Homework 2: RoboLang Parser

For this homework, you will build upon the ANTLR regular expressions you made in Homework 1 to create a parser.

Step 1. Check out the project

I have created you a Subversion repository space at the following URL:

https://utopia.cs.memphis.edu/course/comp4040-2013fall/uuids/*YOUR_UUID*/

Where you should replace *YOUR_UUID* with your UUID (i.e., your UofM email name; mine is sdflming).

In your space, you will find an Eclipse project named **homework2**. Using Eclipse, checkout this project.

Within the project are several files:

- **RoboLang.g4** – This is where you should create your parser.
- A number of ***X*.robo** files – These are input files to use to test your parser.

You should use ANTLRWorks to edit and test your parser.

Be warned that I may test your parser on different input when I grade it! Feel free to create additional input files.

Step 2. Copy over your regular expressions from hw1

The **RoboLang.g4** file will be empty when you check out the **homework2** project, and you will need to copy your regexes from **hw1** into the file.

Step 3. Add a few more tokens

You must update your scanner to match on the following three tokens.

Token name	**Pattern to match**	**Token name**	**Pattern to match**
DISTANCE	distance	RARROW	->
ALERT	alert	LARROW	<-
WALL	wall		

Step 4. Define the CFG for RoboLang

NOTE: Although I will describe the CFG all at once here, you should figure out a way to incrementally add and test its features. If you try to implement the CFG in one “big bang”, you likely to wind up with a big mess on your hands.

Define the CFG for RoboLang as follows.

Robot Declaration

A robo file contains one and only one robot declaration. A robot declaration has this form:

```
robot robot-identifier ->
    … body of the robot declaration …
<- robot
```

Note that the above is not written in the form of a CFG production. It’s up to you to figure out what the productions should be. I will use italics to signify that appropriate text needs to be filled in.

So, as an example, a robot Bob might be declared like this:

```
robot Bob ->
    … elided stuff …
<- robot
```

Note that whitespace characters are basically ignored by this grammar, except in so far as they are used to separate tokens. Thus, the following would also be a valid way to declare Bob:

```
robot Bob->… elided stuff …<-robot
```

And this would also be valid:

```
robot
Bob
->
… elided stuff …
<-
robot
```

The body of the robot declaration may contain variable declarations or robot-behavior declarations.

Variable Declarations

Variable declarations must come first in the body of a robot declaration, and there may be any number of them. All variables in RoboLang are numbers.

Variable declarations have the following form:

```
var variable-identifer := arithmetic-expression ;
```

The assignment part is optional. If the assignment part is omitted, the variable defaults to 0. Thus, the following are valid variable declarations:

```
robot Bob ->
    var x ;
    var y := 20 ;
    … elided stuff …
<- robot
```

Robot-Behavior Declarations

There are several types of robot-behavior declarations:

- The **main** behavior declaration specifies the robot's default behavior. There must be one and only one main declaration
- There are two types of **alert** behavior declarations, and there may be 0 or 1 of each of them in the body of the robot declaration:
    - The **alert robot** behavior declaration specifies what the robot should do if it scans another robot. Scanning happen automatically as the robot's main behavior executes.
    - The **alert wall** behavior declaration specifies what the robot should do if it runs into a wall.

The robot behavior declarations may come in any order, but they must follow the variable declarations.

Here is an example of what the behavior declarations should look like:

```
robot Bob ->
    var x ;
    var y := 20 ;

    main ->
        … elided stuff …
    <- main

    alert robot ->
        … elided stuff …
    <- alert

    alert wall ->
        … elided stuff …
    <- alert

<- robot
```

Robot-Behavior Bodies

The bodies of robot-behavior declarations look similar to most typical imperative programming languages. Here are some examples.

Assignment Statement:

```
variable-identifier := arithmetic-expression ;
```

While Loop:

```
while conditional-expression do
    … body statements …
od
```

Note that **do** and **od** are matched. Also note that while loops can contain other while loops.

Branching Conditional:

```
if conditional-expression then
    … body statements …
elsif conditional-expression then
    … body statements …
elsif conditional-expression then
    … body statements …
else
    … body statements …
fi
```

Note that the **if** and **fi** are matched. The if part may be followed by any number of **elsif** parts and ended by 0 or 1 **else** parts

Special Commands:

Command	**Form**	**Comments**
ahead	`ahead` *`arithmetic-expression`* `;`	Moves robot ahead by value of expression.
back	`back` *`arithmetic-expression`* `;`	Moves robot backward by value of expression.
right	`right` *`arithmetic-expression`* `;`	Turns robot right by value of expression degrees.
left	`left` *`arithmetic-expression`* `;`	Turns robot left by value of expression degrees.
fire	`fire` *`arithmetic-expression`* `;`	Fires with strength of value of expression. Uses that much energy.
energy	`energy`	Returns the robot’s current energy level. Should used as an identifier in arithmetic expressions.
scan	`scan ;`	Forces the robot to scan.

bearing	`bearing`	Returns the bearing of a scanned robot with respect to current robot. Should be used as an identifier in arithmetic expressions. May only be used in the **alert robot** behavior body.
noop	`noop ;`	Does nothing (“no op”).

Arithmetic Expressions:

Follow C/Java syntax for arithmetic expressions. Allow them to contain numbers or identifiers. Support the following operators: +, -, *, /, (, and).

Conditional Expressions:

The while loop and if/elsif statements take conditional expressions. Follow the C/Java syntax for conditional expressions. Allow them to contain arithmetic expressions (0 evaluates to false and non-0 evaluates to true), and support the following conditional operators: ||, &&, !, (, and).

Special Notes

- You need not detect the use of undeclared identifiers.
- You must make your grammar unambiguous.
- Your grammar must properly handle operator association and precedence.
- Make sure your grammar continues to support the comments from **hw1**.

Examples

See the robo files included with homework2 for some examples of the above syntactic elements.

Step 5. Submit your work

To submit, simply commit your completed **RoboLang.g4** file to the **homework2** project in the SVN repository. Feel free to add/commit your test files as well.