CSCI Coding Standards Philosophy

Purpose of this document:

This is a working draft of our department coding standards philosophy. It is not meant to define at a granular level a set of coding standards. Instead, its goal is to provide high-level guidance for what constitutes "quality code" in a way that should apply across courses at all levels of our program. Note: I'm not sure that the format shown here needs to be the one we will use. Our goal at this point is to discuss and refine. ☺

Why do developers have *coding standards*?

At the end of the day, source code is for humans, and so we aim for these two principles:

1. Source code should be *human readable*
2. Source code should be *maintainable*

Human Readable Code

One a programmer's task is to write code that *other people can read and understand*. Programmers do this by following these practices when creating code:

A. Document code with informative comments
B. Follow good naming practices
C. Use layout to communicate code structure
D. Code should be formatted for multiple *views*

Each of these will be described in more detail below. Programmers use a combination of all of these practices to make their code *readable*. They will need to use all of them, but how much of each is used will depend on the kind of program they're writing and their personal style.

Document Code with Informative Comments

There are two main reasons for comments in finished code:

- To explain complex code blocks or chunks of code
    - They should explain what the code *does*, not how it is done
- To explain how to use the code:
    - At a program-level, a *header comment* should
        - give an overview of the purpose of a program or file
        - describe how to use it
        - identify who is responsible for it (authors)
    - At a function-level, a comment that includes:
        - Function Inputs
        - Function Outputs
        - A description how to use the function

Programmers can also use comments while they're developing their code:

- To outline their plan before they start coding
- To add *todos* to remind themselves about incomplete tasks

Follow Good Naming Practices

When writing code, programmers need to name variables, constants, methods, and functions. Choosing good names makes it easier to understand what the code means and does.

- Variable and constant names/identifiers:
  - Should describe what data they contain
  - Often will be *noun words*
- Functions and Method names/identifiers:
  - Should describe what they *do*
  - Will often contain a *verb*
- Use consistent naming styles.

Use Layout to Communicate Code Structure

Programs can be complicated and include different *levels* of code, as well as *blocks* of code that logically belong together. Programmers use white space to help make these *levels* and *blocks* easier to see.

- Indentation:
  - Use indentation to *group* lines of code together that belong in the same block
- Blank Lines:
  - Use blank lines to *separate* chunks of code

Code should be formatted for Multiple Views

Programs will be viewed by different people and not just the way that the person who wrote it will view it. We should format our code in mind that it will be viewed:

- In different window sizes
- As a printout (for labs and assignments)

To make sure code structure doesn't get destroyed in the process:

- Avoid long lines of code or comments
  - When source code is printed, long lines *wrap* and can disrupt the layout created using whitespace.
  - When multiple files are viewed side by side, the window may need to be narrow, which can make the viewer have to scroll side to side to read long lines.

Maintainable Code

Many of the programs written in first year won't be programs have to be maintained or added to, but that won't always be the case! So, now's the time to start practicing creating code that someone may want to:

- Patch or apply bug fixes
- Port to another platform
- Update to add more features
- Update to use different libraries or frameworks

There are a couple of simple guidelines for writing *maintainable* code:

I. Write modular code
II. Use "just enough" scope

Write Modular Code

Well-written code should make it easy to reuse functionality without having to rewrite it in multiple places. Put code that can be used more than once in a function or method and call the function when needed instead of copy-pasting the same code over and over.

The advantages of this are:

- It saves keystrokes when typing
- When this code needs to be updated or fixed, it only needs to be changed in one place
- It will help structure the code and make it easier to read

Use "Just Enough" Scope

To avoid accidentally overwriting a variable, variables should be declared where they need to be used, and not elsewhere. Some techniques for achieving this include:

- Avoiding *global* scope variables
- Use global constants if a value that will never change, is needed in many places

How to Make Your Instructor Call You in for a Chat

We love chatting with our students! There are a few coding practices that might be considered bad practice, and your instructor sees you using them they might want to chat with you about why to avoid them. Here are just a few:

- Using gotos in your labs or assignments
- *Recursively* calling the `main()` function
- Declaring constants for values like 1 or 2

Tips for Success When Creating a Program

- Structure the solution before writing code
  - Work on paper or the whiteboard
- Aim for good code standards as soon as you start writing code. Don't plan on "fixing" it later.
- Add comments to code that reflect the plan before starting to write coding
- Use an incremental coding style:
  - Write small logical chunks at a time
  - Test and fix before adding more complexity
  - Divide and conquer to solve complex problems
- When first learning to program, avoid tools that do the work for you
  - Typing code helps beginner programmers develop muscle memory
    - Copy-paste doesn't help with this
    - Auto-complete can bypass this, too
  - Typing code helps programmers learn what they're doing
- Comment out code rather than outright deleting it, until the solution is complete

Appendix:

The following are not meant to be included in this Standards document. Rather they reflect some things we discussed during the creation of the document, some helpful tips, etc.

How to promote good code standards:

- Practice what you preach
  - Provide lots of examples of well-written code
  - If you have a good reason to *break the rules* (like using shortened identifiers when writing on the board), be sure to explain why you're doing so
- Provide opportunities for students to learn to identify well-written code
  - Peer code reviews
  - Let students *critique* or *fix* poorly styled code

    - Have students view code on git hub and discuss
- Reward good coding standards with formative or summative feedback

Huizhu's Priority List for Good Code:

1. **Correctness**. Google uses the "usability by customer". In my mind, the correctness of a program is determined by the program's customers. Whoever uses a program is the one giving out specifications. Whoever gives out the specification gets to decide what's the program's correct behaviour.
2. **Readability**. Easy to understand, therefore easy to debug and maintain.
3. **Robustness**. A program should reasonably anticipate exceptions and handle these exceptions elegantly.
4. **Evolvability**. Although I should follow Google's standard and call it "Extensibility/Modifiability".
5. **Efficiency**. Use merge sort instead of bubble sort!

Sample code standards from the real world:

- Google Common Lisp Style Guide: https://google.github.io/styleguide/lispguide.xml
- Microsoft C# Coding Conventions: https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions
- Microsoft All-In-One Code Framework project team's Convention: https://github.com/sphinxlogic/All-In-One-Code-Framework/blob/master/All-In-One%20Code%20Framework%20Coding%20Standards.docx

Some standard naming styles to help students find their style:

- https://en.wikipedia.org/wiki/Camel_case
- https://en.wikipedia.org/wiki/Letter_case#Kebab_case
- https://en.wikipedia.org/wiki/Snake_case
- You can read up more on programming naming conventions, in general, here: https://en.wikipedia.org/wiki/Naming_convention_(programming)