Regular expression examples

- **Identifiers:** ['_' 'a'-'z' 'A'-'Z'] ['_' 'a'-'z' 'A'-'Z' '0'-'9']*
- **Integer literals** ['0'-'9']+
- **The keyword** `if` "if"
- **The left-shift operator** "<<"
- **Floating-point literals**

['0'-'9']+ '.' ['0'-'9']+ (['E' 'e'] ['+' '-']? ['0'-'9']+)?

| ['0'-'9']+ ['E' 'e'] ['+' '-']? ['0'-'9']+

Ocamllex example 1

- In this and following examples, the trailer is omitted; it will contain definitions like the ones on the previous slide.
- What does this do?

```
rule main = parse
    ['0'-'9']+                      { "Int" }
  | ['0'-'9']+'.'['0'-'9']+         { "Float" }
  | ['a'-'z']+                      { "String" }
```

"79" → "Int"

"79.3" → "Float"

"xyz" → "String

" 79" → nothing

↑ space

Ocamllex examples 2 and 3

```
let digit = ['0'-'9']
rule main = parse
   digit+                { "Int" }
 | digit+'.'digit+       { "Float" }
 | ['a'-'z']+            { "String" }
```

Same as previous

```
{   type token = Int | Float | Ident    }
rule main = parse
   ['0'-'9']+                    { Int }
 | ['0'-'9']+'.'['0'-'9']+       { Float }
 | ['a'-'z']+                    { Ident }
```

Same, but returns values of type Token instead of strings

Ocamllex examples 4 and 5

```
{  type token = Int | Float | Ident  }
rule main = parse
   ['0'-'9']+                        { Int }
 | ['0'-'9']+'.'['0'-'9']+           { Float }
 | ['a'-'z']+                        { Ident }
 | _                                 { main lexbuf }
```

same, except "73" → Int (i.e. ignores unhandled characters)
"%-a" → Ident

```
{ type token = Int of int | Float of float | Ident of string }
rule main = parse
   ['0'-'9']+ as x                       { Int (int_of_string x) }
 | ['0'-'9']+'.'['0'-'9']+ as x          { Float (float_of_string x) }
 | ['a'-'z']+ as id                      { Ident id }
 | _                                     { main lexbuf }
```

same, but returned tokens include value of the token (int, float, or identifier)

Ocamllex example 6

```
{ type token = Int of int | Float of float | Ident of string | EOF }

rule main = parse
   ['0'-'9']+ as x                    { Int (int_of_string x) }
 | ['0'-'9']+'.'['0'-'9']+ as x       { Float (float_of_string x) }
 | ['a'-'z']+ as id                   { Ident id }
 | _                                  { main lexbuf }
 | eof                                { EOF }
```

Same, but returns EOF token if at end of input

Difficult cases...

- **Write regular expressions for various kinds of comments:**
  - **C++-style (//...)**

    "//" [^'\n']*

  - **C-style (/*... */, no nesting)**

    See ostermiller.org/findcomment.html

  - **OCaml-style ((*... *), with nesting)**

    Impossible

Handling C-style comments

```
let open_comment = "/*"
let close_comment = "*/"
rule main = parse
     digits '.' digits as f   { Float (float_of_string f) }
   | digits as n              { Int (int_of_string n) }
   | letters as s             { Ident s }
   | open_comment             { comment  lexbuf }
   | eof                      { EOF }
   | _                        { main lexbuf  }
and comment = parse
     close_comment            { main lexbuf }
   | _                        { comment lexbuf }
```

"% -/* ignore */ ab" → Ident "ab"