Computer Language Processing

Lab 2

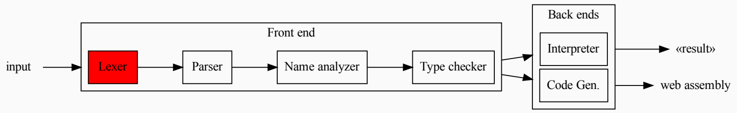
Benoît Maillard

Fall 2022

Labs overview

- Lab01 – Interpreter
- Lab02 – Lexer
- Lab03 – Parser
- Lab04 – Type Checker
- Lab05 – Codegen (Code Generator)
- Lab06 – Compiler extension

Pipeline


input
Front end
Lexer
Parser
Name analyzer
Type checker
Back ends
Interpreter
Code Gen.
«result»
web assembly

Lexer vs Parser

- Lexer
  - ▷ Input: sequence of characters
  - ▷ Output: sequence of grouped characters (tokens)
- Parser
  - ▷ Input: sequence of tokens (from the lexer)
  - ▷ Output: abstract syntax tree

Amy tokens

```
enum Token extends Positioned with Product:
  case KeywordToken(value: String)
  case BoolLitToken(value: Boolean)
  case PrimTypeToken(value: String)
  case OperatorToken(name: String)
  case DelimiterToken(value: String)
  case IdentifierToken(name: String)
  case IntLitToken(value: Int)
  case StringLitToken(value: String)
```

Example

- Input

```
val s: String = "Hello world";
s
```

- Output:

```
KeywordToken(val)(1:1)
IdentifierToken(s)(1:5)
DelimiterToken(:)(1:6)
PrimTypeToken(String)(1:8)
OperatorToken(=)(1:15)
StringLitToken(Hello world)(1:17)
DelimiterToken(;)(1:30)
IdentifierToken(s)(2:1)
EOFToken()(2:2)
```

- Display: *TokenType(args)(line:column)*

Another example

- Input

  ```
  val : a + if else s: String;
  ```

- Not a valid Amy program
- But valid input for the lexer!

Working with Silex

Silex

- Lexing library
- Write rules made of regular expressions

```
word("0b") ~ many1(oneOf("01"))
  |> { (cs, range) =>
    transformToToken(cs).setPos(range._1)
  }
```

Silex

- Lexing library
- Write rules made of regular expressions

```
word("0b") ~ many1(oneOf("01"))
  |> { (cs, range) =>
    transformToToken(cs).setPos(range._1)
  }
```

- Accepted inputs: *0b01*, *0b1000*, *0b1*, ...

Writing Silex expressions

```
def elem(char: Character): RegExp
def elem(predicate: Character => Boolean): RegExp
def oneOf(chars: Seq[Character]): RegExp
def word(chars: Seq[Character]): RegExp
```

Writing Silex expressions II

```
def many(regExp: RegExp): RegExp
def many1(regExp: RegExp): RegExp
def opt(regExp: RegExp): RegExp

sealed abstract class RegExp {
    def |(that: RegExp): RegExp
    def ~(that: RegExp): RegExp
}
```

Amy keywords example

```
word("abstract") | word("case") | word("class") |
word("fn") | word("else") | word("extends") |
word("if") | word("match") | word("object") |
word("val") | word("error") | word("_") | word("end")
  |> { (cs, range) =>
    KeywordToken(cs.mkString).setPos(range._1) },
```

Error handling and EOF

```
val lexer = Lexer(
  word("true")
    |> { (cs, range) =>
      BoolLitToken(true).setPos(range._1) },
  ... // other rules

) onError {
  (cs, range) =>
    ErrorToken(cs.mkString).setPos(range._1)
} onEnd {
  pos => EOFToken().setPos(pos)
}
```

Some advice

- Read the handout carefully
- Don't forget to call *setPosition* on tokens
- Write as many tests as possible