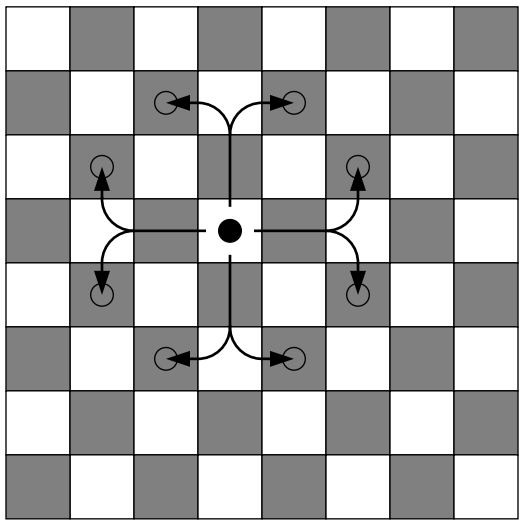
Project 1

Knight's Tour

Introduction

In the game of chess, which is played on an 8×8 board, a knight may move two squares in any of the four basic directions and then one square perpendicular to that direction. Thus, there are 8 possible knight's moves, assuming none of them move the knight off the board. The Knight's Tour Problem asks the question, can a knight move from square to square, landing on every square exactly once?



For example, if the board is 5×5, then the following diagram shows one solution, beginning in square $(0, 0)$ and ending in square $(4, 0)$.

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

In this project, we will write a program that will solve the Knight's Tour Problem for a board of any size, either by finding a tour or by determining that there is no tour. For example, there is no knight's tour on a 4×4 board.

Data Structures

The `Point` Class

We will use the `Point` class from the first semester. I will provide you with my version of the `Point` class. (You may have called it the `Point2D` class.) A `Point` object consists of a pair of integers representing the x and y coordinates of a point in the plane.

The Possible Moves

Use a `typedef` statement to define a `Move` object to be the same as a `Point` object. When the pair of numbers (x and y) represent a move from the current position, it seems more natural to call it a move than to call it a point.

Create a list of possible moves. This should be an array containing 8 `Moves`. These moves should be $(2, 1)$, $(1, 2)$, $(-1, 2)$, etc., representing each possible offset from the current position. See the first diagram above. (You may list the 8 possible moves in any order you like.) The `Point` class allows addition and subtraction of points, so you can make a move by adding an offset to the current position.

The `TourBoard` Class

The TourBoard class will have data members, `m_rows` and `m_cols`, that will record the size of the board. The data member `m_board` will be a two-dimensional array of integers. Declare the array to be 15×15. Then make sure that `m_rows` and `m_cols` are never greater than 15. The integer stored in the array is the order in which that square was visited. See the second diagram above. The value `0` means that the square has not yet been visited.

The goal is to solve the Knight's Tour Problem for an traditional 8×8 board, but to speed up the testing of your program, you may want to begin with a 5×5 board. A 5×5 board can be solved roughly 1000 times faster than the 8×8 board, depending on the starting point and the order in which the possible moves are tried.

The Algorithm

The program should begin by prompting the user for the number of rows and the number of columns in the board. These values are used to create the board. Then the user is prompted for the starting position. This will be entered as a `Point` object.

Our algorithm will be recursive. I suggest that you write a function `solveProblem()` that will receive the TourBoard object and the current position.

The `solveProblem()` function should first check to see whether the problem has been solved by checking whether the number of squares visited so far equals the number of squares on the board. If it does, then return `true`.

Otherwise, begin trying the possible moves. Use a `for` loop to work sequentially through the list of possible moves. For each move in the list, first check whether it is legal. If it is, then make that move and call `solveProblem()` (recursively) with the updated values of the board and the current position.

If `solveProblem()` returns `true`, then return `true`. If `solveProblem()` returns `false`, then undo that move and try the next possible move. If all possible moves have been tried, but without success, then return `false`.

When execution returns to the `main()` function, `solveProblem()` will return either `true` or `false`. If it returns `true`, then display the board, writing in each cell the order in which that square was visited. If it returns `false`, then print a message indicating that a knight's tour is not possible for that board.

This assignment will be turned in in two parts:

- By 5:00 pm Friday, January 19, place the files `tourboard.h` and `tourboard.cpp` in a folder named `TourBoard_Class` and drop it in the dropbox.
- By 5:00 pm Friday, January 26, place the files `tourboard.h`, `tourboard.cpp`, and `KnightsTour.cpp` in a folder named `Project_1` and drop it in the dropbox.