Passerine Monthly Meeting

2022-03-05

Agenda

- Welcome/Introduction (5 min)
- Changes coming in `big-refactor` (10 min)
  - What has been done
  - What is left to do
- A tour of Passerine’s Codebase (20 min)
  - Overview of the compilation pipeline
  - How macro expansion and type-checking fit in
  - Places where you can help with `big-refactor` if interested
- Show and tell: what have you been working on? (As long as needed)
- Live coding some lexer and parser stuff (30 minutes)
  - Might open up VSCode live share if people want to hop in

Welcome to this month's Monthly Meeting!

- Happen first Saturday of the month.
- Anyone can present or share something they've been working on.
- Fairly open-ended discussion.
- Notes & slides are published in the `vrtbl/meetings` repo.
- Meeting will now be recorded for those who can't make it.

Changes coming in `big-refactor`

Done:

- Better bytecode verification and debugging tooling
- Better support for Wasm by fixing closure mangling technique
- Completely rewritten lexer and parser:
  - Support for unicode
  - 5x performance gains
  - Ready for new macro system
- Removed FFI to get ready for effect-based system injection.
- VM refactoring (3x perf gains)
- New error format, better messages

WIP:

- Support for records
- Support for lists
- Get end-to-end pipeline working

To be done:

- Support for token-based macros
- Limited effect system for system injection
- Tests and documentation
- Minor cleanup
- Type definitions
- Language prelude

What's next after `big-refactor`?

Short run:

- Type-checking (out of scope of big)
- Building out the standard library
- Compiling to MiniVM and LLVM

Long run:

- Qualm distributed runtime + Wasm support
- Vaporization memory management
- Building out the Aspen package manager
- Writing a 'real-world' application in Passerine!
- Self-hosting Passerine

Dilemma

People want to develop new features for Passerine, me included.

Developing against `master` is counterproductive, because of `big-refactor`.

So everyone's waiting for `big-refactor` to get merged.

But `big-refactor` has a ways to go before being merged.

I've been *really* short on free-time, so project has ground to a halt.

I've put *so much* work into `big-refactor`, but there's basically nothing to show for it so far.

How you can help big-refactor get merged

- Testing:
  - Write test cases and report failing ones.
  - Current focus is lexer and parser.
- Documentation:
  - Document functions in big-refactor
  - Help out with the codex!
- If you have a particular feature you’d like to help implement, I’m down to call, show you the ropes, and help get a PR in together.

Tour of Passerine’s Codebase

Macro Expansion

- After the lexing step, before the parsing step.
- Compile macro functions down to bytecode
- Convert Tokens to Passerine object
- Call macro function with converted object
- Convert result back to Tokens, splice in.

Type Checking

- Right before code generation
- All symbols have been hoisted and uniquely renamed
- Generate a set of constraints
- Try to solve constraints using standard unification
- Wrap all AST nodes in a Typed<...> Node, which identified the type of the Expr
- Compiler then has all type layout information it needs to compile native (barring monomorphization)

Show and Tell

Live Coding