Formal Languages and Compilers

14 September 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 two sections: *header* and *simulation* 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 character "I", a "_", a date in the format DD/MM/YYYY between 03/09/2021 and 05/03/2022. Remember that the months of September and November have 30 days, while the month of February has 28 days. The date is optionally followed by a ":" and by a time in the format HH:MM between 08:00 and 17:35. Example: `I_28/02/2022:08:10`.
- `<tok2>`: It is composed of the character "J", a "_", and by an even number of repetitions, at least 6, of hexadecimal numbers. Each hexadecimal number is between 3b and aE3. The hexadecimal numbers can be separated by the characters "+", "-" or "*". Example: `J_3b*3B+3C-aaa-123+9a+99+9B`.
- `<tok3>`: It is composed of the character "K", a "_", and it is followed by a word composed of an odd number (at least 5) of lowercase alphabetic letters, and optionally followed by 3 or more repetitions of the words "00", "11", "10", or "01" in any combination. Example: `K_hellooo0011001101`.

Header section: grammar

In the *header* section the tokens `<tok1>` and `<tok2>` can appear **in any order and number** (**even 0 times**), instead, `<tok3>` can appear only **0, 1, or 2 times**.

Simulation section: grammar and semantic

The *simulation* section is composed of an `<init>` instruction followed by a list of **at least 3** `<command>` in **odd** number (i.e., 3, 5, 7,...).

The `<init>` instruction is the word "`INIT`", a command `<height_ass>`, a ",", a command `<speed_ass>`, and a ";". The order of `<height_ass>` and `<speed_ass>` can be **inverted**, and both are **optional**. The command `<height_ass>` is the word "`HEIGHT`", a "=" and an `<exp>`, while `<speed_ass>` is the word "`SPEED`", a "=" and an `<exp>`. The command `<height_ass>` stores in a global variable *height* the result of `<exp>`, while `<speed_ass>` stores in a global variable *speed* the result of `<exp>`. In the case of the absence of one or both commands, the value stored in the corresponding variable is 0. For instance, `INIT SPEED=2 MUL 3, ;` stores *speed=6* and *height=0*. **Variables *height* and *speed* are the only global variables allowed in all the examination, an they can be written only in `<height_ass>` and `<speed_ass>` commands.**

`<exp>` is a common mathematical expression in which operands can be *integer* numbers or the content of variables *height* or *speed* (i.e., the words "`HEIGHT`" or "`SPEED`"), while operators can be "`SUM`" (addition) or "`MUL`" (multiplication).

The three possible commands in the list of `<command>` are `<height_ass>`, `<speed_ass>` or `<cond>`, all followed by a ";". The `<cond>` command is the word "`COND`", a "[", a `<comparison>`, a "]", the word "`UPDATE`" a list of `<ass>`, and the word "`DONE`".

The list of <ass> is a not empty list of <height_ass> and <speed_ass> commands. They are executed if the result of the <comparison> is *true* (**use inherited attributes** to access the value of the result of <comparison> and decide if the command is executed). <comparison> is **optional**, and if not present it must be considered as *true*.

<comparison> is an <exp>, a <comparison_operator>, and another <exp>. Possible comparison operators are ">" (which returns a *true* value if the first <exp> is greater than the second), and a "=" (which returns a *true* value if both <exp> are equal).

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:

```
I_28/02/2022:08:10;           +-- tok1 --+
K_hellooo0011001101 ;         +-- tok3 --+
I_31/12/2021 ;                +-- tok1 --+
K_hello ;                     +-- tok3 --+
J_3b*3B+3C-aaa-123+9a+99+9B;  +-- tok2 --+

====  +-- division between header and simulation sections --+

INIT HEIGHT = 20, SPEED = 10;   +-- HEIGHT=20, SPEED=10 --+

HEIGHT = HEIGHT SUM 10 MUL 2;   +-- HEIGHT=20+10*2=20+20=40 --+

COND [ HEIGHT SUM 1 > SPEED MUL 2 ]  +-- 41>20 -> true --+
UPDATE
    HEIGHT = 10 SUM 3;          +-- HEIGHT=10+3=13 --+
    SPEED = 10 SUM 3 SUM SPEED; +-- SPEED=10+3+10=23 --+
DONE;

COND [ ]                        +-- Always true --+
UPDATE
    HEIGHT = 3;                 +-- HEIGHT=3 --+
DONE;
```

Output:

```
HEIGHT 20
SPEED 10
HEIGHT 40
HEIGHT 13
SPEED 23
HEIGHT 3
```

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