Formal Languages and Compilers

9 February 2023

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*, *food* and *recipes* sections, separated by means of the sequence of characters "`$$$$`" (at least 4 and in even number). Comments are possible, and they are delimited by the starting sequence "`(*-`" and by the ending sequence "`-*)`".

Header section: lexicon

The *header* section can contain 2 types of tokens, each terminated with the character ";":

- `<tok1>`: it starts with a word composed of at least 6 and at most 17 repetitions of the following words of two characters: "%*", "*%", or "%%" (as a consequence, the number of characters of the first part of this token ranges between 12 and 34). The word is followed by 3 or 6 hexadecimal numbers of 2, 3 or 6 characters. The hexadecimal numbers are separated by the character "+".
- `<tok2>`: it is composed of three dates separated by the character "$" or by the character "%". A date has the format YYYY/MM/DD and it is in the range between 2022/11/15 and 2023/03/30, with the exclusion of the days 2022/12/13 and 2023/02/14. Remember that the month of November has 30 days, and the month of February has 28 days. The token is optionally terminated by the character "-" and a binary number between 1011 and 10111.

Header section: grammar

In the header section, the token `<tok1>` must appear **exactly one time**, instead, `<tok2>` must appear in the header section **two or more times** and it can appear in any position (manage this requirement with grammar).

Food section: grammar and semantic

The *food* section is composed of a list of `<food_category>` in **even** number, including **zero** (i.e., 0, 2, 4, 6,...). Each `<food_category>` is composed of a `<food_category_name>`, a ":", a "`[`", a `<food_list>`, a "`]`", and a ";".

A `<food_list>` is a non-empty list of `<food>` separated with ",". A `<food>` is a `<food_name>`, a ":", a `<kg_price>` (i.e., a *real* and *positive* number with *two decimals*), and the word "`EURO/kg`". Tokes `<food_category_name>` and `<food_name>` are *quoted strings*.

At the end of this section, all the information needed for the following *recipes* section must be stored into an entry of a global symbol table with key `<food_category_name>`. **This symbol table is the only global data structure allowed in all the examination, and in the food section can only be written, while in the recipes section can only be read**.

In addition, at the end of this section, the translator must print the less expensive and the more expensive `<food_name>`, i.e., the `<food_name>` with the lower and the higher values in terms of `<kg_price>`, respectively. For the output format see the example.

Recipes section: grammar and semantic

The *recipes* section is composed of a list, which can be **empty**, of <recipe>. Each <recipe> is a <number> (i.e., an *positive integer* number), a <recipe_name> (i.e., a *quoted strings*), a ":" (which is **OPTIONAL**), an <ingredient_list>, and a ";". An <ingredient_list> is a non-empty list of <ingredient> separated with ",". Each <ingredient> is a <food_category_name>, a ".", a <food_name>, a <weight>, and the word "kg".

In this section, for each <recipe>, the translator must print the <recipe_name>, and for each <ingredient>, the translator must print the <food_name> associated to the <ingredient>, the <price>, and the word "EURO". The <price> is obtained by multiplying the <kg_price> (which can be accessed through the couple <food_category_name>.<food_name> from the symbol table), with the <weight>, and with the <number> (which can be accessed through **inherited attributes**).

In addition, for each <recipe> the translator must print the word "TOT:", the sum of all the <price> of the <recipe>, and the word "EURO".

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:

```
(*- Header section -*)
2022/11/15$2023/03/30%2022/02/28-1111; (*- tok2 -*)
%%*%*%*%%%%*%%2a+3B+aBC ;              (*- tok1 -*)
2022/12/04$2023/01/02$2023/01/01 ;     (*- tok2 -*)

$$$$ (*- division between header and food sections -*)
(*- Available food -*)
"low price" : [
    "flour": 1.00 EURO/kg, "tomato sauce" : 8.00 EURO/kg,
    "mozzarella cheese" : 15.00 EURO/kg
];

"medium price" : [
    "flour": 1.50 EURO/kg, "tomato sauce" : 10.00 EURO/kg,
    "mozzarella cheese" : 21.00 EURO/kg, "tomatoes" : 13.00 EURO/kg
];

$$$$$$ (*- division between food and recipes sections -*)
(*- 2*0.6*1.00 = 1.20 -*)
2 "bread" "low price"."flour" 0.6 kg;   (*- Note that the ":" is optional -*)
(*- 3*0.4*1.50+3*0.2*8.00+3*0.1*21.00 = 1.80+4.80+6.30 = 12.90 -*)
3 "pizza" : "medium price"."flour" 0.4 kg, "low price"."tomato sauce" 0.2 kg,
            "medium price"."mozzarella cheese" 0.1 kg;
```

Output:

```
Less expensive: "flour" 1.00 EURO/kg
More expensive: "mozzarella cheese" 21.00 EURO/kg
----
"bread"
"flour" 1.20 EURO
TOT: 1.20 EURO
"pizza"
"flour" 1.80 EURO
"tomato sauce" 4.80 EURO
"mozzarella cheese" 6.30 EURO
TOT: 12.90 EURO
```

Weights: Scanner 8/30; **Grammar** 9/30; **Semantic** 10/30