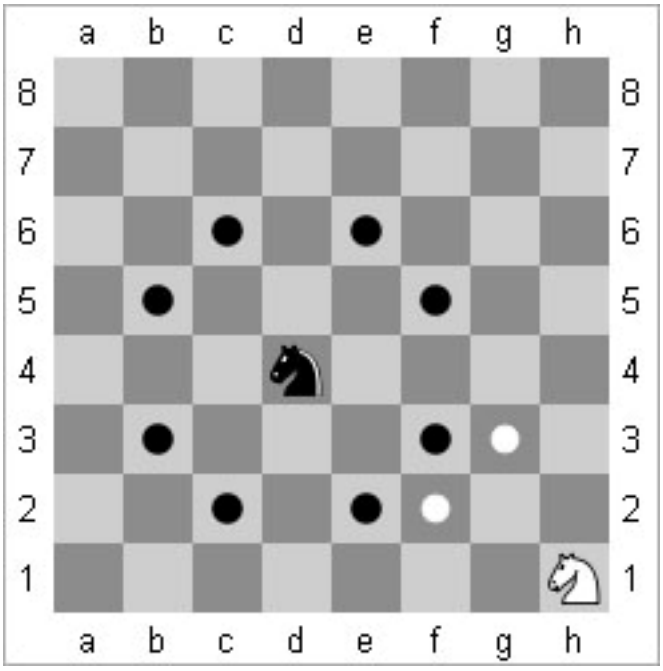
CS007A: Knight's Path Problems – Part 1

A knight's path is the path a knight takes in moving around the chess board. In general, a knight is known to move from its current position on a chess board to a new position by either going up or down 1 or 2 and then going left or right 2 or 1, making an "L" shape which is squares in one direction and 2 squares in the other direction. So a black knight on an a standard chess board at column d and row 4 (as shown below) can move to 8 positions (black circles), while the white knight in the corner at h1 has only two moves (white circles).



There are some special knight's paths we will study:
(1) a *knight's tour* visits each square exactly once and
(2) a *knight's circuit* visits every square exactly once and then can return to the original square on the last move.

It can be shown that there is no Knight's tour of a 4x4 chess board and there are knight's tours of the 5x5 board, but no knight's circuit on a 5x5 (verify).

Challenged to produce a program that would let us experiment with knight's tours in class on 3/28/12 we decided that the game loop (in pseudo code) could include:

```
initialize board and initial position
while (it's possible to move)
   display board
   get move
   update board
```

The variables needed include the state of the board (which squares have been visited on which move) the current position of the knight (perhaps a single integer between 0 and SIZE^2 – 1 counting the squares from left to right and down) and variable to choose the player's move.

Here's some code we came up with to make this work during lab on 4/27/12:

```
// G. Hagopian  Knight's tour version 01

#include <iostream>
#include <iomanip>
using namespace std;
const int SIZE = 8;
void displayBoard(int [][SIZE], const int);
void getMove(int [][SIZE], int&, const int, int, bool&);
void start(int [][SIZE], int&, const int);

int main()
{

	int board[SIZE][SIZE] = {0};
	bool stuck = 0;
	int move = 2;
	int currPos;
	start(board, currPos, SIZE);
	// game loop
	while(!stuck)
	{
		displayBoard(board,SIZE);
		getMove(board, currPos, SIZE, move, stuck);
		++move;
	}
	return 0;
}

void start(int b[][SIZE], int& cp, const int S)
{
	int row, col;
	cout << "\nWhere do you want to start? ";
	cin >> row >> col;
	b[row][col] = 1;
	cp = 8*row+col;
}

void displayBoard(int b[][SIZE], const int S)
{
	for(int i = 0; i < S; i++)
	{
		for(int j = 0; j < S; j++)
			cout << setw(3) << b[i][j];
		cout << endl << endl;
	}
}

void getMove(int b[][SIZE], int& currPos, const int S, int move, bool& stuck)
{
	int row, col;
	cout << "\nEnter a move: ";
	cin >> row >> col;
	currPos = currPos + 8*row + col;
	b[currPos/SIZE][currPos%SIZE] = move;
}
```

Observe that we have three functions corresponding roughly to our pseudocode. (1) `start()` initializes the set up and allows the user to choose an initial position. Then we enter the game loop where (2) `displayBoard()` and (3) `getMove()`. Also note that the current position of the knight, `cp` or **`currPos`**, is a key piece of information that is passed by reference to the `start()` function and the `getMove()` function.

Your task is to modify this code to

(1) Make sure the user only supplies a legal move that stays on the board and doesn't move into a previously visited position. It might be helpful to define a global variable for the (in general) eight possible moves:

```
int move[8][2] = {{2,-1},{1,-2},{-1,-2},{-2,-1},{-2,1},{-1,2},{1,2},{2,1}};
```

(2) Include a `stuck()` function that returns false if the knight has a place to move and true if not. For this purpose, it may be helpful to maintain and update a matrix of access numbers for each position on the board. Each number represents how many squares the square is accessible from, originally this matrix needs to be updated each time you move and is used to help strategize about how to visit all the squares on a knight's tour.

```
int access[8][8] = {{2,3,4,4,4,4,3,2},
                    {3,4,6,6,6,6,4,3}
                    {4,6,8,8,8,8,6,4},
                    {4,6,8,8,8,8,6,4},
                    {4,6,8,8,8,8,6,4},
                    {4,6,8,8,8,8,6,4},
                    {3,4,6,6,6,6,4,3},
                    {2,3,4,4,4,4,3,2}};
```