Formal Languages and Compilers

03 July 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 three sections: *header*, *houses*, and *preferences* sections, separated by means of the sequence of characters "`$$$`" (at least 3 in odd number). 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>`: consists of an even number (at least 4) of exclamation marks (i.e., "!") followed by an even number ranging from -18 to 286, or an odd number (at least 5) of question marks (i.e., "?").
- `<tok2>`: is a date with the format "DD/MM/YYYY" or the format "YYYY/MM/DD" between 02/07/2023 (or 2023/07/02) and 06/10/2023 (or 2023/10/06). Remember that the month of September has only 30 days.
- `<tok3>`: is an hour in the format HH:MM or HH:MM:SS, between 07:37:19 and 22:39:23.

Header section: grammar

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

Houses section: grammar and semantic

The *houses* section is composed of a list of houses with **at least 2** `<house>` in **even** number (i.e., 2, 4, 6,...).

Each `<house>` is the word "house", a `<type>` (i.e., a quoted string), the word "start", a `<room_list>`, and the word "end". The `<room_list>` is a non-empty list of `<room>` separated with ",", where each `<room>` is a `<room_name>` (i.e., a quoted string) and a `<size>` (i.e., an unsigned integer that represents the size in square meters). All the data of this section must be stored in a symbol table with `<type>` as the key. **This symbol table is the only global data structure allowed in all the examination, and it can be written only in this section.**

Preferences section: grammar and semantic

The *preferences* section is composed of a list that can be **empty** of `<if>` commands. Each `<if>` command is the word "if" followed by a `<bool_exp>`, the "then" word, a `<print_list>`, and the word "fi".

A `<bool_exp>` can contain the following logical operators: `and`, `or`, `not`, and round brackets to define the scope. Operands are a `<room_ref>`, the symbol `==` and a `<size>`. The `<room_ref>` is a `<type>`, a "." (i.e., a dot), and a `<room_name>`. The couple `<room_ref>.<type>` can be used to access the `<size>`, which was stored in the symbol table in the previous *houses* section. If the value obtained from the symbol table, which is associated with the couple `<room_ref>.<type>`, is equal to the `<size>`, the operand is associated with a *true* value; otherwise, it is associated with a *false* value.

The `<print_list>` is a non-empty list of `<print>` commands. A `<print>` command is the word "print" followed by a *quoted string*, and by a ";".

If the result of the computation of the `<bool_exp>` is *true*, the `<print>` commands listed in `<print_list>` are executed. In particular, the quoted string associated to each print command is printed into the screen.

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:

```
08:34:10;   <* tk3 *>
2023/09/07; <* tk2 *>
10:12;      <* tk3 *>
!!!!!!-12;  <* tk1 *>
$$$$$
house "three-room" start
   "kitchen" 10, "living" 12, "bedroom" 8, "bathroom" 3
end
house "one-room" start
   "kitchen-bedroom" 13, "bathroom" 2
end
$$$
<* false or true and true = true *>
if "three-room"."living" == 6 or "three-room"."living" == 12 and "three-room"."kitchen" == 10 then
   print "house";
   print "found";
fi
<* not ( true or true) = false *>
if not ("one-room"."kitchen-bedroom" == 13 or "one-room"."bathroom" == 2) then
   print "not found";
fi
```

Output:

```
"house"
"found"
```

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