A short guide to writing good fucking C++ code.

2018-12-20

GOOD FUCKING C++ ADVICE

This page intentionally left… shit, it's no longer blank.

n°1: Keep It Fucking Simple

Simple is not easy and it’s not about having few elements. Simple is not subjective. Simple comes from simplex which means being composed of a single element. As opposed to complex which means being composed of several elements.
Don't mix things together that can be handled separately. Make complex things by combining simple things, not by making a mess.

n°2: Make Interfaces Fucking Obvious and Unsurprising

An interface defines types and operations using those types. Make sure the types are in tune with their usage. Make the function names accurate and consistent. Common usage should be easy. Incorrect usage should not compile. The results shall be the expected ones with no additional surprises. If in doubt, leave it out.

Inspired by:

- The C++ Core Guidelines[1]
- Simplicity Matters by Rich Hickey[2]
- How To Design A Good API and Why it Matters by Joshua Bloch[3]
- General wisdom

And of course:

- Good Fucking Design Advice[4]

[1]: https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines
[2]: https://www.youtube.com/watch?v=rI8tNMsozo0
[3]: https://www.youtube.com/watch?v=heh4OeB9A-c
[4]: https://goodfuckingdesignadvice.com

n°10: Don't Optimize the Wrong Fucking Thing

Write code at a high abstraction level and leave the job of optimizing for the specific hardware to the compiler.
Measure critical performance indicators continuously over the whole project's lifetime. Optimize only the critical paths.
Micro-optimizing random stuff without real data is just a waste of time and code.

n°3: Write Fucking Pure Functions

Pure functions doing a single thing are easy to reason about, easy to test and easy to reuse. Don't complicate your code and make it untestable by putting internal states and side-effects everywhere.

n°4: Make Data Fucking Immutable

Immutable data is more simple — its value does not depend upon time. Any value such as a string, a date, a color, etc., is better left immutable. If you need a new value, create a new one.
Mutable data makes code hard to understand because you are never quite sure what the value was for each point in time. Mutable data may suffer data-races.

n°9: Don't Overuse Fucking Inheritance

Don't rely on inheritance as a code reuse mechanism. Inheritance strongly couples the internals of every class in the hierarchy to each other. Use pure functions or composition to reuse code. Use inheritance only when there is a true "is-a" relationship.

n°8: Don't Overuse Fucking Threads and Locks

Thread and locks are a low-level operating system mechanism. They are the worst concurrency mechanism available. Threads are too coarse for most tasks and allow for data-races. Locks can cause deadlocks and degrades performance by causing lock-contention.
Use a concurrency mechanism such as the actor model, an event-oriented architecture, or an async/tasks library.

n°5: Use Fucking STL Algorithms

Don't needlessly reinvent the wheel. Avoid naked loops and unnamed algorithms. When there is an STL algorithm for the task at hand, use it. You may even find out that it performs better.

n°6: Use Fucking Resource Acquisition Is Initialization

Acquire resources in the constructor and release them in the destructor of a lifetime managing object. Use this to ensure that resources are created and released only once at the right time.

n°7: Don't Transfer Ownership Through Fucking Raw Pointers

Transfer ownership through smart pointers such as std::unique_ptr. Don't rely on someone having to read the documentation and manually do the right thing at the right time, only once, and in all code paths.