# Assignment 2: Ada Programming (25%)

Choose one of the following two tasks:

## 1. MAZE SEARCHING

A maze is an area surrounded by walls; in between you have a path from starting position to ending position. Maze searching is a classic data-structures type problem, which can be solved using elements such as stacks, and backtracking, or recursion. A maze can be represented as a 2D array of characters, each representing a cell. For example:

```
* * * * * * * * * * * * *
* o . . . . . . . . . . *
* . * . * * . * . * . * *
* * . . . * * . . * . * *
* . * . * . * * . * . * *
* * . * . * . * . * * * *
* . * * * * * . . . . * *
* . * * . . * . * . . * *
* * . * * * . * . . * * *
* . . . . . * * * * * * *
* . * . * . . . . . e * *
* * * * * * * * * * * * *
```

with the following character representations:

| | | |
|---|---|---|
| walls | * | (asterisk) |
| open space | . | (period) |
| start | o | (lowercase ‘O’) |
| finish | e | (lowercase ‘E’) |

The maze is surrounds by walls, and as it is traversed, the periods are replaced with o’s. Note that the maze above is shown with spaces in between for clarity. So the input format would be:

```
12 12
************
*o.........*
*.*.**.*.*.*
**...**..*.*
*.*.*.**.*.*
**.*.*.*.***
*.*****....*
*.**..*.*..*
```

```
**.***.*..**
*.....******
*.*.*.....e*
************
```

You may choose your own maze configuration, as long as it is clear (i.e. you may make a more complex one, or a more simple one using 1's and 0's... as long as it works!). Each cell in the maze is either open, or blocked by an internal wall. From any open cell, you may move left, right, up, or down to an adjacent empty cell. To solve a maze, you must find a path of open cells from a given start cell, '**o**', to a specified end cell, '**e**'.

The problem is to choose the path. If we find any dead-end before ending point, we have to backtrack and change the direction. The direction for traversing be North, East, West and South. We have to continue "move and backtrack" until we reach the ending point. Now from the starting position, the algorithm moves to the next free path. If a dead-end is encountered, the algorithm back-tracks and makes the cells in the path ones (wall). This continues until the end point is reached. There are two possible non-recursive approaches to solving the maze:

- a *depth-first* approach using a stack
- a *breadth-first* approach using a queue

For example, a solution using a stack might use an algorithm of the form:

```
create an empty stack named S;
push start onto S;
while S is not empty do
    current_cell ← pop from S;
    if current_cell is the finish-point then
        output "Maze traversed ok";
        S ← empty stack;
    else if current_cell is not a wall and
            current_cell is not marked as visited then
        mark current_cell as visited;
        push the cell to the east onto S;
        push the cell to the west onto S;
        push the cell to the north onto S;
        push the cell to the south onto S;
    end if
end while
```

## TASK

Design an Ada program for traversing the maze using either a stack or queue in the form of an Ada package. Test your program on a series of mazes, including easy and non-trivial.

## 2. SUDOKU

A Sudoku puzzle is a special kind of Latin square popularized by the Japanese puzzle company Nikoli in 1986, with Sudoku loosely translated to *single number*. Latin squares, so named by the 18th century mathematician Leonard Euler (1707-1783), are $n \times n$ matrices that are filled with $n$ symbols in such a way that the same symbol never appears twice in the same row or column. A standard Sudoku is a 9 × 9 grid with the additional property of having nine blocks (or subgrids) of 3 × 3 cells which contain the digits 1 to 9, with no repeat values allowed. This popular puzzle game demonstrates man versus machine, incorporating aspects of backtracking and recursion.

## TASK

Design and implement an Ada program to solve Sudoku puzzles. You may use any technique you wish: back-tracking, recursion, concurrency, stacks, queues, lists, or add some human problem solving techniques. Test your puzzle with real Sudoku solutions. Input should be in the form of a 9x9 Sudoku matrix stored in an ASCII file. You may use an existing algorithm, but make sure to reference any work you use.

**References:**

- Crook, J.F., "A pencil-and-paper algorithm for solving Sudoku puzzles", *Notices of the AMS*, Vol.56(4), pp.460-468 (2009).
- http://en.wikipedia.org/wiki/Sudoku_algorithms

See if your solution can solve this puzzle, said to be amongst the hardest[1].

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 5 | 3 | | | | | |
| 8 | | | | | | | 2 | |
| | 7 | | | 1 | | 5 | | |
| 4 | | | | | 5 | 3 | | |
| | 1 | | | 7 | | | | 6 |
| | | 3 | 2 | | | | 8 | |
| | 6 | | 5 | | | | | 9 |
| | | 4 | | | | | 3 | |
| | | | | | 9 | 7 | | |

[1] http://www.dailymail.co.uk/news/article-1304222/It-took-months-create-long-crack–worlds-hardest-Sudoku.html

## SUBMISSIONS

Your submission should consist of the following items:

- A small design document (2-4 pages) detailing your algorithm, and rationalizing your decisions. What structures did you select? Include some form of flow-chart to visually depict your algorithm.
- The code (well documented and styled appropriately of course).