# Document your code

- Write self-documenting code
  - Clear algorithm first
    - Optimize later
  - Explicit variable names
    - Shorter names for shorter scope
  - No magic constant (number)
    - Named constants act as documentation
- Document the interface
  - e.g. Doxygen-style comments
- Document the behavior with comments in the code
  - Document the goal, the intent
  - Document the motivations, “why”
  - Do not paraphrase the code !!!

# Be coherent

- A coherent indentation scheme
  - use automated tools, indent, astyle, clang-format, …
- A coherent capitalization scheme
  - `E.g. ALL_CAPS` for constants, `CamelCase` for types, `snake_case` for variables and functions, …
- A coherent naming scheme
  - `m_` for class members, `s_` for static members, …
- Coherent include order
  - Most general to most specific, `<file>` : external, `“file.h”` : local

# Enforce modularization

- No hidden information channel between modules
  - No global variable (Global constants can be OK)
- No (global) `using` statement in headers
  - Affect in all files including the header
- Hide implementation details in classes
  - No public variable
  - Use accessor (set/get)
  - Only access classes via an interface
- … When possible (see perf. issues)

# Minimize mistakes potential

- Minimal scope for variables
  - Late declaration
  - Initialize at declaration
- `Const` everywhere
  - Most variables are never modified
- Use `override` keyword
  - Catch typos when overriding a virtual function
- Minimize namespace pollution
  - `using` for specific identifiers, not whole namespaces
- Any virtual function => a virtual destructor

# Memory handling

- In C++: a course on itself
- In 3 words: **NO RAW POINTER**
  - Who is responsible to allocate, deallocate, when, …
  - So also no
    - `malloc/free`
    - `new/delete`
- Instead use **containers** and:
  - Reference for a view
  - `unique_ptr` when single owner
  - `shared_ptr` for multiple owners
  - `weak_ptr` for a view on a `shared_ptr`
  - `const` everywhere