CRYSTAL

CodeCamp | SF2017

C bindings

tapping a bigger ecosystem

What are they?

- Allows use of code exported in compilation units
- Usually compiled from C source files
- Low-level bindings
- https://crystal-lang.org/docs/syntax_and_semantics/c_bindings/

Compilation and linking process

- What is a compiler?
- What is a linker?
- Compilation unit
- Object files
- Libraries: static vs. shared libraries
- The C runtime

Example use case: embedding Lua

- What is Lua? http://www.lua.org/
- Why embed Lua?
- Lua VM and execution model
- The Lua stack

Discovering what to bind

- Library documentation
  - http://www.lua.org/manual/5.2/manual.html#4
- C header files
  - https://github.com/lua/lua/blob/v5-2/lua.h
- Exported library symbols
  - `nm /usr/local/lib/liblua.dylib`
- ABI specifications
- Calling conventions
- Name mangling

Binding definitions

- `lib` declarations
- `fun`
- `struct`, `enum` and `union`
- `type` and `alias`

Conventions

- Naming libraries
- Naming function definitions
- Higher-level abstractions

Type mapping

- How are Crystal objects laid out in memory
- Relation to C types
- `to_unsafe`
- `out` parameters
- Procs and callbacks
- Closures

Caveat: memory handling

- Dealing with separate/different memory handling models
- Interaction with Crystal’s GC
- Mixing allocation models
- Avoiding memory or resource leaks
- Finalizers
- Avoiding premature deallocations

Caveat: threading and I/O

- Mixing concurrency models
- Mixing execution models: asynchronous, synchronous, evented
- More in the Concurrency & Processes segment

Automatic generation of bindings

- Available tools for boilerplate code generation
- `crystal_lib`: https://github.com/crystal-lang/crystal_lib
- Limitations

Higher level abstractions

- Leverage Crystal language features to provide better abstractions
- Preferably without losing performance
- Wrapper classes
- Boxing
- Example: https://github.com/veelenga/lua.cr

We are Manas.

We build unconventional software_

https://manas.tech