# Coding Style

## General Recommendations

The main goal of these recommendations is to improve the readability, understanding and maintainability of the source code. Violation of these recommendations is allowed only if it enhances the readability of the source code.

## Header Files

Every .cpp file should have an associated .h file. Files should be saved in the same directory. An exception to this is the main.cpp file, which does not have to have an associated .h file. Other small files containing unit testing can also be excluded from having an associated .h file.

## Include Statements

Include statements should be sorted and grouped by their hierarchical position in the system, with low level files included first. Include statements should be located at the top of a file only. Local libraries should be placed below the external libraries.

// Example of includes.

#include <iostream> // Main external library.

#include <fstream> // Main external library.

#include <cmath> // Minor external library.

#include <algorithm> // Minor external library.

#include “Functions.h” // Main local library.

#include “OtherFunctions.h” // Minor local library.

using namespace std; // ONLY USE THIS IN .CPP FILES.

## The #define Guard

All header files should have the #define guard to avoid duplicate declarations.

Formatting for the #define guard should be <*FILE>\_H\_*

## Naming Conventions

1. Names representing types must be in mixed case starting with upper case. Example: *Name, FileName*
2. Variable names must be in mixed case starting with lower case.

Example: *name, filename*

1. Constants must be all upper case using underscore to separate words.

Example: *MAX\_NAME, PI*

1. Names representing methods should be verbs and written in mixed case starting with lower case.

Example: getName( ), computeArea( ), setName( )

1. Names of classes should begin with an uppercase letter.

Example: Class MyClass

## Variables

1. Variables must be initialized upon declaration.
2. Variable names must be clear and meaningful.
3. Variables must be declared at the top of a functions unless needed otherwise.
4. Matching variable types must be in the same line unless it is really hard to read.
5. Use of global variables is not permitted; in C++ there is no need for global variables.
6. Class variables must always be declared as private members.

// Example of variable declarations.

void functionName(){

int n1 = 0, n2 = 0, n3 = 0; // Simple.

float tax = 1.5, pie = 3.14; // Simple.

MyClass instance1(var1, var2, var3, var4, var5, var6); // Complex.

MyClass instance2(var1, var2, var3, var4, var5, var6); // Complex.

// Code.

}

## Tabs, Spacing and Indentation

Tabs must be set to four characters.

The bracing must follow the style whereby the opening brace begins on the same line of the statement and ends on a new line. Only use ‘endl’ if there isn’t a “ “ you can add a ‘\n’ to. If you are outputting the last line of a group/menu, do a double newline, to create a gap between it and the next output. If a cout runs on for a long time, just let it word wrap. Group lines of code together in a way that everything is neatly displayed and easy to read. Use this example code for the exact format of where spaces are used and where they aren’t.

// This uses word wrapping and correct spacing.

for(int i = 0; i < 5; i ++){

cout << “Variable 1: ” << var1 << “\nVariable 2: “ var2 << “\nVariable 3: “ var3 << endl;

}

Unless an if/else statement is an easy-to-read 1-liner, you must use braces.

// This is a simple if statement without braces.

if(x = 1)

Cout << “Yay!\n”;

// Example of usual if/else statements.

if(input == “1”){

cout << “Input Name: ”;

cin >> name;

doFunction(name);

}

else if(input == “2”){

cout << “Input Age: ”;

cin >> age;

doFunction(age);

}

else if(input == “3”){

cout << “Input Country: ”;

cin >> country;

doFunction(country);

}

// Example of grouping code:

Cout << “Input Name: “;

Cin >> name;

runFunction(name);

if(test == true){

cout << “Running test…\n”;

runTest();

cout << “Test completed.\n\n”;

}

cout << “Main Menu:\n”;

cout << “1) Search.\n”;

cout << “2) Edit.\n”;

cout << “3) Exit.\n”;

cout << “Choice: “;

cin >> choice;

runAnotherFunction();

## Classes

Classes must follow the same format for bracing, with the opening brace on the same line of the class name and the closing brace on a new line. The labels for *private:* and *public:* are not indented. If there is nothing in public/private/protected, then don’t write them.

Example: // This only has public functions.

Class MyClass {

public:

MyClass( ); // Constructor.

~MyClass( ); // Destructor.

void myMethod ( );

};

One exception to this bracing style is permitted- inline functions may have their closing brace on the same line as their opening brace.

## Code Comments

Comments should be used to explain what a section of code does if it is not clear.

It is preferable that if the code is too complex to understand that it be re-written. Single line “//” commenting is to be used, even for multi-line comments. Comments must have a space between the code and the //. Also, include a space immediately after “//”, and the comment should start with an upper case character and end with a period, as in standard English. If a comment is supposed to explain multiple lines of code, put it directly above the first line of said code.

// For example, these are cins.  
cin >> x;

cin >> y;

cin >> z;

Block commenting “/\* \*/” should only be used for commenting out sections of code for debugging purposes.

## Functions

Functions should be kept as small as possible. If a function exceeds approximately 40 lines, consider separating it into smaller parts.

There is no hard limit to the size of a function, as at times it is unavoidable that a function grow in size- this can be acceptable if there is no alternative method of reducing it into smaller parts.

## Versioning

At the top of each code file is a comment block which must be modified each time the file is modified. Increment the version number before synching it to Github.