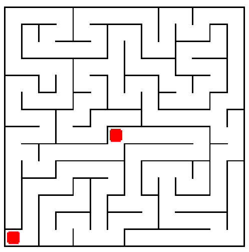
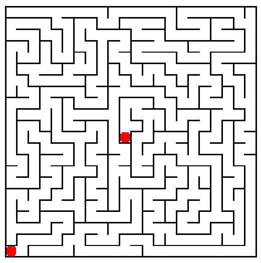
Exercise 1: Perfect maze generation

Write a program that takes a command-line argument n, and generates a random n-by-n perfect maze. A maze is *perfect* if it has exactly one path between every pair of points in the maze, i.e., no inaccessible locations, no cycles, and no open spaces.

Here's a nice algorithm to generate such mazes. Consider an n-by-n grid of cells, each of which initially has a wall between it and its four neighboring cells. For each cell (x, y), maintain a variable `north[x][y]` that is **`true`** if there is wall separating (x, y) and $(x, y + 1)$. We have analogous variables `east[x][y]`, `south[x][y]`, and `west[x][y]` for the corresponding walls. Note that if there is a wall to the north of (x, y) then `north[x][y] = south[x][y+1] =` **`true`**. Construct the maze by knocking down some of the walls as follows:

1. Start at the lower level cell $(1, 1)$.
2. Find a neighbor at random that you haven't yet been to.
3. If you find one, move there, knocking down the wall. If you don't find one, go back to the previous cell.
4. Repeat steps 2 and 3 until you've been to every cell in the grid.

Hint: maintain an $(n + 2)$-by-$(n + 2)$ grid of cells to avoid tedious special cases.

Exercise 2: Getting out of the maze

Given an n-by-n maze (like the one created in the previous lab), write a program to find a path from the start cell $(1, 1)$ to the finish cell (n, n), if it exists. To find a solution to the maze, run the following algorithm, starting from $(1, 1)$ and stopping if we reach cell (n, n).

```
explore(x, y)
-------------
  - Mark the current cell (x, y) as "visited."
  - If no wall to north and unvisited, then explore(x, y+1).
  - If no wall to east and  unvisited, then explore(x+1, y).
  - If no wall to south and unvisited, then explore(x, y-1).
  - If no wall to west and  unvisited, then explore(x-1, y).
```

Exercise 3: Word ladders

Write a program that takes two 5 letter strings from the command line, and reads in a list of 5 letter words from standard input, and prints out a word ladder connecting the two strings (if it exists). Two words can be connected in a word ladder chain if they differ in exactly one letter. As an example, the following word ladder connects `green` and `brown`.

```
green greet great groat groan grown brown
```

A list of 5 letter words can be found in the file `words5.txt`.

Exercise 4: Nine-letter word

Find a nine-letter English word such that remains an English word after successively removing each of its letters (in an appropriate order). Build a digraph with words and vertices and an edge from one word to another if it can be formed by adding one letter. A possible solution is:
`startling` → `starting` → `staring` → `string` → `sting` → `sing` → `sin` → `in` → `i`.

A list of $20,068$ English words can be found in the file `words.txt`.