CSCI 334:
Principles of Programming Languages

Lecture 15: Parsing, part 2

Instructor: Dan Barowy
Williams

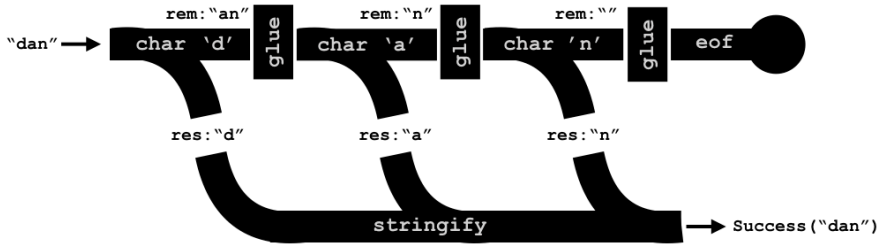
Topics

Using parser combinators

Your to-dos

1. Lab 7, **due Sunday 11/5** (partner lab)

Parser Combinators



Basic Primitives

- Input

```
type Input = string * int * bool
```

- Output

```
type Outcome<'a> =
| Success of result: 'a * remaining: Input
| Failure of fail_pos: int * rule: String
```

Basic Primitives

- A parser is

```
type Parser<'a> = Input -> Outcome<'a>
```

- Keep in mind: a parser *is a function*.

Refesher: terminal parsers

```
pchar(c: char): Parser<char>
```

```
> let input = prepare "ddd";;
val input: Input = ("ddd", 0, false)

> let d = pchar 'd';;
val d: Parser<char>

> d input;;
val it: Outcome<char> = Success ('d', ("ddd", 1, false))
```

Refresher: combining parser

```
pseq
   (p1: Parser<'a>)
   (p2: Parser<'b>)
   (f:'a -> 'b -> 'c)
   : Parser<char>
```

```
> let dd = pseq d d (fun (x,y) -> x.ToString() + y.ToString());;
val dd: Parser<string>

> dd input;;
val it: Outcome<string> = Success ("dd", ("ddd", 2, false))
```

Example: brace language

- An *expression* is a sequence of *terms*, consisting of *at least one term*.
- A *term* is either '`aaa`', '`bbb`', or a *brace expression*.
- A *brace expression* is '`{`', followed by an *expression*, followed by '`}`'.

Example: brace language

```
<expr>   ::= <term>+
<term>   ::= aaa
           |  bbb
           |  <brace>
<brace>  ::= { <expr> }
```

Let's write a parser for this language.

Recap & Next Class

Today:

Writing a parser

Next class:

WCMA