CPSC 121 Computer Science I

Gonzaga University

Daniel Olivares

Content used in this lesson is based upon information in the following sources:

- Dr. Gina Sprint's materials.

Overview

This document outlines the general expectations of code organization and style for CPSC 121. This coding standard and guideline have been written up for several reasons which should help writing high quality code that is easy to understand and develop. We will apply code reviews to validate code quality, so it is important that all students use the same style of coding. Style in this sense means using common constructs, writing proper documentation and code comments, and organizing code to a common layout.

Although complying with coding guidelines may seem to appear as unwanted overhead or limit creativity, this approach has already proven its value for many years in industry and here.

Additional goals include:

- Preventing common mistakes and pitfalls.
- Preventing language constructs that are less comprehensive.
- Promoting good design
- Improving readability and extensibility of the code.
- Facilitate the ease of learning to program
- Help you develop superior coding skills and habits.

This standard is a summary of known good habits and industrially accepted practices. It is a simplified version for Academic use.

Commenting and code layout is used as a mechanism (maybe the only reliable one) for the code writer (author) to effectively communicate purpose, functionality, logic, intent, decision processes, and motivation for any piece of code to the code reader. Anytime there is ambiguity, incompleteness, or obfuscation in the code there is the opportunity for mistakes and misunderstanding to occur. We hope to teach you how to minimize these types of problems and in that effort also help you learn to be a great software developer.

Naming Convention Definitions

The following naming conventions may be referred to in this document. An example of each convention is given for a label given to describe the area of a circle.

- Camel case (AKA lower camel case)
    - Example: `circleArea`
- Pascal case
    - Example: `CircleArea`
- Snake case (generally not used in C++, may be used for file names such as text files)
    - Example: `circle_area`
- Macro case
    - Example: `CIRCLE_AREA`

File Names

Source files (e.g. .cpp files) should have a name that describes the functionality/purpose of the code in the file. For example, a C++ program to compute the area of various shapes could be called Area.cpp instead of a non-informative name, such as Program.cpp.

Input/output files (e.g. .txt files, .csv files, etc.) should have a name that describes the content of the file.

Programmer-Defined Identifiers

All identifiers should be named informatively. For example, do not name a variable, `variable`. Thoughtful identifier names improve the readability of your code to yourself and others.

Variables

A variable name should describe the value of the variable and should use camel case. For example, `circleRadius` for storing the radius of a circle.

Constant variables should use macro case. For example, `GRAVITATIONAL_CONSTANT` for storing the gravitational constant, G.

The use of non-constant global variables, variables defined outside of a function, should be avoided.

All variables should be declared at the top of the function they are defined in.

Functions

A function name should describe the computation/function/purpose of the function, begin with a verb, and use camel case. For example, `computeCircumference()` to compute the circumference of a circle.

Functions should have function prototypes placed in a header file as part of the 3-file format.

Types

A programmer-defined type name should describe the semantics of the type and use Pascal case. For example, `Circle` to represent information and functionality related to a circle type, such as a struct.

Comments

Liberally comment your code! It helps you and others who may read your code. Also, should you use an algorithm, idea, or any other content that is not your own, use comments in your code to properly **cite your sources**.

Inline Comments

To describe the purpose of a line of code, it is beneficial to place comments throughout your code, not just when defining classes/interfaces/methods. For example:

```
// relevant area of a circle formula: pi * r ^ 2
```

Source File Comments

Begin all source code files with a header section that indicates

- Programmer's name
- Version and date
- The course ("CPSC 121")
- Assignment number (e.g., "Programming Assignment #1")
- Brief description of what program does. If you use any sources other than those provided in the course, list them here.
- Any sources to cite?

For example

```
/*
 Name: Gina Sprint
 Class: CptS 121, Spring 2019
 Date: January 9, 2019
 Programming Assignment: PA1
 Description: This program computes...
 Notes: I utilized this source url: ...
*/
```

Function Comments

Every function declaration should have a comment block before it. The comment block should describe the function, function parameters, relevant side effects of the method (does it change the object state?), relevant pre/post conditions, and the result if the method returns a value.

For example:

```
/*************************************************************
 * Function: convertTimeToSeconds ()
 * Date Created: 1/17/18
 * Date Last Modified: 1/17/18
 * Description: This function converts the times from min
 *              and sec to just sec.
 * Input parameters: The time in minutes and seconds.
 * Returns: The runner's time in seconds only.
 * Pre: The minutes and seconds of the time had to
 *                 have been scanned in by the user.
 * Post: The runner's time in seconds only is returned.
 *************************************************************/

double convertTimeToSeconds (int minutes, double seconds)
{
      double overall_seconds = 0.0;

      /* Need the time in seconds only in order to calculate ft/s and m/s */
      overall_seconds = SECONDS_PER_MINUTE * minutes + seconds;

      return overall_seconds;
}
```

3 File Format

As part of the 3 file format, code should be broken up into three files:

1. Main.cpp: Contains the main() function definition and includes your header file
2. <name>.cpp file: contains all other function definitions and includes your header file
    a. <name> is a name for the file that describes the functionality in it
3. <name>.h file (header file): contains header file guard code, all standard library inclusions, using statements, constant global variable declarations, and function prototypes for the functions in the <name>.cpp file

Example header file guard code for a header file called CircleArea.h:

```
#ifndef CIRCLE_AREA_H
#define CIRCLE_AREA_H

// all other header file code goes here

#endif
```

Other Good Styling

Insert blank space before and after commas and operators such as +, =, *, /, //. Use indentation appropriately so it is clear which statements belong with which block. Be consistent. Beyond what is listed in this document, there will be several opportunities to add your own style to your code. Choose a style and stick with it.