Formal Languages and Compilers

16 September 2022

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 three sections: *header*, *distance*, and *route* sections, separated by means of the sequence of 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 ";":

- `<tok1>`: It is composed of the characters "X_" followed by 3, 12, or 15 repetitions of a binary number (between 101 and 1011011). Each binary number is followed by a "*" or a "+". Example: `X_101+1000+1001*`.
- `<tok2>`: It is composed of the characters "Y_" followed by 2 or 5 words separated by the character "#" or "$". Each word is composed of at least 6 characters in the set "x", "y" or "z", which are disposed in any order and, in total, in even number. Example: `Y_xyzxyz#xxyyzzzy`.
- `<tok3>`: It is composed of the characters "Z_" followed by a date with the format YYYY/MM/DD in the range between 2022/09/10 and 2023/03/15. Remember that the months of September and November have 30 days, while the month of February has 28 days. This first part of the token is optionally followed by a hour with the format :HH:MM between :09:11 end :17:13. Example: `Z_2023/02/28:09:30`.

Header section: grammar

In the header section `<tok1>` and `<tok2>` can appear **in any order and number** (**also 0 times**), instead, `<tok3>` can appear only **0, 1 or 4 times**.

Location section: grammar and semantic

The *distance* section contains a non-empty list of `<distance>` commands. Each `<distance>` command is a `<city>`$_a$ (i.e., a *quoted string*), a `<list_of_pos>`, and a ";". The `<list_of_pos>` is a non-empty list of `<pos>` separated with "," (i.e., *comma).* Each `<pos>` is a `<city>`$_b$, a `<dist>` (i.e., a *real number* that indicates the distance from `<city>`$_a$ to `<city>`$_b$), and the word "`km`". **All the needed information of this section must be stored in a global data structure. This data structure is the only global variable allowed in all the examination, and it can be written only in this section.**

Route section: grammar and semantic

The *route* section is composed of **at least 4** `<command>` in **even** number (i.e., 4, 6, 8,...). Each `<command>` can be an `<elevation_cmd>` or a `<route>` command.

An `<elevation_cmd>` command is the word "`ELEVATION`, a `<list_of_places>` separated with ",", and a ";". Each component of the list (i.e., a `<places>`) is a `<city>`, an `<elevation>` (i.e., an *integer number*), and the word "`m`". This command computes the elevation performing the sum of the differences between the `<elevation>` of the next `<city>` and that of the current `<city>` in the list (see the example for more details about this computation).

A `<route>` command is the word “`ROUTE`, followed by a `<cal>` (which represents the consumed kilocalories to perform a kilometer with a bike), **optionally** the word “`kcal/km`”, a “:”, a non-empty `<list_of_pair_of_cities>` separated with “,”, and a “;”. A `<pair_of_cities>` is a `<city>`$_a$, a `<city>`$_b$, and a `<modif>` (i.e., a *real number*). The translator must retrieve from the global structure the distance from `<city>`$_a$ to `<city>`$_b$, perform the multiplication between this distance, the `<modif>` and the `<cal>`, obtaining a `<partial_cal>` value. **In this context, the value `<cal>` must be accessed through inherited attributes.** The compiler must print `<city>`$_a$, `<city>`$_b$, `<partial_cal>`, and the word “`kcal`”.

In addition, at the end of each `<route>` command, the translator must print the sum of all the `<partial_cal>` computed in the command (see the example for the output).

Goals

The translator must execute the language, and it must produce the output reported in the example. For any detail not specified in the text, follow the example.

Example

Input:

```
X_110*1011000+111*;                                 (+- tok1 -+)
Y_xxxyyy$xyzxyzxyzx#xxxyyy$xxxyyy$xxxyyy ;          (+- tok2 -+)
Z_2023/01/01 ;                                      (+- tok3 -+)
Y_xyzxyz#xxyyzzzy;                                  (+- tok2 -+)

====  (+- division between header and distance sections -+)
"Biella" TO "Miagliano" 6.5 km,
         TO "Candelo" 5.9 km;
"Miagliano" TO "Piedicavallo" 13.0 km,
            TO "Bielmonte" 24.0 km,
            TO "Sordevolo" 13.2 km,
            TO "Oropa" 15.2 km;
"Candelo" TO "Miagliano" 11.7 km,
          TO "Viverone" 23.2 km,
          TO "Parco Burcina" 12.7 km,
          TO "Graglia" 16.6 km;

====  (+- division between distance and route sections -+)
(+- (537-400)+(1043-537)=137+506=643 -+)
ELEVATION "Biella" 400 m, "Miagliano" 537 m, "Piedicavallo" 1043 m;
(+- 6.5*15*1.1+13.0*15*1.2=107.25+234.0=341.25 kcal -+)
ROUTE 15 : "Biella" "Miagliano" 1.1, "Miagliano" "Piedicavallo" 1.2;

(+- (340-400)+(537-340)+(1186-537)=-60+197+649=786 -+)
ELEVATION "Biella" 400 m, "Candelo" 340 m, "Miagliano" 537 m, "Oropa" 1186 m;
(+- 5.9*20*0.9+11.7+20*1.1+15.2*20*1.4=106.2+33.7+425.6=565.50  -+)
ROUTE 20 kcal/km : "Biella" "Candelo" 0.9, "Candelo" "Miagliano" 1.1, "Miagliano" "Oropa" 1.4;
```

Output:

```
ELEVATION 634 m
"Biella" "Miagliano" 107.25 kcal
"Miagliano" "Piedicavallo" 234.0 kcal
Tot: 341.25 kcal
ELEVATION 786 m
"Biella" "Candelo" 106.2 kcal
"Candelo" "Miagliano" 33.7 kcal
"Miagliano" "Oropa" 425.6 kcal
Tot: 565.50 kcal
```

Weights: Scanner 9/30; **Grammar** 9/30; **Semantic** 9/30