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

This final project will use GOLD to perform syntactic analysis of the ROBOT programs.

Below we define a language for programs for the robot.

A program for the robot is simply a secuence of command blocks 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.
- 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 board or land off the board when leaping.

Task 1. The task for this project is to use GOLD to perform syntactical analysis for the Robot. Specifically, you have to complete the definition of the lexer (a finite state transducer) and the parser (a pushdown automaton). You only have to determine whether a given robot program is syntactically correct: you do not have to interpret the code.

Attached you will find a ZIP folder with the following files:

1. `LexerParserRobot202320.gold`: the main file (the one you must run to test your program via console).
2. `Lexer202320.gold`: the lexer
3. `ParserRobot202320.gold`: the parser

The lexer reads the input and generates a token stream. The parser only accepts a couple of commands.

- `walk(n)` where n is a number
- `turnto(north)`
- `walk(id, right)` where id is an identifier

You have to modify `Lexer202320.gold` so it generates tokens for the whole language. You only have to modify procedure `initialize`. You also have to modify `ParserRobot202320.gold` so you recognize the whole language.

Important: for this project, we asume that the language is case sensitive. This is to say it does distinguish between lower case and upper case letters. For example, `tuRnTo` will be interpreted as an idetifier, not as the keyword `turnto`.

You do not have to verify whether or not variables and functions have been defined before they are used.