Lab 4 Graded - CIT315

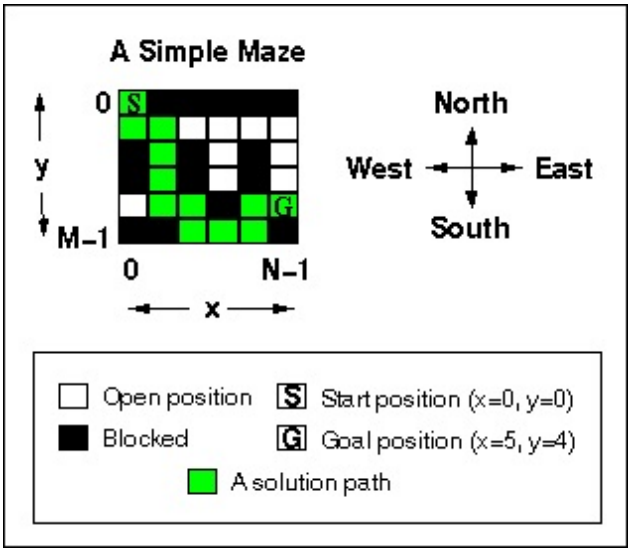
Goals

The primary goal of this project is to write, test and complete an *ANSIC* program which uses *functions*, *pointers* and *recursion*. You will use multiple recursion to solve a **maze puzzle** which is a common use for everyday tasks and the basis of many AI problems.

Maze

A robot is asked to navigate a maze. It is placed at a certain position (the starting position) in the maze and is asked to try to reach another position (the goal position). Positions in the maze will either be open or blocked with an obstacle. Positions are identified by (x,y) coordinates.

Diagram



Example of a simple maze: At any given moment, the robot can only move 1 step in one of 4 directions. Valid moves are:

- Go North: (x,y) to (x,y-1)
- Go East: (x,y) to (x+1,y)
- Go South: (x,y) to (x,y+1)
- Go West: (x,y) to (x-1,y)

Note that positions are specified in zero-based coordinates (i.e., 0...size-1, where size is the size of the maze in the corresponding dimension).

The robot can only move to positions without obstacles and must stay within the maze.

The robot should search for a path from the starting position to the goal position (a solution path) until it finds one or until it exhausts all possibilities. In addition, it should mark the path it finds (if any) in the maze.

Representation

To make this problem more concrete, let's consider a maze represented by a matrix of characters. An example 6x6 input maze is:

```
S#####
.....#
#.####
#.####
...#.G
##...#

'.'-  where the robot can move (open positions)
'#'-  obstacles (blocked positions)
'S'-  start position (here, x=0, y=0)
'G'-  goal (here, x=5, y=4)

A path in the maze can be marked by the '+' symbol...

A path refers to either a partial path, marked while the robot is still searching:

+#####
++++.#
#.####
#.####
...#.G
##...#

(i.e., one that may or may not lead to a solution). Or, a solution path:

S#####
++...#
#+####
#+####
.++#+G
##+++#

which leads from start to goal.
```

Specifications

1. Construct the array exactly as it is written in the specifications above
2. You must use mutual recursion to navigate the maze
3. After each move you must test for the goal state, to determine if you are in the *correct end point*
4. If you are in the end point, then the program ends, otherwise keep going

You must have a function called **mazeGo**, where the **general form** is the following:

```
??? mazeGo( ???   ???)
{
   if(success}
exit
 else
 {
mazeGo(east)
mazeGo(west)
mazeGo(south)
mazeGo(north)
}
}
```

Hints

- Use a two dimensional array to create the model of the whole problem
- Use mutual recursion to move around in the space and find the path
- If needed, pass the array by reference from one function to another.

Submission

THIS LAB IS GRADED!! Submit the project file(s) by the required date and time on Brightspace.

Rubric - 25 points total

- 5 points - Correct use of 2-dim array for the maze model.
- 10 points - Use of mutual recursion in the correct format to solve the problem.
- 5 points - Ability to navigate the maze to a successful goal.
- 2 points - Documentation project title and short description at the top.
- 2 points - Inline documentation describing the input, processing and output parts of the program.
- 1 points - Your name, email and lab time in comments at the top.