C Coding Style
EEE243

Overview

This document specifies the required coding style for EEE243.

Code must always be written to maximize clarity for a reader. This includes well-chosen names for constants, variables, and functions; consistent formatting, and appropriate hierarchical program structure. If you are thoughtful about your code structure and naming, you will remove the need for most inline comments as your code will be self-explanatory.

Errors and warnings

Your code must not display any errors or warnings in CLion and must compile without errors or warnings. If there's a green check mark at the top right of the CLion editor pane, and no warnings or errors show in the build output when you to a complete build of your project (clean, then build), your code meets this requirement.

Formatting and indentation

Follow the default CLion style for formatting and indentation. You can auto-format your code any time by using the menu command *Code > Reformat Code* or *Code > Reformat File*. The equivalent keyboard commands (Ctrl-Alt-L and Ctrl-Alt-Shift-L on Windows) are well worth memorizing.

Names

Names must always be chosen to clearly indicate the purpose of the item named. In the particular case of loop indices, short names like `i` and `j` are acceptable. Names must follow the following conventions:

- **Variable and function parameter names:** all lower-case except the start of successive words are capitalized, as in `studentName`. Normally nouns. This style is called "camel case".
- **Function names:** camel case.
  - where the function's primary purpose is to compute a value, the name should be the value computed, like `averageGrade`.
  - where the function's primary purpose is its side effects, the name should be a verb like `updateStudentRecord`.

- **preprocessor constants and macros (created with `#define`)**: upper-case, words separated with underscores, like `TICKS_PER_CM`. This style is called "capitalized snake case".
- **`structs`, `typedefs` and `enums`**: capitalized camel case, like `StudentRecord`, normally nouns.

Numeric constants ("magic numbers")

Literal numbers other than 0, 1, 2, and -1 should not normally appear in the body of your code. All other numbers in your code must be given descriptive constant names using the `#define` directive, and the constant name used in the code instead. The only exceptions are where the number has a well-understood meaning and is unlikely to change, e.g., `90` used as ninety degrees.

Line length

Lines must not exceed 80 characters. Where the required statement needs more characters than that, the statement must be broken across lines intelligently to preserve a readable appearance. The automatic reformatting option in CLion does not automatically break the lines. However, you can configure CLion to help you with the line length by going to *File > Settings > Editor > Code Style* on Windows or *CLion > Settings > Editor > Code Style* on Mac. There, you can set the *Hard wrap at* to 80, select the *Wrap on typing* option, and set a *Visual guide* at 80.

Braces and indentation

The opening brace goes on the same line as the block-opening statement (function definition, `for`, `if`, `while`, etc.). Blocks are indented. The closing brace is on its own line, indented to the same level as the opening statement. CLion's auto formatting will take care of this for you.

Single statement blocks

All blocks must be surrounded by braces, even if only one statement long. For example, don't do this

```
if (count > LIMIT)
    error = true;
```

but rather do this:

```
if (count > LIMIT) {
    error = true;
}
```

File comment

Each file must begin with comment of the following form, at the very top of the file:

```
/*
 * Description of the purpose of this file or module, possibly
 * spanning several lines if necessary. Should be terse.
 *
 * Author : Your name(s)
 * Version: Today's date
 */
```

Function comment

Each function in a `.c` or `.cpp` file must have a function comment. The one exception is the `main` function, since a main's description would normally just repeat the file comment.

An example of a function comment is below. The description should be terse but informative. If there is anything "tricky" in the function's implementation, it should be stated here. Every parameter must have a description, as must the return value if there is one.

```
/*
 * Drive straight forward, while displaying "Driving". Due to
 * motor inaccuracy, the actual distance driven may be different
 * from the distance requested.
 *
 * speed: in native motor-control units
 * distance: to drive, in centimetres
 * returns: the distance actually driven, in centimetres
 */
double driveForward(int speed, double distance) {
```

In-line comments

In-line comments may be included where they are necessary to clarify. Appropriate choice of names and good program structure will often make inline comments unnecessary. An in-line comment may be either at the end of the line it describes, or on the line immediately prior. Lines with comments must not exceed the 80-character limit (above). In-line comments should use the `//` comment marker rather than `/* */`.