Formal Languages and Compilers

11 September 2024

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*, *camping*, and *users* sections, separated by means of the sequence of characters "`*`" (the number of characters is odd and the minimum number is 5). 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 type of tokens, each terminated with the character ";":

- `<tok1>`: Starts with "A:", followed by a word composed of at least 7 characters from the set "!", "@", or "#", arranged in any order and in an odd number. Optionally followed by a ":" and a binary number between 101 and 10110.
- `<tok2>`: Starts with "B:", followed by a word composed of 3 to 22 repetitions of "&&", "%%", or "$$" (any order is possible). Followed by 2 or 3 words of 4, 6, or 8 alphabetic characters separated by "#".
- `<tok3>`: Starts with "C:", followed by a time in the format HH:MM between 10:32 and 18:27 or in HH:MM am/pm format between 10:32 am and 06:27 pm.

Header section: grammar

In the *header* section tokens can appear **in any order**. In addition, `<tok3>` can appear **0 or more times**, `<tok2>` must appear **exactly 2 times**, and `<tok1>` must appear **exactly 1 time**. Remember to manage this requirement with grammar.

Camping section: grammar and semantic

The *camping* section is composed of a list of **at least 2** `<sols>` in **even** number (i.e., 2, 4, 6,...).

Each `<sol>` is composed of a `<type>` (i.e., a *quoted string between ' characters*), a ":", a list of `<characteristics>` separated with ",", and a ";". A `<characteristic>` is a `<dim>` (i.e., a *quoted string between ' characters*), followed by a `<price>` (i.e., a *unsigned real number with two decimals*), and by the word "euro/day".

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

Users section: grammar and semantic

The *users section* is **optionally** started by a `<min_sum>` instruction, and followed by a **possibly empty** list of `<reservation>` instructions.

The `<min_sum>` instruction is the "`MIN_SUM`" word, followed by a "(", by a list of `<items>`, a ")" and a ";". An `<item>` is the combination `<type>.<dim>` (which identifies the `<price>` associated to the combination `<type>.<dim>`, and accessible through the symbol table). The `<min_sum>` instruction prints the *minimum* and the *sum* between `<prices>` associated to the listed `<items>`.

A `<reservation>` instruction is a `<discount>` (i.e., an *unsigned real number*), the **optional** presence of the symbol `<electricity>` (i.e., the character "E"), a `<user_name>` (i.e., a *quoted string between ' characters*), a ":", a list of `<durations>` separated with "," and terminated with ";". A `<duration>` is an `<item>` followed by a `<days>` (i.e., an *unsigned integer number*). For each `<reservation>` instruction, the translator must multiply the `<days>` with the `<price>` and with the `<discount>`. If the `<electricity>` (i.e., the character "E") is present, to the result of the previous computation, the compiler must add 10.00 euro. To perform the computation, **use inherited attributes** to access the symbols `<discount>` and `<electricity>`. Finally, for each `<reservation>` and for each `<duration>`, the translator must print the `<user_name>` and the summation of all the sub-results computed for each `<duration>`.

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

Example

Input:

```
(+ Header section +)
A:!@!@###!#:10000 ;                (+ tok1 +)
B:&&%%&&abcd#AbCdEf;               (+ tok2 +)
C:11:56 am;                        (+ tok3 +)
B:&&&&&&abcdefgh#ABCDEF#ABCDEF;    (+ tok2 +)
******
(+ Camping section +)
'roulotte' : 'small' 40.00 euro/day,
             'big' 100.00 euro/day;
'tend' : 'small' 20.00 euro/day,
         'medium' 30.00 euro/day,
         'big' 40.00 euro/day;
*******
(+ Users section +)
(+ SUM: 40.00+20.00+30.00=90.00 MIN_SUM command is optional +)
MIN_SUM('roulotte'.'small', 'tend'.'small', 'tend'.'medium');
0.5 E 'Stefano': 'roulotte'.'small' 3,    (+ (3*40.00)*0.5+10.00=120.00*0.5+10.00=70.00 +)
                 'tend'.'medium' 2;       (+ (2*30.00)*0.5+10=60.00*0.5+10.00=40.00 +)
1.0 'Gabriele' : 'tend'.'big' 5;          (+ (5*40.00)*1.0=200.00 +)
```

Output:

```
MIN: 20.00 SUM: 90.00
'roulotte'.'small' 70.00
'tend'.'medium' 40.00
'stefano' 110.00
'tend'.'big' 200.00
'gabriele' 200.00
```

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