Introduction

The Linux style guidelines that we have pointed to you provide an excellent base for formatting your C code. Unfortunately it doesn't exactly describe some edge cases in styling your code. This document was creating to clarify a few things and fill in these gaps in the official Linux style guidelines.

Comments

C99 style comments are perfectly valid for inline documentation of code. Don’t use them for comments before functions, however, as those should be different as they’re important comments. Use the standard /* */ comments for commenting functions.

The reason for using C99 ("//") comments within code is so that you can comment out large sections of code with multiline comments. This is useful in debugging code or testing different variations of the same code. And this is a more common occurrence than you’d think!

For-loops

For-loops are a fundamental part of most programs. They also have very specific guidelines for their formatting. Like all control statements there should be a space between the “for” and the opening parenthesis. Each semi-colon within the for-statement itself will also get its own space afterwards. The expressions done within the for-statement also get their normal spacing. The end of the line will have an opening curly-brace to start the for-loop off.

Here is a properly formatted for-loop declaration:

```c
for (j = 0; j < 22; j++) {
    // CODE GOES HERE!
}
```

Statements

Statements in C are what are generally thought of as a line of code: small sections of code that end with a semi-colon. These can contain variables, function calls, operations, or compound expressions of all of these. The cardinal rule with statements is that you should never have more than one on a line. This means there should only be one semi-colon per line (with the exception of the for-statement).

As an example this is a poorly formatted line of code:
int j = 10; q = sqrt(j);

Array initialization

Arrays are initialized with comma-separated values within a pair of curly-braces:

int nums[4] = {1, 2, 3, 4};

This list of values should not have a space after the opening curly-brace or before the closing curly-brace. The main reason for this is that it’s unnecessary and doesn’t increase readability. It also makes multi-dimensional arrays easier to read without that extra spacing:

int nums[3][3] = {{1, 1, 1}, {2, 2, 2}, {3, 3, 3}};

80-column lines

The Linux style guidelines stipulate an 80-column width to your code. Generally a column means a single character, but that’s not quite always the case. Newline characters don’t count (the return at the end of a line) and tabs count for 8 (generally displayed as 4 characters in text editors).

The column count is generally displayed at the bottom of the text editor next to the line number. This is where it’s shown in both Notepad++ and MPLAB. Now the only problem with this number is that it is normally counting a tab as only four characters when they should really be 8. To change this right-click on an open text file and select Properties->’C’ File types and set the tab size to 8. Then the column reading at the bottom will be correct. Another option is to set MPLAB to use Notepad++ as your editor as it’s easier to change settings in that once than to change it in MPLAB for every project.

Now all lines in C should be no longer than 80 columns. And if you split a line to be shorter than 80 columns then all of the resulting lines should meet this limit also.

Naming conventions

Naming conventions is another important part of proper formatting. For us we will use the camelCase convention for variables and filenames, Pascal-case for function names, and ALLCAPS for constants. camelCase is when all words are pushed together and the first letter of each word is capitalized save the first one. Pascal-case is similar but capitalizes the first word. ALLCAPS is self-explanatory.

<table>
<thead>
<tr>
<th>camelCase</th>
<th>expectedValue, stdIO, iAmAVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pascal-case</td>
<td>Init, ToString, PrintToScreen</td>
</tr>
<tr>
<td>ALLCAP</td>
<td>SIZE, FP_DELTA</td>
</tr>
</tbody>
</table>

Naming conventions extend to the actual names you use. The variable should effectively and briefly describe what the variable is holding. expectedValue is a good variable name, though you wouldn’t want to name a variable something much longer than that. You may use tmp, temp, scratch for variables that are just temporary holding values but are not used for more than a couple of lines. Variables that are a single letter such as i, j,
k, and l are commonly integers that are used in for or while loops. Do not use single-letter or double-letter variables outside of loops!