C Programming Style Guide

Introduction

A good programmer needs to develop a coding style that is both readable and maintainable (by both you and others). So it is not sufficient for you to simply understand sequence, selection, iteration, recursion, functions and the other material in CS 100. You must also be able to apply these concepts in programs that exhibit proper style.

The standard speech you will hear on style is that you should be able to take any piece of code that you write and hand it to another individual who understands the langauge and they can easily read it, understand it, and modify it without hassles.

Now that you are actually writing problems of more than just a few lines, we are introducing some **required** style guidelines that will be enforced for your programming projects. The purpose of these guidelines is to help ensure that using proper programming style becomes second nature to you. Please realize that failure to follow these style guidelines will result in lost points on the projects.

Comments

Make sure that your program is properly commented. For any program that you write, you should be able to hand that code to someone else and expect them to be able to understand your logic and processes. This includes:

- Header Comments: You must provide a series of comments at the top of any program or module that describes the purpose of the file and the authorship. Such comments are known as it header comments. Remember, if you are modifying the work of another author (and have permission to do so), you must follow the original author's instructions for modification. At a minimum, you must add comments indicating your authorship and the reasons for the modifications.
- Function Comments: You must provide comments immediately prior to each function describing the purpose of the function. The more well-named your function and formal parameters are, the more succinct these comments can be.
- Inline Code Comments: You must provide comments of sections of code whose purpose would not be immediately obvious to a reader of the code. A good rule of thumb for this is as follows - if you had to think about the code before writing it (it was not incredibly obvious what was needed), then you should include a comment for this section of code.

Spacing, Indentation and Braces

With these items, consistency is the key to readability. We expect to see a consistent pattern with spacing, indentation and braces throughout the entire program. Shifting from one convention to another causes confusion on the part of the person reading the code. This includes:

- Indentation: You must use consistent indentation schemes to indicate the body of functions and the statements or blocks belonging to conditionals and loops.
- Braces: There are several standard conventions for where to place the braces with functions, loops, and conditionals. The convention that you pick must be used consistently throughout your program.
- Spacing: Blank lines for readability and spacing around operators and variables within a statement also define your coding style. You should ensure that whatever style you select is done consistently throughout the program.

Input

Input is free-format. This generally means that any amount of whitespace may separate and terminate tokens in the input. Specific project instructions may alter these restrictions.

Output

There are two types of output, token-oriented output and line-oriented output. There should be no whitespace preceding the first printable character and no whitespace following the last printable character of token-oriented output. For line-oriented output, each line should be terminated with a single newline with no whitespace preceding the newline. As with token-oriented output, there should be no preceding whitespace on any line. Specific project instructions may alter these restrictions.

C-Specific Requirements

The following style requirements are specific to the C programming language.

- Non-parameterized Functions: For all functions that you write which take no arguments, you must indicate the lack of formal parameters with the keyword `void`, as in `int f(void);`
- Default Types: You must explicitly state all types; using default types is forbidden.

- Function Length: Functions should never (well, hardly ever) exceed 25 lines in length. That is, the entire function should be visibile in a default terminal window. One can generally make a function smaller by replacing code representing some task with a well-named function to perform that task.
- Line Length: No line should exceed more than 80 characters, as line wrapping makes code difficult to read. With today's screen resolutions and good eyesight, it is possible to declare terminal windows that have widths of 150 to 200 characters. However, not everyone takes advantage of this capability. Lines that are longer than 80 characters can quickly turn code unreadable when viewed in a different environment.
- Main Signatures: The main function must have one of the following signatures, depending on whether or not you are using command-line arguments in your program:

```
int main(int argc,char *argv[]);
int main(int argc,char **argv);
int main(void);
```

- Compiler Warnings: You must compile with the `-Wall` and `-Wextra` flags. You must remove all compiler errors and warnings before submitting your code.
- Assertions: You should liberally use assertions to enforce preconditions and postconditions in your functions.

Using an Object Style

See `http://beastie.cs.ua.edu/C-first.html`