Assignment 5: Twin Knights' Tour

Due: 23:59, Tue 22 Nov 2022 File names: `TwinKnightsTour.cpp` `tour.cpp` Full marks: 100

Introduction

The objective of this assignment is to practice object-oriented programming. You will write a class and a client program to walk a *twin knights' tour*.

In the game of chess, a knight (馬) is a piece moving like the letter L (「日」字) on a chessboard. That is, it moves either two squares horizontally and one square vertically (2H1V) or 1H2V. A *twin knights' tour* is to put two knights on a chessboard, each making moves, such that they never visit a square more than once. Note that the two knights can eventually end up in squares where both can have no more possible moves but other squares remain unvisited. Figure 1 shows one twin knights' tour, in which both knights ♘ at (rol,col) = (0,0) and ♞ at (0,7) have no more moves eventually but some squares, e.g., (2,5), (3,7), are unvisited. Your class and client program will let users put two knights somewhere on a chessboard and moves them until no more moves can be made.

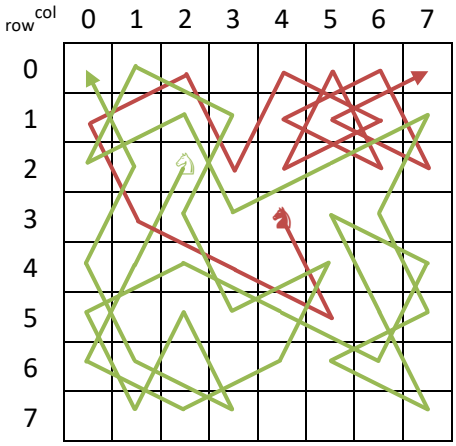

Figure 1: An example 6×6 twin knights' tour. The symbols ♘ and ♞ denote the starting positions of the two knights

Program Specification

You shall write your program in two source files `TwinKnightsTour.cpp` and `tour.cpp`. The former is the implementation of the class `TwinKnightsTour`, while the latter is a client program of class `TwinKnightsTour` which performs the program flow. You are recommended to *finish the `TwinKnightsTour` class first* before writing the client program. When you write the `TwinKnightsTour` 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.*

Class `TwinKnightsTour` (`TwinKnightsTour.cpp`)

You are given the interface of the `TwinKnightsTour` class in the header file `TwinKnightsTour.h`. You *shall not modify* the contents of this header file. Descriptions of the class are given below.

```
class TwinKnightsTour {
public:
    TwinKnightsTour(int r1, int c1, int r2, int c2);
    void print() const;
    bool isValid(char knight, int r, int c) const;
    bool hasMoreMoves() const;
    bool move(char knight, int r, int c);
    static const int N = 8;
private:
    string board[N][N];
    int posR1, posC1, posR2, posC2;
    int steps1, steps2;
    int consec1, consec2;
};
```

Public Class (`static`) Named Constant

`static const int N = 8;`

A class-wide named constant to denote the board size. Your class and client program should be *scalable* to other values for this named constant. That is, they shall still *work normally for other board size*. When grading, we may modify N to any values in the range 4–20 for further testing.

Public Data Members

`string board[N][N];`

The configuration of a twin knights' tour is represented by a two-dimensional array of `string`. The elements `board[0][0]`, `board[0][N-1]`, `board[N-1][0]`, and `board[N-1][N-1]` denote the top-left, top-right, bottom-left, and bottom-right corners of the board respectively. Each array element stores either a dot "`.`", one letter "A"–"Z"/"a"–"z", or two letters "AA"–"AZ", "BA"–"BZ", …, "ZZ" (or "aa"–"az", "ba"–"bz", …, "zz"). A dot "`.`" means that the square was not visited by any knight before. Uppercase strings "A"–"ZZ" denote the move sequence of the one knight ♘: A→B→C→…→Z→AA→AB→…→AZ→BA→…, where "A" is the starting position of ♘. (The sequence is like the column ordering in spreadsheet applications.) Lowercase strings "a"–"zz" denote similarly for the other knight ♞. E.g., the tour in Figure 1 would be stored in the array `board` as shown in Figure 2. Note that "`.`", "A"–"Z", and "a"–"z" are single-symbol strings but not characters.

row \ col	0	1	2	3	4	5	6	7
0	AG	J	f	.	h	k	n	q
1	e	.	L	I	m	p	i	O
2	K	AF	A	g	j	N	l	o
3	.	d	H	M	a	U	P	.
4	AE	B	Y	c	.	F	.	V
5	Z	.	AB	G	X	b	T	Q
6	C	AD	.	.	E	R	W	.
7	.	AA	D	AC	.	.	.	S

Figure 2: The contents of data member `board` for the twin knights' tour in Figure 1

`int posR1, posC1, posR2, posC2;`

Denote the current positions of the two knights on the chessboard. (`posR1`,`posC1`) is the current position of ♘, while (`posR2`,`posC2`) is the current position of ♞.

`int steps1, steps2;`

The data members `steps1` and `steps2` store respectively the number of moves that ♘ and ♞ have already done since the beginning.

`int consec1, consec2;`

To balance the two knights, there is a restriction that one knight cannot move three consecutive steps. That is, if a knight has moved twice consecutively, then the next move must be by the other knight. The data members `consec1` and `consec2` store the number of consecutive moves by ♘ and ♞ respectively.

Public Constructor and Member Functions

`TwinKnightsTour(int r1, int c1, int r2, int c2);`

This constructor creates a twin knights' tour where the knights ♘ and ♞ are initially positioned at (`r1`,`c1`) and (`r2`,`c2`) respectively. It shall set all array elements of the data member `board` as "`.`" (unvisited) except the initial position, which shall be set as "A" for ♘ and "a" for ♞. It shall also set the data members (a) `posR1`, `posC1`, `posR2`, and `posC2` using the parameters `r1`, `c1`, `r2`, and `c2` respectively, and (b) all the other data members as 0. You can assume that parameters `r1`, `c1`, `r2`, and `c2` are always in [0 … N−1], and the two positions are always different.

`void print() const;`

Prints the configuration of the twin knights' tour in the format shown in Figure 3. The following printing format shall be observed:

- The current positions of ♘ and ♞ are specially printed as @ and # respectively. (The actual `board` contents would still be letter strings; they are just printed specially to easily identify the current positions.)
- The row numbers are printed with a width of 2. That is, if a row number is single-digit, then there shall be a space before the digit.
- The board contents are printed with a width of 3. That is, "`.`", "@", "#", "A"–"Z", or "a"–"z" shall be preceded by two spaces, while "AA"–"ZZ" or "aa"–"zz" shall be preceded by one space.

```
    0  1  2  3  4  5  6  7
 0  @  J  f  .  h  k  n  #
 1  e  .  L  I  m  p  i  O
 2  K AF  A  g  j  N  l  o
 3  .  d  H  M  a  U  P  .
 4 AE  B  Y  c  .  F  .  V
 5  Z  . AB  G  X  b  T  Q
 6  C AD  .  .  E  R  W  .
 7  . AA  D AC  .  .  .  S
```

width 2 width 3

Figure 3: Printing format of a twin knights' tour

`bool isValid(char knight, int r, int c) const;`

Checks whether the parameter `knight` in the tour can be moved from its current position to the destination at (`r`,`c`). Note that the knight is *not* actually moved in this member function; it just checks if the move is valid or not. It shall return `true` if *all* the following conditions are satisfied:

- `knight` is either '@' or '#' (meaning ♘ and ♞ respectively);
- `r` and `c` form a proper position ($0 \le r,c < N$);
- the proposed destination is an unvisited square;
- the proposed move is not a third consecutive move for the same knight;
- the destination is an L-shape move (either 2H1V or 1H2V).

If any of the above conditions is `false`, the member function shall return `false`.

`bool hasMoreMoves() const;`

Checks whether either knight has more possible moves to make. This member function shall return `true` if at least one knight can make at least one valid move; and shall return `false` otherwise. This member function can be implemented by *calling* `isValid(…)` several times.

`bool move(char knight, int r, int c);`

Tries to move the parameter `knight` from its current position to the destination at (`r`,`c`). This member function *should call* `isValid(…)` in its implementation to verify whether the move can actually be made. If the destination forms a valid move, this member function shall update all relevant data members to reflect the corresponding knight's change of position, and return `true`. Otherwise (that is, the move is invalid), this member function shall update nothing and return `false`. E.g., a valid move by ♘ would: (a) update its move sequence and its current position, (b) increment `steps1` and `consec1`, (c) reset `consec2` to 0, and (d) return `true`. When updating the move sequence in `board`, you have to find out the next string in the sequence, e.g., from "C" to "D". Note that there are some special cases that you need to handle, e.g., from "Z" to "AA", from "AZ" to "BA", and so on. You also have to handle lowercase strings similarly.

Client Program (`tour.cpp`)

Your main program is a client of the `TwinKnightsTour` class. You create a `TwinKnightsTour` object here and call its member functions to implement the following program flow.

1. The program starts with prompting the user to enter the two knights' starting positions. You can assume that *this user input is always four integers*, where the first two inputs are the row and column of ♘ (@), and the next two inputs are the row and column of ♞ (#).
2. If either input position is invalid (row or column outside [0 … N-1]) or the two positions coincide, display a warning and go back to step 1.
3. Create a TwinKnightsTour object from the input positions.
4. Prompt the user to make a knight's move. You can assume that *the input is always a character followed by two integers*, in which the character is the knight and the integers are the destination of the move. You need to check if a user input is valid or not. (See definition in the description of the `isValid(…)` member function of `TwinKnightsTour` class.) You shall call the `isValid(…)` or `move(…)` member functions to do the checking here.
5. If the input is valid, move the corresponding knight to the destination. Otherwise, print a warning message and go back to step 4.

6. After moving a knight, check if there are still more possible moves. (At least one knight can make at least one move.) If so, go back to step 4 to obtain the next user input destination.
7. When there are no more possible moves, display the message "*No more moves!*".

Some Points to Note

- You *cannot declare any global variables* in *all* your source files (except `const` ones).
- You can write extra functions in any source files if necessary. However, extra *member* functions (instance methods), no matter `private` or `public`, are *not* allowed.
- Your `TwinKnightsTour` class shall *not* contain any `cin` statements. All user inputs shall be done in the client program (`tour.cpp`) only.
- The `TwinKnightsTour` class shall *not* contain any `cout` statements *except* in the `print()` member function for printing the tour. There shall be `cout` statements in the client program `tour.cpp` also.

Sample Run

In the following sample run, the 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* (same text, symbols, letter case, spacings, etc.). Note that *there is a space after the ':' in the program printout*. More sample runs are available in Blackboard.

```
Knights' starting positions (row1 col1 row2 col2): 1 8 3 4↵
Invalid position(s)!
Knights' starting positions (row1 col1 row2 col2): 1 1 -1 3↵
Invalid position(s)!
Knights' starting positions (row1 col1 row2 col2): 2 3 2 3↵
Invalid position(s)!
Knights' starting positions (row1 col1 row2 col2): 2 2 3 4↵
    0  1  2  3  4  5  6  7
 0  .  .  .  .  .  .  .  .
 1  .  .  .  .  .  .  .  .
 2  .  .  @  .  .  .  .  .
 3  .  .  .  .  #  .  .  .
 4  .  .  .  .  .  .  .  .
 5  .  .  .  .  .  .  .  .
 6  .  .  .  .  .  .  .  .
 7  .  .  .  .  .  .  .  .
Move (knight row col): @ -1 2↵
Invalid move!
Move (knight row col): @ 1 5↵
Invalid move!
Move (knight row col): @ 4 1↵
```

```
    0  1  2  3  4  5  6  7
 0  .  .  .  .  .  .  .  .
 1  .  .  .  .  .  .  .  .
 2  .  .  A  .  .  .  .  .
 3  .  .  .  .  #  .  .  .
 4  .  @  .  .  .  .  .  .
 5  .  .  .  .  .  .  .  .
 6  .  .  .  .  .  .  .  .
 7  .  .  .  .  .  .  .  .
Move (knight row col): @ 6 0↵
    0  1  2  3  4  5  6  7
 0  .  .  .  .  .  .  .  .
 1  .  .  .  .  .  .  .  .
 2  .  .  A  .  .  .  .  .
 3  .  .  .  .  #  .  .  .
 4  .  B  .  .  .  .  .  .
 5  .  .  .  .  .  .  .  .
 6  @  .  .  .  .  .  .  .
 7  .  .  .  .  .  .  .  .
Move (knight row col): @ 7 2↵
Invalid move!
Move (knight row col): # 5 5↵
    0  1  2  3  4  5  6  7
 0  .  .  .  .  .  .  .  .
 1  .  .  .  .  .  .  .  .
 2  .  .  A  .  .  .  .  .
 3  .  .  .  .  a  .  .  .
 4  .  B  .  .  .  .  .  .
 5  .  .  .  .  .  #  .  .
 6  @  .  .  .  .  .  .  .
 7  .  .  .  .  .  .  .  .
Move (knight row col): # 3 4↵
Invalid move!
Move (knight row col): @ 7 2↵
    0  1  2  3  4  5  6  7
 0  .  .  .  .  .  .  .  .
 1  .  .  .  .  .  .  .  .
 2  .  .  A  .  .  .  .  .
 3  .  .  .  .  a  .  .  .
 4  .  B  .  .  .  .  .  .
 5  .  .  .  .  .  #  .  .
 6  C  .  .  .  .  .  .  .
 7  .  .  @  .  .  .  .  .

                ⋮          (Some moves are skipped to save space. See Blackboard for full version.)
```

```
    0  1  2  3  4  5  6  7
 0  .  J  f  .  h  k  n  .
 1  e  .  L  I  m  .  i  O
 2  K  .  A  g  j  N  l  #
 3  .  d  H  M  a  U  P  .
 4  @  B  Y  c  .  F  .  V
 5  Z  . AB  G  X  b  T  Q
 6  C AD  .  .  E  R  W  .
 7  . AA  D AC  .  .  .  S
Move (knight row col): # 1 5↵
    0  1  2  3  4  5  6  7
 0  .  J  f  .  h  k  n  .
 1  e  .  L  I  m  #  i  O
 2  K  .  A  g  j  N  l  o
 3  .  d  H  M  a  U  P  .
 4  @  B  Y  c  .  F  .  V
 5  Z  . AB  G  X  b  T  Q
 6  C AD  .  .  E  R  W  .
 7  . AA  D AC  .  .  .  S
Move (knight row col): @ 2 1↵
    0  1  2  3  4  5  6  7
 0  .  J  f  .  h  k  n  .
 1  e  .  L  I  m  #  i  O
 2  K  @  A  g  j  N  l  o
 3  .  d  H  M  a  U  P  .
 4 AE  B  Y  c  .  F  .  V
 5  Z  . AB  G  X  b  T  Q
 6  C AD  .  .  E  R  W  .
 7  . AA  D AC  .  .  .  S
Move (knight row col): @ 0 0↵
    0  1  2  3  4  5  6  7
 0  @  J  f  .  h  k  n  .
 1  e  .  L  I  m  #  i  O
 2  K AF  A  g  j  N  l  o
 3  .  d  H  M  a  U  P  .
 4 AE  B  Y  c  .  F  .  V
 5  Z  . AB  G  X  b  T  Q
 6  C AD  .  .  E  R  W  .
 7  . AA  D AC  .  .  .  S
Move (knight row col): # 0 7↵
    0  1  2  3  4  5  6  7
 0  @  J  f  .  h  k  n  #
 1  e  .  L  I  m  p  i  O
 2  K AF  A  g  j  N  l  o
 3  .  d  H  M  a  U  P  .
 4 AE  B  Y  c  .  F  .  V
 5  Z  . AB  G  X  b  T  Q
 6  C AD  .  .  E  R  W  .
 7  . AA  D AC  .  .  .  S
No more moves!
```

Submission and Marking

- Your program file names should be `TwinKnightsTour.cpp` and `tour.cpp`. Submit the *two files* in Blackboard (https://blackboard.cuhk.edu.hk/). You do *not* have to submit `TwinKnightsTour.h`.
- Insert *your name*, *student ID*, and *e-mail* as comments at the beginning of all your files.
- Besides the above information, your program should *include suitable comments as documentation* in all your files.
- You can submit your assignment multiple times. Only the latest submission counts.
- Your program should be *free of compilation errors and warnings*.
- ***Do NOT plagiarize.*** Sending your work to others is subjected to the same penalty as the copier.