, assembly language, and Forth.

There are more than eight different formatting schemes. Only ONE of them is correct for each programmer. I use a variation on the Whitesmiths system, which is about third in popularity. This format was reinforced on my Macintosh computer, which used APIs built on a Pascal language base. However, I had learned Pascal a few years earlier which made reading the code easy. With time, this formatting style became second nature. It allows me to ‘see’ the code better.

I like tabs set to 3 spaces. I can notice this visual offset quickly. I find 2 spaces is too small, while any more than 3 is wasteful of screen space. However, you should try to avoid writing code with more than four or five levels of indentation. If you require that much indentation you need to factor your code into smaller functions. There may be times when you have to go deeper, but be careful, you are adding to the complexity of debugging. If you write simpler code you will spend less time debugging it.

C requires braces around multi-line statements. I like to keep the braces at the same indentation level as the code it encloses. This gives me a nice visual chunk to examine. Placing the braces at the previous indentation level just confuses me. Do I look here, or do I look there, simply wastes my time thinking about useless matters.

I use white space between chunks of code. For instance, at the very top of each file is an area for #include statements, followed by #define statements, then global (to this file) variable declarations. I put a blank line between each type for visual separation. I also use blank lines between "phrases" of my code. All of this work makes the code easier to read. It is very simple to write obfuscated code, simply ignore any formatting conventions.

I add a block of comments preceding complex functions, so I can read them before I read the code. I also add comments inside a function, either before a line, or following the statement on the same line with the // comment characters. If I want to comment out chunks of code I will use the /* */ comment form. The // comments can nest within the /* */ comments without a problem.

I set the conditional compilation commands flush with the left edge so the blocks are prominent. This helps me recognize the different program paths. The #define flags are always capitilized to show their type; it also allows me to find them more easily.