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).
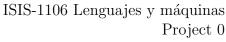
Robot Description

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, explained in the following.

A program for the robot is a sequence of instructions.

- An instruction can be a command, a control structure or a function call.

    - A command can be any one of the following:

        * (defvar name n) where name is a variable's name and n is a number or a constant used initializing the variable.

        * (= name n) where name is a variable's name and n is a number or a constant The result of this instruction is to assign the value of the number n to the variable.

        * (move n): where n is a number or a variable or a constant. The robot should move n steps forward.

        * (skip n): where n is a number or a variable or a constant. The robot should jump n steps forward.

        * (turn D): where D can be :left, :right, or :around (defined as constants). The robot should turn 90 degrees in the direction of the parameter in the first to cases, and 180 in the last case.

        * (face O): where O can be :north, :south, :east, or :west (all constants). The robot should turn so that it ends up facing direction O.

        * (put X n): where X corresponds to either :balloons or :chips, and n is a number or a variable. The Robot should put n X's.

        * (pick X n): where X is either :balloons or :chips, and n is a number or a variable. The robot should pick n X's.

* `(move-dir n D)`: where `n` is a number or a variable. `D` is one of `: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.

* `(run-dirs Ds)`: where `Ds` is a non-empty list of directions: `:front`, `:right`, `:left`, `:back`. The robot should move in the directions indicated by the list and end up facing the same direction as it started.

* `(move-face 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.

* (**null**): a instruction that does not do anything

* These are the constants that can be used:
    · Dim : the dimensions of the board
    · myXpos: the x postition of the robot
    · myYpos: the y position of the robot
    · myChips: number of chips held by the robot
    · myBalloons: number of balloons held by the robot
    · balloonsHere: number of balloons in the robot's cell
    · ChipsHere: number of chips that can be picked
    · Spaces: number of chips that can be dropped

* A control structure can be:

    Conditional: (`if` condition B1 B2): Executes `B1` if condition is true and `B2` if `condition` is false. B1 and B2 can be a single command or a Block

    Repeat: (`loop` condition B): Executes `B` while `condition` is true. B can be a single command or a block.

    RepeatTimes: (`repeat` n B) where `n` is a variable or a number. `B` is executed `n` times. B is a cingle command or a block.

    FunctionDefinition: (`defun` name (Params)Cs) where `name` is the function name, (`Params`) is a list of parameter names for the function (separated by spaces) and `Cs` is a sequence of commands for the function.

* A function call is the function's name followed by parameter values within parenthesis, as in (`funName a1 a2 a3`).

* A condition can be:
    · (**facing**? O) where `O` is one of: `:north`, `:south`, `:east`, or `:west`
    · (**blocked**?) This is true if the Robot cannot move forward.
    · (**can-put**? X n) where `X` can be chips or balloons, and `n` is a value.

· (**can-pick**? X n) where X can be chips or balloons, and n is a value.

· (**can**-move? O) where O is one of: :north, :south, :west, or :east

· (isZero? V) where V is a value.

· (**not** cond) where cond is a condition.

Blocks are sequences of instructions delimited by parenthesis ().

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 print "yes" if the syntax is correct and print "no" otherwise.

You must verify that used variable names have been previously defined and in the case of functions, that they have been previously defined and are called with valid parameter values.

Below we show an example of a valid robot program.

```
1  (defvar rotate 3)


5  (if (can-move? :north ) (move-dir 1 :north) (null))

7  (
8  (if (not (blocked?)) (move 1) (null))
9  (turn :left)
10 )

12 (defvar one 1)

14 (defun foo (c p)
15     (put :chips c)
16     (put :balloons p)
17     (move rotate))
18 (foo 1 3)

20 (defun goend ()
21     (if (not (blocked?))
22     ((move one)
23         (goend))
24     (null)))

26 (defun fill ()
27  (repeat Spaces (if (not (isZero? myChips)) (put :chips 1) ))
28 )

30 (defun pickAllB ()
31  (pick :balloons balloonsHere)
32 )

34 (run-dirs  :left :up :left :down :right)
```