Formal Languages and Compilers – JFlex & Cup

Using the JFLEX lexer generator and the CUP parser generator, realize a Java program capable of recognizing a language that allows to describe the operative itinerary of a vehicle exploring/monitoring a region that can ideally be represented as a bi-dimensional space. The program should parse and interpret a file (specified as the first parameter in the command line) with the description of an itinerary according to such language.

Input Language

The itinerary specification file starts with an header section which contains 3 fields:

1. An alphanumeric strings of 8 characters starting with an uppercase character and representing the vehicle code.
2. The current date according the ***gg/mm/aaaa***
3. A textual description of the itinerary enclosed by <***!*** ….. ***!***>

The vehicle code and the date must be present while the description is optional, they should appear in the file according to the previous order.

The next section is separated by means of ***"%%"*** symbol and contains the specifics for an itinerary. The itinerary language comprises a set of primitives allowing to control the vehicle:

- **Angle(α):** rotates the vehicle until it is oriented along the direction forming an angle of α with the x axis (counter clockwise). α can be specified both in radians and in degrees and can be signed; in radians it is expressed as a fraction or a real number followed by the symbol ***'PI'*** (as ***3/4PI*** or ***-0.5PI)*** while in radians is expressed simply as a real number.
- **Move(k T):** moves the vehicle along the current direction. **T** can represent either a distance of **k** meters (**T** is *'M'* or *'m'*) or a time of **k** seconds (**T** is *'S'* or *'s'*). **k** is a real number and must be positive
- **Speed (k):** sets the speed of the vehicle to **k** meters per second, **k** is a real number and must be positive.
- **AcquireData (k):** stops the vehicle for **k** seconds and acquire data

In the specification file these primitives are separated by means of **';'** symbols and they form a sequence of commands and can come in any order but **Speed(k)** must be specified as the first command and after each **AcquireData** command.

The language allows for the definition of a **for** statement that repeats a certain amount of times a set of commands enclosed by ***'['*** and ***']'*** . The statement is expressed as:

- **for (u)[**

  ……..

  ]

where **u** is an integer positive number greater than 0 and specifies the amount of time the execution of the list of command should be repeated. The set of commands within the for body must begin with a Speed primitive.

The language allows also for the definition of a choice statement that executes a set of commands enclosed by ***'['*** and ***']'*** if they satisfy a given condition. In this language conditions can concerns the

distance from the starting point or the total time elapsed and support the relational operators **'>'** and **'<'**. The statement is expressed as:

- **if (T relOp k) [**

  ……

  ]

where **k** is a real number and **T** can be *'S'* or *'s'* (representing the time elapsed) otherwise **T** can be *'M'* or *'m'* for the distance from starting point of the itinerary.

The language could therefore contain 3 kinds of blocks (in any order and in any number): simple command lists, for blocks and if blocks. Notice that **for** and **if** statements could not be nested (*no **for** or **if** blocks within a **for** or **if** block).*

Program Objective

The program is required to verify if the file is conform to the syntax of the language and if that is true, in addition, to calculate the final position with the respect of the starting point as well as the total of the time elapsed. Global class variables for handling the current speed and angle are allowed.

NB: Notice that the DeltaX and DeltaY relative a to a segment of length L along a direction forming an angle Alpha with the *X* axis can be retrieved as:

deltaX = k * cos(alpha) deltaY = k * sen(alpha)

For this pourpose in java are available the static methods of the class Math: java.Math.sin e java.Math.cos (*and the constant java.Math.PI)* they both takes a double as input parameter and return a double value.

Example

```
AX23FF34
02/11/2006
<! Route B !>
%%
Speed( 3.5 );
Move( 10 M );
for( 4 )[
     Speed( 3);
     Angle(-2/3PI);
     Move(12.5 S)
]
if( s > 3.5)[
     AcquireData( 34 );
     Speed(1);
]
Angle(0);
Move( 3 S);
```

INPUT

OUTPUT

```
Syntax correct
The final position is X = 34.5,
Y = -12.56. The total time
elapsed is 134.5 sec
```