Formal Languages and Compilers

06 February 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 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 3 types of tokens, each terminated with the character ";":

- `<tok1>`: Starts with "X:", followed by a real number with one decimal between -4.4 and 12.3.
- `<tok2>`: Starts with "Y:", followed by at least 5 repetitions of the character "!" or "$" (which can be mixed). Followed by a sequence of lowercase alphabetic characters in an odd number. Optionally followed by a binary number between 11 and 10101.
- `<tok3>`: Starts with "Z:", followed by 3, 4, or 7 hexadecimal numbers of 3 or 5 characters each, separated by "!" or "?".

Header section: grammar

In the header section `<tok1>` and `<tok3>` can appear in **any order and number** (**also 0 times**), instead, `<tok2>` can appear **only 0, 1, or 3 times**.

Code section: grammar and semantic

The *code* section is composed of a list that can be **empty** or composed of **at least 5** `<instruction>` in **odd** number (i.e., 0, 5, 7, 9,...).

Two types of `<instructions>` are defined:

- `<ass>`: It is a `<list_of_id>` followed by the symbol "=", a `<bool_expr>`, and a ";". `<list_of_id>` is a non-empty list of `<id>` (same regular expression of C identifiers) separated with ",". The results of a `<bool_expr>` can be `true` or `false`. When the `<ass>` instruction is executed, the result of the `<bool_expr>` is stored on each `<id>` of the `<list_of_id>`, where `<id>` must be used as the key of an entry of a global symbol table, and the associated value is the result of `<bool_expr>`. **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.**
- `<MULTIIF>`: It is the word "`MULTIIF`", followed by `<exp1>`, by the word "`DIV`", by `<exp2>`, by the word "`DIV`", by `<exp3>`, and by a ";". `<exp1>` is the word "`EXP1`" followed by a `<bool_expr1>`, and `<exp2>` is the word "`EXP2`" followed by a `<bool_expr2>`. Instead, `<exp3>` is the word "`EXP3`" followed by a `<bool_expr3>`, a `<true_stmt>`, the word "`ELSE`", a `<false_stmt>`, and a ";". Both `<true_stmt>` and `<false_stmt>` are a "{" followed by

a <print_list>, and a "}". The <print_list> is a not empty list of <print>, where each <print> is the word "print" followed by a *quoted string* and terminated with a ";". The *quoted strings* associated to the <print> instructions of the <true_stmt> are printed if <bool_expr1>, <bool_expr2>, and <bool_expr3> are all equal to true or all equal to false, otherwise the quoted strings associated to the <print> instructions of the <false_stmt> are printed. <bool_expr1>, <bool_expr2>, and <bool_expr3> have exactly the same grammar and semantic of <bool_expr>.

<bool_expr> is a boolean expression and can contain the following logical operators: and, or, not, and round brackets. Operands can be true, false, and an <id>. The current value associated with an <id> (i.e., true or false) can be obtained from the symbol table.

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:

```
Y:!??!!?!abc ;                  [( tok2 )]
X:-3.4 ;                        [( tok1 )]
Y:!!!!!hello100;                [( tok2 )]
Z:2ab!128Ad?12345?ABCDF;        [( tok2 )]

+++ [( division between header and code sections )]

[( Assignment operations )]
a, b = true or false and not false;  [( a=b=true or false=true )]
c = a and false;                     [( c=false )]

[( MULTIIF instruction )]
MULTIIF EXP1 c and false        [( false )]
    DIV
    EXP2 not true and false     [( false )]
    DIV
    EXP3 c and false or false   [( false )]
    { print "One";              [( executed )]
      print "Two";              [( executed )]
    }
    ELSE {
      print "Four";
      print "Five";
    }
;
d, e = true;                     [( d=e=true )]
f = false and ( true or true );  [( f=false and true=false )]
```

Output:

```
a=true
b=true
c=false
"One"
"Two"
d=true
e=true
f=false
```

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