C Style Guidelines

NOTE: There are code examples at the end of this page

1. **Meaningful names for variables, constants and functions. Do not use camelcase;** use underscores for multi-word variables. For example, use hot_water_temperature instead of hotWaterTemperature. Using a variable name with a single character is only appropriate if the variable is the iteration variable of a loop; otherwise using a single character variable is not correct. For example, if a variable represents a sum, name it **sum** instead of **s**.
2. **Good indentation (3 or 4 spaces). Use a proper editor (e.g., emacs, vi) that assists with indentation.**
3. **If variables have the same type, declare them on the same line if possible.**
4. Leave one blank line between variable declarations and the first line of code in a function.
5. Consistent style (e.g., use of curly brackets).
6. **Opening brace must be on the same line as conditional or function.**
7. Define values as constants when needed (do not use variables for constants). Do not use numbers in your expressions if those numbers have a special meaning (e.g., 3.1415); instead define constants (e.g., using #define) for them.
8. **#defined constants must be in uppercase (e.g., #define MAX_LEN 10 and NOT #define max_len 10).**
9. In your code you should leave one blank space between operators (e.g., x = 5 + 7).
10. Leave one space after a comma.
11. If the code is too complicated to be read on its own, simplify/split/rename variables.
12. **Use braces; avoid loops and conditionals without them.**
13. Use parentheses for clarity, especially with bit operations.
14. **Avoid global variables where they are unnecessary.**
15. **If p is a pointer to a structure, you should access members by using p->member rather than (*p).member**.
16. **If p is a pointer to a structure, do not surround the -> operator with spaces (use p->id instead of p -> id).**
17. You should avoid source lines exceeding 80 characters. You can use the **linecheck** command in the grace system to verify you have lines with the appropriate length.
18. Use the indent utility (after setting the alias as described in the grace info folder indent_utility_info.txt) and run it against a copy of your code. Compare the indentation generated by the utility and your original code.
19. **You must avoid code duplication.**
20. Make sure you read all the information on this page. You will lose credit if you don't follow the suggestions we have defined.

Take a look at the examples below, so you know what we are expecting when it comes to style.

Regarding Indentation

For this course, make sure your indentation is OK in emacs. Graders will use that particular editor to verify you are using good indentation. One reason why indentation can look different in several editors is the tab settings.

Code Organization

- **Use the universal convention to organize your code:**
    1. #include <>
    2. #include ""
    3. #defines
    4. Data Types (e.g., structures)

5. Globals
6. Prototypes
7. Code

- The main() function is either first or last.
- #includes and #defines in the middle of code are usually frowned upon.

Functions

- **You must avoid code duplication by calling appropriate functions (rather than cutting and pasting code).**
- **Input parameters must appear before out parameters.**
- **Annotate helper functions with static.**
- Annotate unmodified parameters with const.
- And annotate functions intended to be used from outside with extern.
- Functions should represent a reasonable unit of complexity or be reused frequently.
- If the parameter list extends beyond 80 characters, arrange parameters line by line. Here is an example:

```
void process(int my_first_value, int temperature, float pressure,
             int altitude, char color) {
   /* Code here */
}
```

Comments

- **All code must have some comments (there is no such thing as self-documenting code).**
- Describe the intent of a block of code.
- A description should appear at the top of each function.
- Describe the use of a variable where it is declared, if not obvious.

Miscellaneous

- **Use assert** - Macro that prints error message and terminates the program if the provided expression is false (e.g., assert(x != 0)). Assert can help with readability, mechanically exposing pre and post conditions more strongly than comments (which can become obsolete). Keep in mind that aborting execution may be inappropriate for submitted tests. We likely expect an error message.
- About return statements:
  - Return at the end is OK.
  - Return at the beginning for erroneous input is OK.
  - Return in the middle of a loop is sketchy.
- About loops:
  - If a loop should end by a condition, use loop control rather than break.
  - Use a for loop when initialization, comparison, and increment are appropriate (not a while loop).
- Avoid using the indent command as a last step; it can undo any manual beautification.

Examples

- style_example1.c (styleExamples/style_example1.c)
- style_example2.c (styleExamples/style_example2.c)
- style_example3.c (styleExamples/style_example3.c)

Style Notes

Additional notes can be found at style notes (styleExamples/style_notes.txt).

References

- Dr. Neil Spring (http://www.cs.umd.edu/~nspring (http://www.cs.umd.edu/~nspring))
- "C Programming Style" Paul Krzyzanowski (http://www.pk.org/rutgers/notes/pdf/Cstyle.pdf)
- http://www.cs.arizona.edu/~mccann/cstyle.html (http://www.cs.arizona.edu/~mccann/cstyle.html)
- http://books.openlibra.com/pdf/c_handbook.pdf (http://books.openlibra.com/pdf/c_handbook.pdf)
- For C++ (http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml)
- http://users.ece.cmu.edu/~eno/coding/CCodingStandard.html (http://users.ece.cmu.edu/~eno/coding/CCodingStandard.html)