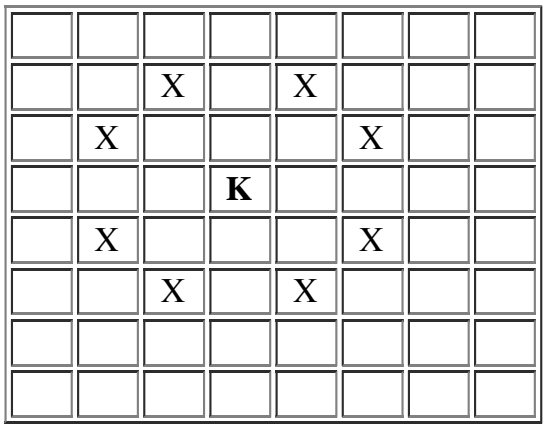
EECS 268: Spring 2016

Laboratory 7

Due: This lab is due before your next lab begins.

You studied the 8 Queens problem in the text. Another classical chess-themed problem to solve that can be attacked with recursive backtracking is the Knight's Tour. In one move, a knight can travel two columns to the left or right, and then one row up or down; or it can move one column to the left or right followed by two rows up or down. For example, if the knight is currently in the square marked "**K**", it can move to any one of the eight squares marked with an "X" in one move. Obviously if the knight is near one of the edges of the board, it will have fewer than eight choices because it needs to remain on the board!



While chess boards are 8x8 as shown above, the Knight's Tour problem is defined for boards of any dimension – even rectangular ones. The Knight's Tour problem can be stated as follows. Given a rectangular board of size *m* x *n* and an initial knight position (*row*, *col*), find a series of valid moves that result in the knight landing exactly once on each square.

Theoretical analysis of this problem includes the result that – for an *m* x *n* board with the smaller dimension being at least 5 – there is always a solution. In fact there are very large numbers of solutions for boards of that size and larger.

The Assignment

You are to implement a recursive backtracking solution to the Knight's Tour problem. You are to get the four parameters *m*, *n*, *initial-knight-row*, *initial-knight-col* from the `argv` commands. (Use `atoi` to convert `argv[i]` into an integer.) Make sure the board size is at least 1x1, and make sure the initial position of the knight is within the board. You can either dynamically allocate the board to the exact size specified, or you may assume the board will never be larger than 8x8.

Try to solve the problem and report whether a solution was found. Then print the final board, indicating the exact sequence of moves taken. Some example outputs follow.

Executing: lab7 6 6 3 5

```
Success!
[ 8][27][16][19][10][25]
[15][ 2][ 9][26][17][20]
[28][ 7][18][ 1][24][11]
[ 3][14][ 5][32][21][ 0]
[ 6][29][34][23][12][31]
[35][ 4][13][30][33][22]
```

Executing: lab7 6 8 3 5

```
Success!
[30][ 7][10][25][28][41][44][23]
[ 9][ 2][29][42][11][24][27][40]
[ 6][31][ 8][ 1][26][43][22][45]
[ 3][18][ 5][12][15][ 0][39][36]
[32][13][16][19][34][37][46][21]
[17][ 4][33][14][47][20][35][38]
```

Executing: lab7 4 4 2 1

```
Failed!
[  ][  ][  ][  ]
[  ][  ][  ][  ]
[  ][ 0][  ][  ]
[  ][  ][  ][  ]
```

Executing: lab7 8 8 0 0

```
Success!
[ 0][ 9][30][63][32][25][52][61]
[11][ 6][27][24][29][62][33][50]
[ 8][ 1][10][31][26][51][60][53]
[ 5][12][ 7][28][23][34][49][40]
[ 2][17][ 4][35][48][39][54][59]
[13][20][15][22][45][56][41][38]
[16][ 3][18][47][36][43][58][55]
[19][14][21][44][57][46][37][42]
```

Warning: the computational complexity of this algorithm can be significant, and actual times depend not only on the board size, but also *very much* on the initial position as well as the order in which your code tries the 8 possible squares to which a knight can move. My implementation produced the immediately preceding result (i.e., "8 8 0 0") in about 2 seconds on cycle2. When I tried "8 8 3 5", however, I wound up just killing the program after it had been running for 1½ hours!

Suggestion: try your code on small boards (maybe 5x5) until you are confident it works. Then try larger boards, non-square boards, and different starting points.

Note also: Since there are multiple solutions for most inputs, your code may produce valid solutions different from the ones shown above. The solution you find will depend on the order in which you try the eight possible moves. The only reliable way to see if a tour is correct is to visually verify that the solution you generated is (or is not) correct.

Grading Criteria

Grades will be assigned according to the following criteria:

- Program's use of recursion is appropriate and either correctly finds a solution or reports that none exists: 50%
- Appropriate class definitions and implementations: 20%
- Error checking the input: 10%
- Well-formatted output: 10%
- Style, including programming style, paragraphing conventions, and internal documentation: 10%

Submission

Once you have created the tarball with your submission files, email it to your TA. The email subject line must look like "[EECS 268] SubmissionName" as follows:

```
[EECS 268] Lab 07
```

Note that the subject should be exactly like the line above. Do not leave out any of the spaces, or the bracket characters ("[" and "]"). In the body of your email, include your name and student ID.