# Formal Languages and Compilers

22 September 2023

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 *commands* 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 "A", a "_", and 4, 7, or 11 repetitions of an odd number between $-273$ to 457. Numbers are separated by the character "*". Example: `A_-153*7*83*457`
- `<tok2>`: It is composed of the character "B", a "_", and a date between "2023/09/22" (or "2023/September/22") and "2024/02/07" (or "2024/February/07") with the format YYYY/MM/DD (or YYYY/`<month_name>`/DD). Remember that the months of September and November have 30 days. Example: `B_2023/09/22`
- `<tok3>`: It is composed of the character "C", a "_", and by an even number of repetitions, at least 4, of hexadecimal numbers. Each hexadecimal number is composed of 4 or 8 characters. The hexadecimal numbers can be separated by the characters "*", "$", or "&". Example: `C_12aa*abCd$12345678$abcc`

### Header section: grammar

In the *header* section `<tok1>` must appear exactly **1 time**, `<tok2>` can appear **1 or 2 times**, instead `<tok3>` can appear in **any order** and number (**also 0 times**). There are no restrictions on the order of tokens in the sequence.

### Commands section: grammar and semantic

The *commands* section is composed of a list of **at least 4** `<command>` in **even** number (i.e., 4, 6, 8,...).

The three types of commands are `INS`, `CMP`, and `SUM`, and they can contain `<math_exp>`. A `<math_exp>` is a typical mathematical expression enclosed in square brackets containing "+", "-", "*", "/" operators, and parenthesis. Operands are *unsigned integer numbers.*

Each instruction is terminated by the ";" character and it has the following syntax:

- `INS <math_exp>`: executes the mathematical operation included in `<math_exp>`, and prints the result.

- `CMP <math_exp1>, <math_exp2>`: compares the previous two results of the commands previously executed (i.e., $R_{-1}$ and $R_{-2}$). If $R_{-1} == R_{-2}$ (i.e., $R_{-1}$ is equal to $R_{-2}$), the `CMP` command executes `<math_exp1>` and prints the result, otherwise if $R_{-1} <> R_{-2}$ (i.e., $R_{-1}$ is not equal to $R_{-2}$) the command executes `<math_exp2>` and prints the result. There is a more compact version of `CMP` in which `<math_exp2>` and the "," do not exists (i.e., `CMP <math_exp1>`), in this case if $R_{-1} <> R_{-2}$ the function will return the value 0.

- `SUM <exp_list>`: executes the sum of the elements in `<exp_list>` and prints the result. `<exp_list>` is a list of `<math_exp>` separated by ",".

## 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:

```
B_2023/10/25;                         [[+ tok2 +]]
A_-1*-3*123*15*-7*9*11;               [[+ tok1 +]]
B_2024/January/01;                    [[+ tok2 +]]
C_1234$abcde123$1ABd*1234$30aB$1234; [[+ tok3 +]]
##
INS [16];                     [[+ 16 +]]
INS [10+3*2];                 [[+ 16 +]]
CMP [2*4*5];                  [[+ 2*4*5=40 because 16==16 +]]
INS [3];                      [[+ 3 +]]
CMP [(10+2)*2], [10] ;        [[+ 10 because 40 <> 3 +]]
SUM [1], [1+1], [1+1+1] ;     [[+ 1+2+3=6 +]]
SUM [10*2], [2];              [[+ 20+2=22 +]]
CMP [2];                      [[+ 0 because 6 <> 22 +]]
```

### Output:

```
16
16
40
3
10
6
22
0
```

**Weights: Scanner** 8.5/30; **Grammar** 9/30; **Semantic** 9.5/30