Formal Languages and Compilers

26 June 2025

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*, *objects*, and *queries* sections, separated by means of the sequence of characters "`++++`" (at least 4 in even number). Comments are possible, and they are delimited by the starting sequence "`<++`" and by the ending sequence "`++>`", or in C++ style (i.e., starting from the sequence "`//`" until the end of the line).

Header section: lexicon

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

- `<tk1>`: it is the character "!" followed by a binary number between 101 and 1011110, or it is the character "?" followed by a odd number between −123 and 2565". The part of the token starting with "?" is optionally followed by a word composed of the symbols "i" and "j" without consecutive equal symbols (i.e., near the symbol "i" only "j" symbols are possible, and near the symbol "j" only "i" symbols are possible). Example: `?19ijiji`.
- `<tk2>`: it is the character "$", followed by a date with the format "YYYY/MM/DD" between 2025/06/16 and 2026/02/24. Remember that the months of June, September, and November have 30 days. The date is optionally followed by an hour with the format ":HH:MM" between :05:18 and :10:47. Example: `$2025/08/01:07:12`.

Header section: grammar

In the *header* section, the `<tk1>` token must appear **one or two times**, while `<tk2>` token must appear **at least one time**. Manage this requirement with grammar.

Objects section: grammar and semantic

The *objects* section is composed of a list of objects with **at least 2** `<objects>` in **even** number (i.e., 2, 4, 6,...). Each `<object>` is the word "obj", followed by an `<attr_list>`, the word "name", an `<obj_name>` (i.e., a *quoted string*), and the word "end". The `<attr_list>` is a non-empty list of `<attr>` separated with ",", where each `<attr>` is a `<attr_name>` (i.e., a *quoted string*) and a `<attr_value>` (i.e., an *unsigned integer number*). All the data of this section must be stored in a symbol table with `<obj_name>` 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.**

Queries section: grammar and semantic

The *queies* section is composed of a list that can be **empty** of `<query>` commands. Each `<query>` command is the symbol "`?`" followed by a `<bool_exp>`, a `<print_function>`, and the word "`-?`".

A `<bool_exp>` can contain the following logical operators: `AND`, `OR`, `NOT`, and round brackets to define the scope. Operands are an `<obj_ref>`, the symbol `-eq` (for equality) or `-neq` (for not equality) and an `<attr_value>`. The `<obj_ref>` is an `<obj_name>`, a "." (i.e., a dot), and an `<attr_name>`. The couple `<obj_name>.<attr_name>` can be used to access the `<attr_value>`, which was stored in the symbol table in the previous *objects* section. If the value obtained from the symbol table, which is associated with the couple `<obj_name>.<attr_name>`, is equal to the `<attr_value>` (in the case of `-eq`), or is not equal to `<attr_value>` (in the case of `-neq`), the operand is associated with a *true* value; otherwise, it is associated with a *false* value.

<print_function> is composed of a <print_true> followed by <print_false>, or vice-versa, where one of the two can be not present (see the last <query> of the example where only <print_false> is present).

<print_true> is the word "IS_TRUE" followed by a <print_list>. Each element of <print_list> is the word "print" followed by a <print_str> (i.e., a *quoted string*). If <bool_exp> is *true*, the <print_str> strings are printed into the screen.

<print_false>, works similarly to <print_true>, with the difference that the word "IS_TRUE" is substituted with "IS_FALSE" and that <print_str> strings are printed if <bool_exp> is *false*.

To decide if print or not <print_str> strings, **use inherited attributes and the parser stack**.

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:

```
<++ header section ++>
!1001;              // tk1
$2025/07/02;        <* tk2 *>
$2026/01/01:06:00;  // tk2
?-121;              <* tk1 *>
++++++
<++ objects section ++>
obj "weight" 3, "price" 4, "height" 5 name "pen" end
obj "type" 2, "size" 4, "price" 10
  name "clock"
end
++++
<++ queries section ++>
<++ true AND true = true ++>
? "pen"."price" -eq 4 AND "clock"."price" -neq 12
IS_TRUE
    print "Correct prices"
    print "ok"
IS_FALSE
    print "Not correct price"
-?

<++ NOT (true OR false) = NOT true = false ++>
? NOT ( "clock"."size" -eq 4 OR "pen"."height" -eq 3 )
IS_FALSE print "size check false"
IS_TRUE print "size check true"
-?

<++ false ++>
? "pen"."weight" -neq 3 IS_FALSE print "weight check false" -?
```

Output:

```
"Correct prices"
"ok"
"size check false"
"weight check false"
```

Weights: Scanner 7/30; **Grammar** 10/30; **Semantic** 10/30