Formal Languages and Compilers

30 June 2020

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 *code* 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 2 types of tokens, each terminated with the character ";":

- `<tok1>`: it is a date in the format YYYY/MM/DD from 2020/01/12 to 2020/07/13. Remember that the months of April and June have 30 days, while the month of February 2020 had 29 days. The date is followed by a "?", and an odd number of hours, at least 3, separated by the characters "*" or "$". The hour has the format HH:MM, where HH is a number between 00 and 23, while MM is a number between 00 and 59.
- `<tok2>`: it starts with the character "!", followed by an integer and even number between -16 and 136, or by 2, 7 or 23 repetitions of the strings "xx", "yy", "aa", "bb" (any sequence of these strings is possible).

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.

Code section: grammar and semantic

The *code section* is composed of a list of `<commands>`. The list can be possibly **empty**, or with an **odd** number of elements, **at least 5**. As a consequence, the list can be composed of 0, 5, 7, 9,... elements.

Two types of commands are possible:

- *Assignment*: it is a `<variable>` (same regular expression of C identifiers), followed by a "=", and a `<value>` (i.e., a quoted string). This command stores the `<value>` into an entry of a global symbol table with key `<variable>`. **This symbol table is the only global data structure allowed in all the examination, and it can be written only by means of an assignment command**. Each time an *assignment* command is executed, the command prints into the screen the `<variable>` name and the associated `<value>`.
- *IF*: it has the following syntax:

  `IF [ <bool_expr> ] [ <ass_list_TRUE> ] ELSE [ <ass_list_FALSE> ]`

  where `<bool_expr>` represents the result of a boolean expression (i.e., a `TRUE` or a `FALSE` value). `<ass_list_TRUE>` and `<ass_list_FALSE>` are two **non empty** lists of *assignment* commands. The *assignment* commands reported in `<ass_list_TRUE>` are executed if the result of `<bool_expr>` is `TRUE`, while the commands in `<ass_list_FALSE>` are executed if the result of `<bool_expr>` is `FALSE`. To manage the execution (or not) of an *assignment* command check within the grammar semantic action of the command, using inherited attributes, the result of `<bool_expr>`.
  The `ELSE [ <ass_list_FALSE> ]` part of the `IF` command is **optional**.

`<bool_expr>` can contain the following logical operators: `&` (and), `|` (or), `!` (not), and round brackets. Operands can be `TRUE` (the true constant), `FALSE` (the false constant), and a *comparison* instruction that has the following syntax `<variable> == <quoted_string>`. If the `<value>` associated to `<variable>` (accessed through the symbol table) is equal to `<quoted_string>`, the result of the *comparison* instruction is `TRUE`, otherwise the result is `FALSE`.

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:

```
2020/02/29?10:30*22:00$22:30*23:00*23:30; ((-- tok1 --))
!-8 ;                                     ((-- tok2 --))
!xxxxyyxxaaaayy ;                         ((-- tok2 --))
2020/07/02?12:00*12:30*13:00;             ((-- tok1 --))

%%% ((-- division between header and execution sections --))

a = "one";
b="two" ;

((-- TRUE | TRUE & FALSE = TRUE --))
IF [ a == "one" | TRUE & b =="three" ] [
  c="1";
  d="2";
] ELSE [
  c="3" ;   ((-- not executed --))
]

((-- ! FALSE & !!TRUE = TRUE & TRUE = TRUE --))
IF [ ! a=="two" & !!a=="one" ] [
  e="4";
]

f = "end";
```

Output:

```
a "one"
b "two"
c "1"
d "2"
e "4"
f "end"
```

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