CSC215 Programming Techniques – Due April 16 at Midnight
Programming Assignment #3: The Knights Tour

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). You must use the mcs gitlab repository csc215s18programs.git and create a project named prog3 within it. This should be a multiple source file program with files named prog3.cpp, menu.h and menu.cpp.

Starting the program

Your program will have one or two arguments added to the program executable name. There are four ways your program could be started.

1) C:\> prog3.exe
2) C:\> prog3.exe tourfile
3) C:\> prog3.exe -fancy
4) C:\> prog3.exe -fancy tourfile

In the above list, 1 & 2 are required and will do some basic formatting to the screen and the file.
List items 3 & 4 are extra credit and will neatly display the board to the screen and the file. You will need to get a good editor to view the fancy output. (notepad++ or VSCODE) If your program does not work correctly, you will receive no extra credit. Sometimes windows 10 will display the correct characters encoding scheme.

Option 1)

This will present a menu to the user show the board size and default coordinates of where the knight will start. The default board size is 8x8 and the default starting position is in row 7 column 7. As the user make changes, you must update your menu. When the user selects option 3, you sole the tour with the selected values and show your solution to the screen. Error checking is a must, you can not change the board size to 5 while starting position is 7,7. You can not change the starting location to 9,7 on an 8x8 board.

Sample run

```
  1) Change Board Size from 8x8
  2) Change starting location from [7,7]
  3) Exit and Solve Tour
Enter Choice: 1

Enter the size of the NxN Board (>3): 5

  1) Change board size from 5x5                ← note the board size is updated.
  2) Change starting location from [7,7]
  3) Exit and Solve Tour
Enter Choice: 2
```

```
Enter Starting Coordinates [ row , col ]: 2 4

  1) Change board size from 5x5
  2) Change starting location from [2,4]  ← noprogte that the starting position changed
  3) Exit and Solve Tour
Enter Choice: 3

Tour # 1
        5x5 starting at (2 , 4)

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

Option 2)

If an input file is given at the command prompt, it will contain tours in the following format. You are not guaranteed that a tour will be in the file(empty) but you are guaranteed that if one exists, it is valid. There will be many tours inside this file and you will need to process every tour. 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. The tours in the file will be valid.

Example	explanation
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 tours for you to solve.
The output file “Solutions.tours” would contain the following.

```
Tour # 1
        5x5 starting at (0 , 0)

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

```
Tour # 2
        6x6 starting at (3 , 2)

        18  21  28   9  12   3
        29  10  19   2  27   8
        20  17  22  11   4  13
        33  30   1  24   7  26
        16  23  32  35  14   5
        31  34  15   6  25  36

Tour # 3
        5x5 starting at (1 , 3)

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

Outputting a tour to the file or the screen:

The output file name will be named "Solutions.tours". As you solve the tours from the input file, you will output the solution / no solution to the file in the following format. If the file already exists, **you will append your answers to the end of the file**. You will start numbering your tours from 1. Do not attempt to start numbering where the file tours leave off.

```
Tour: # 1
          nxn  starting at (r,c)

          #   #   #   #   #
          #   #   #   #   #
          #   #   #   #   #
          #   #   #   #   #
          #   #   #   #   #
```

Example:
So if this was the 5th tour in the file and n was 5 and you started at 2 4

```
Tour: # 5
        5 X 5 starting (2, 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:

```
Tour: # 6
        4 X 4 starting at (0, 0)

        No solutions for this case.
```

Setting up to solve a Tour:

For practice, you will need to dynamically allocate a 2d array for each tour. 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. The zero would indicate that you have not visited that spot on the board. Remember to add 2 to the starting row and column if you use this method. You can elect to dynamically allocate just the space required and handle when you try and step off the board with a base case.

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 tour shows where the knight will start. If you throw away the board, you will be at row 2, column 2. 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	**S**	0	o	-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 tour

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.

	h		a	
g				b
		X		
f				c
	e		d	

<u>It is required to do the moves in this order or grading will take forever and you will not get the correct results</u>

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 (5x5 and starting point of 2 2)

0	0	0	0	0
0	0	0	0	0
0	0	**1**	0	0
0	0	0	0	0
0	0	0	0	0

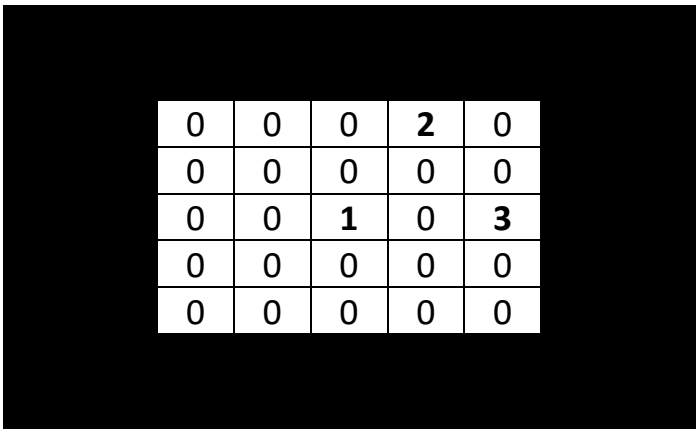
Try move a: Up 2 right 1

Open and would be the second step.

0	0	0	**2**	0
0	0	0	0	0
0	0	**1**	0	0
0	0	0	0	0
0	0	0	0	0

Move a,b,c step you off the board

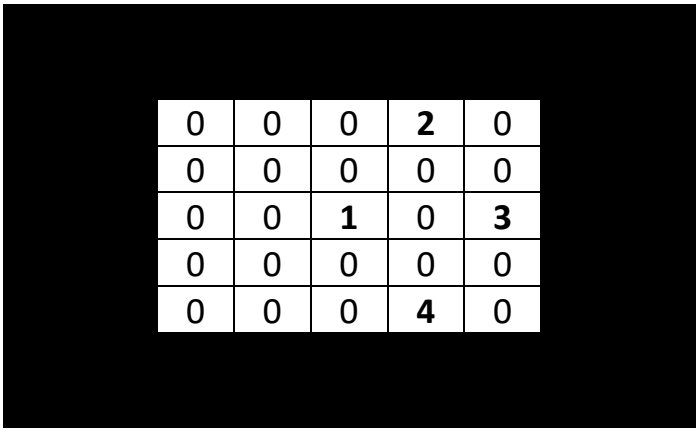
Move c is open and would be the third step.



0	0	0	**2**	0
0	0	0	0	0
0	0	**1**	0	**3**
0	0	0	0	0
0	0	0	0	0

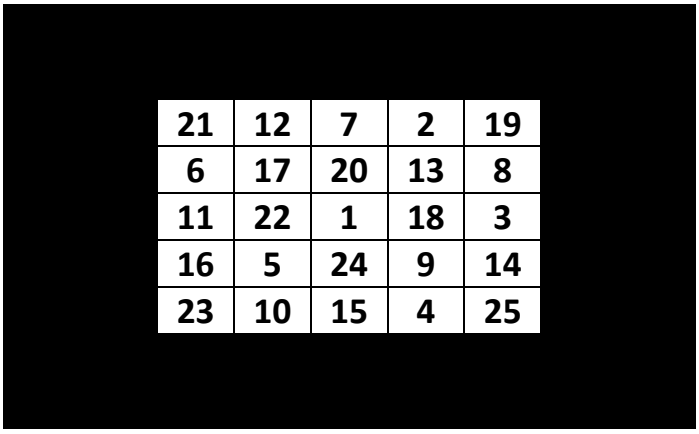
Move a,b,c,d are invalid

E is open and would be the 4th step.



0	0	0	**2**	0
0	0	0	0	0
0	0	**1**	0	**3**
0	0	0	0	0
0	0	0	**4**	0

This pattern would repeat until you find a solution or have exhausted all possibilities.



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

Solution found, output the solution to the file or screen and start next tour if required.

Do not find all solutions

Stop the brute force algorithm after the first solution is found.

Menu

You will use the menu class that your team developed in program 2. Copy all the member functions to a single file "menu.cpp" and copy menu.h to your project directory. If you have all your menu functions in multiple files, you must put them into a single source.

Algorithm

1. Verify command line arguments
2. If menu is required, see algorithm 1
3. If file is given, see algorithm 2

Algorithm 1

1) Present menu
    a. If option 1 is selected, prompt for board size, error check and change menu if valid.
    b. If option 2 is selected, prompt for starting coordinates, error check and change menu if valid.
    c. If option 3 is selected, break loop and solve tour.
    d. Goto 1)
2) Dynamically allocate 2d array and initialize the board
3) Solve the tour using the brute force backtracking method.
4) Output the solution or "No solution" to the screen formatted neatly with information in columns.
5) Free the array
6) Exit program

Algorithm 2

1) Open the input file
2) Read in the first tour if it exists
3) Dynamically allocate 2d array and initialize the board
4) Solve the tour using the brute force backtracking method.
5) Output solution or "No solution" in the proper format to the file
6) Free the array
7) Goto 2 until no more tours are left if solve.
8) Exit program

Hints

1. Work on program for short periods of time.
2. Start with small tour, 8x8 can 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. And you will lose points for not working on it throughout the two weeks.
5. Do your own work.

Turning in your program

You need to have prog3.cpp with doxygen information. Make sure you put a @file tag at the top all c++ files. I will only accept the 3 files (prog3.cpp, menu.h and menu.cpp) because it is a fairly small program. Your program must be in the csc215s18programs git repository set up similarly to prog1. The menu class should already be document. The file must be committed to the repository to get graded.

No timeline is given, and you are expected to work on your program often.

Create a project name prog3.sln
Add the source file prog3.cpp, menu.h and menu.cpp to the solution.

The Extra Credit.

If -fancy is supplied as a command line option, you can get some extra credit if you use the extended ascii characters to format your data to the screen and to the file. Be warned that you may not see what you want when viewing the file. If you decide to implement the extra credit, options 1 and 2 must work as described. Options 3 and 4 will have the term –fancy in argv[1]. You must validate all command line arguments. Do not just assume that if there are three options that -fancy exists.

You may not use my nqueens formatting but can use it for guidance. Come up with your own formatting of the output and not just the board. Google “extended ascii table” to see what is available.