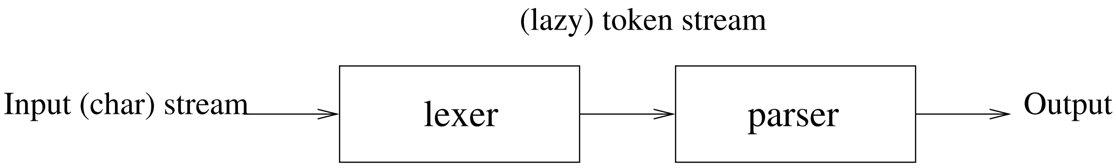
UPMC/master/info/APS-4I503
TD-TME – Syntaxe

P. Manoury

janvier 2016



1 La calculette

LEX

```
/* http://www.linux-france.org/article/devl/lexyacc/minimanlexyacc-4.html */
%{

  /*#include "global.h"*/
#define YYSTYPE double
#include "y.tab.h"

#include <stdlib.h>

%}

nls "\n"|"\r"|"\r\n"
nums [0-9]+("."[0-9]+)?
%%

[ \t]  { /* On ignore */ }

"+"   return(PLUS);
"-"   return(MINUS);

"*"   return(TIMES);
"/"   return(DIV);

"("   return(LPAR);
")"   return(RPAR);

{nls}  { return(NL); }
```

```
{nums}    {
      yylval=atof(yytext);
      return(NUM);
    }
```

YACC

```
/* http://www.linux-france.org/article/devl/lexyacc/minimanlexyacc-4.html */
%{

  /*#include "global.h"*/
#define YYSTYPE double

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

%}

%token  NUM
%token  PLUS MINUS TIMES  DIV
%token  LPAR RPAR
%token  NL

%left PLUS MINUS
%left TIMES DIV
%left NEG

%start input
%%

input:
    /* Vide */
  | input line
  ;

line:
    NL        { printf("Resultat :\n"); }
  | exp NL    { printf("Resultat : %f\n",$1); }
  ;

exp:
    NUM                  { printf("num: %f\n",$1); $$ = $1; }
  | exp PLUS exp         { $$ = $1 + $3; }
  | exp MINUS exp        { $$ = $1 - $3; }
  | exp TIMES exp        { $$ = $1 * $3; }
  | exp DIV exp          { $$ = $1 / $3; }
  | MINUS exp %prec NEG  { $$ = -$2; }
  | LPAR exp RPAR        { $$ = $2; }
  ;

%%
```

```
int yyerror(char *s) {
  printf("error: %s\n",s);
  return 1;
}

int main(void) {
  yyparse();
  return 0;
}
```

JFLEX

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * Copyright (C) 2000 Gerwin Klein <lsf@jflex.de>                          *
 * All rights reserved.                                                    *
 *                                                                         *
 * Thanks to Larry Bell and Bob Jamison for suggestions and comments.      *
 *                                                                         *
 * License: BSD                                                            *
 *                                                                         *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

%%

%byaccj

%{
  private Parser yyparser;

  public Yylex(java.io.Reader r, Parser yyparser) {
    this(r);
    this.yyparser = yyparser;
  }
%}

nums = [0-9]+ ("." [0-9]+)?
nls  = \n | \r | \r\n

%%

/* operators */
"+"  { return Parser.PLUS; }
"-"  { return Parser.MINUS; }
"*"  { return Parser.TIMES; }
"/"  { return Parser.DIV; }

/* parenthesis */
"("  { return Parser.LPAR; }
")"  { return Parser.RPAR; }

/* newline */
```

```
{nls}   { return Parser.NL; }

/* float */
{nums}  { yyparser.yylval = new ParserVal(Double.parseDouble(yytext()));
         return Parser.NUM; }

/* whitespace */
[ \t]+ { }

\b     { System.err.println("Sorry, backspace doesn’t work"); }

/* error fallback */
[^]    { System.err.println("Error: unexpected character ’"+yytext()+"’"); return -1; }
```

(B)YACC -J

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * Copyright (C) 2001 Gerwin Klein <lsf@jflex.de>                          *
 * All rights reserved.                                                    *
 *                                                                         *
 * This is a modified version of the example from                          *
 *   http://www.lincom-asg.com/~rjamison/byacc/                            *
 *                                                                         *
 * Thanks to Larry Bell and Bob Jamison for suggestions and comments.      *
 *                                                                         *
 * License: BSD                                                            *
 *                                                                         *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

%{
  import java.io.*;
%}

%token NL                    /* newline  */
%token <dval> NUM            /* a number */
%token PLUS MINUS TIMES DIV  /* operators */
%token LPAR RPAR             /* parethesis */

%type <dval> exp

%left MINUS PLUS
%left TIMES DIV
%left NEG          /* negation--unary minus */


%%

input:   /* empty string */
       | input line
       ;

line:    NL      { if (interactive) System.out.print("Expression: "); }
```

```
       | exp NL  { System.out.println(" = " + $1);
                   if (interactive) System.out.print("Expression: "); }
       ;

exp:     NUM                   { $$ = $1; }
       | exp PLUS exp          { $$ = $1 + $3; }
       | exp MINUS exp         { $$ = $1 - $3; }
       | exp TIMES exp         { $$ = $1 * $3; }
       | exp DIV exp           { $$ = $1 / $3; }
       | MINUS exp  %prec NEG  { $$ = -$2; }
       | LPAR exp RPAR         { $$ = $2; }
       ;

%%

  private Yylex lexer;


  private int yylex () {
    int yyl_return = -1;
    try {
      yylval = new ParserVal(0);
      yyl_return = lexer.yylex();
    }
    catch (IOException e) {
      System.err.println("IO error :"+e);
    }
    return yyl_return;
  }


  public void yyerror (String error) {
    System.err.println ("Error: " + error);
  }


  public Parser(Reader r) {
    lexer = new Yylex(r, this);
  }


  static boolean interactive;

  public static void main(String args[]) throws IOException {
    System.out.println("BYACC/Java with JFlex Calculator Demo");

    Parser yyparser;
    if ( args.length > 0 ) {
      // parse a file
      yyparser = new Parser(new FileReader(args[0]));
    }
    else {
      // interactive mode
      System.out.println("[Quit with CTRL-D]");
      System.out.print("Expression: ");
      interactive = true;
    yyparser = new Parser(new InputStreamReader(System.in));
    }

    yyparser.yyparse();

    if (interactive) {
      System.out.println();
      System.out.println("Have a nice day");
    }
  }
```

ocamllex

```
(* https://caml.inria.fr/pub/docs/manual-ocaml/lexyacc.html *)
{
  open Calc_yacc        (* The type token is defined in parser.mli *)
  exception Eof

}
rule token = parse
    [’ ’ ’\t’]     { token lexbuf }     (* skip blanks *)
  | [’\n’ ]        { EOL }
  | [’0’-’9’]+(’.’[’0’-’9’])? as lxm { NUM(float_of_string lxm) }
  | ’+’            { PLUS }
  | ’-’            { MINUS }
  | ’*’            { TIMES }
  | ’/’            { DIV }
  | ’(’            { LPAREN }
  | ’)’            { RPAREN }
  | eof            { raise Eof }
```

ocamlyacc

```
(* https://caml.inria.fr/pub/docs/manual-ocaml/lexyacc.html *)
%token <float> NUM
%token PLUS MINUS TIMES DIV
%token LPAREN RPAREN
%token EOL
%left PLUS MINUS        /* lowest precedence */
%left TIMES DIV         /* medium precedence */
%nonassoc UMINUS        /* highest precedence */
%start main             /* the entry point */
%type <float> main
%%
  main:
  expr EOL                { $1 }
  ;
  expr:
    NUM                     { $1 }
  | LPAREN expr RPAREN      { $2 }
  | expr PLUS expr          { $1 +. $3 }
  | expr MINUS expr         { $1 -. $3 }
  | expr TIMES expr         { $1 *. $3 }
  | expr DIV expr           { $1 /. $3 }
  | MINUS expr %prec UMINUS { -. $2 }
  ;
```

«Main» ml

```
let _ =
  try
    let lexbuf = Lexing.from_channel stdin in
      while true do
        let result = Calc_yacc.main Calc_lex.token lexbuf in
          Printf.printf"Résultat: %f\n" result; print_newline(); flush stdout
      done
  with Calc_lex.Eof ->
    exit 0
```

Makefile

```
LEX_J  = jflex
YACC_J = ~/tmp/yacc.macosx -J
JAVAC = javac

LEX_C = flex
YACC_C = yacc -d
GCC = gcc

LEX_ML = ocamllex
YACC_ML = ocamlyacc
OCAMLC = ocamlc

java: calc_j.lex calc_j.y
$(LEX_J) calc_j.lex
$(YACC_J) calc_j.y
$(JAVAC) Parser.java

c: calc_c.lex calc_c.y
$(YACC_C) -d calc_c.y
$(LEX_C) calc_c.lex
$(GCC) -c lex.yy.c
$(GCC) -c y.tab.c
$(GCC) lex.yy.o y.tab.o -ll -lm

ml: calc_ml.mll calc_ml.mly
$(LEX_ML) -o calc_lex.ml calc_ml.mll
```

```
$(YACC_ML) -b calc_yacc calc_ml.mly
$(OCAMLC) -c calc_yacc.mli
$(OCAMLC) -c calc_lex.ml
$(OCAMLC) -c calc_yacc.ml
$(OCAMLC) calc_lex.cmo calc_yacc.cmo calc_ml.ml
```

2 APS0

Syntaxe concrète

```
Prog   ::=  [ Cmds ]
Cmds   ::=  Stat | Dec ; Cmds | Stat ; Cmds
Dec    ::=  VAR ident Type
        |   CONST ident Type Expr
Stat   ::=  SET ident Expr
        |   IF Expr Prog Prog
        |   WHILE Expr Prog
Expr   ::=  true | false
        |   num | ident
        |   ( not Expr ) | ( and Expr Expr ) | ( or Expr Expr )
        |   ( eq Expr Expr ) | ( lt Expr Expr )
        |   ( add Expr Expr ) | ( sub Expr Expr )
        |   ( mul Expr Expr ) | ( div Expr Expr )
Type   ::=  bool | int
```

À la prolog

```
Prog   ::=  prog([ Cmds ])
Cmds   ::=  Stat | Dec , Cmds | Stat , Cmds
Dec    ::=  var("ident",Type)
        |   const("ident",Type,Expr)
Stat   ::=  set("ident",Expr)
        |   if(Expr,[ Cmds ],[ Cmds ])
        |   while(Expr,[ Cmds ])
Expr   ::=  true | false
        |   num | ident
        |   not( Expr) | and( Expr, Expr) | or( Expr, Expr)
        |   add(Expr, Expr) | sub( Expr, Expr) | mul( Expr, Expr) | div( Expr, Expr)
        |   eq( Expr, Expr) | lt( Expr, Expr)
Type   ::=  bool | int
```