Formal Languages and Compilers

24 July 2018

Using the JFLEX lexer generator and the CUP parser generator, realize a JAVA program capable of recognizing and executing the programming language described in the following.

Input language

The input file is composed of two sections: *header* and *commands* sections, separated by means of the two characters "##". Comments are possible, and they are delimited by the starting sequence "/-" and by the ending sequence "-/".

Header section: lexicon

The *header* section can contain 3 types of tokens, each terminated with the character ";":

- <token1>: It is an hexadecimal number between -27 and 256C, followed by the character "_", followed by a word with at least 5 alphabetic characters in an odd number, followed by the character "_" and optionally ended with the word SOS or the word XYZZX (where the number of Y must be odd and the number of Z must be even: i.e., XYZZX, XYYYYYZZX, XYYYZZZZX,...).

- <token2>: It is a hour in the format HH:MM:SS or HH:MM or HH:MM am / pm (without spaces) between 09:21:12 and 17:43:34. Examples of correct hours are 09:21:12, 09:21, 09:21am, 10:20:02, 10:20, 10:20am, 11:59am, 12:00pm, 14:30:12, 14:30, 02:30pm. Examples of not correct hours are 08:10, 10:00pm, 09:21:11, 9:21. Remember that 12:00pm is noon, 12:01pm corresponds to 12:01, and 01:00pm corresponds to 13:00.

- <token3>: It is the sequence of characters "$$" followed by a binary number containing 3 or 5 characters "1" (e.g., $$010001010, $$0110100101), or it is the sequence of characters "&&" followed by a word composed of the symbols "X" and "O" without consecutive equal symbols (i.e., near to the symbol "X" we can have only "O", near the symbol "O" we can have only "X"). Example: &&X, &&, &&OXOX, &&XOXOXOXO.

Header section: grammar

In the *header section* tokens can appear in **any order**. In addition, <token1> must appear **exactly 2 times**, <token2> must appear **exactly 1 time**, while <token3> can appear **0 or more times**.

Commands section: grammar and semantic

The *commands section* is composed of a <distance_list> (**at least 3** <distance> in **odd** number), followed by an <compute_list> (the list can be **empty**, or if not empty, the number of <compute> is **odd**):

- <distance>: Each <distance> command is composed of a <departure_city> (a *quoted string*), the symbol "->", a <destination_city_list> and a ";". Elements of the list are separated with commas. Each element is composed of a <destination_city> (a *quoted string*), a <distance> (a *real number*) that represents the distance between the departure and the destination cities in kilometers, and the word "km". All the semantic information defined in this section and useful in the following section, can be stored into a global variable. **It is the only global variable allowed in all the examination, and it must only contain the information derived from the <distance> command. Solutions using other global variables will not be accepted.** In addition, the command must produce the output reported in the example.

- <compute>: Each <compute> command is composed of the word "COMPUTE", a <departure_city>, the word "TO", a <destination_city>, a <type>, a ":", an <element_list> and a ";". <type> can be the string "TIME", or the string "EXPENSE", which can be **optionally** followed by the word "EXTRA" and a <discount> (a *real number with two decimals*).

  When <type> is "TIME", <element_list> is a <time_list>, which is a not empty list of <time> separated by commas. Each <time> is composed by a <percent> (an *unsigned integer* number), a "%", a <speed> (an *unsigned integer* number) and the word "km/h". For each element of the list (i.e., <time>), the COMPUTE command has to access with inherited attributes to the symbols <departure_city> and <destination_city>, in order to read and compute from the global variable content the distance between

the two cities. The command has to multiply this distance by <percent>, divide by 100 and divide by the <speed>, obtaining a time expressed in hours. This time must be printed, together with a progressive number (starting from 0 at any new execution of the same command), as reported in the output of the example. At the end, the command has to print the sum of all the times.

When <type> is equal to "EXPENSE", <element_list> is an <expense_list>, which is a not empty list of <expense> separated by commas. Each <expense> is a <code> (a *quoted string*) followed by an <exp>, a "-" and a <disc>. An <exp> is a <price$_a$> (*real number with two decimals*) and the word "euro/km", while a <disc> is the word "DISC", a <price$_b$> (same regular expression of <price$_a$>) and the word "euro". <exp> and <disc> can appear **in any order**, and <dist> is **optional** (see the example). For each element of the list (i.e., <expense>), the COMPUTE command computes the expense multiplying the distance between the two cities by <price$_a$>, subtracting <price$_b$> (if present), and subtraction <discount> (if present). At the end, the command has to report the minimum value between all expenses. For the output see the example.

Goals

The translator must execute the language, and it must produce the output reported in the example.

Example

Input:

```
/- Header Section -/
-1B_aBcDefG_;        /- token1 -/
$$01101;             /- token3 -/
12:59pm;             /- token2 -/
3C_abcde_XYYYYYZZX ; /- token1 -/
&&XOXOX ;            /- token3 -/

##

/- Commands Section -/

"Milano" -> "Torino" 150.0 km,
            "Genova" 150.2 km,
            "Venezia" 269.0 ;

"Roma" -> "Napoli" 226.0 km,
          "Bari" 431.0 km,
          "Torino" 670.0 km ;

"Torino" -> "Bologna" 332.0 km ;

COMPUTE "Roma" TO "Torino" TIME: 25 % 100 km/h,  /- 670.0*25/100/100=1.675 -/
                                 25 % 50 km/h,   /- 670.0*25/100/50=3.35 -/
                                 50 % 125 km/h ; /- 670.0*50/100/125=2.68 -/

COMPUTE "Milano" TO "Torino" EXPENSE EXTRA 10.00 :
   "codeA" 0.28 euro/km - DISC 9.50 euro,  /- 150.0*.28-9.50-10.00=22.50 -/
   "codeB" DISC 12.00 euro - 0.29 euro/km, /- 150.0*.29-12.00-10.00=21.50 -/
   "codeC" 0.26 euro/km;                  /- 150.0*.26-10.00=29.00 -/

COMPUTE "Milano" TO "Torino" EXPENSE :
    "codeD" 0.26 euro/km ; /- 150.0*.26=39.00 -/
```

Output:

```
0 1.675
1 3.35
2 2.68
TOT: 7.705
"codeA" 22.50
"codeB" 21.50
"codeC" 29.00
MIN: 21.50
"codeD" 39.00
MIN: 39.00
```

Weights. Scanner: 8/30; **Grammar:** 9/30; **Semantics:** 10/30