Formal Languages and Compilers

04 July 2022

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 *execution* 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 "D", a "-", a date in the format DD/<month>/YYYY between 04/July/2022 and 15/January/2023. Remember that the months of September and November have 30 days. The date is optionally followed by a "-" and another date with the same format of the previous one. Example: `D-05/July/2022-01/January/2023`.
- `<tok2>`: It is composed of the character "R", a "-", and at least 4 and at most 15 repetitions of the following 3 words composed of 2 characters: "XX" or "YY" or "ZZ", which may appear in any order. The first part of the token is optionally followed by an even number (at least 4) of "?" characters. Example: `R-XXYYXXYYZZ??????`.
- `<tok3>`: It is composed of the character "N", a "-", and it is followed by an odd number of repetitions, at least 5, of hexadecimal numbers. Each hexadecimal number is between 2A and bC3. The hexadecimal numbers can be separated by the characters "+", "/" or "*". Example: `N-2a/2A*aA1/123*12B*12c*12f`.

Header section: grammar

The *header* section contains one of these two possible sequences of tokens:

1. **at least 3**, and in **odd** number (3, 5, 7, 9,...) repetitions of `<tok2>`, followed by 2 or 3 or 9 repetitions of `<tok1>` and `<tok3>`, which can appear in **any possible order**. Example: `<tok2> <tok2> <tok2> <tok2> <tok3> <tok1> <tok3>`
2. `<tok1>` and `<tok2>` can appear in **any order** and **number** (**also 0** times), instead, `<tok3>` must appear **exactly 3 times**. The **first token** of the sequence **must** be `<tok3>`. Example: `<tok3> <tok1> <tok2> <tok1> <tok3> <tok3> <tok1> <tok2>`

Execution section: grammar and semantic

The *execution* section is composed of a **possibly empty** list of **at least 5** `<command>` in **odd** number (i.e., 0, 5, 7, 9,...).

Two types of `<command>` are defined, namely `<ass>` and `<if>`.

The `<ass>` is a `<var>` (same regular expression of C identifiers), follower by a "=", and a `<bool_exp>`. This command stores the result of the `<bool_exp>` into an entry of a global symbol table with key `<var>`. **This symbol table is the only global data structure allowed in all the examination, and it can be written only by means of an `<ass>` command.**

`<bool_exp>` can contain the following logical operators: `&` (and), `|` (or), `!` (not), and round brackets. Operands can be TRUE (the true constant), FALSE (the false constant), or a `<var>` (which represents the value stored in the symbol table by an `<ass>` command).

The `<if>` command has the following syntax:
`IF <bool_exp1> <list_comp> FI`
where `<bool_exp1>` represents the result of a `<bool_exp>` (i.e., `TRUE` or `FALSE`). `<list_comp>` is a non-empty list of `<comp>`, and each `<comp>` is an `OR` or an `AND` followed by `<bool_exp2>` (that represents the result of a `<bool_exp>`), the word `DO`, a `<print>`, and the word `DONE`.

Each time `<bool_exp1>` `<comp>` `<bool_exp2>` is `TRUE` (i.e., at least one `<bool_exp>` is `TRUE` in the case of `<comp>` equal to `OR`, or both `<bool_exp>` are `TRUE` in the case of `<comp>` equal to `AND`), the `<print>` instruction is executed. The `<print>` instruction consists in the word `PRINT` followed by a *quoted string* and a "`;`". When executed the quoted string is printed. **For the implementation of the `<if>` command use inherited attributes to access the values of `<bool_exp1>` to decide to execute or not the `<print>` instruction.**

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: of type 1  --]]
R-XXXXXXXX ;                        [[-- tok2 --]]
R-XXYYXXYYXX???? ;                  [[-- tok2 --]]
R-XXYYXXYYZZXXYYXX;                 [[-- tok2 --]]
D-05/August/2022 ;                  [[-- tok1 --]]
N-2B*Bc2*bC2/aaa/aAa;               [[-- tok3 --]]
D-10/August/2022-03/January/2023 ; [[-- tok1 --]]

====  [[-- division between header and execution sections --]]
x = TRUE & FALSE;               [[-- x = FALSE --]]
y = TRUE;                       [[-- y = TRUE --]]
z = FALSE | FALSE & FALSE;      [[-- z = FALSE | FALSE = FALSE --]]

IF ! x | (TRUE & TRUE)       [[-- TRUE --]]
  OR x | x DO   [[-- TRUE OR FALSE -> Executed --]]
    PRINT "one";
  DONE
  AND FALSE DO  [[-- TRUE AND FALSE -> Not Executed --]]
    PRINT "two";
  DONE
  AND TRUE DO   [[-- TRUE AND TRUE -> Executed --]]
    PRINT "three";
  DONE
FI
x = TRUE;       [[-- x = FALSE --]]
```

Output:

```
"one"
"three"
```

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