Coding Style Guide For SDI

# Formatting

Keep lines of code at reasonable length to allow for easy reading and prevent confusion, preferably below 80 characters long.

## Use tabs for indentation

**Example:**

If(x != 1){

Cout<<”Hello World”; //One tab of indentation

}

## Use spaces around operators.

**Example:**

Total = 21 + 42; //Space on each side of “+”

## Use space after special characters such as commas, colons, semicolons.

**Example:**

Total = 21 + 42

A , B = 5, 21;

## Do not use spaces after (, [, {or before ), ], }

**Example:**

If (homer != “Simpson”){

};

## Conditionals on different lines

**Example:**

If(x = 3){

…

}

Else{

…

}

# Naming

* Variables and methods are to be named using the snake case method, e. g total\_payment.
* Classes are to be named using the camel case method e. g CalculateInterest.
* Constant variables are to be named using snake case method in upper case e. g INTEREST\_RATE.
* Names of methods which only return a true or false value are to end with a question mark e.g. IsArrayEmpty?

# Headers

Headers are to be included in a specific order:

1. C System Headers
2. C++ Standard Library
3. Non-Standard Library Headers
4. Project Headers

# Namespaces

If required Namespaces may be used within the project, when setting names for namespaces make sure that:

* The name of the namespace clearly suggests what the namespace refers to e. g when creating a namespace dedicated to the GUI of the project the namespace could be called “gui”.
* Namespaces are only used when required and are not used to distinguish areas of the project which could be effectively distinguished by using better naming for methods and functions.
* Namespace names should all be in lower case.
* When ending the definition of a namespace show clearly by using a comment e. g “// namespace gui”

# Non-member functions

Functions which are not bound to a class instance and can be used globally should not use any external variables, they are required to be within a specific namespace so that it is known what part of the program they specifically refer to and it is easier to identify them.

You are not to group non-member functions within a specific class specifically created for these functions as this is unnecessary.

# Declaring Variables

When declaring variables, it is recommended to declare them in as local of a scope as possible and close to the first use of that variable in order to improve code maintenance and readability. This helps others have a better idea about what the variable relates to as well as what it has been initialized as.

**Example:**

float pi = 3.14 //both declaration and initialization have been done on the same line.

Vector<int> position = {5, 10}; //declaration and initialization together.

# Comments

Comment consistency is key, make sure the comments you are writing are in depth that anybody looking at the code could understand what is happening.

To write comments in this project the // syntax will be used.

## Class:

Class comments need to be in depth enough to allow the reader to understand exactly what this class is doing and how and when they can use the class. They should point to any considerations that the reader should make before using the class and have extra detail in scenarios where the class can be accessed by multiple threads where rules regarding this should be included. In the case of more complicated classes the comments should include a more abstract example of what the class is doing.

## Function:

When declaring functions, the comments should mention what the function aims to achieve if there are any inputs and outputs and what they will be. They should also state if any of the function arguments can be null pointers or what kinds of arguments the function is expecting. If the function performs better in certain scenarios or can be used with specific arguments in order to achieve specific results this should also be commented. Make sure that comments are relevant and not stating facts which the person looking at the code is most likely aware of.

Variable:

Comments can be written for variables if required however in most cases the variable name should be descriptive enough to allow a good understanding of its function. This is also applicable to class attributes. Global Variables are the only ones where an in-depth explanation is required, this should include the purpose of the variable in detail, and why it needs to be a global variable.

**Example:**

Int result\_of\_sum //stores result of sum between number\_1 and number\_2 after being used in function x

Comments are encouraged, the more detailed comments explaining the code the better however, do not state the obvious and simply describe what the code does as this can be seen by a skilled C++ programmer. Try to explain the reasoning to why the code has been written like this and any in depth concepts which may be hard to spot within it.

Comments can be used to mark issues within the code in order to fix them later yourself or by another person. In order to be able to identify these issues easily the comments which represent them need to be within a certain format.

**Example:**

//FIX(Name of individual who written the code):comment(name of individual who spotted the issue)

//FIX(Richard):Algorithm not sorting in correct order, fix to sort ascending(Matt)

This way it is clear that the comment is in relation to an issue, who it is that is most likely to fix it as they have written the code, the issues there are with the code and the person who has spotted the issue just in case the original writer wants more information or input.