Coding Standards

C++ Version
We should be using C++17.

Code Format
Tab indentation should be 4 spaces, chance this setting in your editor.
Any brace style is fine, but braces should always be included in loops and if statements to improve readability.

Dependencies
Every .h file should have a corresponding .cpp file UNLESS that .h file is pure virtual.

Do not use any non-standard dependencies that we haven't agreed on as a group.

Only include what is necessary, if you use a symbol from another file in your code, that file should be included.

Use **#ifndef guards** in every .h NOTE: **pragma once** is non-standard
Eg.

```
#ifndef FILE_H
#define FILE_H
Class File{}
#endif
```

Forward Declarations
Avoid forward declarations unless absolutely necessary, rather include header files you need.

Naming
Use meaningful variable names that accurately describe their purpose and function.
Variable and Function names are in **camelCase**

Boolean variables have the “is” prefix. i.e isAvailable
Type names begin with a **Capital** letter i.e Type* myType …
Constants have the following format: **MY_CONSTANT or CONSTANT**
File names should start with a Capital Letter and have no spaces or underscores.

Comments

We will comment “As we go” with Doxygen, using /** */ comments etc.
At every function give a brief description of what the function does, include explanations for parameters if they are not primitive data types.

Exceptions

Use exceptions sparingly - Exceptions should be used only when necessary to handle unexpected situations. Overuse of exceptions can make the code harder to read and maintain.

Catch exceptions at the appropriate level - Exceptions should be caught at the lowest possible level where they can be handled. This helps to prevent the exception from propagating up the call stack and causing unexpected behaviour.

Use specific exception types - Use specific exception types instead of catching the base exception class. This allows for more targeted handling of exceptions and makes the code easier to read and maintain.

Testing

Write unit tests “as you go”, ensure that your functions, classes etc are functioning correctly and document the results in a txt.

Logging

Remove any log or print statements before submission that aren’t necessary. Be thorough!