# Formal Languages and Compilers

04 July 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 to manage *inline skate athletes*.

## Input language

The input file is composed of three sections: *header*, *athletes*, and *evaluation* sections, separated by means of the sequence of characters "`*`" (the number of characters is even and the minimum number is 4). 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 type of tokens, each terminated with the character ";":

- `<tok1>`: Starts with "A:", followed by the character "!" followed by a binary number containing 4, 6, or 11 characters 1 or 0. Alternatively, it starts with the character "B:", followed by the character "@" followed by a time in the format HH:MM:SS between 07:21:13 and 19:45:54.
- `<tok2>`: Starts with "C:", followed by a real number with two decimals between 10.53 and 13.74 or the word "value". This first part is followed by 2 or 10 repetitions of "@" or "!" (even mixed together). Finally, it is ended by an odd number of lowercase alphabetic letters or an even number of uppercase alphabetic letters.

### Header section: grammar

Two sequences of tokens are possible in the *header* section:

1. **three** `<tok1>` and **any number** of `<tok2>` (**even 0**). This sequence must **start with** a `<tok1>`, the second and third repetitions of `<tok1>` can be in **any position** of the sequence. Example: `<tok1> <tok1> <tok2> <tok2> <tok1>`.
2. **at least 6**, and in **even number** (6, 8, 10,...) repetitions of `<tok2>`, followed by **4 or 13 repetitions** of `<tok1>`. Example: `<tok2> <tok2> <tok2> <tok2> <tok2> <tok2> <tok1> <tok1> <tok1> <tok1>`.

### Athletes section: grammar and semantic

The *athletes* section is composed of a list of **at least 3** `<athlete>` in **odd** number (i.e., 3, 5, 7,...).

Each `<athlete>` is composed of a `<name>` (i.e., a *quoted string*), a ":", a list of `<attr>` separated with ",", and a ";". An `<attr>` is an `<attr_name>` (i.e., a *quoted string*), followed by an `<expr>`. An `<expr>` is a typical mathematical expression containing "+", "-", "*", "/" operators, and parenthesis. Operands are *unsigned real numbers*.

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

### Evaluation section: grammar and semantic

The *evaluation section* is a **possibly empty** list of `<eval>`. An `<eval>` is a `<name>`, a `<mult>` (i.e., an *unsigned real number*), the word "POINTS" (which is **optional**) a ":", a list of `<val>` separated with ",", and a ";". Each `<val>` is a `<value>` (i.e., an *unsigned real number*), an `<attr_name>`, and a `<score>`. The `<score>` is the word "LOW", "MEDIUM", or "HIGH", which corresponds to the number 0.0, 1.0, or 2.0, respectively.

For each `<eval>` the tranlator must print the `<name>`. Then, for each `<attr_name>` it must print the sub-result, i.e., the result of the multiplication between `<mult>`, the `<value>`, the result to the expression associated to the couple `<name>.<attr_name>` (accessible through the symbol table), and the number associated to the `<score>`. **Use inherited attributes** to access the symbols `<attr_name>` and `<mult>`. Finally, the translator must print the summation of all the sub-result. 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:!011101 ;                [++ tok1 ++]
C:11.70!@!!@@@!@!hello;    [++ tok2 ++]
B:!07:22:30  ;             [++ tok1 ++]
A:!0000;                   [++ tok1 ++]
C:value!!EVENWORD ;        [++ tok2 ++]
****
[++ Athletes section ++]
"Gabriele" : "speed" 1.05+1.0*.05, "resistance"  (.5+0.5)*1.2;[++ "speed" 1.1 "resistance" 1.20 ++]
"George" : "speed" 1.0, "technique" 1.10, "resistance" 0.9;
"Armando" : "speed" 1.0, "technique" 1.10;
******
[++ Evaluation section ++]
[++ 2.0*10.0*1.1*2.0 + 2.0*20.0*1.20*1.0 = 44.0 + 48.0 = 92.0 ++]
"Gabriele" 2.0 POINTS : 10.0 "speed" HIGH, 20.0 "resistance" MEDIUM;
[++ 1.5*20.0*1.10*1.0 + 1.5*30.0*1.0*1.0 + 0.0 = 33.0 + 45.0 + 0.0 = 78.0 ++]
"George" 1.5 : 20.0 "technique" MEDIUM, 30.0 "speed" MEDIUM, 30.0 "resistance" LOW;
```

### Output:

```
"Gabriele"
"speed" 44.0
"resistance" 48.0
TOTAL: 92.0
"George"
"technique" 33.0
"speed" 45.0
"resistance" 0.0
TOTAL: 78.0
```

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