# C++ Style Guide

## ;DROP TABLE * --;

11/9/2022 v1.0

Goal / Background:
To keep our code base consistent in readability and patterns.

General Rules:
Avoid repeated code as often as possible.
Try to make self-documenting code when possible. Refer to rule below.
If the method is far too complicated to justify, in a single method comment, put line comments where necessary.
There will be exceptions naturally, but stick to these standards as much as necessary.
Warnings can likely be ignored unless it is severe, but obviously fix all compiler errors where needed.

Good code is:
Simple but effective
Self-documented
Well formatted
Consistent

Comments:
All methods have block style comments, in the header file with an @brief @param @return sections.
Every file has its own block comment at the top identifying implementation, a course reference, and our group name.
Short inline comments will be used where helpful.

Files:
Pascal case the file name, header files for every cpp file.
Header files will utilize the filename, and _ (for example, FILENAME_H)
Source files should include all necessary headers directly, not indirectly through other (or corresponding) header files.

Naming:
Camel cases

Header Files:
All documentation stays in the header file. All implementation goes inside of the cpp.

Format Standards:
camelCase naming convention for variables, methods, instance/privates
Temporary variables can generally follow any convention that is easy to read or understand.

Exceptions (if any):
Any exceptions will be reviewed by the whole team.

Brackets:
Like function{
}

Not function
{
}
C++ Defensive Coding Conventions
Try to utilize private variables as much as necessary.
Pass variables by method call to minimize segmentation faults.
Custom classes should be in separate files, complete with their own h files.
Utilize branches to minimize the amount of sudden change without merge approval
Try to use descriptive names for variables.
Vague names are awful unless they are being used for temporary variables or iterators
Connections can work implicitly and explicitly.
Declare variables in the most limited scope possible.