Formal Languages and Compilers

06 July 2018

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 *drone* sections, separated by means of two characters "##". The sequences "$$" or "??" identify the start of a comment. A comment finishes at the end of the line.

Header section: lexicon

The *header* section can contain 3 types of tokens, each terminated with the character ";":

- <token1>: It begins with 3, 30 or 300 repetitions of the character "!", followed by an odd number (at least 5) of lowercase alphabetic letters or by an even number (at least 6) of uppercase alphabetic letters. The first part of the token is optionally followed by a binary number between 10 and 101110.
- <token2>: It is composed of an odd number of words (at least 3), separated by the character "*", "$" or "%". Each word is composed of 2, 4 or 7 repetitions of the strings "xx", "xy", "yx" or "yy" in any combination.
- <token3>: A date in the format "<month> DD, YYYY" (where <month> can be June, July, August, September, October or june, july, august, september, october) or in the format "YYYY/MM/DD" between June 13, 2018 and October 23, 2018 (or between 2018/06/13 and 2018/10/23) with the exclusion of the day July 02, 2018 (or 2018/07/02). Remember that the months of June and September have only 30 days.

Header section: grammar

In the header section <token1> and <token2> can appear in **any order** and **number** (also **0 times**), instead, <token3> can appear only **0, 2 or 3 times**.

Drone section: grammar and semantic

The *drone section* starts with the `SET` instruction, which sets the current state of a drone in terms of position (coordinates `X`, `Y` and `Z`) and battery (`B`). The `SET` instruction has the syntax `SET POS X Y Z - BATTERY B;`, where `X`, `Y`, `Z` and `B` are *signed integer* numbers, `POS X Y Z` sets the position of the drone to the values `X`, `Y` and `Z`, while `BATTERY B` sets the battery level to `B`. The **order** of `POS X Y Z` and `BATTERY B` inside the `SET` instruction can be **inverted** (i.e., `POS` followed by `BATTERY` or `BATTERY` followed by `POS`).

After the `SET` instruction, there is a list of <commands>, each terminated with a ";". The number of commands is **odd** and **at least 7**.

The following four commands are defined:

- `PRINT`: The word `PRINT` followed by a *quoted string*. This instruction prints the quoted string.
- `CHANGE`: The word `CHANGE` followed by the option `POS <offsetX> <offsetY <offsetZ>`, or by `BATTERY <offsetB>`, or by both separated by the character "-" (i.e., `POS` followed by `BATTERY`, or `BATTERY` followed by `POS`). The two options change the current position of the drone (`POS.X`, `POS.Y`, `POS.Z`) of the quantities `<offsetX> <offsetY <offsetZ>`, or the `BATTERY` of the quantity `<offsetB>`, respectively.
- `COMPUTE`: This command has the following syntax `COMPUTE(<stat>, <values_list>)`. `<values_list>` is a non empty list of `<value>` separated by "," (commas). `<value>` can be a *signed integer* number or a `<state_value>`, i.e., the keywords `POS.X`, `POS.Y`, `POS.Z`, `BATTERY`, which represent the current state of the drone in terms of its position or battery, respectively. `<stat>` can be equal to `AVG` or `MAX`. It sets the command to compute the *average* or the *maximum* between the `<value>` listed in `<values_list>`.
- `IN_CASE`: This command is the more complex. It has the syntax `IN_CASE <state_value> <cases_list>`. `<state_value>` is one keyword between `POS.X`, `POS.Y`, `POS.Z` and `BATTERY`. `<cases_list>` is a non empty list of `<case>`. Each `<case>` starts with an `<interval>`, followed by the word `DO`, an `<instruction_list>`, and ended by the word `DONE`. `<instruction_list>` is a list composed by commands `PRINT` or `CHANGE`,

which can appear in any order. If the value of the current state identified by `<state_value>` is within the interval identified by `<interval>`, the instructions listed in `<instruction_list>` are executed. Three types of `<interval>` are possible: `LOWER <number>` (*true* if `<number>` is lower than the value identified by `<state_value>`), `HIGHER <number>` (*true* if `<number>` is bigger than the value identified by `<state_value>`), `BETWEEN <number_1> AND <number_2>` (*true* if the value identified by `<state_value>` is in the range between `<number_1>` and `<number_2>`, extremes included). All `<numbers>` are *signed integers*.

To simplify the command, the `<state_value>` used to check if the value represented by one of the keywords `POS.X`, `POS.Y`, `POS.Z` or `BATTERY` is inside or outside of an `<interval>`, is not changed instantly by the instructions executed inside the `IN_CASE` command, but only at its end (i.e., for the first `IN_CASE` command of the example, `POS.X` is changed only at the end of the `IN_CASE` command, that is after the three `<intervals>` were evaluated and all the instructions executed).

Goals

The translator must execute the language previously described. **No global variable are allowed. Solutions that uses global variables will not be accepted.** Each command may change the state of the drone (in terms of `X`, `Y`, `Z` and `B`). Use the parser stack to store the last state, and use inhered attributes to access from any command the previous state, which must be checked and/or modified by the command. For the output of the commands look the example.

Example

Input:

```
$$ Header section

xxxx*xyxyxxyy$yyyy*xyxy*xxxy;   $$ <token2>
!!!abcdefg ;                    $$ <token1>
july 03, 2018;                  $$ <token3>
!!!ABCDEFGH10111;               $$ <token1>
2018/10/22 ;                    $$ <token3>


##


$$ Drone section
$$ Instruction SET POS 0 0 0 - BATTERY 100; is also possible
SET BATTERY 100 - POS 0 0 0 ;    ?? BATTERY: 100 - POS: 0 0 0

PRINT "START PROGRAM";
CHANGE POS -2 +3 2;                $$ BATTERY: 100 - POS: -2 3 2
CHANGE BATTERY -3 - POS 1 3 3;     $$ BATTERY: 97 - POS -1 6 5

COMPUTE (AVG, POS.X, POS.Y, 4);    $$ Returns: (-1 + 6 + 4) / 3 = 9.0
COMPUTE (MAX, POS.X, POS.Y, BATTERY, -2);  $$ Returns: max(-1,6,97,-2) = 97.0

$$ Inside IN_CASE only CHANGE and PRINT instructions are allowed
$$ POS.X is equal to -1 (and remains equal to -1 for all the IN_CASE command)
IN_CASE POS.X LOWER 3 DO  $$ True
                PRINT "1";
                CHANGE BATTERY -2 ;               $$ BATTERY: 95 - POS: -1 6 5
                CHANGE POS -1 -1 -1;              $$ BATTERY: 95 - POS: -2 5 4
              DONE
              HIGHER 2 DO  $$ False
                PRINT "2";
                CHANGE POS -1 -2 -3 - BATTERY -5 ;
              DONE
              BETWEEN -5 AND 6 DO  $$ True
                PRINT "3";
                CHANGE POS -2 -2 -2 - BATTERY -4 ; $$ BATTERY: 91 - POS: -4 3 2
              DONE ;
CHANGE BATTERY -2 ;                $$ BATTERY: 89 % - POS: -4 3 2
$$ POS.Z is equal to 2
IN_CASE BATTERY LOWER 3 DO  $$ False
                  PRINT "4";
                DONE;
PRINT "END PROGRAM";
```

Output:

```
BATTERY: 100 - POS: 0 0 0
"START PROGRAM"
BATTERY: 100 - POS: -2 3 2
BATTERY: 97 - POS -1 6 5
9.0
97.0
"1"
BATTERY: 95 - POS: -1 6 5
BATTERY: 95 - POS: -2 5 4
"3"
BATTERY: 91 - POS: -4 3 2
BATTERY: 89 - POS: -4 3 2
"END PROGRAM"
```