Introduction:
One of the most popular tools for travelers planning a trip is MapQuest, an Internet site which allows for route planning. An important part of the software for this site is finding feasible paths from a source to a destination, if one exists. We can think of path-finding in a maze in a similar light. There is a source, a destination, and (hopefully) at least one path in between them.

In this assignment, you will read in a maze from a text file. The maze will contain a start point, an end point, blockages, and grid points which can be used to form a path. You will write a series of Java classes which traverse the maze from the start point to find a path to the end point, if one exists.

Input Maze

The test input maze appears as follows:

```
BBBB....BBB
.........BBB
BBBBBB.BBB
BB.BB...BB
...BB.B.BB
BB.BB.B...
BB.BB.B.E.
B.....BS..
BBBBBB.BBB
BBBBBB.BBB
```

The goal of your maze program is to locate a path from the start point S to end point E using available grid points (.). Grid points labeled with a B are blocked. These points can't be used to form a path. Note that the input maze above is a sample. Your program is expected to work with any maze of square size.
**Task Overview**

In this assignment you will perform the following specific tasks.

1. Read in the maze using the constructor method in the Maze method that we provide. Note that this class handles mazes of any size as long as they are square (e.g. height equals width).
2. Print out the maze using the provided method.
3. Locate the start point in the maze (labeled with S)
4. Starting at S, use a stack made from a linked list to iteratively search for feasible paths from S to E to determine if a path exists. Intermediate paths (the partial path found so far) should be stored on the stack. Once the end point E is found, the path from S to E should be stored on the stack in reverse order (e.g. E is on the top of the stack).
5. After a successful path search, replace the character (E, S, or .) at each grid point along the found path with a *.
6. Print out the entire maze, including the new characters for found path.
7. If no path is possible between S and E, print out a message indicating that no path is possible rather than performing steps 5 and 6.

**Getting started**

The best way to complete any programming assignment (including this one) is to take tasks step-by-step. The first decision you need to make is the number of classes you will write. For this assignment, you should have at least four (preferable more) classes, including the Maze and GridPoint classes, which are provided. Please do not place all the methods for your entire project in one class. At a minimum, consider creating a Stack and a driver class which will call objects of other classes (e.g. Maze).

**Required output**

Please provide the following output from your program:

1. Print out the initial maze
2. If a path from S to E exists, replace the characters along the path with a * and print out the revised maze.
3. If a path does not exist, print out an error message indicating that there is no feasible path

Hints and suggestions

The key to this project is determining how to use the stack to keep track of the intermediate path from start to end points. Convince yourself how to do this before you start coding. Consider making a Coordinate class whose objects can be pushed on the stack. A Coordinate for a grid position should only be pushed on the stack once. Ask yourself the following questions before you start coding:

• How do I determine in my code if there is no path between S and E?
• How can I ensure that the final stack only contains Coordinates which are located between S and E?
• When should I pop a value off the stack?

Requirements

You have some flexibility in building your solution to this assignment but there are several requirements:

• You must write your own stack class and your own linked list class. You cannot use the built-in Java library functions for these classes for this assignment.
• Your code must work on a square maze of any size. The mazes used for grading will be different from the ones distributed as samples for this assignment.
• Your program must find a path if one exists and only assign * characters to grid points on the path.
• Your program must successfully determine if a path doesn’t exist.
• From a given grid position, you can only move to the grid position above, below, left, or right. No diagonal moves are allowed.
• Your program does not have to locate the shortest path from S to E.

What to submit:

Please submit all .java files for your project. All code should be well commented.

Reminder: The course honesty policy requires you to write all code yourself, except for the code that we give to you. Your submitted code will be compared with all other submitted code for the
course to identify similarities. Note that our checking program is not confused by changed variable or method names. *We mean it!*

**Grading:**

- Code works (50%)
- Comments (20%)
- Program structure (20%)
- Readability (10%)

**Extra Credit**

We will provide 15 points extra credit for programs that will *always* find the shortest path in terms of number of grid points from S to E, if multiple paths exist.