Formal Languages and Compilers

24 July 2014

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

- <token1>: it begins with the character "=", followed by an **odd number** (**at least 3**) of uppercase alphabetic letters or by an **even number** (**at least 4**) of lowercase alphabetic letters, then in the token is present a "?" character, that is **optionally** followed by **3 or more** repetitions of the words ("xx", "xy", "yx" or "yy") in **any combination**.
- <token2>: it is composed of **3, 5 or 8 exadecimal numbers**. Each exadecimal number is composed of **2 or 4** characters. The exadecimal numbers are separated by the character **"." or ":"**.
- <email>: a word composed of numbers, letters and characters "_" and ".", the character "@", a words composed of numbers and letters, a "." and the word "it", or "com", or "net".

Header section: grammar

The *start* section contains **exactly one** token of type <token2>, **exactly one** token of type <email> and **any number, even 0**, of tokens of type <token1>. Tokens can appear in the *start* section in **any order**.

Program section: grammar and semantic

The *program* section is composed of a list (eventually **empty** or with an **even** number of elements) of <instructions>. Each <instruction> is terminated with the character ";".

The programming language defines the following <instructions>:

- *Variable assignment*: an <identifier> (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 stores the value of <expression> in the <identifier> key of a symbols table. **The symbols table is the only global variable allowed in all the examination.**
- Function `DISTANCE()`: is a function that computes the total distance of a list of points. The function `DISTANCE` is the name "DISTANCE", followed by a "(", a <list_of_points> and a ")". <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 "]". As example, the function `DISTANCE([0, 1], [2, 3], [4, 5])` must return the following value:

  $$returned\_value = \sqrt{[(2-0)^2 + (3-1)^2] + [(4-2)^2 + (5-3)^2]} = \sqrt{16} = 4$$

  To perform the square root of a double number `n` use the *java* function `double Math.sqrt(double n)`.
- `VALUE`: The word "VALUE", followed by an <expression> (exp_1), followed by a non-empty list of <intervals> separated with ",". Each <interval> is composed of the word "IN", a <range>, the word "WRITE" and a **quoted string**. A <range> is a "[", an <expression> (exp_s), a ",", an <expression> (exp_e) and a "]". The `VALUE` 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**), <identifiers> (whose numeric values can be accessed through the symbols table) and the returned values of the function `DISTANCE()`. 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 in each <interval> of the `VALUE` 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:

```
/* Start section */

a1.ab12.AB:1234:123b;       /* token2 */
=ABCDE?xxxyxxxx;            /* token1 */
stefano_123.xyz@polito.it ; /* email */
=abcdef?;                   /* token1 */

##
/* Program section */

/* Variable assignments */
x1 = 0.0;
y1 = 0.0+1.0;       /* y1 = 1.0 */
x2 = 1.;            /* x2 = 1.0 */
y2 = 1.0*(0.5+0.5); /* y2 = 1.0 */

/* DISTANCE command */
dist = DISTANCE( [0.0, DISTANCE([0.0, 0.0], [0.0, 0.0]) ], /* [0.0, 0.0] because... */
                                    /* ...DISTANCE([0.0, 0.0], [0.0, 0.0]) = 0.0 */
                 [x1+2.5*x1, y1],                          /* [0.0, 1.0] */
                 [x2, y2] );                               /* [1.0, 1.0] */
/* The previous command:
   dist = DISTANCE([0.0, 0.0], [0.0, 1.0], [1.0, 1.0])
   stores in the variable dist the distance between the three points
   [0.0, 0.0], [0.0, 1.0] and [1.0, 1.0] that is 2.0 (i.e., dist = 2.0)
*/

/* VALUE command */
VALUE dist+0.0 IN [x1 : 1.5+1.0] WRITE "Near !", /* TRUE because 0.0 <= dist <= 2.5 */
               IN [3.0 : 4.5] WRITE "Middle !",
               IN [4.0 : 7.0] WRITE "Far !" ;
```

Output:

```
"Near !"
```