Formal Languages and Compilers

01 July 2021

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*, *warehouse* and *products* 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 2 types of tokens, each terminated with the character ";":

- `<tok1>`: It is composed of the character "X", a "`-`", and a hour in the format `HH:MM:SS` between "03:51:47" and "23:45:34". The hour is optionally followed by an odd number of repetitions, at least 5, of the words "`aa`", "`ab`", "`ba`" and "`bb`". Example: `X-10:01:59aabbaabbaa`.
- `<tok2>`: It is composed of the character "Y", a "`-`", and 4, 123, or 257 repetitions of a binary number between 101 to 1010001. Numbers are separated by the character "`-`". Example: `Y-101-101100-1010001-101`.

Header section: grammar

The *header* section contains one of these two possible sequences of tokens:

1. at **least 4**, and in **even** number (4, 6, 8,...) repetitions of `<tok1>`, followed by **3 or 9 or 12** repetitions of `<tok2>`
2. **2 or 3** `<tok2>`, and **any number** of `<tok1>` (**even 0**) in any position of the sequence except for the first. This sequence **must start** with a `<tok2>`, the second and/or third repetitions of `<tok2>` can be in **any position** of the sequence.

Warehouse section: grammar and semantic

The *warehouse* section is composed of a list of **at least 2** `<material_type>` in **even** number (i.e., 2, 4, 6,...). Each `<material_type>` is composed of a "{", a `<material_list>`, a "}", a `<material_type_name>`, and a ";".

A `<material_list>` is a non-empty list of `<material>` separated with ",". A `<material>` is a `<material_name>`, a `<unit_price>` (i.e., a *real* and *positive* number with *two decimals*), and the word "`euro/kg`". Tokes `<material_type_name>` and `<material_name>` are *quoted strings*.

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

In addition, at the end of this section, for each `<material_type>` the translator must print the `<material_type_name>` and the `<material_name>` with the lower and higher values of `<euro>`, respectively (for the output format see the example).

Products section: grammar and semantic

The *products* section is composed of a list, which can be **empty**, of `<product>`. Each `<product>` is a `<tax>` (i.e., a *real* and *positive* number with *two decimals*), the word "`euro`", a `<product_name>` (i.e., a *quoted string*), a ":", and a non-empty list of `<element>` separated with ",", and a ";". Each `<element>` is a `<material_type_name>` a "{", a non-empty list of `<component>` separated with "," and a "}". Each `<component>` is a `<material_name>`, a `<quantity>` (i.e., a *unsigned integer*), and the word "`kg`".

In this section, for each `<product>`, the translator must print the `<product_name>`, and for each `<material_name>` related to the `<product>`, the translator must print the `<material_name>` and the `<price>`. The `<price>` is obtained by adding the `<tax>` to the result of the multiplication between the `<quantity>` and the `<unit_price>`, which can be obtained from the symbol table from `<material_type_name>` (which can be accessed through **inherited attributes**) and `<material_name>` (see the example).

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: second type of header *)
Y-101-101100-1010001-101 ;     (* tok2 *)
X-10:01:59aabbaabbaa;          (* tok1 *)
Y-1011-101000-1010001-111  ;   (* tok2 *)
X-23:44:59;                    (* tok1 *)

#### (* division between header and warehouse sections *)
(* Row material in the warehouse *)
{
    "plastic" 12.00 euro/kg,
    "iron" 5.00 euro/kg,
    "steel" 10.00 euro/kg
} "component";

{ "gpl" 0.63 euro/kg, "methane" 1.05 euro/kg } "energy";

#### (* division between warehouse and products sections *)
2.00 euro "Steel pots" : "component" { "steel" 10 kg , "plastic" 1 kg }, "energy" { "methane" 1 kg };
1.00 euro "Plastic glasses" : "component" { "plastic" 1 kg }, "energy" { "methane" 1 kg };
```

Output:

```
"component": less: "iron", more: "plastic"
"energy": less: "gpl", more: "methane"
---
"Steel pots"
"steel" 102.00 euro
"plastic" 14.00 euro
"methane" 3.05 euro
"Plastic glasses"
"plastic" 13.00 euro
"methane" 2.05 euro
```

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