README (All Rubric Points REQUIRED)

CRITERIA	MEETS SPECIFICATIONS	Status
A README with instructions is included with the project	The README is included with the project and has instructions for building/running the project. If any additional libraries are needed to run the project, these are indicated with cross-platform installation instructions. You can submit your writeup as markdown or pdf.	Done
The README indicates which project is chosen.	The README describes the project you have built. The README also indicates the file and class structure, along with the expected behavior or output of the program.	Done
The README includes information about each rubric point addressed.	The README indicates which rubric points are addressed. The README also indicates where in the code (i.e. files and line numbers) that the rubric points are addressed.	Done

Compiling and Testing (All Rubric Points REQUIRED)

CRITERIA	MEETS SPECIFICATIONS	Status
The submission must compile and run.	The project code must compile and run without errors. We strongly recommend using cmake and make, as provided in the starter repos. If you choose another build system, the code must compile on any reviewer platform.	Done

Loops, Functions, I/O

CRITERIA	MEETS SPECIFICATIONS	Status
The project demonstrates an understanding of C++ functions and control structures.	A variety of control structures are used in the project. The project code is clearly organized into functions.	Yes
The project reads data from a file and process the data, or the program writes data to a file.	The project reads data from an external file or writes data to a file as part of the necessary operation of the program.	Yes
The project accepts user input and processes the input.	The project accepts input from a user as part of the necessary operation of the program.	Yes

Object Oriented Programming

CRITERIA	MEETS SPECIFICATIONS	Status
The project uses Object Oriented Programming techniques.	The project code is organized into classes with class attributes to hold the data, and class methods to perform tasks.	Yes
Classes use appropriate access specifiers for class members.	All class data members are explicitly specified as public, protected, or private.	Yes
Class constructors utilize member initialization lists.	All class members that are set to argument values are initialized through member initialization lists.	Yes
Classes abstract implementation details from their interfaces.	All class member functions document their effects, either through function names, comments, or formal documentation. Member functions do not change program state in undocumented ways.	Yes

Classes encapsulate behavior.	Appropriate data and functions are grouped into classes. Member data that is subject to an invariant is hidden from the user. State is accessed via member functions.	Yes
Classes follow an appropriate inheritance hierarchy.	Inheritance hierarchies are logical. Composition is used instead of inheritance when appropriate. Abstract classes are composed of pure virtual functions. Override functions are specified.	Yes
Overloaded functions allow the same function to operate on different parameters.	One function is overloaded with different signatures for the same function name.	Yes
Derived class functions override virtual base class functions.	One member function in an inherited class overrides a virtual base class member function.	Yes
Templates generalize functions in the project.	One function is declared with a template that allows it to accept a generic parameter.	Yes

Memory Management

CRITERIA	MEETS SPECIFICATIONS	Status
The project makes use of references in function declarations.	At least two variables are defined as references, or two functions use pass-by-reference in the project code.	Yes
The project uses destructors appropriately.	At least one class that uses unmanaged dynamically allocated memory, along with any class that otherwise needs to modify state upon the termination of an object, uses a destructor.	Yes
The project uses scope / Resource Acquisition Is Initialization (RAII) where appropriate.	The project follows the Resource Acquisition Is Initialization pattern where appropriate, by allocating objects at compile-time, initializing objects when they are declared, and utilizing scope to ensure their automatic destruction.	Yes
The project follows the Rule of 5.	For all classes, if any one of the copy constructor, copy assignment operator, move constructor, move assignment operator, and destructor are defined, then all of these functions are defined.	Yes
The project uses move semantics to move data, instead of copying it, where possible.	For classes with move constructors, the project returns objects of that class by value, and relies on the move constructor, instead of copying the object.	Yes
The project uses smart pointers instead of raw pointers.	The project uses at least one smart pointer: unique_ptr, shared_ptr, or weak_ptr. The project does not use raw pointers.	Yes

Concurrency

CRITERIA	MEETS SPECIFICATIONS	Status
The project uses multithreading.	The project uses multiple threads in the execution.	Yes
A promise and future is used in the project.	A promise and future is used to pass data from a worker thread to a parent thread in the project code.	Yes
A mutex or lock is used in the project.	A mutex or lock (e.g. std::lock_guard or `std::unique_lock) is used to protect data that is shared across multiple threads in the project code.	Yes
A condition variable is used in the project.	A std::condition_variable is used in the project code to synchronize thread execution.	Yes