Assignment 5: Drunk Knight's Path

Due: 20:00, Mon 22 Nov 2021 File names: `KnightsPath.cpp` `walk.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 *drunk knight's path*.

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 two squares horizontally and one square vertically (2H1V) or 1H2V. A *drunk knight's path* is a sequence of knight moves on a chessboard such that the knight never revisits a square and "turns back in a next move". "Never turns back" means that a knight cannot move in the two directions that is behind itself. Figure 1 shows an example, where the knight has just moved from position B1 to C3. Its next destination can only be the five squares E2, E4, D5, B5, or A4, but cannot be A2 and D1. Figure 2 shows a drunk knight's path for a 6×6 board. Note that the knight can eventually end up in a square where it has no more possible moves but other squares remain unvisited. An example can be seen after 10 moves in Figure 2. (The knight can move to *neither* D0 nor E3, as they are turn-backs.) Your overall program will let users put a knight somewhere on a chessboard and moves it until no more moves can be made.

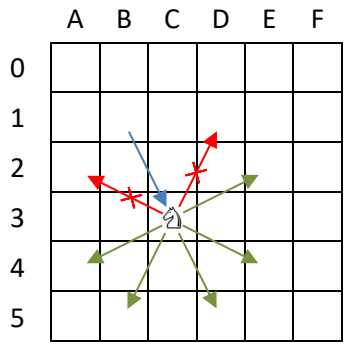

Figure 1: A Knight Never Turns Back.

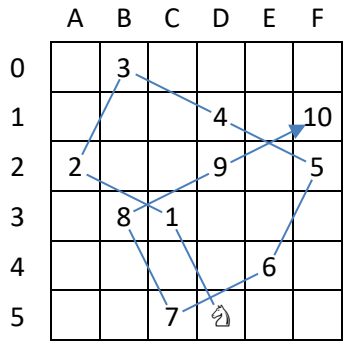

Figure 2: An Example 6×6 Drunk Knight's Path. The symbol ♘ denotes the starting position of the knight, while a number k in a square means the knight position after k moves.

Program Specification

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

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

```
class KnightsPath {
public:
    const static int N = 6;
    KnightsPath(int r, int c);
    void print() const;
    int getSteps() const;
    bool isValid(int r, int c) const;
    bool hasMoreMoves() const;
    bool move(int r, int c);
private:
    int board[N][N];
    int currentR, currentC;
    int steps;
    int previousR, previousC;
};
```

Public Class Data Member

`const static int N = 6;`

A class (`static`) named constant denoting the board size. Your program shall be *scalable* to other values for N. That is, your program should still *work normally for other board size*. We may change N to other values in the range 1–10 when grading.

Private Data Members

`int board[N][N];`

The configuration of a drunk knight's path is represented by an N × N two-dimensional `int`-array. 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. It stores the number of moves that the knight took to reach that position in a path. A value k means the knight reaches that position after k moves. A value 0 means the knight was at that position initially. A special value -1 means that position is not yet visited by the knight.

`int currentR, currentC;`

The current position of the knight on the chessboard. They store the row and column indices in `board` respectively.

`int steps;`

Stores the number of moves that the knight has already made since the beginning of the walk.

`int previousR, previousC;`

The immediate last position of the knight on the chessboard. That is, the knight has just moved from row `previousR`, column `previousC` to row `currentR`, column `currentC`.

Figure 3 shows the contents a `KnightsPath` object representing the drunk knight's path in Figure 2.

board (row \ col)

	0	1	2	3	4	5
0	-1	3	-1	-1	-1	-1
1	-1	-1	-1	4	-1	10
2	2	-1	-1	9	-1	5
3	-1	8	1	-1	-1	-1
4	-1	-1	-1	-1	6	-1
5	-1	-1	7	0	-1	-1

currentR 1 previousR 2

currentC 5 previousC 3

steps 10

Figure 3: Contents of a `KnightsPath` object for the Drunk Knight's Path in Figure 2

Public Constructor and Member Functions

`KnightsPath(int r, int c);`

This constructor creates a drunk knight's path where the knight is initially positioned at row `r`, column `c`. All elements of the array data member `board` shall be initialized to -1 (unvisited) except the starting position, which shall be initialized to 0. The data members (a) `currentR` and `currentC` shall be initialized using the parameters `r` and `c`, (b) `steps` shall be initialized to 0, and (c) `previousR` and `previousC` shall be initialized to -1. You can assume that the parameters `r` and `c` are always in the range [0 … N-1].

`void print() const;`

Prints out the drunk knight's path in the format shown in Figure 4. Symbols 'K' and 'k' denote the knight's current and starting positions respectively. Symbol '`.`' denotes an unvisited square. In the special case of the knight currently at the starting position, print a lowercase 'k' instead of uppercase 'K'. The number of moves that the knight has made is also printed.

```
   A  B  C  D  E  F
0  1  .  .  .  3  .
1  .  K  2  .  .  .
2  .  k  .  4  .  .
3  .  .  .  .  .  .
4  .  .  .  .  .  .
5  .  .  .  .  .  .
Steps: 5
```

Figure 4: Printing Format of a Drunk Knight's Path

`int getSteps () const;`

Returns the number of steps the knight has walked, that is, the data member `steps`.

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

Checks whether the knight in the path can be moved from the current position (row `currentR`, column `currentC`) to the destination at row `r`, column `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:

- `r` and `c` form a proper position ($0 \le r,c < N$);
- the destination is an unvisited square;
- the destination is 2H1V or 1H2V from the current position;
- the destination is not at a back direction.

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

Hint: One way to determine whether a knight is turning back is to check whether the parameter position (`r`, `c`) is "too close" to the knight's previous position (`previousR`, `previousC`). The example in Figure 1 shows that the two disallowed turn-back positions A2 (row 2, column 0) and D1 (row 1, column 3) have a distance of $\sqrt{2}$ and 2 from the previous position B1 (row 1, column 1) respectively. The allowed positions E2, E4, D5, B5, A4 have distances $\sqrt{10}$, $\sqrt{18}$, $\sqrt{20}$, 4, $\sqrt{10}$ from B1, which are all larger than $\sqrt{2}$ and 2. Nonetheless, you are free to use any method to determine turn-backs if you find appropriate.

`bool hasMoreMoves() const;`

Checks whether the knight has more possible moves to make. This member function shall return `true` if there is at least one square on the board that would form a valid move; and shall return `false` otherwise. This member function can be implemented by *calling* `isValid(…)` several times.

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

Tries to move the knight from its current position to the destination at row `r`, column `c`. This member function *should call* `isValid(…)` in its implementation. If the destination forms a valid move, this member function shall update the data members `board`, `currentR`, `currentC`, `steps`, `previousR`, `previousC` accordingly to reflect the knight's change of position, and return `true`. Otherwise (that is, the move is not valid), this member function shall update nothing and return `false`.

Client Program (`walk.cpp`)

Your main program is a client of the `KnightsPath` class. You create a `KnightsPath` object here and call its member functions for the following program flow to walk a knight on the board.

1. The program starts with prompting the user to enter the knight's starting position. You can assume that *this user input is always one letter followed by an integer*, denoting the column and row position of the chessboard.
2. If the user enters an invalid position (row or column outside the board), display a warning and go back to step 1.
3. Create a `KnightsPath` object using the input position.
4. Prompt the user to enter the destination of the knight's move. You can assume that *the input is always one letter followed by an integers*. But you still need to check if a user input is valid or not. (See definition in the description of the `isValid(…)` member function of `KnightsPath` class.) You should call the `isValid(…)` or `move(…)` member functions to do the checking here.
5. If the input is valid, you should move the knight to the destination. Otherwise, you should print a warning message and go back to step 4.

6. After moving the knight, check if the knight still has more possible moves. If so, you should go back to step 4 to obtain the next user input destination.
7. When there are no more possible moves, display the message "*Finished. No more moves!*".
8. Furthermore, if the knight has visited more than half of the board squares, display the message "*Well done!*". Otherwise, display "*Still drunk? Walk wiser!*".

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 `KnightsPath` class shall *not* contain any `cin` statements. All user inputs shall be done in the client program (`walk.cpp`) only.
- In all column input, *lowercase letters are considered invalid*. Only uppercase letters can be valid. You have to *convert the uppercase letter to the corresponding column index* before calling the relevant member functions.
- The `KnightsPath` class shall *not* contain any `cout` statements *except* in the `print()` member function (for printing the board). You have `cout` statements in `print()` and `walk.cpp` only.

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*.

```
Enter starting position (col row): G 1↵
Invalid. Try again!
Enter starting position (col row): D -1↵
Invalid. Try again!
Enter starting position (col row): f 2↵
Invalid. Try again!
Enter starting position (col row): F 5↵
   A  B  C  D  E  F
0  .  .  .  .  .  .
1  .  .  .  .  .  .
2  .  .  .  .  .  .
3  .  .  .  .  .  .
4  .  .  .  .  .  .
5  .  .  .  .  .  k
Steps: 0
Move the knight (col row): E 3↵
```

Lowercase invalid!

Print lowercase k rather than uppercase K at the beginning.

```
   A  B  C  D  E  F
0  .  .  .  .  .  .
1  .  .  .  .  .  .
2  .  .  .  .  .  .
3  .  .  .  .  K  .
4  .  .  .  .  .  .
5  .  .  .  .  .  k
Steps: 1
Move the knight (col row): C 4↵
   A  B  C  D  E  F
0  .  .  .  .  .  .
1  .  .  .  .  .  .
2  .  .  .  .  .  .
3  .  .  .  .  1  .
4  .  .  K  .  .  .
5  .  .  .  .  .  k
Steps: 2
Move the knight (col row): A 2↵
Invalid move. Try again!
Move the knight (col row): E 3↵
Invalid move. Try again!
Move the knight (col row): D 2↵
Invalid move. Try again!
Move the knight (col row): E 5↵
Invalid move. Try again!
Move the knight (col row): D 6↵
Invalid move. Try again!
Move the knight (col row): a 3↵
Invalid move. Try again!
Move the knight (col row): B 2↵
   A  B  C  D  E  F
0  .  .  .  .  .  .
1  .  .  .  .  .  .
2  .  K  .  .  .  .
3  .  .  .  .  1  .
4  .  .  2  .  .  .
5  .  .  .  .  .  k
Steps: 3
Move the knight (col row): A 0↵
```

Turn-backs not allowed!

Lowercase invalid!

```
   A  B  C  D  E  F
0  K  .  .  .  .  .
1  .  .  .  .  .  .
2  .  3  .  .  .  .
3  .  .  .  .  1  .
4  .  .  2  .  .  .
5  .  .  .  .  .  k
Steps: 4
Finished! No more moves!
Still drunk? Walk wiser!
```

Five squares visited. Not more than half (18).

Submission and Marking

- Your program file names should be `KnightsPath.cpp` and `walk.cpp`. Submit the *two files* in Blackboard (https://blackboard.cuhk.edu.hk/). You do *not* have to submit `KnightsPath.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.