CSE/ECE 474 Code standard

Vikram Iyer
Alex Ching
Zachary Englhardt
Deeksha Prabhu

Rev. March 2024

This document describes the formatting requirements for C source code in ECE474.

Libraries and online code

You are welcome to use example code online either directly or as a guide. For any such code you use, you **must** cite the source. This must be included in your source code. This is an easy step that you should get in the habit of doing.

Constants

All constants should be in **symbolic** form, defined at the top of the code, or in a `.h` file included at the top of your code. This is an important feature of maintainable code. All constant names and variable names must be descriptive. Descriptive names are names that tell you something about what the variable or constant is used for. Example:

```
if (a > 31) { ...
```

This is a dangerous piece of code. First, it is hard to understand. Why is 31 important? Why do we care if `a` is greater than 31? Second, it is hard to modify. Usually, this constant must appear in more than one place. Suppose we need to scale up the software to solve a bigger problem? How do we change all of the 31s to 301? Sure you can use a global search and replace, but what if there are other 31s (like `x=247`**`31`**`4`) that we do not want to change? Finally, what the heck is "`a`" anyway? The variable should have a **name that means something** in the application (ideally means something to the end-user).

Solution:

```
#define  PACKETLENGTH 31
 ...
if (buffer_size > PACKETLENGTH) { …
```

Constants should be in **ALL CAPS** so that they are easily distinguished from variables. We recommend using enumerated constants when appropriate.

Example:

Thanks for valuable guidance from Prof. Blake Hannaford

```
enum days {MON, TUE, WED, THU, FRI, SAT, SUN}
```

The statement above is a shorthand for

```
#define MON 0
#define TUE 1
#define WED 2
...
```

The identifier `days` is not used. You just have to put something there. Now we can write:

```
if(day == FRI) { ...   // Do Friday tasks  }
```

Comments

Liberally comment the code. Aim for an average of one comment per 3 lines. **Comments should add value** to the code. For example, the following comment is **NOT** useful.

```
a = b + c  ;  // add b to c
```

Instead, use something like:

```
// add buffer length to transmit byte count
pack_size = b_length + tbytes;
```

This comment tells us stuff that we didn't know already from just looking at the code.

File headers

The beginning of each source file must contain a multi-line comment of the form

```
/* University of Washington
 * ECE/CSE 474,  [DATE]
 *
 *   [Team Member Name1]   (alphabetical order)
 *   [Team Member Name2]
 *   [Team Member Name3]
 *
 *   [Name of Assignment]
 *
 *   Acknowledgments:   [Sources of any borrowed code in this
 *                        source file]
 */
```

Thanks for valuable guidance from Prof. Blake Hannaford

Each function must have a header comment of the form

```
/*************************************************
 * [Function Prototype] (a copy here in the comment)
 *      Argument1 - explanation
 *      Argument2 - explanation
 *         ...
 *      ArgumentN - explanation
 *      Returns - [description of return value]
 *
 *      Text description of how function works.
 *
 *      Author [lead author, assisting authors]
 *
 *      Acknowledgments:   [Sources of any borrowed code in this
 *                          function]
 */
```

Thanks for valuable guidance from Prof. Blake Hannaford