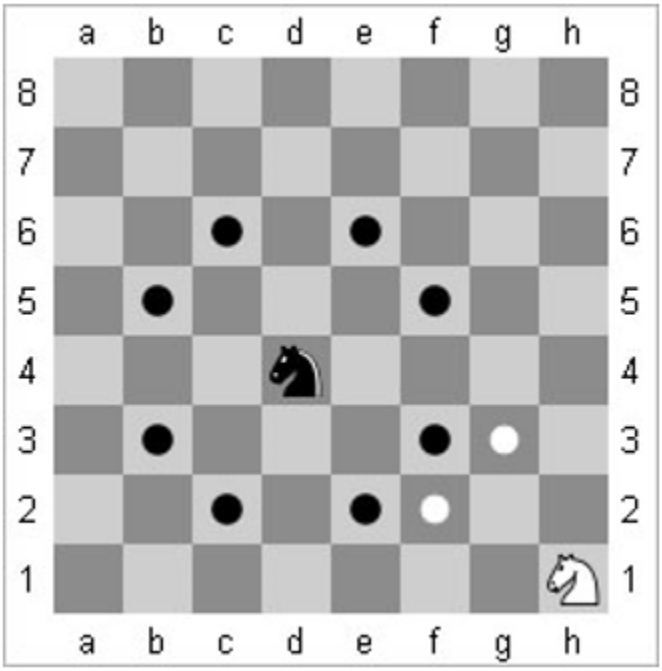
CS 7A - Fall 2015 - CS007A: Knight's Path Problems Part 1. Due 10/28/15

Background Theory

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 we might start with a game loop (in pseudo code) like this:

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

The variables we may need 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 a variable to choose the player's move.

Here's some code we could start with:

```
/// GH working on knights tour 20151015
#include "..\std_lib_facilities.h"
int bWidth{8}, bLength{8};

///prototypes
void displayBoard(vector <int>);

/// let user choose a legal move and update
void getMove(vector<int>& brd,
             int& move,
             int& currPos,
             bool& stuck);

///initialize the board and set first position
void start(vector <int >, int &);

int main() {
    vector<int> board(bWidth*bLength);
    bool stuck = false;
    int move{1};
    int currPos{0};
    start(board , currPos);
    ///game loop
    while(!stuck) {
        displayBoard(board);
        getMove(board , currPos , move , stuck);
    }
    return 0;
}

///initialize the board and set first position
void start(vector<int> brd, int& cp) {
    int row , col;
    cout  << "\nWhat row and column position do you want to start ?";
    cin  >> row  >> col;
    cout << "\n THat's at " << row*bWidth+col << endl;
    brd[row*bWidth+col] = 1;
    cp = row*bWidth+col;
}

void displayBoard(vector <int> board)  {
    for(int i = 0; i < board.size(); ++i) {
        cout << board[i];
        if((i+1)%bWidth == 0)cout << endl;
    }
}

void getMove(vector<int>& brd,
             int& currPos,
             int& move,
             bool& stuck) {
    int row , col;
```

```
53      cout  << "\nChange␣in␣rows␣and␣columns␣(-2,1)␣for␣up␣two-right␣1):";
        do {
55          cin  >> row  >> col;
            if(abs(row*col) != 2
57             || currPos + 8*row + col<0
               || currPos + 8*row + col>63)
59              cout << "\nThat's␣not␣legal␣move.␣␣Try␣again:␣\n";
        } while(abs(row*col) != 2
61              || currPos + 8*row + col<0
                || currPos + 8*row + col>63);
63      ///check that this is a legal move
        currPos = currPos + 8*row + col;
65      ++move;
        brd[currPos]=move;
67  }
```

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 we execute (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 make it conform with the constraints of a proper knight's tour: (1) the knight must be only able to make proper "L"-shaped knight's moves and (2) the knight cannot revisit a square it's already visited.

1 Problems

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. Since this is always eight possibilities, a two-dimensional `array` structure is appropriate:

   ```
   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. If this structure were two-dimensional array on an 8x8 board, it might look like this

   ```
   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}};
   ```

   But we want it to be more flexible so that we can play on a board of any dimensions. Build the corresponding `vector<int>` sturcture and incorporate it into your code so that a user can search for knight's tours.

Here're some Youtubes showing relevant information Knight's Tour - Numberphile
Can You Solve the Knight's Tour?