Formal Languages and Compilers

02 September 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 a token composed of **3 or more** characters "#" (e.g, ###, ####, etc.) or composed of an **even** number (**at least 4**) of characters "*" (e.g, ****, ******, etc.). Semantic actions are required only in the *program* section. The input file can contain C stile **comments** (i.e., /* <comment> */).

The *start* section can contain 2 types of tokens, each terminated with the character ";":

- <hour>: is a hour with the formats HH:MM:SS and values between 10:45:12 and 13:20:10 (correct examples are: 10:45:13, 13:20:10, 12:59:58)
- <code>: starts with a word composed of **at least 5** character in the set "%", "+" or "-", disposed in **any order** and in **odd** number (e.g., +-+%%, ++++++-, ++%%%-++-). This first word is followed by another word composed of letters and numbers (the **last character** of the word is a **letter**). At the end of this token, it is **optionally** present a word composed of an **odd number** between -43 and 123.

Header section: grammar

In the *start* section, the token <code> must appear exactly **two times**, while for <hour> these is not any restriction, i.e., it can apper in any position inside the *start* section and with any cardinality, even zero (**Manage this requirement with grammar**).

Program section: grammar and semantic

The *program* section is composed of a list (**empty** or with an **odd** number of elements) of <commands>. Each <command> is terminated with the character ";".

The programming language defines the following <commands>:

- *Point definition*: a <point_name> (i.e., a word that begins with a letter or the character "_" and followed by letters, numbers and characters "_"), an "=", the character "[" a <list_of_point_attributes> and the character "]". <list_of_point_attributes> is a list (possibly **empty**) of <point_attributes> separated by commas ",". Each <point_attribute> is composed of an <attribute_name> (the character "x" for the *x coordinate*, "y" for the *y coordinate* and the character *z* for the *z coordinate*), the character ":" and an <attribute_value> (a **real** number). If an <attribute_name> is not present in the <list_of_point_attributes>, assign to it a default value equal to 0.0. This instruction stores points values into a global structure. **The global structure is the only global variable allowed in all the examination.**
- `WHEN`: is the word WHEN followed by a <boolean_expression>, followed by a non-empty <list_of_conditions>. A <condition> is the word IS, followed by a <condition_value> (i.e., the word TRUE or FALSE), followed by a <list_of_print_commands> (one or more <print_command>). A <print_command> is the word PRINT, followed by a **quoted string** and ended with a ";". If <boolean_expression> is true and <condition_value> is TRUE, all the <print_commands> referred to this <condition> are executed (i.e., the quoted strings are printed). Similarly, if <boolean_expression> is false and <condition_value> is FALSE, all the <print_commands> referred to this <condition> are executed. Otherwise, no action is performed. Look the example.

  <boolean_expression> has the same meaning of comparison/boolean expressions of a typical if instruction of the C programming language. <boolean_expression> can contain real numbers or <point_coordinate> (a <point_name>, followed by a ".", followed by "x", "y" or "z", to indicate the *x*, *y* and *z* coordinates of the point, respectively). To access the <attribute_value>, a lookup on the global structure filled by the *point definition* command must be performed. The only required comparison operators are "<" and ">", and the required boolean operators are AND, OR and NOT (look the example).

To execute the `WHEN` command use inherithed attributes to access the value of <boolean_expression> in order to decide if execute or not the <print_commands>. **Solutions that do not use inherited attributes to this extent will not be accepted.**

- `Z_STATS`: is the word Z_STATS, followed by a "(", a non-empty <list_of_points_names> and a ")". <list_of_points_name> is a list of <points_names> separated by commas ",". This function prints the *minimum*, the *maximum* and the *mean* values between the *z* coordinates of the listed <points_names>. Look the example.

Goals

The translator must execute the programming languages of the last section.

Example

Input:

```
/* Start section */
--%%%A1234b11;
12:50:55;
++--%%%aBCd;

******
/* Program section */

p1 = [x:3.0, y : 5.0, z: 40.0];      /* p1.x=3.0   p1.y=5.0   p1.z=40.0 */
p2 = [ z: 30.0, x: .5, y: 2. ];      /* p2.x=0.5   p2.y=2.0   p2.z=30.0 */

Z_STATS(p1, p2);                     /* MIN: 30.0   MAX: 40.0   AVG: 35.0 */

p3 = [ x: 3.0, z: 20.0 ];            /* p3.x=3.0   p3.y=0.0   p3.z=20.0 */

Z_STATS(p1, p2, p3);                 /* MIN: 20.0   MAX: 40.0   AVG: 30.0 */

WHEN p2.z > 35.0 IS FALSE PRINT "NOT HIGH"; ;
/* printed because boolean_expression is FALSE */

WHEN NOT p1.z > 50.0 OR p2.x < 1.0 AND p3.z < 10.0 IS TRUE
                                      PRINT "IS TRUE";
                                      PRINT "printed because boolean_expression=TRUE";
                                  IS FALSE
                                      PRINT "IS FALSE";
                                      PRINT "not printed because boolean_expression=TRUE";
                                  IS TRUE
                                      PRINT "IS TRUE 2";
;

/* NOT FALSE OR TRUE AND FALSE = TRUE OR TRUE AND FALSE = */
/* = TRUE OR FALSE = TRUE -> boolean_expression = TRUE    */
```

Output:

```
MIN: 30.0   MAX: 40.0   AVG: 35.0
MIN: 20.0   MAX: 40.0   AVG: 30.0
"NOT HIGH"
"IS TRUE"
"printed because boolean_expression=TRUE"
"IS TRUE 2"
```