

[◀ Return to Classroom](#)

C++ Capstone Project

REVIEW


CODE REVIEW



HISTORY

Meets Specifications

Congratulations !

Commendable Job. You have really worked hard on this project to give it such a good user experience.

Making the required changes and submitting the project in such a short span of time shows your dedication towards it. The hard work did pay off and you have passed this nanodegree with flying colors. 

- Firstly, its great to see that you have chosen multiple points covering each section which makes this project outshine other projects where people tend to cover points just from OOP section. 
- Secondly, its good to see that you have maintained a proper README file containing all the information from detailed project description to steps to install and run the project successfully. 

Good luck with whatever comes next! :)

We look forward to receiving your future project submissions soon .

Have a Good Day/Night and Stay Safe 🙏

Keep learning and stay Udacious



README (All Rubric Points REQUIRED)

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.


You can submit your assignment as many times as you want.

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.

README gives the description of the project 

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.

README indicates which rubric points are addressed. 

Compiling and Testing (All Rubric Points REQUIRED)

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.

Loops, Functions, I/O

A variety of control structures are used in the project.

The project code is clearly organized into functions.

The project reads data from an external file or writes data to a file as part of the necessary operation of the program.


The project accepts input from a user as part of the necessary operation of the program.

Object Oriented Programming


The project code is organized into classes with class attributes to hold the data, and class methods to perform tasks.

Project follows the OOP structure and has various classes with class attributes and methods 

All class data members are explicitly specified as public, protected, or private.

Access specifiers have been used for class data members 

All class members that are set to argument values are initialized through member initialization lists.

Class members are initialized through member initialization lists. 

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.

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.

Encapsulation and data hiding 

Inheritance hierarchies are logical. Composition is used instead of inheritance when appropriate. Abstract classes are composed of pure virtual functions. Override functions are specified.

One function is overloaded with different signatures for the same function name.

One member function in an inherited class overrides a virtual base class member function.

Virtual base class member function is overridden by derived class member function 


One function is declared with a template that allows it to accept a generic parameter.

Memory Management

At least two variables are defined as references, or two functions use pass-by-reference in the project code.

Functions use pass-by-reference in the project code. 

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.


Destructor is used for unmanaged dynamically allocated memory 

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.

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.


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.

The project uses at least one smart pointer: `unique_ptr`, `shared_ptr`, or `weak_ptr`. The project does not use raw pointers.

Smart pointer `shared_ptr` has been used instead of traditional raw pointers 


Concurrency

The project uses multiple threads in the execution.

Multithreading has been used 

A promise and future is used to pass data from a worker thread to a parent thread in the project code.

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.

`std::unique_lock<std::mutex>` has been used for shared data protection 

A `std::condition_variable` is used in the project code to synchronize thread execution.

 [DOWNLOAD PROJECT](#)

[RETURN TO PATH](#)

