You’re writing D wrong

An old man yells his pet peeves at the clouds

Átila Neves, Ph.D.
DConf 2023
Get off my lawn

- Warning: opinions incoming.
- “An expert is someone who’s made every mistake possible”
- Rule #0: always use your head.
Refactoring is key

- Code that doesn’t change is dead.
- Nothing matters in a tiny script.
The moral of the story

- The API is important.
- The implementation is not. *

* Unless it is:

- Performance.
- Bugs.
- Ease of working with.
- ...
Stop writing classes*

- **null**: the billion dollar mistake.
- Reference semantics are terrible (action at a distance)
- What we want: value semantics.
- What we don’t want: performance penalties.
- **C++**: `std::shared_ptr<const T>`
- Don’t pay for what you don’t need.
- Not even needed: [https://github.com/atilaneves/tardy](https://github.com/atilaneves/tardy)
- Look ma, no class! `new MyStruct`

* Unless you actually need them
• Treat your implementation like your underwear drawer.
• API design needs to be explicit, not implicit.
• If your clients can, they will.
• Don’t get married to randoms off the street.
• If a private function is deleted, does it make a sound?
Don't test private functions*

- Implementation details don’t matter.
- If the behaviour can't be tested publicly . . .

* Or be willing to delete the tests
Think carefully about dependencies

- Think about who imports whom.
- Avoid cyclic dependencies.
- Organise code into D packages where it makes sense.
- dpp has a package module just for dependencies:
  - source/dpp/translation/type/package.d
  - source/dpp/translation/enum_.d
  - source/dpp/translation/macro_.d
  - ...
- D packages are like OOP class hierarchies.
• (also because of dependencies)
• What about tooling?
• Functions with too many imports stink.
Stop caring about endianness

- Rob Pike’s “The byte order fallacy”
- Mixed-endian machines exist

```c
// How to decode a 32-bit integer encoded in big-endian:
```
Stop writing auto

- But don’t explicitly write types either.
- Options: const or scope.
- What would be great:

  InputRange!(const int) rng = algo();
unittests should be @safe pure

- And objects should be const in them.
- And probably CTFE runnable (linker? pfft)
- Not always possible or desirable.
- Test taxonomy is boring. They should be:
  - Fast.
  - Consistent.
  - Flexible (i.e. not brittle).
  - Capture the true behaviour of the code.
- Don’t write tests that depend on the network.
with(immutable ReggaeSandbox("dub")) {
    runReggae("-b", "make");
    make(["VERBOSE=1"]).shouldExecuteOk.shouldContain("-debug -g");
    execute(["touch", inSandboxPath("dub.selections.json")]);
    make.shouldFailToExecute.shouldContain("reggae");
}
Don't write for loops

- Ranges are good. Use ranges.
- 99%* of for loops are map/filter/fold.
- Parallelism makes this worse:

```c
foreach(foo; foos.parallel)
    bars ~= foo;
```

* Made-up number.
Nearly every one of my non-test main functions:

```cpp
int main(string[] args) {
  try {
    run(args);
    return 0;
  } catch(Exception e) {
    import std.stdio: stderr;
    stderr.writeln("Error: ", e.msg);
    return 1;
  }
}
```
Don’t write complicated CI configurations

• ...Unless you can spin up a CI container.
• Otherwise keep it simple so it can be done locally.
scope(exit) is for one-time use

- More than once? Write a struct with a destructor.

```cpp
// intended usage:
auto handle = silly_c_api_setup(options);
scope(exit) silly_c_api_shutdown(handle);

// with a type:
with(ScopedThingie(args)) {
    ...
}
```
Don't write empty parens for function calls

...Because of refactoring

```cpp
struct InfiniteRange {
    enum empty = false;
}
```
Don't write getters. Definitely don't write setters.

- Don’t be nosy.
- Tell, don’t ask.

```rust
code
writeln("x: ", obj.x, " y: ", obj.y); // no
struct Obj {
   void write() @safe scope const {
      writeln("x: ", x, " y: ", y); // yes
   }
}
code
```
Don't use shared

- Use immutable instead.
- Or use a library.
Conclusion

- The API is important.
- The implementation is not.
- Refactoring is key.
Questions?

Slide intentionally left blank