

CSCI 123 – Introduction to Programming Concepts in C++ - Style Guide

This guide explains what your source code must contain and how it should look for the course. This is a brief agreement of how source code will be written in this course. After reviewing the style guide for [Mozilla](#), I've decided to use a style similar to Java style coding and utilize [JavaDoc](#) Style commenting.

1. Introduction

1.1 Why we have coding conventions (style guides)

“Code conventions are important to programmers for a number of reasons:

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.” [1999 Sun]

2. Source Code Header

- 2.1. Source code header will utilize JavaDoc style tags (@date, @author, @filename), to highlight information about the code. The following elements are expected in your header documentation.
- 2.2. Name of the File (example, cashregister.cpp, cashregister.h)
- 2.3. Description of the source code. What is it supposed to do? For the main source file, it should describe the application. For additional source files, it should describe what the particular file is supposed to do, not what the entire program does.
- 2.4. Course Name and Section Number (CSCI 123 00000)
- 2.5. Assignment Name
- 2.6. Date the code was written
- 2.7. Programmer's Name and Student ID
- 2.8. Version (optional)

Example:

```
/**
 * @file          Name of the file(.cpp/.h)
 * @description   Provide a description of the program/file
 *               (what is this file supposed to do)
 * @course        CSCI 123 Section 00000
 * @assignment    Assignment 1,2,3 etc.
 * @date          10/20/2006 etc.
 * @author        (Your Name - student ID - email)
 * @version       1.0/2.0 (optional)
 */
```

3. Function Headers

- 3.1. A description of the function and any assumptions about the function. In addition, the description should explain any algorithms used in the function.
- 3.2. Parameters need to be described with a “@param” tag the name of the parameter and a description of the parameter
- 3.3. Preconditions are described with a “@pre” tag.
- 3.4. Postconditions are described with a “@post” tag.
- 3.5. Return values will be described with a “@return” tag and a description of what the function is returning, including any modifications to global or referenced variables.

Example:

```
/**
 * Function Description
 * @param aNumberOfPeas contains the number of peas
 * @param aNumberOfPods contains the number of pods
 * @pre   aNumberOfPeas and aNumberOfPods are initialized by the
 *        calling program.
 * @post  aNumberOfPeas and aNumberOfPods are multiplied to get the
 *        number of peas in a pod.
 * @return the product of aNumberOfPeas and aNumberOfPods
 */
```

4. Class Declaration

The class header is before the class declaration in the source file

- 4.1. Name of the class
- 4.2. Class Description
- 4.3. Author of the source code

Example:

```
/**
 * Class Name declaration.
 * Brief description of the class.
 * @author: (Your Name - student ID - email)
 */
```

5. Indentation

- 5.1. Indentions should be 4 spaces
- 5.2. Lines of code should not exceed 80 characters

6. Comments

- 6.1. Single Line Comments are denoted by, “//”, two forward slashes. They should be used to describe code statements that aren’t obvious. This should aid in understanding the code.
- 6.2. Multiple Line Comments are denoted by the beginning comment, “/*”, a forward slash and an asterisk and the ending comment, “*/”, an asterisk followed by a forward slash. Sections 1, 2, 3, and 4 are examples of multiple

line comments. Multiple Line comments should be used to describe any additional explanations about your code, data structures, and/or algorithms.

7. Declarations

7.1. One declaration per line. Although multiple variables can be declared on one line, it is recommended that you declare one variable per line.

Example:

```
int courseRequestNumber;    // course request number from the
                             // database
int numberOfCourses;        // total number of courses
```

8. Statements

8.1. if, if-else, if- if-else else statement should follow the following format:

```
if (condition) {
    statements;
}

if (condition) {
    statements;
} else {
    statements;
}

if (condition) {
    statements;
} else if (condition) {
    statements;
} else {
    statements;
}
```

8.2. for statement should follow the following format:

```
for (int i = 0; i <= 10; i++) {
    statements;
}
```

8.3. while statement should follow the following format:

```
while (condition) {
    statements;
}
```

8.4. do-while statement should follow the following format:

```
do{
    statements;
} while (condition);
```

8.5. switch-case statement should follow the following format:

```
switch (condition) {
    case ABC:
        statements;

    case DEF:
        statements;
        break;

    case XYZ:
        statements;
        break;

    default:
        statements;
        break;
}
```

8.6. try-catch statement should follow the following format:

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
}
```

9. White Space

9.1. Indentation will be 4 spaces or one tab. All code enclosed in a block of code will be indented one tab space from the curly braces defining the block. This includes nested blocks of code.

Example:

```
block {
    statements;

    block {
        statements;
    }
}
```

9.2. White space is needed to make your code more readable. When writing operations that require a binary operator, variables and the assigned variable should be separated by a white space.

Example:

```
varResult = var1 + var2;
```

10.Naming Conventions

- 10.1. Variable names must be descriptive. It should describe what is being stored in the variable.
- 10.2. Constant variables will be all uppercase characters and each subsequent word in the variable will be separated by an, “_”, underscore.
- 10.3. Global variables will be prefixed with a lowercase “g” and each subsequent word in the variable will be capitalized.

Example:

g=global (e.g. gPrefService)

- 10.4. Member variables will be prefixed with a lowercase “m” and each subsequent word in the variable will be capitalized.

Example:

m=member (e.g. mLength)

- 10.5. Argument variables will be prefixed with a lowercase “a” and each subsequent word in the variable will be capitalized.

Example:

a=argument (e.g. aCount)

- 10.6. Static variables will be prefixed with a lowercase “s” and each subsequent word in the variable will be capitalized.

Example:

s=static member (e.g. sPrefChecked)

11.Miscellaneous

- 11.1. Don't let your code drift off beyond the 80th column. For example, if you have a long list of comma-delimited identifiers or arguments, break them across multiple lines so you don't go beyond column 80.
- 11.2. Curly braces are to be used for all if/else statements, as well as for all loop statements, even if a statement does not require them.
- 11.3. Each variable is to be declared on its own line, don't provide a comma-separated list.
- 11.4. As mentioned above, there should be no white space between a function identifier and its open parentheses. An open parenthesis in any other context

should be preceded by a blank space (“if” statements, “while” statements, “for” statements, etc.).

12.Code Example

```
1. /**
2.  * @file          ch1_pp1.cpp
3.  * @description   Users to enter the number of peas and
4.  *               pods, then totals the peas in a pod.
5.  * @course        CSCI 123 Section 00000
6.  * @assignment    Programming Project 1
7.  * @date          10/20/2006
8.  * @author        Brad Rippe (00000000) brippe@fullcoll.edu
9.  * @version       1.0
10.  */
11. #include
12. using namespace std;
13.
14. /**
15.  * This is the main function header. Since there isn't a
16.  * class definition we don't use a class header
17.  * @return zero if the application executes successfully
18.  */
19. int main() {
20.     int numberOfPods;
21.     int peasPerPod;
22.     int totalPeas;
23.
24.     cout << "Press return after entering a number.\n";
25.     cout << "Enter number of pods:\n";
26.     cin >> numberOfPods;
27.     cout << "Enter the number of peas in a pod:\n";
28.     cin >> peasPerPod;
29.
30.     totalPeas = numberOfPods * peasPerPod;
31.
32.     cout << "If you have ";
33.     cout << numberOfPods;
34.     cout << " and ";
35.     cout << peasPerPod;
36.     cout << " peas in each pod, then\n";
37.     cout << "you have ";
38.     cout << totalPeas;
39.     cout << " peas in all the pods.\n";
40.
41.     return 0;
42. }
```