This exercise is on the application of stack and queue.

We represent a maze using a 2D array of char.

The symbol '.' represents a path, 'X' represents a block, 'S' represents the start-point, and 'E' represents the end-point (destination).

An entry `A[i][j]` is connected to its 4-neghbors, i.e. `A[i][j-1]`, `A[i][j+1]`, `A[i-1][j]`, and `A[i+1][j]`, if they exist.

The `main()` function reads in the maze from a data file.

You are asked to implement the function `findShortestPath()` to determine a shortest path from the start-point to the end-point.

To solve this problem, you can make use of a 2D array of integer to record the distances of other points from the start-point.

Example:

Initially the array `d[][]` is initialized such that the value at the start-point is zero, and all the other entries are set to -1.

X	.	.	X	X	X	.	.
X	X	.	.	.	X	X	.
.	.	.	.	X	X	.	.
.	.	X	.	X	.	X	X
.	X	X	.	S	.	.	X
.	.	X	X	X	.	.	.
.	.	.	X	X	.	X	X
X	X	.	E	.	.	X	X

Input array `A[][]`

-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	0	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

Initial values of the array `d[][]`

Your program will label the neighboring points of the start-point to have a distance equal to 1, if the neighboring point is not blocked.

For the points labelled with 1, labelled their neighboring points to 2 (if the neighboring point is not blocked and it has not been visited before).

-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	1	0	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

After the 1st pass.

-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	2	-1	2	-1	-1
-1	-1	-1	1	0	1	2	-1
-1	-1	-1	-1	-1	2	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

After the 2nd pass.

-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	3	-1	-1	-1	-1
-1	-1	-1	2	-1	2	-1	-1
-1	-1	-1	1	0	1	2	-1
-1	-1	-1	-1	-1	2	3	-1
-1	-1	-1	-1	-1	3	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

After the 3rd pass.

-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	4	-1	-1	-1	-1
-1	-1	4	3	-1	-1	-1	-1
-1	-1	-1	2	-1	2	-1	-1
-1	-1	-1	1	0	1	2	-1
-1	-1	-1	-1	-1	2	3	4
-1	-1	-1	-1	-1	3	-1	-1
-1	-1	-1	-1	-1	4	-1	-1

After the 4th pass.

The labelling process is repeated until you have reached the end-point.

You can use a `queue<Coordinate>` to control the labelling processing.

If a valid path can be found from the start-point to the end-point, the path is recorded in a `stack<Coordinate>`, with the coordinate of start-point at the top and the coordinate of the end-point at the bottom of the stack.