Formal Languages and Compilers

22 January 2018

Using the JFLEX lexer generator and the CUP parser generator, realize a JAVA program capable of recognizing and executing the programming language described later.

Input language

The input file is composed of two sections: *header* and *program*, separated by means of the token "$$$". Semantic actions are required only in the last section. The input file can contain C stile **comments** (i.e., /* <comment> */).

The *header* section can contain 3 types of tokens terminated with the character ";":

- <token1>: starts with a word composed of **at least 6** characters in the set "*", "#" or "&", which can be disposed in **any order** and in **even** number (e.g., *##*#&, ****&#&##&). This first word is followed by another word composed of letters and numbers (the **last two characters** of the word are **letters**). At the end of this token, can be **optionally** present an **even number** between -24 and 1258.
- <token2>: is composed of **4, 7 or 19 exadecimal numbers**. Each exadecimal number is composed of **3 or 5** characters. The exadecimal numbers are separated by the character **"-" or ":"**.
- <token3>: is a hour with the format "**HH:MM**" (**between 09:31 and 17:46**) or in the 12 hours format with the words "**am**" or "**pm**" to indicate the morning or the afternoon (between the same range, i.e., between 09:31am and 05:46pm).

Header section: grammar

In the *header* section, <token2> and <token3> can appear in **any order and number** (even **0 times**). Instead, <token1> must appear **exactly 2 times**.

Program section: grammar and semantic

The *program* section is composed of a list eventually **empty** or with an **even** number, **at least 4**, of <instructions>. Each <instruction> is terminated with the character ";".

The programming language defines the following three types of <instructions>:

- *Assignment*: an <id> (i.e., a word that begins with a letter or the character "_" and followed by letters, numbers and characters "_"), an "=" and an <expression>. This instruction associates the value <expression> to <id>, which is the key of a symbols table. **This symbols table is the only global variable allowed in all the examination.**
- Function `FZ()`: is the name "FZ", followed by a "(", a <operation>, a ",", a <list_of_points> and a ")". <operation> can be the word "PATH" or the word "MAX". <list_of_points> is a list of elements separated by commas ",", where each element (point) is composed of a "[", an <expression> that represents the x coordinate, a ",", an <expression> that represents the y coordinate and a "]". If <operation> is equal to "PATH", the function "FZ" returns the total length of the segments connecting the points listed as argument of the function FZ. Otherwise, if <operation> is equal to "MAX", FZ returns the length of the longest segment between all the segments connecting the points listed as argument of the function FZ. Both operations (PATH and MAX) analyze the points in the same order as they are listed in the function. To compute the distance between points [x1, y1] and [x2, y2], the following equation has to be used:

  $$result = \sqrt{(y2 - y1)^2 + (x2 - x1)^2}$$

  To perform the square root of a double number `x` use the *java* function `double java.lang.Math.sqrt(double x)`.
- `IF`: is the word "IF", followed by an <expression> (exp_1), followed by a non-empty list of <intervals> separated with ",". Each <interval> is composed of the word "IN", the word "RANGE", a <range>, the word "PRINT" and a **quoted string**. A <range> is a "[", an <expression> (exp_s), a ",", an <expression> (exp_e) and a "]". The `IF` instruction, for each <interval>, if $exp_s \leq exp_1 \leq exp_e$ prints the quoted string.

An <expression> can be composed of <numbers> (only **real numbers**), <id> whose numeric associated values can be accessed through the symbols table, and returned values of the function `FZ()`. Only the mathematical symbols "+", "-", "*", "/", "(" and ")" can be used to performs operations in an <expression>, with their usual meaning.

Goals

The translator must execute the programming languages of the last section. Inherited attributes must be used for each <interval> of the `IF` instruction to access the value exp_1 and to check if the value exp_1 is in the interval between exp_s and exp_e, and consequently print the associated quoted string. **Solutions that do not use inherited attributes to this extent will not be accepted.**

Example

Input:

```
/* Header section */

#&****12home;               /* token1 */
10:40am;                    /* token3 */
#&#&**##1234ab1258;         /* token1 */
12A-12abc-123:456;          /* token2 */

$$$
/* Program section */

/* Assignments */
c_x1 = 0.0;
c_y1 = 0.0+1.0;       /* c_y1 = 1.0 */
c_x2 = 1.+c_y1*0.0;   /* c_x2 = 1.0 */
c_y2 = 1.0*(2.0-1.0); /* c_y2 = 1.0 */

/* FZ command */
dist = FZ( PATH, [0.0, FZ(MAX, [1.0, 0.0], [0.0, 0.0], [0.0, 2.0]) ], /* [0.0, 2.0] because */
                 /* Distance between [1.0, 0.0], [0.0, 0.0] is 1.0
                    and distance between [0.0, 0.0], [0.0, 2.0] is 2.0 that is the bigest */


           /* Distance first segment between [0.0, 2.0] and [0.0, 0.0] is 2.0 */
           [c_y1*0.0, c_x1+2.5*c_x1],                  /* [0.0, 0.0] */
           /* Distance second segment between [0.0, 0.0] and [1.0, 1.0] is sqrt(2)=1.41421... */
           [c_x2, c_y2] );                             /* [1.0, 1.0] */
           /* Total PATH: dist = 2.0+1.41421 = 3.41421 */

/* IF command */
IF dist+c_x1 IN RANGE [ 5.5  :  10.0 ] PRINT "Range 1",
             IN RANGE [ 3.0  :  5.5  ] PRINT "Range 2",    /* TRUE because 3.0 <= dist <= 5.5 */
             IN RANGE [0.0+c_x1 :3.0 ] PRINT "Range 3" ;
```

Output:

```
"Range 2"
```