C*

A Language That Could’ve Been

by Khyber Sen
Unfortunately, I didn’t have time to finish the project, and so extremely little is done at this point.

I’m sorry, it was really hard this semester.

I received barely any help from my team throughout the semester, and then in the last week they decided to leave me as well.

I worked really hard throughout this whole semester.

But evidently, this was not supposed to be a single-person project.
What C* Was Meant To Be

Instead, I’ll discuss in this presentation what the language C* could have been.

Go over the language itself.

Discuss the (intended) architecture of the compiler.
C*: the Language

- A systems programming language
- Semantic simplicity of C
- No hidden costs
- But closer to the expressiveness of Zig and Rust
- A unique fluid and postfix syntax
Major Features of C*

- Expression-oriented: everything is an expression
- Everything is postfix:
  - Except for binary operators
  - But method calls, unary operators, control flow keywords can all be postfix
- Helps the programmer code in a straightforward manner
- I.e., very little jumping back and forth is necessary while coding
- Means IDEs can provide better intellisense since everything is left-to-right

```rust
let line = client_stream.&mut.read_line(buf.&mut)
  .map_err(fn(_, _ = Status.BadRequest).
  .split(fn(b = "\t\r\n\n".contains(b)).match {
      [method, uri, version] => RequestLine { method, uri, version },
      _ => Err(Status.NotImplemented).
  });
```
Major Features of C*

- Algebraic data types: struct and enum
- Pattern matching
- Monadic error handling with the try ? Operator, Option<T>, and Result<T, E>
- Simple methods that are syntactic-sugar
- Defer for resource cleanup
- Slices
- Monomorphized, unchecked (in C++ style) generics
Compiler Architecture

Split into separable and serializable stages

Allows you to develop and test each stage in isolation

Top-level driver CLI splits a compile command into each stage and runs them, similar to clang

Development environment: dune opam esy
Compiler Stages

Source → Lexer → Tokens → Parse → AST → Desugar

Desugured AST → Name Resolution → HIR (high-level) → Type check → THIR (typed high-level) → Further semantic analysis

Lower → MIR (mid-level) → Monomorphize → LIR (low-level) → Codegen to LLVM → LLVM IR

Opt → Assembly → Assemble → Object → Link → Executable
Desugaring

Many features of C* can be desugared into others

- method call => function call
- for Loop => while loop + Option + try ?
- try ? => match
- if, if else => match
- closures => struct + method
- tuples => struct
- defer => closure on stack