**Program 3 Knights Tour**

**Due 4/8/2012 at Midnight**

**Description**

A **knight's tour** is a sequence of moves of a knight on a chessboard such that the knight visits every square once and only once. So given the size of the board and a starting position you are to show how the knight would move to complete its tour. Its starting spot would be labeled with a 1 and the next move would be labeled as 2 and so on. If the n was equal to 8, the final step would be 64 (8x8).

**Starting the program**

Your program will have one argument added to the program executable name. That will be the name of the file that has all the cases. You must error check if the file opens or not and if it doesn’t open, output and error message and exit the program. An example of starting this program would be:

C:\> prog3.exe knightstour.txt

**The Input File**

The input file will be given at the command prompt and will contain the following format. There will be many puzzles inside this file and you will need to process every puzzle. The first number in the file will be the size of the board (n). The next two numbers will be the starting row and starting column respectively. This pattern will repeat until end of file is reach.

Example

5 🡪 5 x 5 board

0 0 🡪start at row 0, col 0

6 🡪 6 x 6 board

3 2 🡪 start at row 3, col 2

5 🡪 5 x 5 board

1 3 🡪start at row 1, col 3

This file has three puzzles for you to solve.

**The Output file:**

The output file name will be generated from the input file’s name. You will remove the extension (not guaranteed to be .txt or to have three characters in it) and then append out.txt to the name. if the filename was knightstour.puz, your output file would be named knightstourout.txt. As you solve the puzzles from the input file, you will output the solution / no solution to the file in the following format.

Case: # = nxn

Start row: y Start column: x

# # # # #

# # # # #

# # # # #

# # # # #

# # # # #

Example:

So if this was the 7th puzzle in the file and n was 5 and you started at 2 4

Case: 7 - 5 X 5

Start row:    2 Start column: 4

  21  12   7   2  19

   6  17  20  13   8

  11  22   3  18   1

  16   5  24   9  14

  23  10  15   4  25

If there is no solution, you will output “No Solutions for this case. Example:

Case: 6 - 4 X 4

Start row:    0 Start column: 0

No solutions for this case.

**Setting up to solve a puzzle:**

For practice, you will need to dynamically allocate a 2d array for each puzzle. Remember to free the array before you allocate the next array. If n was 5, it is recommended that instead of allocating an array of 5x5 that you increase its size by 4 and allocate an array of 9x9. If n was 7 allocate an array of 11x11. Why this is not required, we can pad the array with -1 to prevent the knight from stepping off the board. This method saves us the time of checking the validity of row and column before accessing the array. After you allocate the array, you will need to pad the outer 2 elements to a -1 and all other squares to a 0

Example: n = 5 dynamically allocate a 9x9 array and initialize it as follows:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

Now, because of this padding, you will need to add 2 to both the starting row and column. So row 2 column 4 becomes row 4 column 6. The S in the puzzle shows where the knight will start. If you throw away the board, you will be at row 2, column 4. When you print the solution of, **DO NOT PRINT THE BORDERS.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | **S** | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

**Solving the puzzle**

The knight has 8 possible moves it can do. It can move 2 squares in the horizontal direction and 1 move in the vertical direction or it can move 2 squares in the vertical direction and 1 move in the horizontal direction.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 |  | 2 |  |
| 8 |  |  |  | 3 |
|  |  | X |  |  |
| 7 |  |  |  | 4 |
|  | 6 |  | 5 |  |

**Please do the moves in this order or grading will take forever!**

You will modify the brute force algorithm to try all combinations from a given spot. You may only land on a square that has a zero in it. If it is a zero, move the step into the square and when you leave the square be sure a put a zero back into it, (mark as used and move into solution, mark it unused). Between the marking it used and unused you will make recursive calls to try all possible moves the knight can attempt in the order given above. (border is hidden)

Initial call to the function using starting point

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | **1** | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

Try move 1: Up 2 left 1

Open

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | **2** | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | **1** | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

Move 1, 2, 3, 4, 5 are all invalid but move 6 is open

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | **2** | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | **3** | 0 | **1** | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

Move 1 is open

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | **2** | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | **3** | 0 | **1** | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 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 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | 0 | **4** | 0 | **2** | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | **3** | 0 | **1** | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

Etc ( no tours with 4 in that location)

Etc

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | 21 | 12 | 7 | 2 | 19 | -1 | -1 |
| -1 | -1 | 6 | 17 | 20 | 13 | 8 | -1 | -1 |
| -1 | -1 | 11 | 22 | 3 | 18 | **1** | -1 | -1 |
| -1 | -1 | 16 | 5 | 24 | 9 | 14 | -1 | -1 |
| -1 | -1 | 23 | 10 | 15 | 4 | 25 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

Solution found, output the solution to the file and start next puzzle. **Do not find all solutions**

**Algorithm**

1. Verify command line arguments
2. Create output file name
3. Open input and output file checking for success
4. Read in first puzzle until there are no more
5. Dynamically allocate 2d array
6. Initialize the board
7. Solve the tour/puzzle
8. Output solution or “No solution” with other information to the output file
9. Free the array
10. Repeat 4 until no more puzzles / tours are left

**Hints**

1. Work on program for short periods of time.
2. Start with small puzzles, 8x8 will take hours to run.
3. When stuck, seek help. Do not look at the code for hours trying to find your error.
4. Do not procrastinate. This makes it miserable trying to get the program done at the last minute.
5. Do your own work.

**Turning in your program**

The program is due Monday, April 8 at midnight. You need to have prog3.cpp with doxygen information. Make sure you put a @file tag at the top of the file. You will submit a single source file because this is a short but complex solution. I will not accept multiple file solutions. Use the submit page to turn in prog3.cpp. Do not zip it.