CRYSTAL

CodeCamp | SF2017

Compiler internals

A tour of the Crystal compiler code

manas academyX Twentify stickermule

What is in a compiler?

- Transforms source code into executable code
- Output is object code, strictly speaking
- Code goes through a series of transformations
- Pipeline structure

Compiler pipeline

- Lexer
- Parser
- Semantic phase
- Codegen
- (Linking)

Directory layout

src/compiler/crystal

- `command/` the command line interface
- `syntax/` lexer, parser, AST, visitor, transformer
- `semantic/` type declaration, method lookup, etc.
- `macros/` macro expansion logic
- `codegen/` codegen
- `tools/` doc generator, formatter, init
- `compiler.cr` combines syntax + semantic + codegen
- `types.cr` all possible types in Crystal
- `program.cr` holds definitions of a program (holds Int32, String, etc.)

Crystal compiler pipeline

- Lexer: source code ⟶ tokens
- Parser: tokens ⟶ abstract syntax tree (AST)
- Semantic: AST ⟶ AST
- Codegen: AST ⟶ LLVM module

Compiler algorithm at a glance

src/compiler/crystal/compiler.cr

```
def compile(source : Source | Array(Source), output_filename : String) : Result
  source = [source] unless source.is_a?(Array)
  program = new_program(source)
  node = parse program, source
  node = program.semantic node, cleanup: !no_cleanup?
  codegen program, node, source, output_filename unless @no_codegen
  print_macro_run_stats(program)
  print_codegen_stats(result)
  Result.new program, node
end
```

Lexer

- Tokenizes the source code, ie. encodes plain text into tokens
- Hand coded
- /src/compiler/crystal/syntax/lexer.cr

Parser

- Checks proper syntactic constructions
- Builds the AST tree from the tokens from the lexer
- Hand coded
- /src/compiler/crystal/syntax/parser.cr

Semantic phase

- Apply the visitor pattern to traverse the AST
- Transform and expand the AST in-place
- Macro expansion (recursive)
- Other code transformations (eg. generates new methods)
- Type checking

Program

- Captures all the semantic information of the program being compiled
- Holds all types and top-level methods
- Is passed around in all phases of a compilation (except lexing and parsing, which don’t need semantic info)

Inspect compiler types

```
crystal tool hierarchy
```

Semantic phase algorithm

/src/compiler/crystal/semantic.cr#L21

- **top level**: declare classes, modules, macros, defs and other top-level stuff
- **new methods**: create `new` methods for every `initialize` method
- **type declarations**: process type declarations like `@x : Int32`
- **check abstract defs**: check that abstract defs are implemented
- **class_vars_initializers**: process initializers like `@@x = 1`
- **instance_vars_initializers**: process initializers like `@x = 1`
- **main**: process "main" code, calls and method bodies (the whole program)
- **cleanup**: remove dead code and other simplifications
- **check recursive structs**: check that structs are not recursive (impossible to codegen)

Codegen phase

- Object code is generated by LLVM
- Apply visitor pattern on the AST to create LLVM modules
- Generates LLVM objects, not `.ll` files, using bindings
- Optionally apply compiler optimizations
- Single vs. multiple modules
- Output object files
- Debug metadata generation
- (Optionally) Emit intermediate LLVM files

Linking

- Builds a command line and invokes an external linker
- Program is linked to the system C runtime
- Add all link options specified with `@[Link(...)]` attributes

Compiler tools

- /src/compiler/crystal/command.cr#L135
- (Sort of) extension point!
- All the compiler modules at your disposal

We are Manas.

We build unconventional software_

https://manas.tech