The Present and Future of the D programming Language

Átila Neves, Ph.D.
DConf 2022
Timey Wimey Wibbly Wobbly
Where we want to go

- Minimal recompiling daemon
- Sub 20ms iteration cycles
- Fix the language without breaking user’s code
- Phobos v2 (and v3, v4, ...)
- “Header”-only Phobos
- More attribute inference
How do we get there?

- Crawl before we can walk
The state of the struct

- We have a GC so we’re memory safe (except when not)
- @nogc exists (except not trivial)
- Justified complaints about unfinished features
Unfinished Business

- The preview switches
- shared
- std.experimental
- What else?
Strategy for the previews

- Check to see if druntime/phobos/projects compile
- Check filed issues
- Decide whether to transition
- Print deprecation warnings *unless* \(-\text{revert}\) is used
- Switch the default: \(-\text{revert}\) can still be used
Remaining preview switches

- dip1008 (bug)
- fieldwise
- fixAliasThis
- rvalueRefParam (bug)
- nosharedaccess (bug)
- in
- inclusiveincontracts
- shortenedMethods
• Unclear what the community means by “unfinished”
• Bugs prevent `-preview=nosharedaccess` being made the default
• Focus on using `shared` directly is misguided
• Library help needed with something similar to `fearless`
• And/or structured concurrency in Phobos
fearless — the opposite of BYOM

- Shamelessly “inspired” by Rust’s `std::sync::Mutex`
- Convention Driven Development doesn’t scale
- DIP1000 helps with limiting access to the shared state
struct Foo { int i; }

auto foo = gcExclusive!Foo(42);
{
    int* oldIntPtr; // only here to demonstrate scopes, see below
    scope xfoo = foo.lock(); // get exclusive access
    xfoo.i = 1; // ok, locked mutex
    // ok to assign to a local that lives less
    int* intPtr;
    static assert(__traits(compiles, intPtr = &xfoo.i));
    // not ok to assign to a local that lives longer
    static assert(!__traits(compiles, oldIntPtr = &xfoo.i));
}
void func(Tid tid) @safe {
    receive(
        // ref Exclusive!Foo doesn't compile, use pointer instead
        // look ma, no shared
        (Exclusive!Foo* m) {
            auto xfoo = m.lock;
            xfoo.i++;
        },
    );
}
• Doesn’t seem to have worked as intended
• checkedint merged months ago
• code.dlang.org seems like a better alternative
• std.sumtype validates this approach
• Phobos on dub?
• Unlike allocator and logger, it’s quite small
• The plan: go over it and move to typecons
• I consulted with Robert over what needed finishing
• I looked at all the bugs that were open — no showstoppers
• Recently moved to std.logger
Open questions on std.experimental.allocator

- “Default” go-to allocators
- Synchronization setting global allocator state
- Lifetimes of the allocators themselves. RC?
- More examples of “classic” high-performance allocators in showcase
- Relationship between allocators and the GC?
- What is the “porcelain” of allocators?
Go-to allocators

• It can’t be theAllocator / processAllocator

```cpp
theAllocator = myAllocator;
{
    auto v0 = vector(1, 2);
    theAllocator = otherAllocator;
} // dtor called here: oops
```
Go-to allocators: solutions

- Only allow setting the process allocator once
- Only allow setting the thread allocator at thread creation
- Only allow replacing the allocator if no memory was allocated
- Only allow setting the allocator if the current one is the GC
Synchronization setting global allocator state

- Solution: don’t
- Aliasing makes this worse
Showcase classic allocators

- What are the examples?
Allocators and the GC

• Conceptually the GC is an allocator
• But it has guarantees that no other allocator has
Users shouldn’t be allocating memory themselves
Instead we should have library types to handle that:
  - vector
  - RC smart pointer
  - Unique smart pointer
The focus, again, should be on high-level APIs
Nobody should be calling malloc/free
Allocators: `@nogc`?

- A lot of GC resistance is a perception issue
- `@nogc` is important for that
- But `@nogc` lacking in the allocator interface
The future

- “Header”-only Phobos
- Phobos v2
- Editions
- More attribute inference
- And many more...
• Problem: distributed binary not built with same flags as the user’s
• Origin of the unittest hack (since removed)
• dub?
Phobos v2

Goals:

- Make breaking changes
- No changes to Phobos v1
- Share code between them
- @nogc
• Making breaking changes that don’t break
• Opt-in per module
• Possible that we can’t change everything
• Likely complicated compiler refactoring
More attribute inference

- In practice, most D code seems to be open-source
- If the source is available... infer?
- Not just a D problem: `constexpr Foo getFoo() noexcept const;`
- The priority should be fixing bugs and finishing features
- Only then should we look to expand
- The focus should be in high-level usage
- Help needed fixing bugs
Questions?

Slide intentionally left blank