Assignment 4: Knight’s Tour

Due: 19:00, Wed 25 Nov 2015 Full marks: 100

Introduction

In the game of chess, a knight (馬) is a piece which moves like the letter *L* (「日」字) on a chessboard. That is, it moves either two squares horizontally and one square vertically, or two squares vertically and one square horizontally. A *knight’s tour* is a sequence of moves of a knight on a chessboard such that the knight visits every square of the chessboard *exactly once*. For simplicity, in this assignment, we use a smaller 6 × 6 chessboard. Figure 1(a) shows one possible knight’s tour. Note that if the knight’s moves are not carefully planned, one may end up in a situation where the knight has no more possible moves but some squares on the board remain unvisited. Figure 1(b) shows one such example. You will write a program, using object-oriented programming, to let users put a knight somewhere on a chessboard and make moves to it to solve the knight’s tour problem. Besides, your program should be able to suggest a move for the user.

row \ col	0	1	2	3	4	5
0	32	1	10	13	26	35
1	11	20	33	♘	9	14
2	2	31	12	27	34	25
3	19	28	21	6	15	8
4	22	3	30	17	24	5
5	29	18	23	4	7	16

(a)

row \ col	0	1	2	3	4	5
0						3
1				♘		
2					2	
3			1			
4						
5						

(b)

Figure 1: (a) An Example 6 × 6 Knight's Tour. The symbol ♘ denotes the starting position of the knight, while a number *k* in a square means the knight position after *k* moves. (b) A failed tour after three steps.

Program Specification

You have to write your program in two source files `KnightsTour.cpp` and `tour.cpp`. The former is the implementation of the class `KnightsTour`, while the latter is a client program of class `KnightsTour` which performs the program flow. You are recommended to *finish the `KnightsTour` class first* before writing the client program. When you write the `KnightsTour` class, *implement the member functions and test them individually one by one*. *Your two files will be graded separately*, so you should not mix the functionalities of the two files. Also, you *cannot declare any global variables* in *all* your source files (except `const` ones).

Class `KnightsTour` (`KnightsTour.cpp`)

You are given the interface of the `KnightsTour` class in the head file `KnightsTour.h`. You are *not allowed* to modify the contents of this header file. Descriptions of the class are given below.

```
class KnightsTour {
public:
    KnightsTour(int x, int y);
    void print() const;
    bool hasMoreMoves() const;
    bool isSolved() const;
    bool move(int x, int y);
    void lookAhead(int &r, int &c) const;

    const static int N = 6;
private:
    bool board[N][N];
    int posX, posY;
};
```

Public static Data Member

const static int N = 6;

A *public* named constant denoting that there are six rows and six columns in the chessboard.

Private Data Members

bool board[N][N];

The configuration of a knight's tour is represented by a 6x6 two-dimensional bool-array. The elements `board[0][0]`, `board[0][5]`, `board[5][0]`, and `board[5][5]` denote the top-left, top-right, bottom-left, and bottom-right corners of the board respectively. It stores whether a position in the board was already visited by the knight before. A value of `true` means that position was already visited, while `false` means otherwise.

int posX, posY;

The two data members `posX` and `posY` denote the current position of the knight on the chessboard. They store the row and column indices in `board` respectively. Note that by definition, `board[posX][posY]` is always `true`.

Public Constructor and Member Functions

KnightsTour(int x, int y);

This constructor creates a knight's tour where the knight is initially positioned at row `x`, column `y`. You have to set all array elements of the data member `board` to be all `false` except the initial position. You also have to initialize the data members `posX` and `posY` using the parameters `x` and `y` respectively. You can assume that parameters `x` and `y` are always in [0–5].

void print() const;

Prints out the configuration of the knight's tour. We use symbols 'K', '.', and 'O' to denote the current position of the knight, an unvisited square, and a previously visited square respectively. Figure 2 shows the printout of the tour in Figure 1(a) after five moves.

```
  0 1 2 3 4 5
0 . O . . . .
1 . . . O . .
2 O . . . . .
3 . . . . . .
4 . O . . . K
5 . . . O . .
```

Figure 2: Printing Format of a Knight's Tour

`bool hasMoreMoves() const;`

Checks whether the knight in the tour has more possible moves to make. If all possible destinations of the knight are already visited, then the knight has no more moves. The function should return `true` if a move is possible; and return `false` otherwise.

`bool isSolved() const;`

Returns `true` if the knight's tour is finished successfully, i.e., all squares in the chessboard has been visited already; and returns `false` otherwise.

`bool move(int x, int y);`

Tries to move the knight from its current position to row `x`, column `y`. A move is valid only if *all* the following conditions are satisfied: (a) (x,y) is a proper position (i.e., 0 ≤ x,y ≤ 5), (b) it moves like the letter L, and (c) the destination (x,y) is unvisited before. If the move is not valid, then this member function updates nothing (knight position not updated) and returns `false`. Otherwise, it moves the knight (i.e., update members `board` and `posX` and `posY`) accordingly and returns `true`.

`void lookAhead(int &r, int &c) const;`

This member function "suggests" a move for the knight in the tour. The row and column of the suggested move is written to the *reference parameters* `r` and `c` respectively. The suggested move is decided so that among all valid possible moves (at most eight), we pick the square from which the knight will have the *fewest* onward moves. When counting the number of onward moves, we do not count moves that revisit any visited squares. Figure 3 shows an example to illustrate how a suggestion is made. In the figure, we suppose the knight is at (1,3) and all other squares are unvisited. There are six valid possible moves: (0,1), (0,5), (2,1), (2,5), (3,2), (3,4). Position (0,1) has two onward moves (to (2,0) or (2,2)). Position (0,5) has only one onward move (to (2,4)). Position (2,1) has five onward moves (to (0,0), (0,2), (3,3), (4,0), or (4,2)). Similarly, positions (2,5), (3,2), and (3,4) have three, seven, and five onward moves respectively. So, position (0,5) has the fewest onward moves and thus the reference parameters `r` and `c` are set to 0 and 5 respectively.)

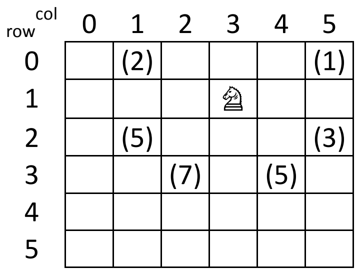

Figure 3: An illustration of move suggestion. The numbers inside brackets denote the number of onward moves that the knight can move from that square. In this example, (0,5) has the fewest onward moves.

Here are some important notes when you implement this member function:

- It is possible that there are two or more choices having the same fewest onward moves. In such case, you break ties with first a smaller row number, and then a smaller column number. For example, in case of ties, (2,3) is more preferred to (3,3); (2,3) is also more preferred to (2,5).
- This look-ahead suggestion is only a heuristic, that is, it is a good suggestion *usually*, but can fail a tour sometimes. You just need to implement this member function accordingly.
- If the knight has no more possible moves, then this member function can suggest nothing. In this case, the reference parameters `r` and `c` should *not* get updated.

- This member function only suggests a move, and the move is not actually made. (And hence the `const` suffix in the prototype.)

Client Program (`tour.cpp`)

Your main program is a client of the `KnightsTour` class with the following flow.

1. The program starts with prompting the user to enter the starting knight position. You can assume that *this user input must be two integers*, the row and column of the chessboard.
2. If the user enters an invalid position (row or column outside [0–5]), you should display a warning and go back to step 1.
3. Create a `KnightsTour` object using the input position and then the program proceeds to the stage of moving the knight.
4. Prompt the user to enter the destination of the knight's move. You can assume that *it must be of two integers*. But you still need to check if a user input is valid or not. (See definition in the description of the `move(…)` member function of `KnightsTour` class.)
5. If the input is valid, you should move the knight to the destination.
6. If the input is not valid, you should print a warning message and obtain a suggested move for the user (using `lookAhead(…)`). Then, move the knight to the suggested destination.
7. After moving the knight, if the knight still has more possible moves, you should go back to step 4 to obtain the next user input destination.
8. When there are no more possible moves, you should determine whether a knight's tour is found or not. Displayed the message "*Finished. Well done!*" or "*Sorry. No more moves!*" accordingly.

Program Output

The following shows some sample output of the program. The **bold blue** text is user input and the other text is the program output. You can try the provided sample program for other input. *Your program output should be exactly the same as the sample program* (i.e., same text, same symbols, same letter case, same number of spaces, etc.). Otherwise, it will be considered as *wrong*, even if you have computed the correct result. Note that in the program output, *only the chessboard is printed from the* `KnightsTour` *class* (`KnightsTour.cpp`). *All other program output (the prompts, warnings, tour end message) are printed from the client program* (`tour.cpp`).

```
Enter knight's starting position (row col): 1 6↵
Invalid. Try again!
Enter knight's starting position (row col): -1 3↵
Invalid. Try again!
Enter knight's starting position (row col): 1 3↵
  0 1 2 3 4 5
0 . . . . . .
1 . . . K . .
2 . . . . . .
3 . . . . . .
4 . . . . . .
5 . . . . . .
Make a move (row col): 0 1↵
```

```
  0 1 2 3 4 5
0 . K . . . .
1 . . . O . .
2 . . . . . .
3 . . . . . .
4 . . . . . .
5 . . . . . .
Make a move (row col): 2 0↵
  0 1 2 3 4 5
0 . O . . . .
1 . . . O . .
2 K . . . . .
3 . . . . . .
4 . . . . . .
5 . . . . . .
Make a move (row col): 0 2↵
Invalid. Make a suggested move for you: 4 1
  0 1 2 3 4 5
0 . O . . . .
1 . . . O . .
2 O . . . . .
3 . . . . . .
4 . K . . . .
5 . . . . . .
Make a move (row col): 6 2↵
Invalid. Make a suggested move for you: 5 3
  0 1 2 3 4 5
0 . O . . . .
1 . . . O . .
2 O . . . . .
3 . . . . . .
4 . O . . . .
5 . . . K . .
Make a move (row col): 4 5↵
  0 1 2 3 4 5
0 . O . . . .
1 . . . O . .
2 O . . . . .
3 . . . . . .
4 . O . . . K
5 . . . O . .
```

.

. *(Skipped to save space. Check Blackboard for full version.)*

.

```
  0 1 2 3 4 5
0 O O O O O .
1 O O K O O O
2 O O O O . O
3 O O O O O O
4 O O O O O O
5 O O O O O O
Make a move (row col): 2 4↵
  0 1 2 3 4 5
0 O O O O O .
1 O O O O O O
2 O O O O K O
3 O O O O O O
4 O O O O O O
5 O O O O O O
Make a move (row col): 0 5↵
0 O O O O O K
1 O O O O O O
2 O O O O O O
3 O O O O O O
4 O O O O O O
5 O O O O O O
Finished. Well done!
```

Submission and Marking

- Your program file names should be KnightsTour.cpp and tour.cpp. Submit the two files in Blackboard (https://elearn.cuhk.edu.hk/). You do not have to submit KnightsTour.h.
- Insert your name, student ID, and e-mail address as comments at the beginning of all your source files.
- Besides the above information, your program should further *include suitable comments as documentation*.
- You can submit your assignment multiple times. Only the latest submission counts.
- Your program should be *free of compilation errors and warnings*.
- *Plagiarism is strictly monitored and heavily punished if proven.* Lending your work to others is subjected to the same penalty as the copier.