Using Namespace is Bad – Use Namespace

Yehezkel Bernat – yehezkelshb@gmail.com

Core C++ Meetup – June ‘18

`using namespace`

- Many examples on the internet include the infamous ‘`using namespace std;`’ directive
- Some people argue that it’s fine and maybe even better than cluttering the code with ‘`std::`’ prefix everywhere
- Some people say that it should be banned from headers (to not affect the includers, who may don’t want it) but it’s fine on source files
- Others ban it everywhere

Horror Story

- Today, I’d like to share with you a nasty bug I have seen
- Then, I’ll leave it to you to guess what is my opinion about `using namespace` ☺

On Linux it would never happen!

- Same algorithm gave a different (and wrong) result when compiled for Android (using gcc), comparing to the result on Windows (MSVC)
- What could it be?

`abs` – absolutely not what we want!

- `<cstdlib>`
  - `std::abs(int)`
  - `std::abs(long)`
  - `std::abs(long long)`
- `<cmath>`
  - `std::abs(float)`
  - `std::abs(double)`
  - `std::abs(long double)`
- (Since C++17, both headers have all the declarations)

`abs` – absolutely not what we want!

- But there is the evil brother – `abs(int)` from C
- C, without function overloading, handles different types by using different names (e.g. `fabs()`)
- `abs` is absolutely only for `int`
- Even if you pass it a `float`, it gets implicitly converted (truncated) to `int` and `int` is returned

So what’s happened there?

- The call site used just `abs()`, without `std::` prefix
  - There was a `using namespace std;` involved there
- Apparently, on Windows, with MSVC, the mix of the included (library) headers contained the declaration for `std::abs()`
  - (even without explicitly including `<cstdlib>`)
- The compiler preferred it over C `abs()`
- When compiling with gcc, only C `abs()` was visible, so this is what the compiler used
- Including `<cstdlib>` solved the issue

`using namespace` is bad

- How can we **prevent** the bug from happening?
- Don’t use `using namespace std;`
- Thus, `std::abs()` must be mentioned
  - (maybe by `using std::abs()`)
- The compiler will stop us if it can’t find the C++ version of the function

When `using namespace` is the only way

- For user-defined literals (UDLs) from a library
- E.g. for chrono literals from the standard library
  - `h`/`min`/`s`/`ms`/`us`/`ns` – C++14
  - `y`/`d` – C++20
- `using namespace std::literals::chrono_literals`
- `using namespace std::literals`
- `using namespace std::chrono_literals`

`using namespace` is Bad – Use Namespace (explicitly)