

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 0, we will try to understand the robot language. That is, given a program for the robot, we will like to see if this satisfies the language specification and robot behavior, explained in the following.

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.

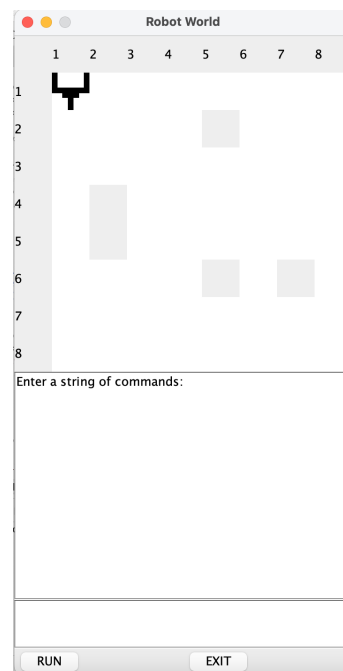


Figure 1: Initial state of the robot's world

The attached Java project includes a simple interpreter for the robot. The interpreter reads a sequence of instructions and executes them. An instruction is a command followed by “;”.

A command can be any one of the following:

- M: to move forward
- R: to turn right
- C: to drop a chip
- B: to place a balloon
- c: to pickup a chip
- b: to grab a balloon
- P: to pop a balloon
- J(n): to jump forward n steps. It may jump over obstacles, but the final position should not have an obstacle.
- G(x,y): to go to position (x,y). Position (x,y) should not have an obstacle.

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.

Below we define a language for programs for the robot.

- A program for the robot is the keyword `ROBOT_R` possibly followed by a declaration of variables, a definition of procedures, and ends with a block of instructions.
 - A declaration of variables is the keyword `VAR` followed by a list of names separated by commas. A name is a string of alphanumeric characters that begins with a letter.
 - Procedure declaration starts with the keyword `PROC` followed by one or more procedure definitions.
 - A procedure definition is the procedure's name followed by “[”, then its parameters, followed by the instructions separated by semicolons (“;”). The parameters are specified by a list of names separated by commas preceded and followed by the symbol “[”]; the definition ends with “]”.
-

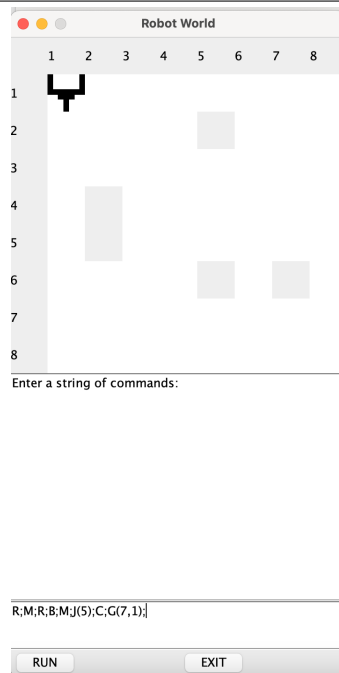


Figure 2: Robot before executing commands

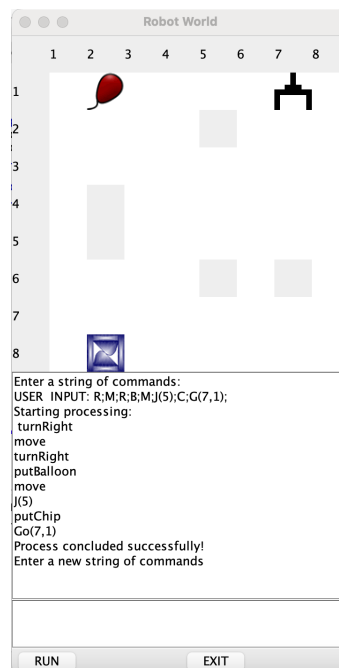


Figure 3: Robot executed commands

-
- A block of instructions is a sequence of instructions separated by semicolons within square brackets `[]`.
 - An instruction can be a command or a control structure or a procedure call
 - A command can be any one of the following:
 - * `assignTo: n , name` where `name` is a variable's name and `n` is a number. The result of this instruction is to assign the value of the number to the variable.
 - * `goto: x, y` – where `x` and `y` are numbers or variables. The robot should go to position `(x, y)`.
 - * `move: n` – where `n` is a number or a variable. The robot should move `n` steps forward.
 - * `turn: D` – where `D` can be left, right, or around. The robot should turn 90 degrees in the direction of the parameter.
 - * `face: O` – where `O` can be north, south, east or west. The robot should turn so that it ends up facing direction `O`.
 - * `put: n , X` – where `X` corresponds to either Balloons or Chips, and `n` is a number or a variable. The Robot should put `n` `X`'s.
 - * `pick: n ,X` – where `X` is Balloons or Chips and `n` is a number or a variable. The robot should pick `n` `X`'s.
 - * `moveToThe: n, D` – where `n` is a number or a variable. `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.
 - * `moveInDir: n, O` – here `n` is a number or a variable. `O` is north, south, west, or east. The robot should face `O` and then move `n` steps.
 - * `jumpToThe: n, D` – where `n` is a number or a variable. `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.
 - * `jumpInDir: n , O` – here `n` is a number or a variable. `O` is north, south, west, or east. The robot should face `O` and then jump `n` steps.
 - * `nop`: The robot does not do anything.
 - a procedure is invoked using the procedure's name followed by `:"` followed by its arguments separated by commas.
 - A control structure can be:
-

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

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

RepeatTimes: `repeat: n Block` – Executes Block `n` times, where `n` is a variable or a number.

– A condition can be:

- * `facing: 0` – where `0` is one of: north, south, east, or west
- * `canPut: n , X` – where `X` can be chips or balloons, and `n` is a number or a variable
- * `canPick: n , X` – where `X` can be chips or balloons, and `n` is a number or a variable
- * `canMoveInDir: n , D` – where `D` is one of: north, south, east, or west
- * `canJumpInDir: n, D` – where `D` is one of: north, south, east, or west
- * `canMoveToThe: n ,0` – where `0` is one of: front, right, left, or back
- * `canJumpToThe: n, 0` – where `0` is one of: front, right, left, or back
- * `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.

Task 1. The task of this project is to use Python or Java to implement a simple yes/no parser. The program should read a text file that contains a program for the robot, and verify whether the syntax is correct.

You must verify that used function names and variable names have been previously defined or in the case of functions, that they are the function's arguments. You must allow recursion.

Spaces and tabulators are separators and should be ignored (outside of instructions).

Below we show an example of a valid program.

```
1 ROBOT_R
2 VARS nom, x, y, one;
3 PROCS
4 putCB [ |c, b| assignTo: 1, one;
5 put : c, chips; put: b , balloons ]

7 goNorth [| |
8 while: canMovetoThe: 1 , north do: [ moveInDir: 1 , north ] od
9 ]

12 goWest [ | | if: canMoveInDir: 1, west then: [MoveInDir: 1 ,
west] else: nop: ]

14 [
15 goTo: 3, 3
16 putcb: 2 ,1
17 ]
```
