Throughout this course we will develop a project in several stages. The project consists of managing and operating a language to program a factory robot in a two-dimensional world. The robot is able to move in the world (delimited by an $n \times n$ matrix); the robot moves from cell to cell. Cells are indexed by rows and columns. The top left cell is indexed as (1,1). North is top; West is left. The robot interacts (picks and puts down) with two different types of objects (chips and balloons). Additionally, note that the robot cannot move on, or interact with obstacles in the world (gray cells).

Robot Description

In this project, Project 1, we will use JavaCC to build an interpreter for the Robot Language introduced in Project 0.

Figure 1 shows the robot facing North in the top left cell. The robot carries chips and balloons which he can put and pickup. Chips fall to the bottom of the columns. If there are chips already in the column, chips stack on top of each other (there can only be one chip per cell). Balloons float in their cell, there can be more than one balloon in a single cell.

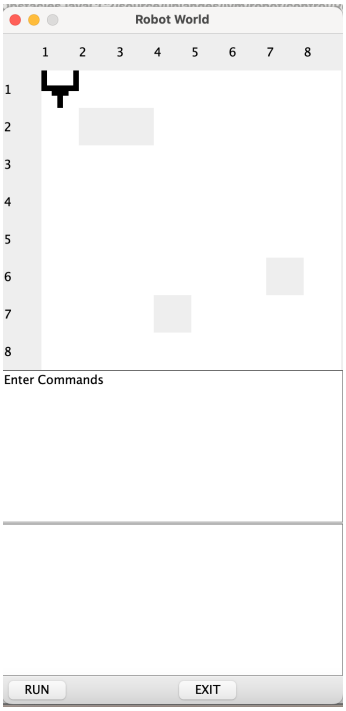

Figure 1: Initial state of the robot's world

The attached Java project includes a simple JavaCC interpreter for the robot.[1] The interpreter reads a sequence of instructions and executes them. An instruction is a command followed by an end of line.

A command can be any one of the following:

- move(n): to move forward n steps
- right(): to turn right
- Put(chips,n): to drop n chips
- Put(balloons,n): to place n balloons
- Pick(chips,n): to pickup n chips
- Pick(balloons,n): to grab n balloons
- Pop(n): to pop n balloons
- Hop(n): To jump n postions forward
- Go(x,y): To go to position x,y

The interpreter controls the robot through the class `uniandes.lym.robot.kernel.RootWorldDec`

Figure 2 shows the robot before executing the commands that appear in the text box area at the bottom of the interface.

Figure 3 shows the robot after executing the aforementioned sequence of commands. The text area in the middle of the figure displays the commands executed by the robot.

Recall the defintion of the language for robot programs.

A program for the robot is simply a secuence of commands and definitions.

- A definition can be a variable defintion or a procedure definition.
  - A variable definition starts with the keyword `defVar` followed by a name followed by an initial value.
  - A procedure definition starts with the keyword `defProc` followed by by a name, followed by a list of parameter in parenthesis separated by commas, followed by a block of commands.

[1] The given interpreter is used for a different robot language, but can be used as a starting point for your own interpreter.

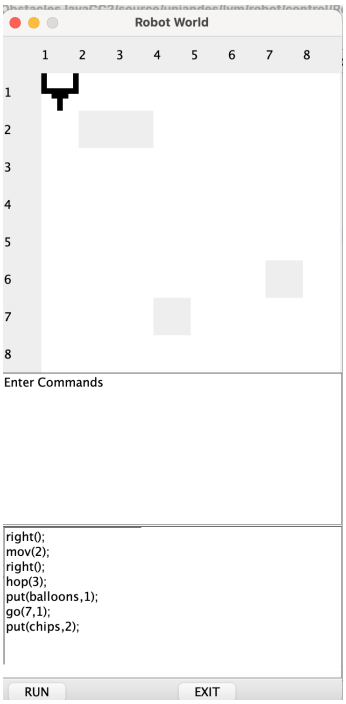

Figure 2: Robot before executing commands

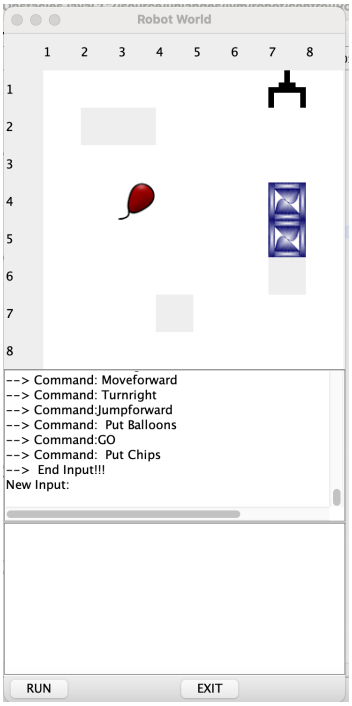

Figure 3: Robot executed commands

- A block of commands is a sequence of commands separated by semicolons within curly brackets .
- A value is a number or a previously defined variable
- Command can be a simple command, a control structure or a procedure call
  - A simple command can be any one of the following:
    * An assignment which is of the form `name = v` where `name` is a variable's name and `n` is a value. The result of this instruction is to assign the value of to the variable.
    * `jump(x,y)` – where `x` and `y` are values. The robot should go to position `(x, y)`.
    * **walk**`(v)` – where `v` is a value. The robot should move v steps forward.
    * **walk**`(v, D)` – where `v` is a value. `D` is a direction, either front, right, left, back. The robot should move `n` positions to the front, to the left, the right or back and end up facing the same direction as it started.
    * **walk** `(v, O)` – where `v` is a value. `O` is north, south, west, or east. The robot should face `O` and then move `n` steps forward.
    * `leap(v)` – where `v` is a value. The robot should jumo v steps forward.
    * `leap(v, D)` – where `v` is a value. `D` is a direction, either front, right, left, back. The robot should jump `n` positions to the front, to the left, the right or back and end up facing the same direction as it started.
    * `leap (v, O)` – where `v` is a value. `O` is north, south, west, or east. The robot should face `O` and then move `n` steps.
    * `turn(D)` – where `D` can be left, right, or around. The robot should turn 90 degrees in the direction of the parameter.
    * `turnto(O)` – where `O` can be north, south, east or west. The robot should turn so that it ends up facing direction `O`.
    * **drop**`(v)` – where v is a value. The robot should drop v chips.
    * `get(v)` – where v is a value. The robot should pickup v chips.
    * **grab**`(v)` – where v is a value. The robot should pick v balloons.
    * `letGo(v)` – where v is a value. The robot put v balloons.
    * `nop()` The robot does not do anything.
  - a procedure is invoked using the procedure's name followed by followed by its arguments in parenthesis separated by commas.
  - A control structure can be:

Conditional: **if** `condition Block1` **else** `Block2` – Executes `Block1` if `condition` is true and `Block2` if `condition` is false.

Loop: **while** `condition Block` – Executes `Block` while `condition` is true.

RepeatTimes: **repeat** `v` **times** `Block` – Executes `Block` `n` times, where `v` is a value.

- A condition can be:
  - **facing**`(O)` – where `O` is one of: north, south, east, or west
  - **can**`(C)` – where C is a simple command. This condition is true when the simple command can be executed without error.
  - **not**`: cond` – where `cond` is a condition

Spaces, newlines, and tabulators are separators and should be ignored.

The language is not case-sensitive. This is to say, it does not distinguish between upper and lower case letters.

Remember the robot cannot walk over obstacles, and when leaping it cannot land on an obstacle. The robot cannot walk off the boar or land off the board when leaping.

Task 1. The task of this project is to modify the parser defined in the JavaCC file `uniandes.lym.robot.control.Robot.jj` (you must **only** send this file), so that it can interpret the new language described above. You may not modify any files in the other packages, nor `uniandes.lym.robot.control.iterpreter.java`.

Below we show an example of a valid program.

```
%% Example 1


defVar nom 0
defVar x 0
defVar y 0
defVar one 0

defProc putCB (c, b)
{
    drop(c);
    letGo(b);
    walk(nom)
}

defProc goNorth()
{
    while can(walk(1,north)) { walk(1,north)}
}

defProc goWest()
{
    if can(walk(1,west))  { walk(1,west); goWest()} else {nop()}
}


{
jump(3,3);
putCB(2,1)
}
```