CS 21 Machine Problem 1: Peg Solitaire

Department of Computer Science
College of Engineering
University of the Philippines - Diliman

1 Problem Statement

Peg solitaire (or Solo Noble) is a board game for one player involving movement of pegs on a board with holes. Some sets use marbles in a board with indentations. The game is known simply as Solitaire in the United Kingdom where the card games are called Patience. It is also referred to as Brainvita (especially in India).

In the traditional game, you have a square board with a cross-like pattern of evenly-spaced holes, with pegs positioned in the holes, except for the center hole. Refer to the illustration below, which shows an English-style traditional board:

```
    o o o
    o o o
o o o o o o o
o o o . o o o
o o o o o o o
    o o o
    o o o
```

In the diagram above, the `o` dots are pegs positioned at holes, and the `.` corresponds to an empty hole. A valid move is to jump a peg orthogonally over an adjacent peg into a hole two positions away and then to remove the jumped peg.

For example, say we move the peg that is two slots directly north of the empty hole. The board configuration after that move looks like this:

```
    o o o
    o . o
o o o . o o o
o o o o o o o
o o o o o o o
    o o o
    o o o
```

The goal of the game is to perform a series of moves such that there is only one remaining peg at the end of the game.

Given a partially completed peg game configuration, your task is to write an assembly program that can determine if the winning condition can be reached .

2 Input

The input to the program will consist of 7 strings, each with 7 characters, describing a row of the board. The input will look like:

```
xx...xx
xx...xx
.......
...o...
.......
xx...xx
xx...xx
```

The `x`'s correspond to inaccessible portions of the board. The `o`'s correspond to pegs in holes. `.`'s correspond to empty holes.

The input format may vary depending on the partial/full implementation you would choose to implement. Refer to the **Scoring** section for different output formats.

A time limit of **10 seconds** will be observed for each test case. Each test case will have a **maximum of 8 pegs**.

Input will be taken via the MARS console.

3 Output

The output of your program will vary, based on the partial/full implementation you would choose to implement. Refer to the **Scoring** section for different output formats.

4 Scoring

Scoring will be based on partial or full implementations. The perfect score for this machine problem is **100 points**.

4.1 A: Check if a solution exists

For this partial implementation, the initial configuration of the board is taken as input, and the output is either **YES**, if there exists a series of moves that leads to a winning condition (one peg left), or **NO**, if winning is not possible. The series of moves can have zero (0) or more moves. Output will be via the MARS console.

Here is a sample initial configuration, with a series of steps that will lead to a winning condition:

```
xx...xx
xxo..xx
..o....
..oo...
.......
xx...xx
xx...xx

---solution---

xx...xx
xxo..xx
..o....
.o.....
.......
xx...xx
xx...xx
xx...xx
xx...xx
.......
.oo....
.......
xx...xx
xx...xx

xx...xx
xx...xx
.......
...o...
.......
xx...xx
xx...xx
```

4.1.1 Output

`YES` or `NO`

This implementation will earn you **40 points**.

4.2 B: Check if a solution exists, where the final peg is at a certain position

For this implementation, the input format will change slightly. Two new symbols will be added, namely `E` and `O` (capital O). One of these two symbols will appear in the input. `E` and `O` symbolize the hole where the last peg needs to end up, where `E` denotes an initially empty hole, and `O` (capital O) denotes an initially occupied hole.

Instead of just checking if a winning condition can be reached through a series of valid moves, the final peg must end up in a specific hole.

Here is a sample initial configuration, with an `O` symbol, and a series of steps that will lead to a winning condition:

```
xx...xx
xxo..xx
..o....
..oO...
.......
xx...xx
xx...xx

---solution---

xx...xx
xxo..xx
..o....
.o.....
.......
xx...xx
xx...xx

xx...xx
xx...xx
.......
.oo....
.......
xx...xx
xx...xx

xx...xx
xx...xx
.......
...o...
.......
xx...xx
xx...xx
```

Here is another sample, this time with an `E`:

```
...xE.o
...oo..
...oo..
.......
.......
.......
.......

---solution---

...xE.o
.....o.
...oo..
.......
.......
.......
.......

...xE.o
.....o.
.....o.
.......
.......
.......
.......

...xEoo
.......
.......
.......
.......
.......
.......

...xo..
.......
```

```
.......
.......
.......
.......
.......
```

4.2.1 Output

`YES` or `NO`

If a winning condition can be reached (final peg ending up in the `E` or `O` position), output **YES**, otherwise output **NO**. Output will be via the MARS console.

This implementation will earn you **65 points**.

4.3 C: Check if a solution exists, where the final peg is at a certain position, and print steps

This implementation takes the same input format as Implementation B. If a valid solution exists, the program must print **YES**, and in addition, print the series of peg moves needed to reach the winning condition. The series of moves can have zero (0) or more moves. If there is no solution, print **NO**.

The solution may not be unique. Any series of moves leading to a winning condition will be marked correct.

Hole positions in the board are denoted using a `row, column` notation. The topmost row of the board configuration is row 1. The leftmost column of the board configuration is column 1.

Refer to the sample output below for exact output format.

```
xx...xx
xxo..xx
..o....
..oO...
.......
xx...xx
xx...xx

---solution---

xx...xx
xxo..xx
..o....
.o.....
.......
xx...xx
xx...xx

xx...xx
xx...xx
.......
.oo....
.......
xx...xx
xx...xx
xx...xx
xx...xx
.......
...o...
.......
xx...xx
xx...xx
```

4.3.1 Output

```
YES
4,4->4,2
2,3->4,3
4,2->4,4
```

or

```
NO
```

Output will be via the MARS console.
This implementation will earn you **90 points**.

5 Implementation

Your assembly programs must follow MIPS conventions and our own CS21 conventions. These include:

1. header comments including your name and section
2. 4-column format for the source code
3. proper initialization and cleanup of functions (`lw` and `sw`, `$sp` decrement/increment, storing `$ra`)
4. proper usage of registers for passing arguments and return values
5. proper usage of preserved and unpreserved registers for function calls
6. using `$gp` for global variables

You may opt to implement auxiliary functions. Indicate the presence of these auxiliary functions in the documentation.

6 Documentation

The documentation is worth **10 points**, and is necessary to earn any implementation points (no documentation, no implementation points). The assembly source code must also be properly documented using comments; this will count towards the documentation points. Note that no points will be given for documentation that does not come with a functional program (documentation-only submissions are banned).

The documentation shall contain a description of the implementation chosen, and a short narrative describing the implementation approach. The documentation should also contain relevant code snippets and short narratives explaining the purpose of each code snippet in the execution of the program. This includes input/output code, function definitions, usage of registers, and parts of the algorithm proper.

7 Submission and Testing

Submission of the documentation and `.asm` source code is via UVLE. Name your assembly source code as **`cs21mp1letter.asm`**, where `letter` denotes the implementation that you chose (`A`, `B`, or `C`). Refer to the UVLe submission bin for the deadline. Please submit your implementation in the corresponding submission bin.

Sample test cases for the different implementations will be provided on UVLe along with this specification. Upon submission of the MP, your programs will be tested on the sample test cases and hidden test cases. The test cases will follow the limits given in the specifications. Your programs will be checked in an **all-or-nothing** manner; please test your program/s diligently.

All forms of academic dishonesty will not be tolerated. We will not hesitate to file necessary academic dishonesty charges against students that would be involved in the copying of another person's code.