Parsing

CC
2016, Autumn

Main tools

- `ocamllex` is a *lexical analiser*

- (`ocamlyacc` or) `menhir` are *parser generators*

A very simple example: Arithmetic

Parser: **exp_par.mly**

```
%token <int> INT
%token PLUS
%token TIMES
%token EOF
%left  PLUS          /* low precedence */
%left  TIMES         /* high precedence */
%start <int> top
%%
top :
   | e = exp; EOF                    { e }

exp:
   | i = INT                         { i }
   | e = exp; PLUS;  f = exp         { e + f }
   | e = exp; TIMES; f = exp         { e * f }
```

Further reading

- RWOC II.16
  https://realworldocaml.org/v1/en/html/parsing-with-ocamllex-and-menhir.html

- Menhir manual
  http://gallium.inria.fr/~fpottier/menhir/manual.pdf

Lexer: **exp_lex.mll**

```
{
open Exp_par
exception SyntaxError of string
}

let int = ['0'-'9'] ['0'-'9']*
let white = [' ' '\t']+
let newline = '\r' | '\n' | "\r\n"
```

```
rule read =
   parse
   | white { read lexbuf }
   | newline   { read lexbuf }
   | int    { INT (int_of_string (Lexing.lexeme lexbuf)) }
   | '+'    { PLUS }
   | '*'    { TIMES }
   | _      { raise (SyntaxError ("Unexpected char: " ^
                                  Lexing.lexeme lexbuf)) }
   | eof    { EOF }
```

Further reading

- RWOC II.16 https://realworldocaml.org/v1/en/html/parsing-with-ocamllex-and-menhir.html

- ocamllex & ocamlyac manual (12.1-12.2) http://caml.inria.fr/pub/docs/manual-ocaml-4.00/manual026.html

- Module Lexing http://caml.inria.fr/pub/docs/manual-ocaml/libref/Lexing.html

Putting it together

```
let _ =
  read_line ()
  |> Lexing.from_string
  |> Exp_par.top Exp_lex.read
  |> print_int;
  print_newline ()
```

```
ocamlbuild -use-menhir -use-ocamlfind exp_test.native
```

Extending the grammar

Parser: **exp_par.mly**

```
%token <int> INT
%token PLUS
%token TIMES
%token EOF
%token COMMA
%left PLUS          /* low precedence */
%left TIMES         /* high precedence */
%start <int list> top
%%
top :
   | el = separated_list(COMMA, exp); EOF { el }

exp:
   | i = INT                          { i }
   | e = exp; PLUS; f = exp    { e + f }
   | e = exp; TIMES; f = exp   { e * f }
```

Top level: **exp_test.ml**

```
let rec read_to_empty buf =
    let s = read_line () in
    if s = "" then buf
    else (Buffer.add_string buf s;
          Buffer.add_string buf "\n";
          read_to_empty buf)

let _ =
  read_to_empty (Buffer.create 1)
   ▷ Buffer.contents
   ▷ Lexing.from_string
   ▷ Exp_par.top Exp_lex.read
   ▷ List.map string_of_int
   ▷ String.concat ",\n"
   ▷ print_endline
```

Errors

```
Immanuel:week2 danghica$ ./
exp_test.native
1+2,
4+5
8*9
Fatal error: exception Exp_par.Error
Immanuel:week2 danghica$
```

```
Immanuel:week2 danghica$ ./exp_test.native
1+3,
5-6,
9
```

```
Fatal error: exception
Exp_lex.SyntaxError("Unexpected char: -")
Immanuel:week2 danghica$
```

```
type position = {
   pos_fname : string;
   pos_lnum  : int;
   pos_bol   : int;
   pos_cnum  : int;
}
```

file name

line number

offset of BoL

offset of the position

Top level: **exp_test.ml**

```
open Exp_lex
open Lexing
open Printf

let print_position lexbuf =
  let pos = lexbuf.lex_curr_p in
  eprintf "Pos %d:%d:%d\n" pos.pos_lnum pos.pos_bol pos.pos_cnum

let parse_with_error lexbuf =
  try Exp_par.top Exp_lex.read lexbuf with
  | SyntaxError msg → prerr_string (msg ^ ": ");
                      print_position lexbuf;
                      exit (-1)
  | Exp_par.Error →   prerr_string "Parse error: “;
                      print_position lexbuf;
                      exit (-1)

let _ =
    read_to_empty (Buffer.create 1)
    ▷ Buffer.contents
    ▷ Lexing.from_string
    ▷ parse_with_error
    ▷ List.map string_of_int
    ▷ String.concat ",\n"
    ▷ print_endline
```

```
Immanuel:week2 danghica$ ./exp_test.native
1,2,3,
4,5,6,
m

Unexpected char: m: Pos 1:0:15
Immanuel:week2 danghica$ ./exp_test.native
1,2,3,
4 5 6
4,5,7

Parse error: Pos 1:0:10
Immanuel:week2 danghica$
```

Issues to bear in mind

- “shift/reduce” or “reduce/reduce” conflicts may be reported

- S/R is often “*benign*” (i.e. disambiguated by other rules): See Sec. 6.1 in the Menhir manual
  - typical : **if-then-else** nesting

- R/R is always considered “*severe*”

- may need to redesign the grammar
  - an art form : adding keywords usually helps

Overly clever

```
%token <int> INT
%token <int → int → int> PLUS
%token <int → int → int> MINUS
%token EOF
%start <int> top
%%
top :
   | exp; EOF { e }

exp:
   | i = INT                          { i }
   | e = exp; o = op; f = exp  { o e f }

op:
   | MINUS {( - )}
   | PLUS  {( + )}
```

```
Immanuel:week2 danghica$ menhir --explain
exp_par_rr.mly
Warning: one state has shift/reduce
conflicts.
Warning: 2 shift/reduce conflicts were
arbitrarily resolved.
```

```
** Conflict (shift/reduce) in state 8.
** Tokens involved: PLUS MINUS
** The following explanations concentrate on token PLUS.
** This state is reached from top after reading:

exp op exp

** The derivations that appear below have the following common factor:
** (The question mark symbol (?) represents the spot where the derivations begin
to differ•)

top
exp EOF
(?)

** In state 8, looking ahead at PLUS, reducing production
** exp → exp op exp
** is permitted because of the following sub-derivation:

exp op exp // lookahead token appears because op can begin with PLUS
exp op exp .

** In state 8, looking ahead at PLUS, shifting is permitted
** because of the following sub-derivation:

exp op exp
       exp op exp
           . PLUS
```

severe!

Weekly task

- design the grammar for your language
    - **parser**, **lexer**, **error-reporting**, **test suite, basic doc**
- for the given AST (at least)
- git / svn set-up