# SPL

## SPL: Lexical Analysis and Parsing

Pim Jager[1] Mart Lubbers[1]

[1]Computer Science: Software Science
Radboud University

March 16, 2016

# Table of Contents

# SPL

## Acronym for: SPL: Parser and Lexer

### Features

- Implementation language: Clean (`http://clean.cs.ru.nl`)

# SPL

## Acronym for: SPL: Parser and Lexer

### Features

- Implementation language: Clean (`http://clean.cs.ru.nl`)
  - Pure language
  - Higher order functions
  - Monads
  - Using parser combinator library YARD

# SPL

## Acronym for: SPL: Parser and Lexer

### Features

- Implementation language: Clean (`http://clean.cs.ru.nl`)
  - Pure language
  - Higher order functions
  - Monads
  - Using parser combinator library YARD
- Positional data available for easy locating of errors.

# SPL

## Acronym for: SPL: Parser and Lexer

### Features

- Implementation language: Clean (`http://clean.cs.ru.nl`)
  - Pure language
  - Higher order functions
  - Monads
  - Using parser combinator library YARD
- Positional data available for easy locating of errors.
- Standardized parser errors. This means you can set it as `buildprg` in `vim` and you can then use the quickfix window!

# YARD

## A minimal home grown monadic parser combinator library

- Inspired by PARSEC: $1pc = 3.375 \cdot 10^{16} yd$ [1].
- Definitons:

```
:: Error = PositionalError Int Int String | Error
    String
:: Parser a b = Parser ([a] -> (Either Error b, [a
    ]))
```

[1] A yard is exactly 36 inch and an inch is exactly the length of 3 barleycorns

# YARD

## A minimal home grown monadic parser combinator library

- Inspired by PARSEC: $1pc = 3.375 \cdot 10^{16} yd$ [1].
- Definitons:

```
:: Error = PositionalError Int Int String | Error
    String
:: Parser a b = Parser ([a] -> (Either Error b, [a
    ]))
```

- Matches longest left-most parser

[1] A yard is exactly 36 inch and an inch is exactly the length of 3 barleycorns

# YARD

## A minimal home grown monadic parser combinator library

- Inspired by PARSEC: $1pc = 3.375 \cdot 10^{16} yd$ [1].
- Definitons:

```
:: Error = PositionalError Int Int String | Error
    String
:: Parser a b = Parser ([a] -> (Either Error b, [a
    ]))
```

- Matches longest left-most parser
- Stops immediately on error

[1] A yard is exactly 36 inch and an inch is exactly the length of 3 barleycorns

# YARD

## A minimal home grown monadic parser combinator library

- Inspired by PARSEC: $1pc = 3.375 \cdot 10^{16} yd$ [1].
- Definitons:

```
:: Error = PositionalError Int Int String | Error
    String
:: Parser a b = Parser ([a] -> (Either Error b, [a
    ]))
```

- Matches longest left-most parser
- Stops immediately on error
  By design!

[1] A yard is exactly 36 inch and an inch is exactly the length of 3 barleycorns

# YARD Combinators

```
instance Functor (Parser a)
instance Applicative (Parser a)
instance Monad (Parser a)
instance Alternative (Parser a)

runParser :: (Parser a b) [a] -> (Either Error b, [a])
(<?>) :: (Parser a b) Error -> Parser a b
fail :: Parser a b
top :: Parser a a
peek :: Parser a a
satisfy :: (a -> Bool) -> Parser a a
check :: (a -> Bool) -> Parser a a
(until) infix 2 :: (Parser a b) (Parser a c) -> Parser
    a [b]
item :: a -> Parser a a | Eq a
list :: [a] -> Parser a [a] | Eq a
eof :: Parser a Void
```

# Table of Contents

# Adapting the grammar

- Remove left recursion
- Fixing associativity
- Added small features such as escape characters `\n\b`

# Adapting the grammar

- Remove left recursion
- Fixing associativity
- Added small features such as escape characters `\n\b`
- Show grammar now. . .

# Two-phase design
## Lexing

Also done with YARD because

- Multiline comments
- Alternatives
- Positions

# Two-phase design
## Parsing

Added some handy primitives

```
parseSColon :: (Parser Token a) -> Parser Token a
parseBlock :: Parser Token [Stmt]
parseOpR :: (Parser Token Op2) (Parser Token Expr) ->
    Parser Token Expr
parseOpL :: (Parser Token Op2) (Parser Token Expr) ->
    Parser Token Expr
parseBBraces :: (Parser Token a) -> Parser Token a
parseBCBraces :: (Parser Token a) -> Parser Token a
parseBSqBraces :: (Parser Token a) -> Parser Token a
trans :: TokenValue (TokenValue -> a) -> Parser Token
    (Pos, a)
trans2 :: TokenValue (TokenValue -> a) -> Parser Token
     a
trans1 :: TokenValue a -> Parser Token a
peekPos :: Parser Token Pos
```

# Table of Contents