Printable code for FLC

Alessio Bincoletto

2021

Contents

1 Labs

1.1 Lab3: Parser for C and mini-C.

1.1.1 Es1

scanner.jflex

```
import java_cup.runtime.*;

%%

%class scanner
%unicode
%cup
%line
%column


%{
  private Symbol symbol(int type) {
    return new Symbol(type, yyline, yycolumn);
  }
  private Symbol symbol(int type, Object value) {
    return new Symbol(type, yyline, yycolumn, value);

  }
%}

nl = \r|\n|\r\n
ws = [ \t]
id = [A-Za-z_][A-Za-z0-9_]*
integer =  ([1-9][0-9]*|0)
double = (([0-9]+\.[0-9]*) | ([0-9]*\.[0-9]+)) (e|E('+'|'-')?[0-9]+)?

%%

"("       {return symbol(sym.RO);}
")"       {return symbol(sym.RC);}
"{"       {return symbol(sym.BO);}
"}"       {return symbol(sym.BC);}
"="       {return symbol(sym.EQ);}
"+"       {return symbol(sym.PLUS);}
"-"       {return symbol(sym.MINUS);}
"*"       {return symbol(sym.STAR);}
"/"       {return symbol(sym.DIV);}
"<"       {return symbol(sym.MIN);}
">"       {return symbol(sym.MAJ);}
"<="      {return symbol(sym.MIN_EQ);}
"=<"      {return symbol(sym.EQ_MIN);}
">="      {return symbol(sym.MAJ_EQ);}
"=>"      {return symbol(sym.EQ_MAJ);}
"&"       {return symbol(sym.AND);}
"|"       {return symbol(sym.OR);}
"!"       {return symbol(sym.NOT);}

"["       {return symbol(sym.SO);}
```

```
"]"      {return symbol(sym.SC);}

"int"    {return symbol(sym.INT_TYPE);}
"double" {return symbol(sym.DOUBLE_TYPE);}

print    {return symbol(sym.PRINT);}
if       {return symbol(sym.IF);}
while    {return symbol(sym.WHILE);}
else     {return symbol(sym.ELSE);}
;        {return symbol(sym.S);}
,        {return symbol(sym.CM);}

{id}"\t"{return symbol(sym.ID, yytext());}
{integer} {return symbol(sym.INT, new Integer(yytext()));}
{double}  {return symbol(sym.DOUBLE, new Double(yytext()));}

"/*" ~ "*/"      {;}

{ws}|{nl}        {;}

. {System.out.println("SCANNER ERROR: "+yytext());}
```

parser.cup

```
import java_cup.runtime.*;
import java.io.*;




parser code {:

    // Redefinition of error functions
    public void report_error(String message, Object info) {
        System.err.print("ERROR: Syntax error");
        if (info instanceof Symbol)
            if (((Symbol)info).left != -1){
                int line = (((Symbol)info).left)+1;
                int column = (((Symbol)info).right)+1;
                System.err.print(" (linea "+line+", colonna "+column+"): ");
            } else System.err.print(": ");
        else System.err.print(": ");
    }
    :};



// Terminal tokens
terminal INT, DOUBLE, ID;
terminal PRINT, IF, WHILE, ELSE;
terminal RO, RC, BO, BC, SC, CM, SO, S;
terminal PLUS, MINUS, STAR, DIV;
terminal MIN, MAJ, MIN_EQ, EQ_MIN, MAJ_EQ, EQ_MAJ, EQ;
terminal AND, OR, NOT;
terminal INT_TYPE, DOUBLE_TYPE;
terminal UMINUS;

```

```
34 // Non terminal tokens
35 non terminal prog, stmt_list, stmt, if, while, assignment, print;
36 non terminal exp;
37 non terminal mineq, mageq;
38
39 non terminal decl_list, decl, var_list, var;
40 non terminal type, array;
41 non terminal if_condition, while_condition;
42 non terminal id;
43
44 // Precedences and associativities
45 // lower precedences
46 precedence left OR;
47 precedence left AND;
48 precedence left NOT;
49 precedence left MIN, MAJ, MIN_EQ, EQ_MIN, MAJ_EQ, EQ_MAJ, EQ;
50 precedence left PLUS, MINUS;
51 precedence left STAR, DIV;
52 precedence left UMINUS;
53 // higher precedences
54
55
56 /////////////////////////////////////
57 // Grammar start
58 /////////////////////////////////////
59
60 start with prog;
61
62
63 prog ::= decl_list stmt_list {: System.out.println("Programm correctly recognized")
       ; :}
64 ;
65
66
67 /////////////////////////////////////
68 // Declarations
69 /////////////////////////////////////
70
71 decl_list ::= decl_list decl |
72 ;
73
74 decl ::= type var_list S
75 ;
76
77 type ::= INT_TYPE | DOUBLE_TYPE
78 ;
79
80 var_list ::= var | var_list CM var
81 ;
82
83 var ::= ID array
84 ;
85
86 array ::= | array SO INT SC
87 ;
88
```

```
89
90  ////////////////////////////////////
91  // Instructions
92  ////////////////////////////////////
93
94  stmt_list ::= stmt_list stmt | stmt
95  ;
96
97
98  stmt ::= if | while | assignment | print | BO stmt_list BC
99  ;
100
101 // Assignment instruction
102 assignment ::= id S | id EQ exp S
103 ;
104
105 // PRINT instruction
106 print ::= PRINT id S
107 ;
108
109
110 // IF instruction
111 if ::= IF if_condition stmt
112 ;
113
114 if_condition ::= RO exp RC
115 ;
116
117
118 // WHILE instruction
119 while ::= WHILE while_condition stmt
120 ;
121
122 while_condition ::= RO exp RC
123 ;
124
125
126 // Expressions
127 exp ::=
128     /* Espressioni logiche */
129     exp AND exp
130     | exp OR exp
131     | NOT exp
132
133     /* Espressioni di confronto */
134     | exp EQ EQ exp
135     | exp MIN exp
136     | exp MAJ exp
137     | exp mineq exp
138     | exp mageq exp
139
140     /* Espressioni aritmetiche */
141     | exp PLUS exp
142     | exp MINUS exp
143     | exp STAR exp
144     | exp DIV exp
```

```
145       | RO exp RC
146       | id
147       | INT
148       | DOUBLE
149       | MINUS INT %prec UMINUS
150       | MINUS DOUBLE %prec UMINUS
151 ;
152
153
154 mineq ::= MIN_EQ | EQ_MIN;
155 mageq ::= MAJ_EQ | EQ_MAJ;
156
157
158 id ::= ID
159        |ID SO INT SC
160        |ID SO ID SC
161 ;
```

1.1.2 Es2

scanner.jflex

```
1  import java_cup.runtime.*;
2
3
4  %%
5
6  %{
7          /* To disable debugging, i.e., printing of recogized token by means of the
               scanner set the constant _DEBUG to false */
8          private static final boolean _DEBUG = true;
9
10 %}
11
12 %class scanner
13 %unicode
14 %cup
15
16
17
18 nl                          =          \n|\r|\r\n
19 NAME                        =          \"[A-Za-z0-9_.,:]+\"_____"//ignora questo commento
       e gli spazi a sx
20 ISBN                        =          [0-9]{2}-[0-9]{2}-[0-9A-Fa-f]{5}-[A-Za-z0-9]
21 INT                         =          [1-9][0-9]*
22 LETTER                      =          [A-Za-z]
23 DATE                        =          {DAY}\/{MONTH}\/{YEAR}
24 DAY                         =          0[1-9]|[1-2][0-9]|3[0-1]
25 MONTH                       =          0[1-9]|1[0-2]
26 YEAR                        =          [0-9]{4}
27
28 %%
29
30
31 {NAME}                      {
32                                        if (_DEBUG) System.out.print("NAME_");
```

```
                                    return new Symbol(sym.NAME);
                                }
->                                  {
                                    if (_DEBUG) System.out.print("ARROW ");
                                    return new Symbol(sym.ARROW);
                                }
{ISBN}                              {
                                    if (_DEBUG) System.out.print("ISBN ");
                                    return new Symbol(sym.ISBN);
                                }
:                                   {
                                    if (_DEBUG) System.out.print("CL ");
                                    return new Symbol(sym.CL);
                                }
{INT}                               {
                                    if (_DEBUG) System.out.print("INT ");
                                    return new Symbol(sym.INT);
                                }
LI                                  {
                                    if (_DEBUG) System.out.print("LI ");
                                    return new Symbol(sym.LI);
                                }
LS                                  {
                                    if (_DEBUG) System.out.print("LS ");
                                    return new Symbol(sym.LS);
                                }
AV                                  {
                                    if (_DEBUG) System.out.print("AV ");
                                    return new Symbol(sym.AV);
                                }
BO                                  {
                                    if (_DEBUG) System.out.print("BO ");
                                    return new Symbol(sym.BO);
                                }
SO                                  {
                                    if (_DEBUG) System.out.print("SO ");
                                    return new Symbol(sym.SO);
                                }
{LETTER}                            {
                                    if (_DEBUG) System.out.print("LETTER ");
                                    return new Symbol(sym.LETTER);
                                }
\%\%                                {
                                    if (_DEBUG) System.out.print("SEP ");
                                    return new Symbol(sym.SEP);
                                }
,                                   {
                                    if (_DEBUG) System.out.print("CM ");
                                    return new Symbol(sym.CM);
                                }
;                                   {
                                    if (_DEBUG) System.out.print("S ");
                                    return new Symbol(sym.S);
                                }
{DATE}                              {
                                    if (_DEBUG) System.out.print("DATE ");
```

```
                                    return new Symbol(sym.DATE);
                                }
{nl}|" "        {;}

.               {System.out.print("SCANNER_ERROR:" + yytext());}
```

parser.cup

```
import java_cup.runtime.*;

terminal NAME, ARROW, ISBN, CL, INT, LS, LI, BO, AV, SO, LETTER, SEP, CM, S, DATE;
non terminal file, authors_list, author_entry, books, collocation, l_g, loans_list,
     loan_entry, loan_books, book;

start with file;

file ::= authors_list SEP loans_list {: System.out.println("File correctly
    recognized"); :};

authors_list ::= authors_list author_entry | author_entry;

author_entry ::= NAME ARROW books S;

books ::= books CM book | book;

book ::= ISBN CL NAME CL INT CL collocation
        | ISBN CL NAME CL INT;

collocation ::= l_g INT LETTER | l_g INT;

l_g ::= LI AV | LI SO | LS AV | LS SO | LS BO;

loans_list ::= loans_list loan_entry | loan_entry;

loan_entry ::= NAME CL loan_books S;

loan_books ::= loan_books CM DATE ISBN | DATE ISBN;
```

1.2 Lab4: Error handling for mini-C.

1.2.1 Es1

scanner.jflex

```
import java_cup.runtime.*;

%%

%class scanner
%unicode
%cup
%line
%column


%{
    private Symbol symbol(int type) {
     return new Symbol(type, yyline, yycolumn);
   }
%}

nl              =       \r | \n | \r\n
number          =       ("+"|"-")?[0-9]+("."[0-9]+(e("+"|"-")[0-9]+)?)?
ws              =       [ \t]
atom            =       [a-z][A-Za-z0-9_]*
variable        =       [A-Z_][A-Za-z0-9_]*


%%



"("             {return symbol(sym.RO);}
")"             {return symbol(sym.RC);}
"."             {return symbol(sym.PT);}
","             {return symbol(sym.CM);}
":-"            {return symbol(sym.SEP1);}
"?-"            {return symbol(sym.SEP2);}
{number}        {return symbol(sym.ATOM);}
{atom}          {return symbol(sym.ATOM);}
{variable}      {return symbol(sym.VARIABLE);}

"/*" ~ "*/"     {;}
{nl}|{ws}       {;}
```

parser.cup

```
import java_cup.runtime.*;
import java.io.*;

parser code {:

    public static int fact_found  = 0;
    public static int error_found = 0;

    /* To run the program type: java parser test.txt */
```

```
10      public static void main(String argv[]){
11          try {
12          /* Scanner instantiation */
13          scanner l = new scanner(new FileReader(argv[0]));
14          /* Parser instantiation */
15          parser p = new parser(l);
16          /* Run the parser */
17          Object result = p.parse().value;
18        } catch (NullPointerException e){
19                System.err.println("Syntax error");
20        } catch (FileNotFoundException e){
21                System.err.println("Errore opening file " + argv[0]);
22        } catch (Exception e){
23                e.printStackTrace();
24        }
25      }
26
27
28      // Redefinition of error functions
29      public void report_error(String message, Object info) {
30          System.err.print("ERROR: Syntax error");
31          if (info instanceof Symbol)
32              if (((Symbol)info).left != -1){
33                  int line = (((Symbol)info).left)+1;
34                  int column = (((Symbol)info).right)+1;
35                  System.err.print(" (row "+line+", column "+column+"): "+message);
36              } else System.err.print(": "+message);
37          else System.err.print(": "+message);
38      }
39      public void syntax_error(Symbol cur_token){}
40
41      // Return actual symbol
42      /* It returns the object of type Symbol in the top of the parser stack.
43      */
44      public Symbol getToken() {
45          return ((Symbol)stack.elementAt(tos));
46      }
47
48  :};
49
50
51
52
53  terminal SEP1, SEP2, ATOM, VARIABLE, RO, RC, CM, PT;
54  non terminal log_prog, elements, element, fact, rule, predicates, predicate,
        interrogation, arguments, argument, fact_find, interrogation_find, no_find;
55
56  start with log_prog;
57
58
59  /*****************************************************/
60  /* Solution 1 */
61  log_prog::= elements interrogation elements {:
62                  if(parser.fact_found == 0 || parser.error_found == 1 )
63                          return null;
64                  System.out.println("Program correctly recognized");
```

```
65
66 :};
67
68 elements ::= | elements element
69 ;
70
71 element ::= fact | rule
72 ;
73 /*****************************************************/
74
75
76
77 /*****************************************************/
78 /* Solution 2: other possible solution */
79 /*
80 log_prog ::= log_prog fact | log_prog rule | fact_find interrogation |
       interrogation_find fact
81 ;
82
83 fact_find ::= fact_find fact | fact_find rule | no_find fact
84 ;
85
86 interrogation_find ::= interrogation_find rule | no_find interrogation
87 ;
88
89 no_find ::= no_find rule |
90 ;
91 */
92 /*****************************************************/
93
94
95 fact ::= predicate PT {:parser.fact_found = 1; :}
96         | error PT {:
97             parser.report_error("Error in a fact\n",parser.getToken());
98             parser.error_found = 1;
99 :};
100
101 rule ::= predicate SEP1 predicates PT
102         | error SEP1 predicates PT {:
103                 parser.report_error("Error in a rule\n",parser.getToken());
104                 parser.error_found = 1;
105 :};
106
107 predicates ::= predicates CM predicate | predicate
108 ;
109
110 interrogation ::= SEP2 predicates PT {: System.out.println("Interrogation
       recognized"); :}
111                 | SEP2 error PT {:
112                         parser.report_error("Error in an interrogation\n",parser.
                            getToken());
113                         parser.error_found = 1;
114 :};
115
116 predicate ::= ATOM RO arguments RC | ATOM
117 ;
```

```
118
119 arguments ::= arguments CM argument | argument
120 ;
121
122 argument ::= predicate | VARIABLE
123 ;
```

1.2.2 Es2

scanner.jflex

```
1  import java_cup.runtime.*;
2
3  %%
4
5  %class scanner
6  %unicode
7  %cup
8  %line
9  %column
10
11
12 %{
13   private Symbol symbol(int type) {
14     return new Symbol(type, yyline, yycolumn);
15   }
16   private Symbol symbol(int type, Object value) {
17     return new Symbol(type, yyline, yycolumn, value);
18
19   }
20 %}
21
22 nl = \r|\n|\r\n
23 ws = [ \t]
24 id = [A-Za-z_][A-Za-z0-9_]*
25 integer =  ([1-9][0-9]*|0)
26 double = (([0-9]+\.[0-9]*) | ([0-9]*\.[0-9]+)) (e|E('+'|'-')?[0-9]+)?
27
28 %%
29
30 "("       {return symbol(sym.RO);}
31 ")"       {return symbol(sym.RC);}
32 "{"       {return symbol(sym.BO);}
33 "}"       {return symbol(sym.BC);}
34 "="       {return symbol(sym.EQ);}
35 "+"       {return symbol(sym.PLUS);}
36 "-"       {return symbol(sym.MINUS);}
37 "*"       {return symbol(sym.STAR);}
38 "/"       {return symbol(sym.DIV);}
39 "<"       {return symbol(sym.MIN);}
40 ">"       {return symbol(sym.MAJ);}
41 "<="      {return symbol(sym.MIN_EQ);}
42 "=<"      {return symbol(sym.EQ_MIN);}
43 ">="      {return symbol(sym.MAJ_EQ);}
44 "=>"      {return symbol(sym.EQ_MAJ);}
45 "&"       {return symbol(sym.AND);}
```

```
"|"      {return symbol(sym.OR);}
"!"      {return symbol(sym.NOT);}

"["      {return symbol(sym.SO);}
"]"      {return symbol(sym.SC);}

"int"    {return symbol(sym.INT_TYPE);}
"double" {return symbol(sym.DOUBLE_TYPE);}

print    {return symbol(sym.PRINT);}
if       {return symbol(sym.IF);}
while    {return symbol(sym.WHILE);}
else     {return symbol(sym.ELSE);}
;        {return symbol(sym.S);}
,        {return symbol(sym.CM);}

{id}       {return symbol(sym.ID, yytext());}
{integer} {return symbol(sym.INT, new Integer(yytext()));}
{double}  {return symbol(sym.DOUBLE, new Double(yytext()));}

"/*" ~ "*/"      {;}

{ws}|{nl}        {;}

. {System.out.println("SCANNER ERROR: "+yytext());}
```

parser.cup

```
import java_cup.runtime.*;
import java.io.*;



parser code {:

    public boolean isCorrect = true;

    // Redefinition of error functions
    /* The report_error function, in this program, is called only when
       an error, not managed by the error symbol, is found.
       Indeed, when errors are recognized by the error symbol, the function
       syntax_error (disabled in this program) is called.
       This program is an example of error function redefinition: two new
       functions are developed, pSynError and pSynWarning, used to print
       syntactical errors and warning, respectively. */
    public void report_error(String message, Object info) {
        System.err.print("ERROR: Syntax error");
        if (info instanceof Symbol)
            if (((Symbol)info).left != -1){
                int line = (((Symbol)info).left)+1;
                int column = (((Symbol)info).right)+1;
                System.err.print(" (linea "+line+", colonna "+column+"): ");
            } else System.err.print(": ");
        else System.err.print(": ");
    }
    public void syntax_error(Symbol cur_token){}

```

```
30          // Return the line number of actual symbol
31      public int getLine() {
32          if (((Symbol)stack.elementAt(tos)).left != -1){
33              return ((Symbol)stack.elementAt(tos)).left+1;
34          }else return -1;
35      }
36      // Return the column number of actual symbol
37      public int getColumn() {
38          if (((Symbol)stack.elementAt(tos)).left != -1){
39              return ((Symbol)stack.elementAt(tos)).right+1;
40          }else return -1;
41      }
42
43
44      :};
45
46
47  action code {:
48
49      private void pSynError(String message){
50          System.err.println("SYN ERROR: line: "+parser.getLine()+" col: "+parser.
                getColumn()+": "+message);
51          parser.isCorrect = false;
52          parser.done_parsing();
53      }
54      private void pSynWarning(String message){
55          System.err.println("SYN WARNING: line: "+parser.getLine()+" col: "+parser.
                getColumn()+": "+message);
56          parser.isCorrect = false;
57      }
58
59  :}
60
61
62  // Terminal tokens
63  terminal INT, DOUBLE;
64
65  terminal PRINT, IF, WHILE, ELSE;
66  terminal ID;
67  terminal RO, RC, BO, BC, S, CM, SO, SC;
68  terminal PLUS, MINUS, STAR, DIV;
69  terminal MIN, MAJ, MIN_EQ, EQ_MIN, MAJ_EQ, EQ_MAJ, EQ;
70  terminal AND, OR, NOT;
71  terminal INT_TYPE, DOUBLE_TYPE;
72  terminal UMINUS;
73
74
75  // Non terminal tokens
76  non terminal prog, stmt_list, stmt, if, while, assignment, print;
77  non terminal exp;
78  non terminal mineq, majeq;
79
80  non terminal decl_list, decl, var_list, var;
81  non terminal type, array;
82  non terminal if_condition, while_condition;
83  non terminal id;
```

```
84
85 // Precedences and associativities
86 // lower precedences
87 precedence left OR;
88 precedence left AND;
89 precedence left NOT;
90 precedence left MIN, MAJ, MIN_EQ, EQ_MIN, MAJ_EQ, EQ_MAJ, EQ;
91 precedence left PLUS, MINUS;
92 precedence left STAR, DIV;
93 precedence left UMINUS;
94 // higher precedences
95
96
97 /////////////////////////////////////
98 // Grammar start
99 /////////////////////////////////////
100
101 start with prog;
102
103
104 prog ::= decl_list stmt_list {: if (parser.isCorrect) System.out.println("Program␣
    correctly␣recognized"); :}
105 ;
106
107
108 /////////////////////////////////////
109 // Declarations
110 /////////////////////////////////////
111
112 decl_list ::= decl_list decl |
113 ;
114
115 decl ::= type var_list S
116     | type error S {: pSynWarning("Error␣in␣declaration"); :}
117 ;
118
119 type ::= INT_TYPE | DOUBLE_TYPE
120 ;
121
122 var_list ::= var
123     | var_list CM var
124 ;
125
126 var ::= ID array
127 ;
128
129 array ::= | array SO INT SC
130 ;
131
132
133 /////////////////////////////////////
134 // Instructions
135 /////////////////////////////////////
136
137 stmt_list ::= stmt_list stmt | stmt
138             | error stmt {: pSynWarning("Error␣in␣statement"); :}
```

```
;


stmt ::= if | while | assignment | print | BO stmt_list BC
    | BO stmt_list error BC {: pSynWarning("Missing ; before }"); :}
    | BO error BC {: pSynWarning("Missing ; before }"); :}
    | error S {: pSynWarning("Error in statement"); :}
;

// Assignment instruction
assignment ::= id S
    | id EQ exp S
    | id EQ error S {: pSynWarning("Error in expression"); :}
    | error EQ exp S {: pSynWarning("Error in assigment"); :}
;


// PRINT instruction
print ::= PRINT id S
          | PRINT error S {: pSynWarning("Error in 'print' instruction"); :}
;


// IF instruction
if ::= IF if_condition stmt ELSE stmt
     | IF if_condition stmt
     | IF if_condition stmt error stmt {: pSynWarning("Error 'else' expected in 'if
        ' instruction"); :}

;

if_condition ::= RO exp RC
    | RO error RC {: pSynWarning("Error in 'if' condition"); :}
    | error exp RC {: pSynWarning("Error '(' expected in 'if' instruciton"); :}
    | RO exp error  {: pSynWarning("Error ')' expected in 'if' instruciton"); :}
;


// WHILE instruction
while ::= WHILE while_condition stmt
;

while_condition ::= RO exp RC
    | RO error RC {: pSynWarning("Error in 'while' condition"); :}
    | error exp RC {: pSynWarning("Error '(' expected in 'while' instruciton"); :}
    | RO exp error  {: pSynWarning("Error ')' expected in 'while' instruciton"); :}
;


// Expressions
exp ::=
    /* Logical expressions */
    exp AND exp
    | exp OR exp
    | NOT exp

```

```
194     /* Comparison expressions */
195     | exp EQ EQ exp
196     | exp MIN exp
197     | exp MAJ exp
198     | exp mineq exp
199     | exp majeq exp
200
201     /* Arithmetic expression */
202     | exp PLUS exp
203     | exp MINUS exp
204     | exp STAR exp
205     | exp DIV exp
206     | RO exp RC
207     | id
208     | INT
209     | DOUBLE
210     | MINUS INT
211     | MINUS DOUBLE
212     | RO error RC {: pSynWarning("Error in expression"); :}
213 ;
214
215
216 mineq ::= MIN_EQ | EQ_MIN;
217 majeq ::= MAJ_EQ | EQ_MAJ;
218
219
220 id ::= ID
221       |ID SO INT SC
222       |ID SO ID SC
223       | error SC {: pSynWarning("Error in vector"); :}
224 ;
```

1.3 Lab5: Simple calculator.

1.3.1 Es1

scanner.jflex

```
import java_cup.runtime.*;

%%

%class Lexer
%unicode
%cup
%line
%column


%{
  private Symbol symbol(int type) {
    return new Symbol(type, yyline, yycolumn);
  }
  private Symbol symbol(int type, Object value) {
    return new Symbol(type, yyline, yycolumn, value);

  }
%}

costante = ("-")?[0-9]+(".")[0-9]+((e|E)("+"|"-")?[0-9]+)?)?
nl = \r|\n|\r\n
ws = [ \t]
scalare = [a-z]
vettore = [A-Z]
operatori = (","|")"|"("|"["|"]"|"+"|"."|"/"|"*"|"-"|"="|";"|"^"|"?")

%%
"+" {return symbol(sym.PLUS);}
"-" {return symbol(sym.MINUS);}
"/" {return symbol(sym.DIV);}
"*" {return symbol(sym.PROD);}
"(" {return symbol(sym.LBR);}
")" {return symbol(sym.RBR);}
"[" {return symbol(sym.LBS);}
"]" {return symbol(sym.RBS);}
"=" {return symbol(sym.EQUALS);}
";" {return symbol(sym.PV);}
"." {return symbol(sym.PT);}
"," {return symbol(sym.CM);}
"^" {return symbol(sym.EXP);}
"?" {return symbol(sym.QP);}
{costante} {
        return symbol(sym.CONST, new Double(yytext()));
        }
{vettore} {return symbol(sym.VECTOR_VAR, new Character(yycharat(0)));}
{scalare} {return symbol(sym.SCALAR_VAR, new Character(yycharat(0)));}

{ws}|{nl} {;}
```

parser.cup

```
import java_cup.runtime.*;
import java.io.InputStreamReader;
import java.util.HashMap;
import java.util.Vector;


init with {:
    symbol_table = new HashMap();
:};


parser code {:
    public HashMap symbol_table;

    public void syntax_error(Symbol current_symbol) {
      StringBuffer m = new StringBuffer("Error");

      if (current_symbol.left != -1) {
        m.append(" in line " + (current_symbol.left+1));
          m.append(", column " + (current_symbol.right+1));
      }
      m.append(", symbol: " + (current_symbol));
      m.append(" : Syntax Error");

         System.err.println(m);
    }
:};



terminal Double CONST;
terminal Character SCALAR_VAR;
terminal Character VECTOR_VAR;
terminal PLUS, MINUS, DIV, PROD, LBR, RBR, LBS, RBS, EQUALS, PV, PT, CM, EXP, QP;
terminal UMINUS;


non terminal Object sessione, expr_list, expr, vect_expr_, scalar_expr_,
    scalar_assign;
non terminal Object vector_assign;
non terminal Double scalar_expr, scalar;
non terminal Double[] vect_expr, vector;

precedence left MINUS, PLUS;
precedence left DIV, PROD, PT;
precedence left EXP;
precedence left UMINUS;

start with sessione;

sessione ::= expr_list QP
;

expr_list ::= expr_list PV expr | expr
;
```

```
55
56 expr ::= scalar_assign | vector_assign | vect_expr_ |scalar_expr_
57 ;
58
59 vect_expr_::= vect_expr:e {:
60         System.out.println("vector expression: [" + e[0] + ","+ e[1]+"]");
61 :};
62
63 scalar_expr_::= scalar_expr:e {:
64         System.out.println("scalar expression: " + e.doubleValue());
65 :};
66
67 // VECTOR EXPRESSIONS
68 vect_expr::= vect_expr:a PLUS vect_expr:b {:
69                 RESULT = new Double[2];
70                 RESULT[0] = new Double (a[0].doubleValue() + b[0].doubleValue());
71                 RESULT[1] = new Double (a[1].doubleValue() + b[1].doubleValue());
72         :}
73 | vect_expr:a MINUS vect_expr:b {:
74                 RESULT = new Double[2];
75                 RESULT[0] = new Double (a[0].doubleValue() - b[0].doubleValue());
76                 RESULT[1] = new Double (a[1].doubleValue() - b[1].doubleValue());
77 :}
78 | scalar_expr:a PROD vect_expr:b {:
79                 RESULT = new Double[2];
80                 RESULT[0] = new Double (a.doubleValue() * b[0].doubleValue());
81                 RESULT[1] = new Double (a.doubleValue() * b[1].doubleValue());
82
83 :}
84 | scalar_expr:a DIV vect_expr:b {:
85                 RESULT = new Double[2];
86                 RESULT[0] = new Double (a.doubleValue() / b[0].doubleValue());
87                 RESULT[1] = new Double (a.doubleValue() / b[1].doubleValue());
88 :}
89 | vect_expr:a PROD scalar_expr:b {:
90                 RESULT = new Double[2];
91                 RESULT[0] = new Double (a[0].doubleValue() * b.doubleValue());
92                 RESULT[1] = new Double (a[1].doubleValue() * b.doubleValue());
93 :}
94 | vect_expr:a DIV scalar_expr:b {:
95                 RESULT = new Double[2];
96                 RESULT[0] =  new Double (a[0].doubleValue() / b.doubleValue());
97                 RESULT[1] =  new Double (a[1].doubleValue() / b.doubleValue());
98 :}
99 | LBR vect_expr:a RBR {: RESULT = a; :}
100 | vector:a{: RESULT=a; :}
101 ;
102
103 // SCALAR EXPRESSIONS
104 scalar_expr::= scalar_expr:a PLUS scalar_expr:b {:
105         RESULT = new Double(a.doubleValue()+ b.doubleValue());
106 :}
107 | scalar_expr:a MINUS scalar_expr:b {:
108         RESULT = new Double(a.doubleValue()- b.doubleValue());
109 :}
110 | scalar_expr:a PROD scalar_expr:b {:
```

```
111         RESULT = new Double(a.doubleValue()* b.doubleValue());
112 :}
113 | scalar_expr:a DIV scalar_expr:b {:
114         RESULT = new Double(a.doubleValue()/ b.doubleValue());
115 :}
116 | MINUS scalar_expr:a  {:
117         RESULT = new Double(- a.doubleValue());
118 :} %prec UMINUS
119 | scalar_expr:b EXP scalar_expr:e {:
120         RESULT = new Double(Math.pow(b.doubleValue(), e.doubleValue()));
121 :}
122 | LBR scalar_expr:e RBR {:
123         RESULT = e;
124 :}
125 | vect_expr:a PT vect_expr:b {:
126         RESULT = new Double(a[0].doubleValue() * b[0].doubleValue() + a[1].
              doubleValue() * b[1].doubleValue());
127 :}
128 | scalar:a{: RESULT = a; :}
129 ;
130
131 // ASSIGNMENTS
132 scalar_assign::= SCALAR_VAR:a EQUALS scalar_expr:b
133 {:
134         parser.symbol_table.put(a,b);
135         System.out.println("assignment: " + a + "->" + b);
136 :};
137
138 vector_assign::= VECTOR_VAR:a EQUALS vect_expr:b {:
139         parser.symbol_table.put(a,b);
140         System.out.println("assignment: " + a + "->[" + b[0]+ "," + b[1]+"]");
141 :};
142
143 scalar::= CONST:a{:RESULT = a;:}|SCALAR_VAR:a {:
144         RESULT = (Double)parser.symbol_table.get(a);
145 :};
146
147 vector::= VECTOR_VAR:a {:
148         RESULT = (Double[]) parser.symbol_table.get(a);
149 :}
150
151 | LBS scalar_expr:a CM scalar_expr:b RBS {:
152         RESULT = new Double[2];
153         RESULT[0] =  a;
154         RESULT[1] =  b;
155 :};
```

1.4 Lab6: Translator from mini C to Pseudo Assembler.

1.4.1 Es1

scanner.jflex

```
import java_cup.runtime.*;

%%

%class scanner
%unicode
%cup
%line
%column


%{
  private Symbol symbol(int type) {
    return new Symbol(type, yyline, yycolumn);
  }
  private Symbol symbol(int type, Object value) {
    return new Symbol(type, yyline, yycolumn, value);

  }
%}

nl = \r|\n|\r\n
ws = [ \t]
id = [A-Za-z_][A-Za-z0-9_]*
integer =  ([1-9][0-9]*|0)
double = (([0-9]+\.[0-9]*) | ([0-9]*\.[0-9]+)) (e|E('+'|'-')?[0-9]+)?

%%

"("       {return symbol(sym.RO);}
")"       {return symbol(sym.RC);}
"{"       {return symbol(sym.BO);}
"}"       {return symbol(sym.BC);}
"="       {return symbol(sym.EQ);}
"+"       {return symbol(sym.PLUS);}
"-"       {return symbol(sym.MINUS);}
"*"       {return symbol(sym.STAR);}
"/"       {return symbol(sym.DIV);}
"<"       {return symbol(sym.MIN);}
">"       {return symbol(sym.MAJ);}
"<="      {return symbol(sym.MIN_EQ);}
"=<"      {return symbol(sym.EQ_MIN);}
">="      {return symbol(sym.MAJ_EQ);}
"=>"      {return symbol(sym.EQ_MAJ);}
"&"       {return symbol(sym.AND);}
"|"       {return symbol(sym.OR);}
"!"       {return symbol(sym.NOT);}

"["       {return symbol(sym.SO);}
"]"       {return symbol(sym.SC);}

```

```
"int"    {return symbol(sym.INT_TYPE);}
"double" {return symbol(sym.DOUBLE_TYPE);}

print    {return symbol(sym.PRINT);}
if       {return symbol(sym.IF);}
while    {return symbol(sym.WHILE);}
else     {return symbol(sym.ELSE);}
;        {return symbol(sym.S);}
,        {return symbol(sym.CM);}

{id}      {return symbol(sym.ID, yytext());}
{integer} {return symbol(sym.INT, new Integer(yytext()));}
{double}  {return symbol(sym.DOUBLE, new Double(yytext()));}

"/*" ~ "*/"      {;}

{ws}|{nl}        {;}

. {System.out.println("SCANNER ERROR: "+yytext());}
```

parser.cup

```
import java_cup.runtime.*;
import java.io.*;




init with {:
    // String buffer used to store output program
    outputBuffer = new StringBuffer();
:};


parser code {:
    // Represent the number of the first usable label
    public int label = 0;

    // It can be "stdout" to write output program to standard
    // output or "file" to dump program in a file.
    public static String dumpOutput;

    // It's true if the semantic check is enabled
    public boolean enableSem = true;

    // String buffer used to store output program
    public StringBuffer outputBuffer;

    // Generation of the next label number
    public int genLabel(){
        label++;
        return label;
    };

    // Redefinition of error functions
    public void report_error(String message, Object info) {
        System.err.print("ERROR: Syntax error");
```

```
36          if (info instanceof Symbol)
37              if (((Symbol)info).left != -1){
38                  int line = (((Symbol)info).left)+1;
39                  int column = (((Symbol)info).right)+1;
40                  System.err.print(" (linea "+line+", colonna "+column+"): ");
41              } else System.err.print(": ");
42          else System.err.print(": ");
43      }
44      public void syntax_error(Symbol cur_token){}
45
46      // Return actual symbol
47      public Symbol getToken() {
48          return ((Symbol)stack.elementAt(tos));
49      }
50
51      // Return semantic value of symbol in position (position)
52      public Object stack(int position) {
53          return (((Symbol)stack.elementAt(tos+position)).value);
54      }
55
56      // Return the line number of actual symbol
57      public int getLine() {
58          if (((Symbol)stack.elementAt(tos)).left != -1){
59              return ((Symbol)stack.elementAt(tos)).left+1;
60          }else return -1;
61      }
62      // Return the column number of actual symbol
63      public int getColumn() {
64          if (((Symbol)stack.elementAt(tos)).left != -1){
65              return ((Symbol)stack.elementAt(tos)).right+1;
66          }else return -1;
67      }
68      :};
69
70
71  action code {:
72      // Disable semantic check
73      private void disableSem(){
74          parser.enableSem = false;
75      }
76      // Return true if semantic is enabled, false otherwise
77      private boolean sem(){
78          return parser.enableSem;
79      }
80
81      // Error management
82      private void pSemError(String message){
83          System.err.println("SEM ERROR: line: "+parser.getLine()+" col: "+parser.
                getColumn()+": "+message);
84
85          parser.done_parsing();
86      }
87      private void pSemWarning(String message){
88          System.err.println("SEM WARNING: line: "+parser.getLine()+" col: "+parser.
                getColumn()+": "+message);
89      }
```

```
90      private void pSynError(String message){
91          System.err.println("SYN_ERROR: line: "+parser.getLine()+" col: "+parser.
                getColumn()+": "+message);
92          parser.done_parsing();
93      }
94      private void pSynWarning(String message){
95          System.err.println("SYN_WARNING: line: "+parser.getLine()+" col: "+parser.
                getColumn()+": "+message);
96          /* Quando c'e' un errore sintattico continuo il parsing ma disabilito la
                semantiva */
97          disableSem();
98      }
99
100     // Write a string in output
101     private void dump(String s){
102         if (parser.dumpOutput == "stdout"){
103             System.out.print(s);
104         }else{
105             parser.outputBuffer.append(s);
106         }
107     }
108     private void dumpln(String s){
109         if (parser.dumpOutput == "stdout"){
110             System.out.println(s);
111         }else{
112             parser.outputBuffer.append(s+"\n");
113         }
114     }
115
116     :}
117
118
119 // Terminal tokens
120 terminal Integer INT;
121 terminal Double DOUBLE;
122
123 terminal PRINT, IF, WHILE,  THEN, ELSE;
124 terminal String ID;
125 terminal RO, RC, BO, BC, S, CM, SO, SC;
126 terminal PLUS, MINUS, STAR, DIV;
127 terminal MIN, MAJ, MIN_EQ, EQ_MIN, MAJ_EQ, EQ_MAJ, EQ;
128 terminal AND, OR, NOT;
129 terminal INT_TYPE, DOUBLE_TYPE;
130 terminal UMINUS;
131
132
133 // Non terminal tokens
134 non terminal prog, stmt_list, stmt, if, while, assignment, print;
135 non terminal Integer[] nt0_while;
136 non terminal Integer nt0_if, nt1_if;
137 non terminal String exp;
138 non terminal mineq, majeq;
139
140 non terminal decl_list, decl, var_list, var;
141 non terminal String type, array;
142 non terminal String if_condition, while_condition;
```

```
143 non terminal String id;
144
145 // Precedences and associativities
146 // lower precedences
147 precedence left OR;
148 precedence left AND;
149 precedence left NOT;
150 precedence left MIN, MAJ, MIN_EQ, EQ_MIN, MAJ_EQ, EQ_MAJ, EQ;
151 precedence left PLUS, MINUS;
152 precedence left STAR, DIV;
153 precedence left UMINUS;
154 // higher precedences
155
156
157 /////////////////////////////////////
158 // Grammar start
159 /////////////////////////////////////
160
161 start with prog;
162
163
164 prog ::= decl_list stmt_list {:
165                                 dumpln("\tEND");
166                                 if (parser.dumpOutput=="stdout"){
167                                     System.out.println(parser.outputBuffer);
168                                 }else{
169                                     try {
170                                         BufferedWriter out = new BufferedWriter(new
                                             FileWriter(parser.dumpOutput));
171                                         String outText = parser.outputBuffer.toString
                                             ();
172                                         out.write(outText);
173                                         out.close();
174                                     }
175                                     catch (IOException e)
176                                         {
177                                             e.printStackTrace();
178                                         }
179
180                                 }
181
182                                 :}
183     ;
184
185
186 /////////////////////////////////////
187 // Declarations
188 /////////////////////////////////////
189
190 decl_list ::= decl_list decl | ;
191
192 decl ::= type var_list S
193     | type error S {: pSynWarning("Error in declaration"); :}
194 ;
195
196 type ::= INT_TYPE {: RESULT = new String("INT"); :}| DOUBLE_TYPE {: RESULT = new
```

```
    String("DOUBLE"); :};

var_list ::= var
    | var_list CM {: RESULT = parser.stack(-2); :} var
;

var ::= ID:x array:y {: dumpln("\t"+parser.stack(-2)+" "+x+y); :};

array ::= {: RESULT = new String(""); :}
    | array:x SO INT:y SC {: RESULT = x+"["+y.toString()+"]"; :}
;


////////////////////////////////////
// Instructions
////////////////////////////////////

stmt_list ::= stmt_list stmt | stmt
            | error stmt {: pSynWarning("Error in statement"); :};


stmt ::= if | while | assignment | print | BO stmt_list BC
    | BO stmt_list error BC {: pSynWarning("Missing ; before }"); :}
    | BO error BC {: pSynWarning("Missing ; before }"); :}
    | error S {: pSynWarning("Error in statement"); :}
;

// Assignment instruction
assignment ::= id:x S {: if (sem()){ dumpln("\t"+x);} :}
    | id:x EQ exp:y S {: if (sem()){ dumpln("\tEVAL "+y.toString()+"\n\tASS "+x);}
      :}
    | id EQ error S {: pSynWarning("Error in expression"); :}
    | error EQ exp S {: pSynWarning("Error in assigment"); :}
;


// PRINT instruction
print ::= PRINT id:x S {: if (sem()){ dumpln("\tPRINT "+x);} :}
        | PRINT error S {: pSynWarning("Error in 'print' instruction"); :}
;


// IF instruction
if ::= IF if_condition nt0_if stmt ELSE nt1_if stmt {:
                                                    if (sem()){
                                                        dump("L"+parser.stack
                                                            (-1)+":");
                                                    }
                                                    :}
     | IF if_condition:e nt0_if stmt {: if (sem()){ dump("L"+parser.stack(-1)+":")
       ;} :}
     | IF if_condition:e nt0_if stmt error nt1_if stmt {: pSynWarning("Error 'else'
       expected in 'if' instruction"); :}

;

```

```
248 if_condition ::= RO exp:e RC {: RESULT=e; :}
249     | RO error RC {: pSynWarning("Error in 'if' condition"); :}
250     | error exp RC {: pSynWarning("Error '(' expected in 'if' instruciton"); :}
251     | RO exp error  {: pSynWarning("Error ')' expected in 'if' instruciton"); :}
252 ;
253
254 nt0_if ::= {:
255             if (sem()){
256                 RESULT=parser.genLabel();
257                 dumpln("\tEVAL "+parser.stack(0)+"\t\t/* if (line "+parser.getLine
                        ()+") */\n\tGOTOF L"+RESULT);
258             }
259             :}
260 ;
261
262 nt1_if ::= {:
263             if (sem()){
264                 RESULT=parser.genLabel();
265                 dumpln("\tGOTO L"+RESULT);
266                 dump("L"+parser.stack(-2)+":");
267             }
268             :}
269 ;
270
271
272 // WHILE instruction
273 while ::= WHILE while_condition nt0_while stmt {:
274                                             if (sem()){
275                                                 Integer[] l=(Integer[])parser.stack
                                                        (-1);
276                                                 dumpln("\tGOTO L"+l[0]);
277                                                 dump("L"+l[1]+":");
278                                             }
279                                             :};
280 while_condition ::= RO exp:e RC {: RESULT=e; :}
281     | RO error RC {: pSynWarning("Error in 'while' condition"); :}
282     | error exp RC {: pSynWarning("Error '(' expected in 'while' instruciton"); :}
283     | RO exp error  {: pSynWarning("Error ')' expected in 'while' instruciton"); :}
284 ;
285
286
287
288 nt0_while ::= {:
289                     if (sem()){
290                         RESULT=new Integer[2];
291                         RESULT[0]=(Integer)parser.genLabel();
292                         RESULT[1]=(Integer)parser.genLabel();
293                         dumpln("L"+RESULT[0]+":\tEVAL "+parser.stack(0)+"\t\t/*
                                while (line "+parser.getLine()+") */\n\tGOTOF L"+RESULT
                                [1]);
294                     }
295                 :}
296 ;
297
298
299 // Expressions
```

```
exp ::=
    /* Logical expressions */
    exp:x AND exp:y  {: RESULT=new String(x+" "+y+" & "); :}
    | exp:x OR exp:y {: RESULT=new String(x+" "+y+" | "); :}
    | NOT exp:x      {: RESULT=new String(x+" ! "); :}

    /* Comparision expressions */
    | exp:x EQ EQ exp:y {: RESULT=new String(x+" "+y+" == "); :}
    | exp:x MIN exp:y   {: RESULT=new String(x+" "+y+" < "); :}
    | exp:x MAJ exp:y   {: RESULT=new String(x+" "+y+" > "); :}
    | exp:x mineq exp:y {: RESULT=new String(x+" "+y+" <= "); :}
    | exp:x majeq exp:y {: RESULT=new String(x+" "+y+" >= "); :}

    /* Arithmetic expressions */
    | exp:x PLUS  exp:y {: RESULT=new String(x+" "+y+" + "); :}
    | exp:x MINUS exp:y {: RESULT=new String(x+" "+y+" - "); :}
    | exp:x STAR  exp:y {: RESULT=new String(x+" "+y+" * "); :}
    | exp:x DIV   exp:y {: RESULT=new String(x+" "+y+" / "); :}
    | RO exp:x RC {: RESULT=x; :}
    | id:x {: RESULT=x; :}
    | INT:x {: RESULT=new String(x.toString()); :}
    | DOUBLE:x {: RESULT=new String(x.toString()); :}
    | MINUS INT:x {: RESULT=new String("-"+x.toString()); :} %prec UMINUS
    | MINUS DOUBLE:x {: RESULT=new String("-"+x.toString()); :} %prec UMINUS
    | RO error RC {: pSynWarning("Error in expression"); :}
;


mineq ::= MIN_EQ | EQ_MIN;
majeq ::= MAJ_EQ | EQ_MAJ;


id ::= ID:x {: RESULT=x; :}
      |ID:x SO INT:y SC {: RESULT=new String(x.toString()+"["+y.toString()+"]"); :}
      |ID:x SO ID:y SC {: RESULT=new String(x.toString()+"["+y.toString()+"]"); :}
      | error SC {: pSynWarning("Error in vector"); :}
;
```

1.5 Lab7: Type checking.

1.5.1 Es1

scanner.jflex

```
import java_cup.runtime.*;

%%

%class Lexer
%unicode
%cup
%line
%column


%{
  private Symbol symbol(int type) {
    return new Symbol(type, yyline, yycolumn);
  }
  private Symbol symbol(int type, Object value) {
    return new Symbol(type, yyline, yycolumn, value);

  }
%}

nl = \r|\n|\r\n
ws = [ \t]
id = [A-Za-z_][A-Za-z0-9_]*
integer =  ([1-9][0-9]*|0)
double = (([0-9]+\.[0-9]*) | ([0-9]*\.[0-9]+)) (e|E('+'|'-')?[0-9]+)?

%%
"("      {return symbol(sym.RO);}
")"      {return symbol(sym.RC);}
"{"      {return symbol(sym.BO);}
"}"      {return symbol(sym.BC);}
"="      {return symbol(sym.EQ);}
"+"      {return symbol(sym.PLUS);}
"-"      {return symbol(sym.MINUS);}
"*"      {return symbol(sym.STAR);}
"/"      {return symbol(sym.DIV);}
"<"      {return symbol(sym.MIN);}
">"      {return symbol(sym.MAJ);}
"<="     {return symbol(sym.MIN_EQ);}
"=<"     {return symbol(sym.EQ_MIN);}
">="     {return symbol(sym.MAJ_EQ);}
"=>"     {return symbol(sym.EQ_MAJ);}
"&"      {return symbol(sym.AND);}
"|"      {return symbol(sym.OR);}
"!"      {return symbol(sym.NOT);}

"["      {return symbol(sym.SO);}
"]"      {return symbol(sym.SC);}

"int"    {return symbol(sym.INT_TYPE);}
```

```
"double" {return symbol(sym.DOUBLE_TYPE);}

print    {return symbol(sym.PRINT);}
if       {return symbol(sym.IF);}
while    {return symbol(sym.WHILE);}
else     {return symbol(sym.ELSE);}
then     {return symbol(sym.THEN);}
;        {return symbol(sym.S);}
,        {return symbol(sym.CM);}

{id}       {return symbol(sym.ID, yytext());}
{integer} {return symbol(sym.INT, new Integer(yytext()));}
{double}  {return symbol(sym.DOUBLE, new Double(yytext()));}

"/*" ~ "*/"      {;}

{ws}|{nl}        {;}

. {System.out.println("SCANNER_ERROR: "+yytext());}
```

parser.cup

```
/*****************************************
 Mini-C to Pseudo Assembler translator

 Author: Stefano Scanzio (stefano.scanzio@polito.it)
 Date: June 2011

 *****************************************/

import java_cup.runtime.*;
import java.io.*;
import java.util.HashMap;




init with {:
    /* String buffer used to store the program output */
    outputBuffer = new StringBuffer();

    /* String buffer used to store errors and warnings */
    errorBuffer = new StringBuffer();
:};


parser code {:
    /* Symbol table for type checking */
    HashMap<String,SymbolType> symbolType_table = new HashMap<String,SymbolType>();

    /* It represents the number of the first usable label */
    public int label = 0;

    /* It can be "stdout" to write output program to standard
       output or "file" to dump program in a file. */
    public static String dumpOutput;

```

```
36      /* It's true if the semantic check is enabled */
37      public boolean enableSem = true;
38
39      /* Number of semantic errors */
40      public int semErrors = 0;
41      /* Number of semantic warnings */
42      public int semWarnings = 0;
43      /* Number of syntactic warnings */
44      public int synWarnings = 0;
45
46      /* String buffer used to store the output of the program */
47      public StringBuffer outputBuffer;
48
49      /* String buffer used to store program errors */
50      public StringBuffer errorBuffer;
51
52      /* Generation of the next label number */
53      public int genLabel(){
54          label++;
55          return label;
56      };
57
58      /* Redefinition of error functions */
59      public void report_error(String message, Object info) {
60          System.err.print("ERROR: Syntax error");
61          if (info instanceof Symbol)
62              if (((Symbol)info).left != -1){
63                  int line = (((Symbol)info).left)+1;
64                  int column = (((Symbol)info).right)+1;
65                  System.err.print(" (linea "+line+", colonna "+column+"): ");
66              } else System.err.print(": ");
67          else System.err.print(": ");
68      }
69      public void syntax_error(Symbol cur_token){}
70
71      /* Return actual symbol */
72      public Symbol getToken() {
73          return ((Symbol)stack.elementAt(tos));
74      }
75
76      /* Return semantic value of symbol in position (position) */
77      public Object stack(int position) {
78          return (((Symbol)stack.elementAt(tos+position)).value);
79      }
80
81      /* Return the line number of actual symbol */
82      public int getLine() {
83          if (((Symbol)stack.elementAt(tos)).left != -1){
84              return ((Symbol)stack.elementAt(tos)).left+1;
85          }else return -1;
86      }
87      /* Return the column number of actual symbol */
88      public int getColumn() {
89          if (((Symbol)stack.elementAt(tos)).left != -1){
90              return ((Symbol)stack.elementAt(tos)).right+1;
91          }else return -1;
```

```
92      }
93      :};
94
95
96  action code {:
97
98      /* Class used to store expression and to do type checking on expressions */
99      class Expr{
100         private String value;
101         private SymbolType type;
102
103         private SymbolType lookupSymbolType(String id){
104             SymbolType type = parser.symbolType_table.get(id);
105             if (type == null){
106                 pSemError("Variable \""+id+"\" not declared");
107                 return new SymbolType(-1, -1);
108             }
109             return type;
110         }
111
112         Expr(String value, SymbolType type){
113             this.value = value;
114             this.type = type;
115         }
116         Expr(String id){
117             this.value = id;
118             this.type = lookupSymbolType(id);
119         }
120
121         Expr(String id, Integer pos){
122             this.value = id+"["+pos.toString()+"]";
123             this.type = lookupSymbolType(id);
124
125             int dim = type.getDim();
126             if (pos>=dim && dim!=-1){
127                 pSemError("Array index ("+pos+") exceed array size ("+dim+")");
128             }
129         }
130         Expr(String id, String pos){
131             this.value = id+"["+pos+"]";
132             this.type = lookupSymbolType(id);
133
134         }
135
136
137         public String toString(){
138             return value;
139         }
140         public SymbolType getSymbolType(){
141             return type;
142         }
143
144         /* Check symbol type. In return unknown type in the case of type error */
145         public SymbolType checkSymbolType(Expr expr){
146             int type1 = type.getType();
147             int type2 = expr.getSymbolType().getType();
```

```
148
149             if (type1==type2){
150                 return type;
151             }else if(type1!=-1 && type2!=-1){
152                 /* If operands are of two different types cast them to double */
153                 pSemWarning("Operation between int and double, int number casted to
                        double");
154                 return new SymbolType(1, 1);
155             }else{
156                 return new SymbolType(-1, -1);
157             }
158         }
159         public void checkSymbolTypeAssignment(Expr expr){
160             int type1 = type.getType();
161             int type2 = expr.getSymbolType().getType();
162
163             if (type1==0 && type2==1){
164                 pSemWarning("Assignment of a double value to an int variable");
165             } else if (type1==1 && type2==0){
166                 pSemWarning("Assignment of an int value to an double variable");
167             }
168         }
169     }
170
171
172
173     /* Disable semantic check */
174     private void disableSem(){
175         parser.enableSem = false;
176     }
177     /* Return true if semantic is enabled, false otherwise */
178     private boolean sem(){
179         return parser.enableSem;
180     }
181
182     /* Error management */
183     private void pSemError(String message){
184         parser.errorBuffer.append("SEM ERROR: line: "+parser.getLine()+" col: "+
                parser.getColumn()+": "+message+"\n");
185         parser.semErrors++;
186     }
187     private void pSemWarning(String message){
188         parser.errorBuffer.append("SEM WARNING: line: "+parser.getLine()+" col: "+
                parser.getColumn()+": "+message+"\n");
189         parser.semWarnings++;
190     }
191     private void pSynError(String message){
192         System.err.println("SYN ERROR: line: "+parser.getLine()+" col: "+parser.
                getColumn()+": "+message);
193         System.err.println("Could not continue parsing");
194         parser.done_parsing();
195     }
196     private void pSynWarning(String message){
197         parser.errorBuffer.append("SYN WARNING: line: "+parser.getLine()+" col: "+
                parser.getColumn()+": "+message+"\n");
198         parser.synWarnings++;
```

```
199         /* When there is a syntactic warning semantic is disable to avoid errors
                due to invalid data structures */
200         disableSem();
201     }
202 
203     /* Functions to dump program output */
204     private void dump(String s){
205             parser.outputBuffer.append(s);
206     }
207     private void dumpln(String s){
208             parser.outputBuffer.append(s+"\n");
209     }
210 
211     :}
212 
213 
214 // Terminal tokens
215 terminal Integer INT;
216 terminal Double DOUBLE;
217 
218 terminal PRINT, IF, WHILE,  THEN, ELSE;
219 terminal String ID;
220 terminal RO, RC, BO, BC, S, CM, SO, SC;
221 terminal PLUS, MINUS, STAR, DIV;
222 terminal MIN, MAJ, MIN_EQ, EQ_MIN, MAJ_EQ, EQ_MAJ, EQ;
223 terminal AND, OR, NOT;
224 terminal INT_TYPE, DOUBLE_TYPE;
225 terminal UMINUS;
226 
227 
228 // Non terminal tokens
229 non terminal prog, stmt_list, stmt, if, while, assignment, print;
230 non terminal Integer[] nt0_while;
231 non terminal Integer nt0_if, nt1_if;
232 non terminal Expr exp;
233 non terminal mineq, majeq;
234 
235 non terminal decl_list, decl, var_list, var;
236 non terminal String type, array;
237 non terminal String if_condition, while_condition;
238 non terminal Expr id;
239 
240 // Precedences and associativities
241 // lower precedences
242 precedence left OR;
243 precedence left AND;
244 precedence left NOT;
245 precedence left MIN, MAJ, MIN_EQ, EQ_MIN, MAJ_EQ, EQ_MAJ, EQ;
246 precedence left PLUS, MINUS;
247 precedence left STAR, DIV;
248 precedence left UMINUS;
249 // higher precedences
250 
251 
252 ////////////////////////////////////
253 // Grammar start
```

```
254 ////////////////////////////////////
255
256 start with prog;
257
258
259 prog ::= decl_list stmt_list {:
260                                 if(sem() && parser.semErrors==0) {
261                                     dumpln("\tEND");
262                                     if (parser.dumpOutput=="stdout"){
263                                         System.out.println(parser.outputBuffer);
264                                     }else{
265                                         try {
266                                             BufferedWriter out = new BufferedWriter(
                                                    new FileWriter(parser.dumpOutput));
267                                             String outText = parser.outputBuffer.
                                                    toString();
268                                             out.write(outText);
269                                             out.close();
270                                         }
271                                         catch (IOException e)
272                                             {
273                                                 e.printStackTrace();
274                                             }
275                                     }
276                                 }else{
277                                     System.err.println("\nOUTPUT COULD NOT BE
                                            PRODUCED DUE TO ERRORS\n");
278                                 }
279                                 System.err.println(parser.errorBuffer);
280
281                                 System.err.println("######################");
282                                 System.err.println("Syntactic Errors : "+parser.
                                        synWarnings);
283                                 System.err.println("Semantic Errors   : "+parser.
                                        semErrors);
284                                 System.err.println("Semantic Warnings: "+parser.
                                        semWarnings);
285
286                                 :}
287     ;
288
289
290 ////////////////////////////////////
291 // Declarations
292 ////////////////////////////////////
293
294 decl_list ::= decl_list decl | ;
295
296 decl ::= type var_list S
297     | type error S {: pSynWarning("Error in declaration"); :};
298
299 type ::= INT_TYPE {: if(sem()){RESULT = new String("INT");} :}
300     | DOUBLE_TYPE {: if(sem()){RESULT = new String("DOUBLE");}
301     :};
302
303 var_list ::= var
```

```
304     | var_list CM {: if(sem()){RESULT = parser.stack(-2);} :} var;
305
306 var ::= ID:x {: if(sem()){
307         dumpln("\t"+parser.stack(-1)+" "+x);
308         if(parser.stack(-1).equals("INT")){
309             parser.symbolType_table.put(x, new SymbolType(0,1));
310         }else if (parser.stack(-1).equals("DOUBLE")){
311             parser.symbolType_table.put(x, new SymbolType(1,1));
312         }
313     }
314                 :}
315     | ID:x SO INT:y SC {: if(sem()){
316         dumpln("\t"+parser.stack(-4)+" "+x+"["+y.toString()+"]");
317         if(parser.stack(-4).equals("INT")){
318             parser.symbolType_table.put(x, new SymbolType(0,y));
319         }else if (parser.stack(-4).equals("DOUBLE")){
320             parser.symbolType_table.put(x, new SymbolType(1,y));
321         }
322     }
323     :};
324
325
326
327
328 ////////////////////////////////////
329 // Instructions
330 ////////////////////////////////////
331
332 stmt_list ::= stmt_list stmt | stmt
333     | error stmt {: pSynWarning("Error in statement"); :};
334
335
336 stmt ::= if | while | assignment | print | BO stmt_list BC
337     | BO stmt_list error BC {: pSynWarning("Missing ; before }"); :}
338     | BO error BC {: pSynWarning("Missing ; before }"); :}
339     | error S {: pSynWarning("Error in statement"); :}
340 ;
341
342 // Assignment instruction
343 assignment ::= id:x S {: if (sem()){ dumpln("\t"+x);} :}
344     | id:x EQ exp:y S {: if (sem()){
345         x.checkSymbolTypeAssignment(y);
346         dumpln("\tEVAL "+y+"\n\tASS "+x);
347     }
348     :}
349     | id EQ error S {: pSynWarning("Error in expression"); :}
350     | error EQ exp S {: pSynWarning("Error in assigment"); :}
351 ;
352
353
354 // PRINT instruction
355 print ::= PRINT id:x S {: if (sem()){ dumpln("\tPRINT "+x);} :}
356     | PRINT error S {: pSynWarning("Error in 'print' instruction"); :}
357 ;
358
359
```

```
// IF instruction
if ::= IF if_condition nt0_if stmt ELSE nt1_if stmt {:
                                                    if (sem()){
                                                        dump("L"+parser.stack(-1)+
                                                            ":");
                                                    }
                                                    :}
     | IF if_condition:e nt0_if stmt {: if (sem()){ dump("L"+parser.stack(-1)+":")
        ;} :}
     | IF if_condition:e nt0_if stmt error nt1_if stmt {: pSynWarning("Error 'else'
        expected in 'if' instruction"); :}

;

if_condition ::= RO exp:e RC {: if(sem()){RESULT=e.value;} :}
    | RO error RC {: pSynWarning("Error in 'if' condition"); :}
    | error exp RC {: pSynWarning("Error '(' expected in 'if' instruciton"); :}
    | RO exp error  {: pSynWarning("Error ')' expected in 'if' instruciton"); :}
;

nt0_if ::= {:
            if (sem()){
                RESULT=parser.genLabel();
                dumpln("\tEVAL "+parser.stack(0)+"\t\t/* if (line "+parser.getLine
                    ()+") */\n\tGOTOF L"+RESULT);
            }
            :}
;

nt1_if ::= {:
            if (sem()){
                RESULT=parser.genLabel();
                dumpln("\tGOTO L"+RESULT);
                dump("L"+parser.stack(-2)+":");
            }
            :}
;


// WHILE instruction
while ::= WHILE while_condition nt0_while stmt {:
                                            if (sem()){
                                                Integer[] l=(Integer[])parser.stack
                                                    (-1);
                                                dumpln("\tGOTO L"+l[0]);
                                                dump("L"+l[1]+":");
                                            }
                                            :};
while_condition ::= RO exp:e RC {: if(sem()){RESULT=e.value;} :}
    | RO error RC {: pSynWarning("Error in 'while' condition"); :}
    | error exp RC {: pSynWarning("Error '(' expected in 'while' instruciton"); :}
    | RO exp error  {: pSynWarning("Error ')' expected in 'while' instruciton"); :}
;
```

```
411 nt0_while ::= {:
412                     if (sem()){
413                         RESULT=new Integer[2];
414                         RESULT[0]=(Integer)parser.genLabel();
415                         RESULT[1]=(Integer)parser.genLabel();
416                         dumpln("L"+RESULT[0]+":\tEVAL "+parser.stack(0)+"\t\t/*
                                while (line "+parser.getLine()+") */\n\tGOTOF L"+RESULT
                                [1]);
417                     }
418                 :}
419 ;
420
421
422 // Expressions
423 exp ::=
424     /* Espressioni logiche */
425     exp:x AND exp:y   {: if(sem()){RESULT = new Expr(x+" "+y+" & ", x.
            checkSymbolType(y));} :}
426     | exp:x OR exp:y  {: if(sem()){RESULT = new Expr(x+" "+y+" | ", x.
            checkSymbolType(y));} :}
427     | NOT exp:x       {: if(sem()){RESULT = new Expr(x+" ! ", x.getSymbolType());}
            :}
428
429     /* Espressioni di confronto */
430     | exp:x EQ EQ exp:y {: if(sem()){RESULT = new Expr(x+" "+y+" == ", x.
            checkSymbolType(y));} :}
431     | exp:x MIN exp:y   {: if(sem()){RESULT = new Expr(x+" "+y+" < ", x.
            checkSymbolType(y));} :}
432     | exp:x MAJ exp:y   {: if(sem()){RESULT = new Expr(x+" "+y+" > ", x.
            checkSymbolType(y));} :}
433     | exp:x mineq exp:y {: if(sem()){RESULT = new Expr(x+" "+y+" <= ", x.
            checkSymbolType(y));} :}
434     | exp:x majeq exp:y {: if(sem()){RESULT = new Expr(x+" "+y+" >= ", x.
            checkSymbolType(y));} :}
435
436     /* Espressioni aritmetiche */
437     | exp:x PLUS exp:y {: if(sem()){RESULT = new Expr(x+" "+y+" + ", x.
            checkSymbolType(y));} :}
438     | exp:x MINUS exp:y {: if(sem()){RESULT = new Expr(x+" "+y+" - ", x.
            checkSymbolType(y));} :}
439     | exp:x STAR exp:y {: if(sem()){RESULT = new Expr(x+" "+y+" * ", x.
            checkSymbolType(y));} :}
440     | exp:x DIV exp:y {: if(sem()){RESULT = new Expr(x+" "+y+" / ", x.
            checkSymbolType(y));} :}
441     | RO exp:x RC {: if(sem()){RESULT=x;} :}
442     | id:x {: if(sem()){RESULT=x;} :}
443     | INT:x {: if(sem()){RESULT = new Expr(x.toString(), new SymbolType(0,1));} :}
444     | DOUBLE:x {: if(sem()){RESULT = new Expr(x.toString(), new SymbolType(1,1));}
            :}
445     | MINUS INT:x {: if(sem()){RESULT = new Expr("-"+x.toString(), new SymbolType
            (0,1));} :} %prec UMINUS
446     | MINUS DOUBLE:x {: if(sem()){RESULT = new Expr("-"+x.toString(), new
            SymbolType(1,1));} :} %prec UMINUS
447     | RO error RC {: pSynWarning("Error in expression"); :}
448 ;
449
```

```
450
451 mineq ::= MIN_EQ | EQ_MIN;
452 majeq ::= MAJ_EQ | EQ_MAJ;
453
454
455 id ::= ID:x {: if(sem()){RESULT = new Expr(x);} :}
456       | ID:x SO INT:y SC {: if(sem()){RESULT = new Expr(x, y);} :}
457       | ID:x SO ID:y SC {: if(sem()){RESULT = new Expr(x, y);} :}
458       | error SC {: pSynWarning("Error in vector"); :}
459 ;
```

2 Exams

2.1 Exam1 (Practice 6)

scanner.jflex

```
import java_cup.runtime.*;

%%

%cup

number          =       [0-9]+
word            =       [a-zA-Z]+
comment         =       "//".*
ident           =       [_a-zA-Z][_a-zA-Z0-9]*


%%

"->"            {return new Symbol(sym.ARROW); }
"-"             {return new Symbol(sym.MINUS); }
"+"             {return new Symbol(sym.PLUS); }
"/"             {return new Symbol(sym.DIV); }
"*"             {return new Symbol(sym.STAR); }
"("             {return new Symbol(sym.OB); }
")"             {return new Symbol(sym.CB); }
";"             {return new Symbol(sym.SC); }
","             {return new Symbol(sym.C); }
"."             {return new Symbol(sym.D); }
":"             {return new Symbol(sym.DD); }
"="             {return new Symbol(sym.EQ); }

{comment}       {;}
{number}        {return new Symbol(sym.NUMBER, new Integer(yytext())); }
{word}          {return new Symbol(sym.WORD, new String(yytext())); }
{ident}         {return new Symbol(sym.ID, new String(yytext())); }

\n|\r|\r\n      {;}
[ \t]           {;}
```

parser.cup

```
import java_cup.runtime.*;
import java.util.*;

/* In this solution the possibility to give the score of 1 to the attributes not
    listed was not handled */

init with {:
    classHash = new HashMap();

    System.out.println("Achieved scores.\n");
:};


parser code {:
```

```
14      public static HashMap classHash;
15
16      /* Return semantic value of symbol in position (position) */
17      public Object stack(int position) {
18          return (((Symbol)stack.elementAt(tos+position)).value);
19      }
20
21  :};
22
23
24  action code {:
25      class Attrib {
26          private String name;
27          private Integer weight;
28
29          Attrib(String name, Integer weight){
30              this.name = name;
31              this.weight = weight;
32          }
33
34          String getName(){
35              return this.name;
36          }
37          Integer getWeight(){
38              return this.weight;
39          }
40
41      };
42  :};
43
44
45  terminal ARROW, MINUS, PLUS, DIV, STAR, OB, CB, SC, C, D, DD, EQ;
46  terminal String WORD, ID;
47  terminal Integer NUMBER;
48
49  non terminal prog, definitions, definition, descriptions, description, sentence;
50
51  non terminal HashMap attrib_list;
52  non terminal Attrib attrib;
53  non terminal Integer point, valutation, scores;
54  non terminal String ident, sentence_elem;
55  non terminal String NT0;
56
57  start with prog;
58
59  prog ::= definitions D descriptions;
60
61
62  ////////////////////////////////////////////////////
63  //DEFINITIONS
64  ////////////////////////////////////////////////////
65
66  definitions ::=         definitions definition
67                          |definition
68  ;
69
```

```
70
71 definition ::= OB attrib_list:attribHash CB ARROW ident:idName
72 {:
73         // The attribute HashMap is completed...
74         // it can be inserted inside the hashTable
75         parser.classHash.put(idName,attribHash);
76 :};
77
78
79
80 attrib_list ::= attrib_list:attribHash C attrib:attrib
81 {:
82         //Insertion of the new attribute inside the attribute HashMap
83         attribHash.put(attrib.name, attrib.weight);
84
85         RESULT = attribHash;
86 :};
87
88
89
90 attrib_list ::= attrib:attrib
91 {:
92         // A new HashMap is created to insert the attributes
93         HashMap hash = new HashMap();
94         // Current attribute is inserted inside the HashMap
95         hash.put(attrib.getName(), attrib.getWeight());
96
97         RESULT = hash;
98 :};
99
100
101 attrib ::=      ident:a DD NUMBER:b {:
102         //Pass an object of the attribute type
103         RESULT = new Attrib(a,b);
104 :};
105
106
107 ////////////////////////////////////////////////////
108 //DESCRIPTIONS
109 ////////////////////////////////////////////////////
110
111 descriptions ::=        descriptions description
112                         |
113 ;
114
115 description ::=         ident DD scores:score EQ sentence SC {:
116         System.out.println(",␣"+score);
117 :};
118
119
120 // The V terminal has the function of the sum operator
121 scores ::= scores:val1 NT0 C valutation:val2
122 {:
123         //I sum the valuations of the present product
124         RESULT = new Integer(val1.intValue() + val2.intValue());
125 :};
```

```
126
127 NT0 ::= {: RESULT = (String)parser.stack(-2); :};
128
129
130 scores ::= valutation:val {: RESULT = val; :}
131 ;
132
133
134 valutation ::= point:punt ident:name {:
135         String identClass = (String)parser.stack(-3);
136
137         //I search inside the hash table the entry related to the current class(
                identClasse)
138         HashMap hash = (HashMap)parser.classHash.get(identClass);
139
140         //I search inside the hash HashMap the weight associated to the current
                attribute
141         Integer weight = (Integer)hash.get(name);
142
143         //Compute the operation
144         RESULT = new Integer(weight.intValue() * punt.intValue());
145 :};
146
147
148 point ::=          STAR          {: RESULT = new Integer(3); :}
149                   |PLUS          {: RESULT = new Integer(2); :}
150                   |DIV           {: RESULT = new Integer(1); :}
151                   |MINUS         {: RESULT = new Integer(0); :}
152 ;
153
154
155 ////////////////////////////////////////////////////////////
156 /////// ELEMENTARY GRAMMAR
157 ////////////////////////////////////////////////////////////
158
159 sentence ::=     sentence sentence_elem:name {: System.out.print(" "+name);  :}
160                  | sentence_elem:name {: System.out.print(name);  :}
161 ;
162
163 sentence_elem ::= WORD:name {: RESULT = name; :}
164                   |NUMBER:num {: RESULT = num.toString();  :}
165 ;
166
167
168 ident ::= ID:a {: RESULT=a; :}
169         | WORD:a {: RESULT=a; :}
170 ;
```

2.2 Exam 2015-09-03 (Practice 7)

scanner.jflex

```
import java_cup.runtime.*;

%%

%unicode
%cup
%line
%column

%{
        private Symbol sym(int type) {
                return new Symbol(type, yyline, yycolumn);
        }

        private Symbol sym(int type, Object value) {
                return new Symbol(type, yyline, yycolumn, value);
        }

%}

/* TOKEN1 regular expression */
token1      = (("%%%%%"("%%")*) | (("**" | "???"){2,3})) {odd_number}
odd_number = "-"(3[135] | [12][13579] | [13579]) | [13579] | [1-9][13579] |
    [12][0-9][13579] | 3([0-2][13579]|3[13])

/* TOKEN2 regular expression */
token2  = {date}("-" | "+"){date}
date    = 2015"/"12"/"(1[2-9] | 2[0-9] | 3[01]) | 2016"/"(01"/"(0[1-46-9] |
    [12][0-9] | 3[01]) | 02"/"(0[1-9] | [12][0-9]) | 03"/"(0[1-9] | 1[0-3]))

/* TOKEN3 regular expression */
token3  = "$"(101 | 110 | 111 | 1(0|1){3} | 1(0|1){4} | 10(1000|0(0|1){3}))

q_string = \"␣~␣\"␣␣␣␣␣␣"//ignora questo commento e gli spazi a sx

uint    = 0 | [1-9][0-9]*

sep             = "##"("##")+

nl              = \r | \n | \r\n
cpp_comment     = "//" .*

%%

{token1}                        { return sym(sym.TOKEN1);}
{token2}                        { return sym(sym.TOKEN2);}
{token3}                        { return sym(sym.TOKEN3);}

{q_string}                      { return sym(sym.QSTRING, yytext());}
{uint}                          { return sym(sym.UINT, new Integer(yytext()));}

{sep}                           { return sym(sym.SEP);}

```

```
"PRINT_MIN_MAX"                 { return sym(sym.MINMAX);}
"PART"                          { return sym(sym.PART);}
"m"                             { return sym(sym.M);}
"m/s"                           { return sym(sym.MS);}
"->"                            { return sym(sym.ARROW);}
"="                             { return sym(sym.EQ);}
"|"                             { return sym(sym.PIPE);}
","                             { return sym(sym.CM);}
";"                             { return sym(sym.S);}
":"                             { return sym(sym.COL);}
"("                             { return sym(sym.RO);}
")"                             { return sym(sym.RC);}
"{"                             { return sym(sym.SO);}
"}"                             { return sym(sym.SC);}

{cpp_comment}                   {;}
\r | \n | \r\n | " " | \t       {;}

.                               { System.out.println("Scanner Error: " + yytext());
    }
```

parser.cup

```
import java_cup.runtime.*;
import java.util.*;
import java.io.*;

init with {:
    table = new HashMap<String, HashMap<String, Integer>>();
:};

parser code     {:

    public HashMap<String, HashMap<String, Integer>> table;

    public void report_error(String message, Object info) {
        StringBuffer m = new StringBuffer(message);
        if (info instanceof Symbol) {
            if (((Symbol)info).left != 1 && ((Symbol)info).right != 1) {
                if (((Symbol)info).left != -1 && ((Symbol)info).right != -1) {
                    int line = (((Symbol)info).left) + 1;
                    int column = (((Symbol)info).right) + 1;
                    m.append(" (line " + line + " column " + column + ")");
                }
            }
            System.err.println(m);
        }
    }

    public Object stack(int position) {
        return (((Symbol)stack.elementAt(tos + position)).value);
    }
:};


////////////////////////////////
```

```
35 ///// SYMBOLS DECLARATION
36 ////////////////////////////////
37
38 terminal TOKEN1, TOKEN2, TOKEN3, M, MS, MINMAX, PART, ARROW, SEP;
39 terminal EQ, PIPE, CM, S, COL, RO, RC, SO, SC;
40 terminal String QSTRING;
41 terminal Integer UINT;
42
43 non terminal prog, header, token1_l, cars, car, race, print_min_max_l, min_max;
44 non terminal Object[] section_names, performances;
45 non terminal HashMap speeds;
46 non terminal Float drive_stats, parts, part;
47 non terminal String NT0, NT1;
48
49
50
51 ////////////////////////////////
52 ///// GRAMMAR
53 ////////////////////////////////
54
55 start with prog;
56
57 prog ::= header SEP cars SEP race;
58
59
60 /******************/
61 /* Header section */
62 /******************/
63 header ::= token1_l TOKEN2 S token1_l TOKEN3 S token1_l
64          | token1_l TOKEN3 S token1_l TOKEN2 S token1_l
65 ;
66
67 token1_l ::= token1_l TOKEN1 S | /* epsilon */;
68
69
70 /****************/
71 /* Cars section */
72 /****************/
73 cars ::= car car | cars car car;
74
75 car ::= QSTRING:s SO speeds:tab SC {:
76               parser.table.put(s, tab);
77         :}
78 ;
79
80 speeds ::= QSTRING:s EQ UINT:u MS {:
81                 RESULT = new HashMap<String, Integer>();
82                 RESULT.put(s, u);
83             :}
84             | speeds:tab CM QSTRING:s EQ UINT:u MS {:
85                 tab.put(s, u);
86                 RESULT = tab;
87             :}
88 ;
89
90 /***************/
```

```
/* Race section */
/***************/
race ::= print_min_max_l performances:s {:
                System.out.println("WINNER: " + s[0] + " " + s[1] + " s");
          :}
;

/* Management of PRINT_MIN_MAX function */
print_min_max_l ::= | print_min_max_l min_max;

min_max ::= MINMAX RO QSTRING:s RC RO section_names:m RC S {:
                    System.out.println("MIN: " + m[0] + " MAX: " + m[1]);
              :}
;

section_names ::= QSTRING:s {:
                        String car = (String)parser.stack(-3);
                        HashMap<String, Integer> speeds = parser.table.get(car);
                        Integer speed = (Integer)speeds.get(s);
                        RESULT = new Object[2];
                        RESULT[0] = speed; // Current min value
                        RESULT[1] = speed; // Current max value
                    :}
                    | section_names:m CM QSTRING:s {:
                        String car = (String)parser.stack(-5);
                        HashMap<String, Integer> speeds = parser.table.get(car);
                        Integer speed = (Integer)speeds.get(s);
                        RESULT = new Object[2];
                        // Update current min and max values
                        if (speed < (Integer)m[0]) {
                            // New min
                            RESULT[0] = speed;
                            RESULT[1] = m[1];
                        } else if (speed > (Integer)m[1]) {
                            // New max
                            RESULT[0] = m[0];
                            RESULT[1] = speed;
                        } else {
                            // No change in min and max
                            RESULT[0] = m[0];
                            RESULT[1] = m[1];
                        }
                    :}
;

/* Part regarding performances */
performances ::= QSTRING:s {: System.out.println(s); :} ARROW parts:x S {:
                        System.out.println("TOTAL: " + x + " s");
                        // To detect the winner car
                        RESULT = new Object[2];
                        RESULT[0] = s; // car name
                        RESULT[1] = x; // result
                    :}
                    | performances:perf QSTRING:s {: System.out.println(s); :} ARROW
                       parts:x S {:
                        System.out.println("TOTAL: " + x + " s");
```

```
                        RESULT = new Object[2];
                        // Check if this car is the current winner
                        if ((Float)perf[1] < x) {
                            // Current winner is an old car
                            RESULT[0] = perf[0];
                            RESULT[1] = perf[1];
                        } else {
                            // Current winner is this car
                            RESULT[0] = s;
                            RESULT[1] = x;
                        }
                    :}
;

// The two markers are used to transfer the car name semantic value in the symbol
    that preceeds the non terminal "part"
parts ::= NT0 part:x {: RESULT = x; :}
          | parts:res PIPE NT1 part:x {: RESULT = res + x; :}
;

NT0 ::= {: RESULT = (String)parser.stack(-2); :};

NT1 ::= {: RESULT = (String)parser.stack(-4); :};

part ::= PART UINT:x COL drive_stats:stat {:
               RESULT = stat;
               System.out.println("PART" + x + ": " + stat + " s");
:};

drive_stats ::= QSTRING:s UINT:u M {:
                        String car = (String)parser.stack(-6);
                        HashMap<String, Integer> speeds = parser.table.get(car);
                        Integer speed = (Integer)speeds.get(s);
                        float result = (float)u.intValue() / (float)speed.intValue();
                        RESULT = new Float(result);
                    :}
                    | drive_stats:stat CM QSTRING:s UINT:u M {:
                        String car = (String)parser.stack(-8);
                        HashMap<String, Integer> speeds = parser.table.get(car);
                        Integer speed = (Integer)speeds.get(s);
                        float result = (float)u.intValue() / (float)speed.intValue();
                        RESULT = new Float(result);
                        RESULT += stat; /* Accumulate the time in result */
                    :}
;
```

2.3 Exam 2020-07-20

scanner.jflex

```
import java_cup.runtime.*;

%%

%cup

evenhex =       ( "-"(5[02468AaCcEe] | [1-4][02468AaCcEe] | [02468AaCcEe]) |
    ([02468ACEace] | [1-9a-fA-F][02468ACEace] | [0-9aA][0-9abAB][0246]) )
token1 =        [abc]{7}([abc][abc])*"#"{evenhex}?
myhour =        ( "07:13:"(2[4-9]|[3-5][0-9]) | "07:1"[4-9]":"[0-5][0-9] | "07:"
    [2-5][0-9]":"[0-5][0-9] | (0[89]|1[0-6])(":"[0-5][0-9]){2} | "17:"[0-2][0-9]":"
    [0-5][0-9] | "17:3"[0-6]":"[0-5][0-9] | "17:37:"([0-3][0-9]|4[0-3]) )
//myhour =      (0[7-9]|1[0-7])":"(1[3-9]|2[0-9]|3[0-7])":"(2[4-9]|3[0-9]|4[0-3])
bynumb =        (101 | 110 | 111 | 1[01]{3} | 10[01]{3} | 110(00|01|10) )
token2 =        {myhour}":"{bynumb}
id =            [a-zA-Z_][a-zA-Z0-9_]*
number =        [1-9][0-9]*
comment =       "(++".*

%%

"compare"       {/*System.out.print("COMP ");*/ return new Symbol(sym.COMP);}
"with"          {/*System.out.print("WITH ");*/ return new Symbol(sym.WITH);}
"end"           {/*System.out.print("END ");*/ return new Symbol(sym.END);}
"+"                     {/*System.out.print("PLUS ");*/ return new Symbol(sym.PLUS)
    ;}
"-"                     {/*System.out.print("MINUS ");*/ return new Symbol(sym.
    MINUS);}
"*"                     {/*System.out.print("STAR ");*/ return new Symbol(sym.STAR)
    ;}
"/"                     {/*System.out.print("DIV ");*/ return new Symbol(sym.DIV);}
"("                     {/*System.out.print("RO ");*/ return new Symbol(sym.RO);}
")"                     {/*System.out.print("RC ");*/ return new Symbol(sym.RC);}
"{"                     {/*System.out.print("BO ");*/ return new Symbol(sym.BO);}
"}"                     {/*System.out.print("BC ");*/ return new Symbol(sym.BC);}
"print"         {/*System.out.print("PRINT ");*/ return new Symbol(sym.PRINT);}
"="                     {/*System.out.print("EQU ");*/ return new Symbol(sym.EQU);
    }
"$$"            {/*System.out.print("SEP ");*/ return new Symbol(sym.SEP); }
";"                     {/*System.out.print("S ");*/ return new Symbol(sym.S);}

{id}            {/*System.out.print("ID ");*/ return new Symbol(sym.ID, new String(
    yytext())); }
{token1}        {/*System.out.print("TOKEN1 ");*/ return new Symbol(sym.TOKEN1);}
{token2}        {/*System.out.print("TOKEN2 ");*/ return new Symbol(sym.TOKEN2);}
{number}        {/*System.out.print("NUMBER ");*/ return new Symbol(sym.NUM, new
    Integer(yytext()));}

{comment}       {/*System.out.print("COMMENT ");*/}

\n|\r|\r\n      {;}
[ \t]           {;}
```

parser.cup

```
import java_cup.runtime.*;
import java.util.*;

init with {:
    varTable = new HashMap<String,Integer>();
:};

parser code {:

        public static HashMap<String,Integer> varTable;

    /* Return semantic value of symbol in position (position) */
    public Object stack(int position) {
        return (((Symbol)stack.elementAt(tos+position)).value);
    }

:};


terminal TOKEN1, TOKEN2, COMP, WITH, END, PLUS, MINUS, STAR, DIV, RO, RC, BO, BC,
    PRINT, EQU, SEP, S, UMINUS;
terminal Integer NUM;
terminal String ID;

non terminal progr, header, headerone, headertwo, oddTok1List, threetwentyoneTok2,
    tripleTok2, emptyTok1List, tok1list;
non terminal commands, command_list, command, assignment, comparation, comp_list,
    comp, print_list, print_command, NT0_comp, NT1_comp;
non terminal Integer expr;

// Precedences and associativities
// lower precedences
precedence left PLUS, MINUS;
precedence left STAR, DIV;
precedence left UMINUS;
// higher precedences

start with progr;

progr ::= header SEP commands;

////HEADER SECTION////

header ::= headerone | headertwo;

headerone ::= oddTok1List threetwentyoneTok2;

oddTok1List ::= TOKEN1 S TOKEN1 S TOKEN1 S TOKEN1 S TOKEN1 S
                                    | oddTok1List TOKEN1 S TOKEN1 S;

tripleTok2 ::= TOKEN2 S TOKEN2 S TOKEN2 S;

threetwentyoneTok2 ::= tripleTok2
                                            | tripleTok2 tripleTok2 tripleTok2
                                              tripleTok2 tripleTok2 tripleTok2
```

```
                                                tripleTok2;

headertwo ::= TOKEN2 S emptyTok1List TOKEN2 S emptyTok1List TOKEN2 S emptyTok1List;

emptyTok1List ::= tok1list | ;

tok1list ::= TOKEN1 S | tok1list TOKEN1 S;

////COMMANDS SECTION////

commands ::= command_list;

command_list ::=  | command_list command;

command ::= assignment | comparation;

assignment ::= ID:name EQU expr:x S {: parser.varTable.put(name,x); /*System.out.
    println("\n"+name+": "+x+";\n");*/ :};

// expressions
expr ::= expr:x PLUS expr:y {: RESULT = x + y; :}
     | expr:x MINUS expr:y        {: RESULT = x - y; :}
     | expr:x STAR expr:y         {: RESULT = x * y; :}
     | expr:x DIV expr:y          {: RESULT = x / y; :}
     | RO expr:x RC                        {: RESULT = x; :}
     | ID:name                             {: RESULT = (Integer)parser.varTable.get(
         name); :}
     | NUM:x                               {: RESULT = x;  :}
     | MINUS NUM:x                         {: RESULT = -x; :} %prec UMINUS
     | RO error RC                         {: System.out.println("Damn man... 
         Expression error"); :}
;

comparation ::= COMP expr WITH comp_list END S;

comp_list ::= NT0_comp comp | comp_list NT1_comp comp;

NT0_comp ::= {: RESULT = (Integer)parser.stack(-1); :};

NT1_comp ::= {: RESULT = (Integer)parser.stack(-2); :};

comp ::= expr BO print_list BC;

print_list ::= PRINT expr:x S {:
                                Integer e1 = (Integer)parser.stack(-5);
                        Integer e2 = (Integer)parser.stack(-4);
                                if(e1 == e2){
                                        System.out.println("print: "+x);
                                }
                        :}
                        | print_list PRINT expr:x S {:
                                Integer e1 = (Integer)parser.stack(-6);
                                Integer e2 = (Integer)parser.stack(-5);
                                if(e1 == e2){
                                        System.out.println("print: "+x);
                                } :};
```

2.4 Exam 2012-06-26

scanner.jflex

```
import java_cup.runtime.*;

%%

%cup

sep      =                  "*****"
htoke =          ("08:31"(":1"[2-9]|":"[2-5][0-9])? | "08:"[3-5][2-9](":"[0-5][0-9])
    ? | (09|1[0-9]|2[0-2])":"[0-5][0-9](":"[0-5][0-9])? | "23:"( ([0-1][0-9]|20)(":"
    [0-5][0-9])? | 21(":"(0[0-9]|10))?) )
evenum  =        "-"(1[0-3][02468]|[1-9][02468]|[2468]) |
    ([02468]|[1-9][02468]|[1-7][0-9][02468]|8[0-1][02468]|82[024])
ctoke =          ([XY]{3}[XY]*)?{evenum}
usercode =       "Usr"((" ."(1[2-9]|[2-9][0-9]|1[0-2][0-9]|13[0-2]))){2})+
quoted  =        \".*\"_______"//ignora questo commento e gli spazi a sx
doublenum =      [1-9][0-9]*\.[0-9]{2} | 0\.[0-9]{2}
word =           [a-zA-Z]+
integer =        [1-9][0-9]*|{evenum}

%%

//"Auction"      {System.out.println("AUCTION ");*/ return new Symbol(sym.AUCTION);}
"->"             {/*System.out.print("ARROW ");*/ return new Symbol(sym.ARROW);}
":"                      {/*System.out.print("DD ");*/ return new Symbol(sym.DD);}
//"min"          {return new Symbol(sym.MIN);}
","                      {/*System.out.print("CM ");*/ return new Symbol(sym.CM);}
//"euro"                 {return new Symbol(sym.EUR);}
";"                      {/*System.out.print("S ");*/ return new Symbol(sym.S);}

{quoted}         {/*System.out.print("QS ");*/ return new Symbol(sym.QS);}
{sep}            {/*System.out.print("SEP ");*/ return new Symbol(sym.SEP, new
    String(yytext())); }
{htoke}          {/*System.out.print("HTOK ");*/ return new Symbol(sym.HTOK);}
{ctoke}          {/*System.out.print("CTOK ");*/ return new Symbol(sym.CTOK, new
    String(yytext()));}
{usercode}       {/*System.out.print("USRCD ");*/ return new Symbol(sym.USRCD, new
    String(yytext()));}
{doublenum} {/*System.out.print("DOUBLE ");*/ return new Symbol(sym.DOUBLE, new
    Double(yytext()));}
{integer}        {/*System.out.print("INT ");*/ return new Symbol(sym.INT, new
    Integer(yytext()));}
{word}           {/*System.out.print("WORD ");*/ return new Symbol(sym.WORD, new
    String(yytext()));}


\n|\r|\r\n       {;}
[ \t]            {;}
```

parser.cup

```
import java_cup.runtime.*;
import java.util.*;

init with{:
```

```
5           capitalTable = new HashMap<String,Double>();
6   :};
7
8   parser code {:
9
10          public static HashMap<String,Double> capitalTable;
11
12      /* Return semantic value of symbol in position (position) */
13      public Object stack(int position) {
14          return (((Symbol)stack.elementAt(tos+position)).value);
15      }
16
17  :};
18
19
20  terminal ARROW, DD, CM, S, QS, SEP, HTOK;
21  terminal Integer INT;
22  terminal Double DOUBLE;
23  terminal String USRCD, WORD, CTOK;
24
25  non terminal progr, header, htoklist, currencies, currency, empty_user_list,
        user_list, user;
26  non terminal empty_auction_list, auction_list, auction;
27  non terminal Object[] advances;
28  non terminal Integer intorctok;
29
30  start with progr;
31
32  progr ::= header SEP currencies SEP empty_auction_list;
33
34  ////HEADER SECTION////
35
36  header ::= HTOK S htoklist CTOK S htoklist CTOK S htoklist
37                  |  htoklist CTOK S HTOK S htoklist CTOK S htoklist
38                  |  htoklist CTOK S htoklist CTOK S HTOK S htoklist;
39
40  htoklist ::= | htoklist HTOK S;
41
42  ////CURRENCIES SECTION////
43  currencies ::= currency currency currency | currencies currency;
44
45  currency ::= DOUBLE WORD WORD DD empty_user_list S;
46
47  empty_user_list ::= | user_list;
48
49  user_list ::= USRCD:usr DOUBLE:capital {:
50                                  Double convrate = (Double)parser.stack(-5);
51                                  parser.capitalTable.put(usr,capital*convrate);
52                                  :}
53                          | user_list CM USRCD:usr DOUBLE:capital {:
54                                  Double convrate = (Double)parser.stack(-7);
55                                  parser.capitalTable.put(usr,capital*convrate);
56                                  :};
57
58  //user ::= USRCD DOUBLE;
59
```

```
60 ////AUCTION SECTION////
61
62 intorctok ::= CTOK:x {: RESULT=Integer.parseInt(x); :} | INT:x {: RESULT=x; :};
63
64 empty_auction_list ::= | auction_list;
65
66 auction_list ::= auction auction | auction_list auction auction;
67
68 auction ::= WORD intorctok:auctnum {: System.out.println("Auction "+auctnum+":");
       :}
69                          DD QS DD intorctok WORD ARROW advances:maxadvance S {:
70                                  String usr = (String)maxadvance[0];
71                                  Double maxoffer = (Double)maxadvance[1];
72                                  Double prevCapital = (Double)parser.capitalTable.
                                        get(usr);
73                                  parser.capitalTable.put(usr,prevCapital-maxoffer);
74                                  System.out.println("Winner is: "+usr+" price "+
                                        maxoffer + " euro");
75                             :};
76
77 advances ::= USRCD:usr DD intorctok:time DD DOUBLE:offer WORD
78     {:
79     Integer duration = (Integer)parser.stack(-8);
80         Double moneyLeft = (Double)parser.capitalTable.get(usr);
81         RESULT = new Object[2];
82         if(offer < 0.0){
83         System.out.println("  "+usr+": Error, advance less than the current auction
               value");
84                 RESULT[0]=null;
85                 RESULT[1]=0.00;
86         }else if(( (Integer)time) > duration){
87                 System.out.println("  "+usr+": Error, advance out of time");
88                 RESULT[0]=null;
89                 RESULT[1]=0.00;
90         }else if(offer > moneyLeft){
91             System.out.println("  "+usr+": Error, available only "+ moneyLeft +"
                   euro");
92                 RESULT[0]=null;
93                 RESULT[1]=0.00;
94         }else{
95                 System.out.println("  "+usr+": New auction price "+ offer +" euro")
                       ;
96                 RESULT[0]=usr;
97                 RESULT[1]=offer;
98         }
99                                             :}
100        | advances:maxadvance CM USRCD:usr DD intorctok:time DD DOUBLE:offer WORD
101        {:
102        Integer duration = (Integer)parser.stack(-10);
103        Double moneyLeft = (Double)parser.capitalTable.get(usr);
104        RESULT = new Object[2];
105        if(offer < (Double)maxadvance[1]){
106                System.out.println("  "+usr+": Error, advance less than the current
                       auction value");
107                RESULT[0]=maxadvance[0];
108                RESULT[1]=maxadvance[1];
```

```
109         }else if(( (Integer)time) > duration){
110                 System.out.println("  "+usr+": Error, advance out of time");
111                 RESULT[0]=maxadvance[0];
112                 RESULT[1]=maxadvance[1];
113         }else if(offer > moneyLeft){
114                 System.out.println("  "+usr+": Error, available only "+ moneyLeft +
                        " euro");
115                 RESULT[0]=maxadvance[0];
116                 RESULT[1]=maxadvance[1];
117         }else{
118                 System.out.println("  "+usr+": New auction price "+ offer +" euro")
                        ;
119                 RESULT[0]=usr;
120                 RESULT[1]=offer;
121         }
122         :};
```

2.5 Exam1 (Practice 6)

scanner.jflex

parser.cup