Introduction to Language Theory and Compilation
Solutions

Session 12: `yacc/bison` parser generator

The files for each exercises are available on the *Université Virtuelle*. You can use this generic CUP and JFlex files.

```
1  import java_cup.runtime.*;//Loading cup (yacc) library
2
3  %%// Options of the scanner
4
5  %class Lexer  //Name
6  %unicode      //Use unicode
7  %line       //Use line counter
8  %column     //Use character counter (by line)
9  %cup        //Use cup Java functions
10
11 %eofval{
12   return new Symbol(Symbols.EOF);
13 %eofval}
14
15 //Extended Regular Expressions
16
17
18 AlphaUpperCase  = [A-Z]
19 AlphaLowerCase  = [a-z]
20 Alpha       = {AlphaUpperCase}|{AlphaLowerCase}
21
22 Integer     = ([1-9][0-9]*)|0
23 Variable      = {Alpha}*
24
25 Space       = "\t"|" "
26 EndOfLine   = "\r"?"\n"
27
28 %%//Identification of tokens
29
30 //Simple symbols detection
31 "+"        {return new Symbol(Symbols.PLUS,yyline, yycolumn);}
32 "^"        {return new Symbol(Symbols.EXPONENT,yyline, yycolumn);}
33 "*"        {return new Symbol(Symbols.TIMES,yyline, yycolumn);}
34 "("        {return new Symbol(Symbols.LPAR,yyline, yycolumn);}
35 ")"        {return new Symbol(Symbols.RPAR,yyline, yycolumn);}
36 ";"        {return new Symbol(Symbols.SEMICOL,yyline, yycolumn);}
37 {EndOfLine}   {return new Symbol(Symbols.END_OF_LINE,yyline, yycolumn);}
38
39 //More complex symbols detection
40 {Integer}   {return new Symbol(Symbols.INTEGER,yyline, yycolumn, new Integer(yytext()));}
41 {Variable}    {return new Symbol(Symbols.VAR,yyline, yycolumn, yytext());}
42
43 //No error on spaces
44 {Space}+      {/* force match to avoid error reporting*/}
45
46 // Report token eror
47   // uncomment to define your own error reporting system, if no standalone
48   // all unmatched characted are reported as error
49 //. {System.err.println("Error on line "+yyline);}
```

```
import java_cup.runtime.*;

parser code {://code to add into generated parser
  private static Parser instance;

  public static void main(String[] args)throws Exception{
    // Simplest interactive shell
    (instance = new Parser(new Lexer(System.in))).parse();
  }
  static void checkPower(Integer power)throws Exception{
    if(power<1) instance.report_fatal_error("exponent must be > 0",power);
  }
:}

// Declaration of terminals
  // without value
terminal SEMICOL,END_OF_LINE;
terminal PLUS, TIMES, EXPONENT;
terminal LPAR, RPAR;
  // with value
terminal Integer INTEGER;
terminal String VAR;

//non terminal
  // without value
non terminal input,line;
  // with value
non terminal Information polynomial,term;

// easy way to handle ambiguous terminals
precedence left PLUS;
precedence left TIMES;
precedence left EXPONENT;

// Definition of the grammar (with non-terminals)

input ::= line input
    |   INTEGER SEMICOL line
      |   line
      ;

line ::= polynomial:p END_OF_LINE;

polynomial ::=  polynomial:a PLUS term:b  {:  :}
           |  term:value            {:  :}
           ;

term     ::= VAR:variable EXPONENT INTEGER:power              {:  :}
      | INTEGER:multiplier VAR:variable EXPONENT INTEGER:power  {:  :}
         |  VAR:variable                                     {:  :}
         |  INTEGER:multiplier VAR:variable                    {:  :}
         |  INTEGER:value                                   {:  :}
         ;
```