Formal Languages and Compilers

19 July 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 two sections: *header* and *run* sections, separated by an even number of "`%`" characters (at least 4). Two types of comments are possible: they can be delimited by the starting sequence "`(((-`" and by the ending sequence "`-)))`", or they can start with the sequence `---` until the end of the line.

Header section: lexicon

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

- `<tok1>`: It is composed of the characters "A_" followed by a binary number containing 2 or 5 characters "0" (e.g., `A_11010`, `A_1001101001`), or it is composed of the characters "A_" followed by a word composed of symbols "*" and "+" (there is no limitation on the number of "*" and "+", even 0 is possible) without consecutive equal symbols (i.e., near to the symbol "+" we can have only "*", and near the symbol "*" we can have only "+"). Example: `A_+*+*+, A_*+*, A_+, A_`.
- `<tok2>`: It is composed of the characters "B_" followed by at least 4 words in even number (i.e., 4, 6, 8, 10,...). Each word is an even number between -32 and 1246. Words are separated by means of the characters "*", "$" or "+". Example: `B_-32$1246+12+0+-4$42`.

Header section: grammar

In the *header* section the token `<tok1>` can appear **in any number** (**even 0**), instead `<tok2>` can appear **zero**, **two**, or **three** times. There is no restriction on the order of both tokens.

Run section: grammar and semantic

The *run* section is used to simulate the power consumption of a car that has two source of power, namely battery and fuel. This section is composed of a `<start>` instruction followed by a list of **at least 4** `<command>` in **even** number (i.e., 4, 6, 8,...).

The `<start>` instruction is the word "`START`", a `<battery_ass>`, a "-", a `<fuel_ass>`, and a ";". The order of `<battery_ass>` and `<fuel_ass>` can be **inverted**, and both are **optional**. The command `<battery_ass>` is the word "`BATTERY`", an `<exp>`, and the word "`kWh`", while `<fuel_ass>` is the word "`FUEL`", an `<exp>`, and the word "`liters`". The command `<battery_ass>` sets the initial value of the *battery* to the result of `<exp>`, while `<fuel_ass>` sets the initial value of *fuel* to `<exp>`. In the case of the absence of one or both parts of the instruction, the missing value is initialized to 100. For instance, `START - BATTERY 10.0;` sets *battery=10.0* and *fuel=100.0*. **In all the examination global variables are not allowed. The current values of *battery* and *fuel* have to be stored in the parser stack and consistently updated.**

`<exp>` is a common mathematical expression in which operands can be *real* numbers or the function `<max>`, while operators can be "`PLUS`" (addition) or "`STAR`" (multiplication). The function `<max>` is the word "`MAX`", a "`(`", a non-empty list of `<exp>` separated with ",", and a "`)`". This function returns the maximum value between the listed `<exp>`.

The two possible commands in the list of `<command>` are `<mod>` or `<use>`, and both are followed by a ";". The `<mod>` command is the word "`MOD`", followed by a `<power_type>` (i.e., "`BATTERY`" or "`FUEL`"), and an `<exp>`. It adds to the type of power identified by `<power_type>` the value represented by `<exp>`.

The <use> command is the word "USE", followed by a <power_type> (i.e., "BATTERY" or "FUEL"), followed by the word "DO", followed by a non-empty list of <cons>, and by the word "DONE". A <cons> is an $\texttt{<exp}_a\texttt{>}$ (i.e., an <exp>), the word "km", an $\texttt{<exp}_b\texttt{>}$ (i.e., an <exp>), and the word "units/km". Each <cons> modifies the power identified by <power_type> by subtracting the value obtained by multiplying $\texttt{<exp}_a\texttt{>}$ and $\texttt{<exp}_b\texttt{>}$. **Use inherited attributes to access the current value of *battery*, of *fuel*, and of <power_type>.**

Each time a modification of the value of *battery* and *fuel* occurs, the translator must print the current value of these two quantities.

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:

```
A_111110011;           (((- tok1 -)))
A_+++++++++ ;          (((- tok1 -)))
B_-12*1132*1244+-4 ;   --- tok2
A_00000;               (((- tok1 -)))
B_-6$-4$-2*0$2$4;      (((- tok2 -)))

%%%%%%  --- division between header and run sections

START BATTERY 60.0 kWh - FUEL 10.0 PLUS 10.0 liters ;  (((- battery=60.0 fuel=20.0 -)))

USE FUEL DO
  10.0 km 0.5 units/km;   (((- fuel=20.0-10.0*0.5=20.0-5.0=15.0 -)))
  5.0 km 1.0 units/km;    (((- fuel=15.0-5.0*1.0=15.0-5.0=10.0  -)))
DONE;

MOD BATTERY MAX(2.0, 3.0, 1.0) PLUS 7.0 ; (((- battery=60.0+3.0+7.0=70.0 -)))

USE BATTERY DO
  5.0 PLUS MAX(5.0, 3.0) km 2.0 STAR 1.0 units/km; (((- battery=70.0-10.0*2.0=50.0 -)))
DONE;

(((- fuel=10.0+MAX(7.0,6.0,10.0)=10.0+10.0=20.0 -)))
MOD FUEL MAX(3.0 PLUS 2.0 STAR 2.0, 3.0 STAR 2.0, MAX(1.0, 3.0, 10.0));
```

Output:

```
battery=60.0 fuel=20.0
battery=60.0 fuel=15.0
battery=60.0 fuel=10.0
battery=70.0 fuel=10.0
battery=50.0 fuel=10.0
battery=50.0 fuel=20.0
```

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