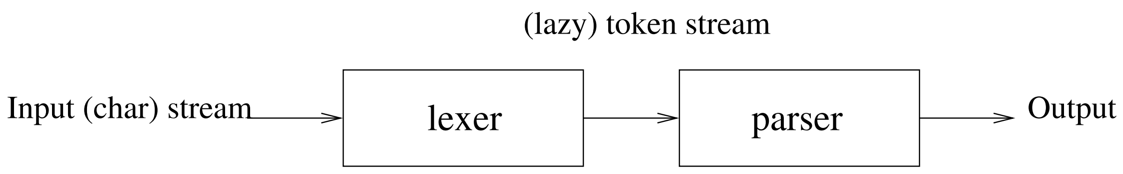
UPMC/master/info/APS-4I503
TD – Syntaxe

P. Manoury

janvier 2015



1 La calculette

JFLEX

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * Copyright (C) 2000 Gerwin Klein <lsf@jflex.de>                          *
 [..]
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

%%

%byaccj

%{
  private Parser yyparser;

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

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

%%

/* operators */
"+" | "-" | "*" | "/" | "^" | "(" | ")"
       { return (int) yycharat(0); }

/* newline */
```

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

/* float */
{NUM}  { 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>                          *
 [..]
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

%{
  import java.io.*;
%}

%token NL          /* newline  */
%token <dval> NUM  /* a number */

%type <dval> exp

%left '-' '+'
%left '*' '/'
%left NEG          /* negation--unary minus */
%right '^'         /* exponentiation        */

%%

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                { $$ = $1; }
       | exp '+' exp        { $$ = $1 + $3; }
       | exp '-' exp        { $$ = $1 - $3; }
       | exp '*' exp        { $$ = $1 * $3; }
       | exp '/' exp        { $$ = $1 / $3; }
       | '-' exp  %prec NEG { $$ = -$2; }
       | exp '^' exp        { $$ = Math.pow($1, $3); }
       | '(' exp ')'        { $$ = $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

```
(* File lexer.mll *)
{
open Parser        (* The type token is defined in parser.mli *)
exception Eof
}
rule token = parse
    [' ' '\t']     { token lexbuf }     (* skip blanks *)
  | ['\n' ]        { EOL }
  | ['0'-'9']+ as lxm { INT(int_of_string lxm) }
  | '+'            { PLUS }
  | '-'            { MINUS }
  | '*'            { TIMES }
  | '/'            { DIV }
  | '('            { LPAREN }
  | ')'            { RPAREN }
  | eof            { raise Eof }
```

ocamlyacc

```
/* File parser.mly */
%token <int> INT
%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 <int> main
%%
main:
    expr EOL                { $1 }
;
expr:
    INT                     { $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**≫

```
(* File calc.ml *)
let _ =
  try
    let lexbuf = Lexing.from_channel stdin in
    while true do
      let result = Parser.main Lexer.token lexbuf in
        print_int result; print_newline(); flush stdout
    done
  with Lexer.Eof ->
    exit 0
```

Compilation

```
ocamllex lexer.mll       # generates lexer.ml
ocamlyacc parser.mly     # generates parser.ml and parser.mli
ocamlc -c parser.mli
ocamlc -c lexer.ml
ocamlc -c parser.ml
ocamlc -c calc.ml
ocamlc -o calc lexer.cmo parser.cmo calc.cmo
```

2 BOPL

Syntaxe concrète

```
PROG          ::=  program CLASSES LOCALS INSTRLIST
CLASSES       ::=  ε | CLASS CLASSES
LOCALS         |   ε | let VARDECS in
INSTRLIST     ::=  begin INSTRSEQ end
CLASS         ::=  class id EXTENDS is VARDECLIST METHODLIST
EXTENDS       ::=  ε | extends CLASSTYPE
VARDECLIST    ::=  ε | vars VARDECS
METHODLIST    ::=  ε | methods METHODS
CLASSTYPE     ::=  Obj | Void | Int | Bool | id
VARDECS       ::=  VARDEC | VARDECS VARDEC
VARDEC        ::=  CLASSTYPE IDS ;
IDS           ::=  IDS , id
METHODS       ::=  METHOD | METHODS METHOD
METHOD        ::=  CLASSTYPE id ( ARGDECLIST ) LOCALS INSTRLIST
ARGDECLIST    ::=  ε | ARGDECS
ARGDECS       ::=  ARGDEC | ARGDECS ; ARGDEC
ARGDEC        ::=  CLASSTYPE id
INSTRSEQ      ::=  INSTR | INSTRSEQ ; INSTR
INSTR         ::=  EXP.id ( ARGLIST )
               |   id := EXP
               |   EXP.id := EXP
               |   return EXP
               |   writeln EXP
               |   if EXP then INSTRLIST else INSTRLIST
               |   while EXP do INSTRLIST
EXP           ::=  null | true | false
               |   num | id
               |   not EXP | EXP and EXP | EXP or EXP
               |   EXP + EXP | EXP - EXP | EXP * EXP | EXP / EXP
               |   EXP = EXP | EXP < EXP
               |   EXP.id | EXP.id ( ARGLIST ) | super.id ( ARGLIST )
               |   new CLASSTYPE | EXP instanceof CLASSTYPE
               |   ( EXP )
ARGLIST       ::=  ε | ARGS
ARGS          ::=  EXP | ARGS , EXP
```

À la prolog

On désigne par ITEM* ou item* une suite, possiblement vide de non-terminaux ou de terminaux séparés par une virgule.

```
Program      ::=  program([Class*], [VarDec*], [Instr*])
Class        ::=  class( id, ClassType, [VarDec*], [Method*])
ClassType    ::=  obj | void | int | bool | id
VarDec       ::=  var( ClassType, id)
Method       ::=  method( id, [VarDec*], ClassType, [VarDec*], [Instr*])
Instr        ::=  call( Exp, id, [Exp*])
               |  setvar( id, Exp)
               |  setfield( Exp, id, Exp)
               |  return( Exp)
               |  write( Exp)
               |  if( Exp, [Instr*], [Instr*])
               |  while( Exp, [Inst*])
Exp          ::=  nil | true | false
               |  num | id
               |  getfield( Exp, id) | call( Exp, id, [Exp*])
               |  new( ClassType) | instanceof( ClassType, Exp)
               |  not( Exp) | and( Exp, Exp) | or( Exp, Exp)
               |  add(Exp, Exp) | sub( Exp, Exp) | mul( Exp, Exp) | div( Exp, Exp)
               |  eq( Exp, Exp) | lt( Exp, Exp)
```