CSCI 1120

Introduction to Computing Using C++
Tutorial 9

10th Nov. 2021

TAs

- Muzhi Li (class A)
- mzli@cse.cuhk.edu.hk
- SHB 1013
- Consultation hours: Mon 3:30-4:30pm, Thu 10:00-11:00am

- Bangqi Fu (class B)
- bqfu21@cse.cuhk.edu.hk
- SHB 913
- Consultation hours: Tue 3:30-5:30pm

Outline

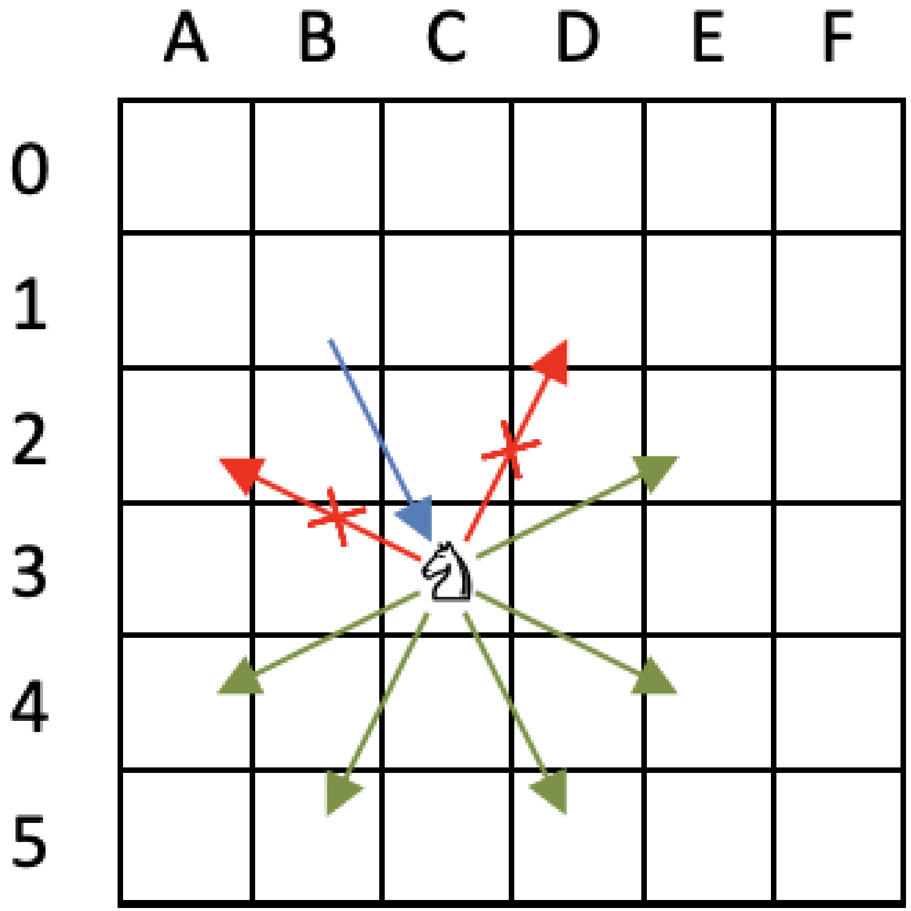
- **Assignment 5**

- 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.

Assignment 5: Drunk Knight’s Path

Introduction

- In the game of chess, a knight (馬) is a piece which moves like the letter L (「日」字) on a chessboard.
- It moves two squares horizontally and one square vertically (2H1V) or 1H2V.
- The knight <span>never</span> **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.



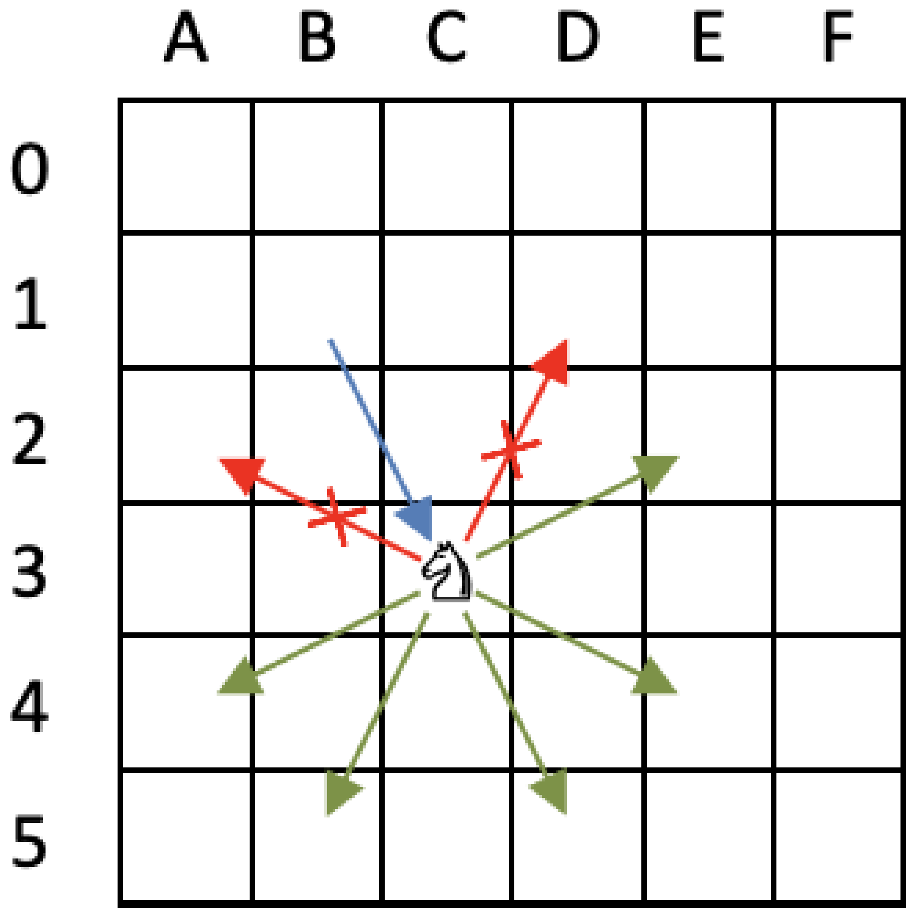
Assignment 5: Drunk Knight's Path

Introduction

- 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.

Example

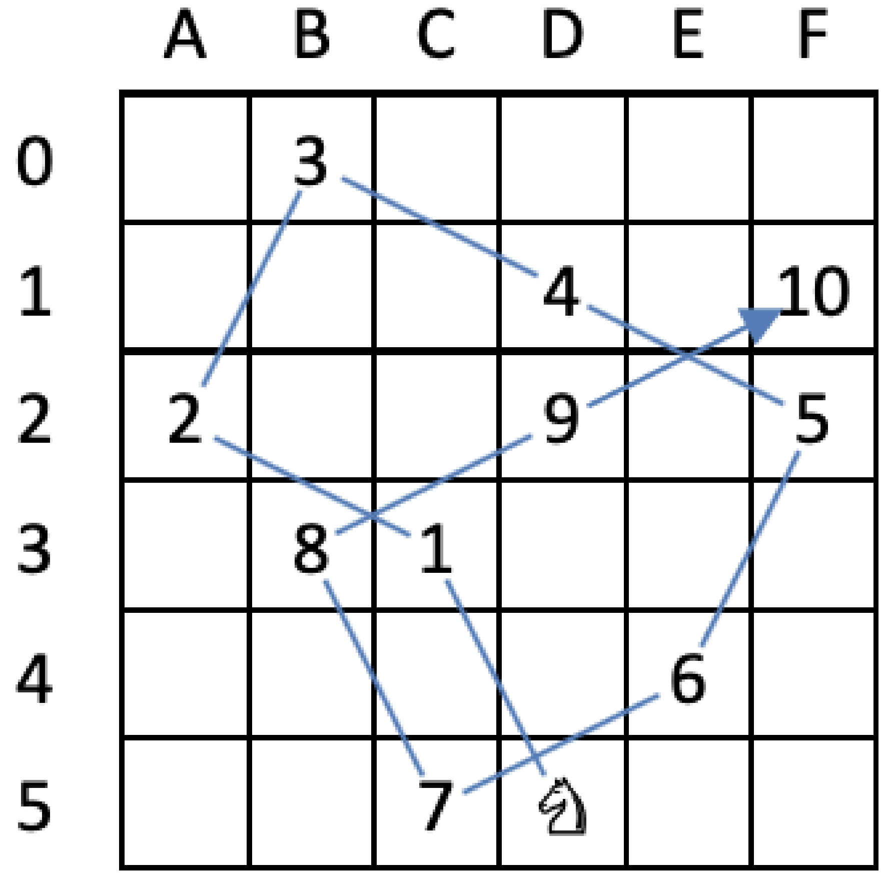
- The knight has just moved from position B1 to C3.
- There are at most 8 possible moves.
- It doesn't go back to B1.
- It doesn't move to the direction backward (A2 and D1).
- 5 moves left.



Assignment 5: Drunk Knight’s Path

➢ Path

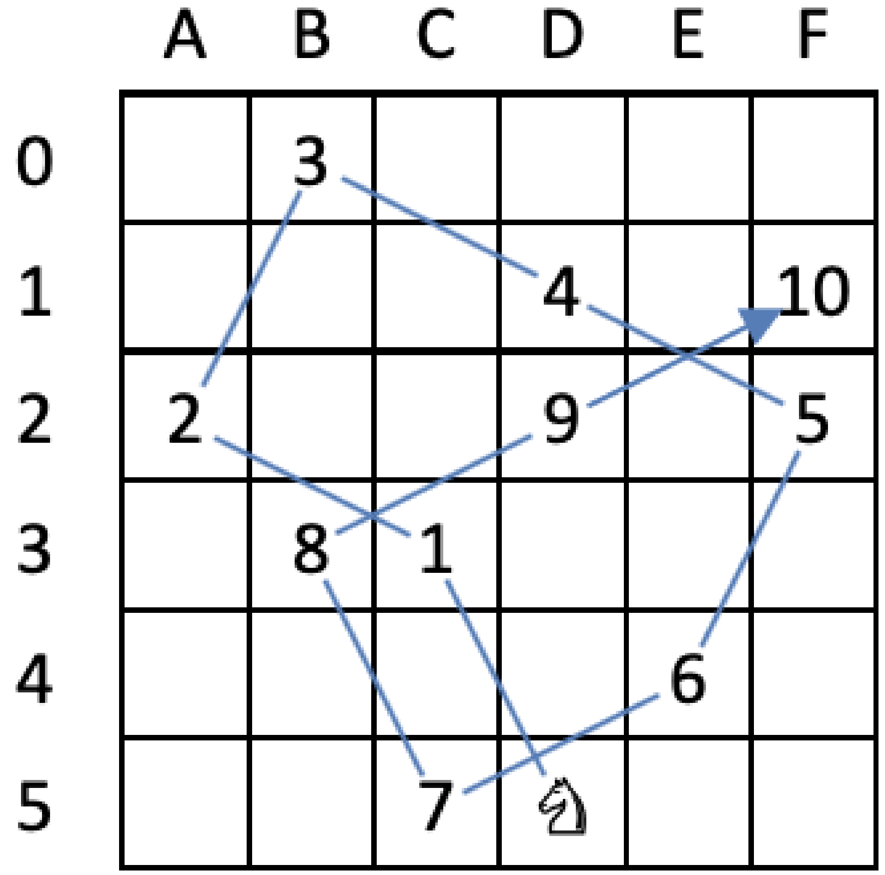
- A drunk knight’s path is a sequence of knight moves on a chessboard that follows the rules stated before.
- On the right shows a drunk knight’s path for a 6 × 6 board.
- The knight can eventually end up in a square where it has no more possible moves.
- The rest of the squares on the board remain unvisited. An example can be seen after 10 moves.



Assignment 5: Drunk Knight’s Path

➢ Path

- After moving from position 9 to position 10, only D0 and E3 are left to choose as they follow the rule of L shape movement.
- However, D0 and E3 are two directions backward.
- Your overall program will let users put a knight somewhere on a chessboard and moves it until no more moves can be made.



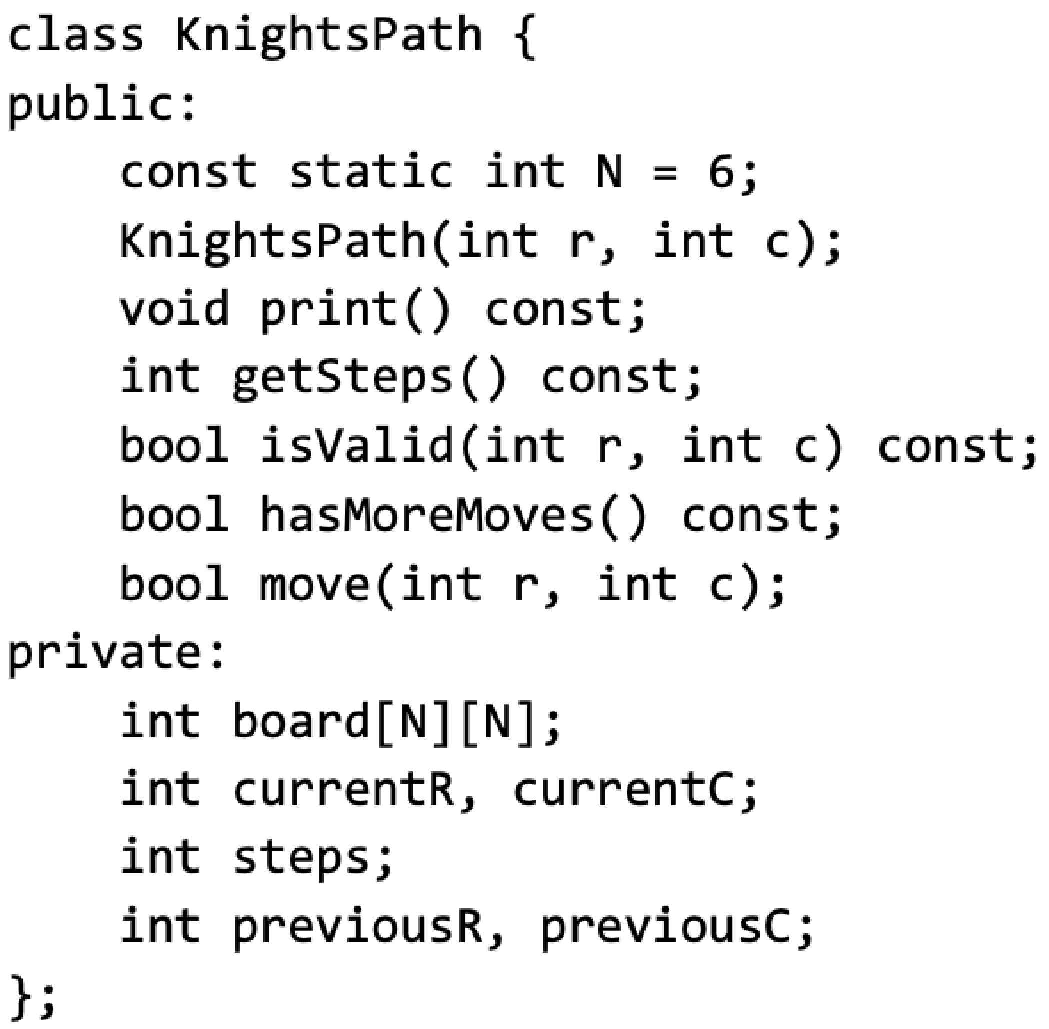
	A	B	C	D	E	F
0		3				
1				4		10
2	2			9		5
3		8	1			
4					6	
5			7	♘		

Assignment 5: Program Structure (KnightsPath.cpp)

➢ File Specification

- You shall write your program in two source files KnightsPath.cpp and walk.cpp.

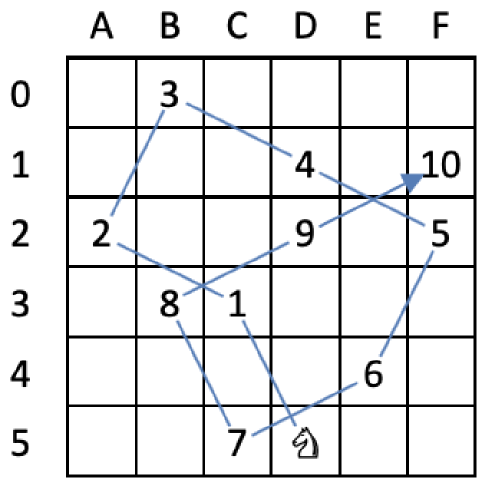
- KnightsPath.cpp: implementation of the class KnightsPath.
- walk.cpp: a client program of class KnightsPath which performs the program flow



```
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;
};
```

Assignment 5: Program Structure (KnightsPath.cpp)

- **Class Specification: data**
  - **const static int N = 6;**
  - A class (static) named constant denoting the board size.
  - Your program shall be scalable to other values for N.
  - Other values in the range 1–10 should be considered.



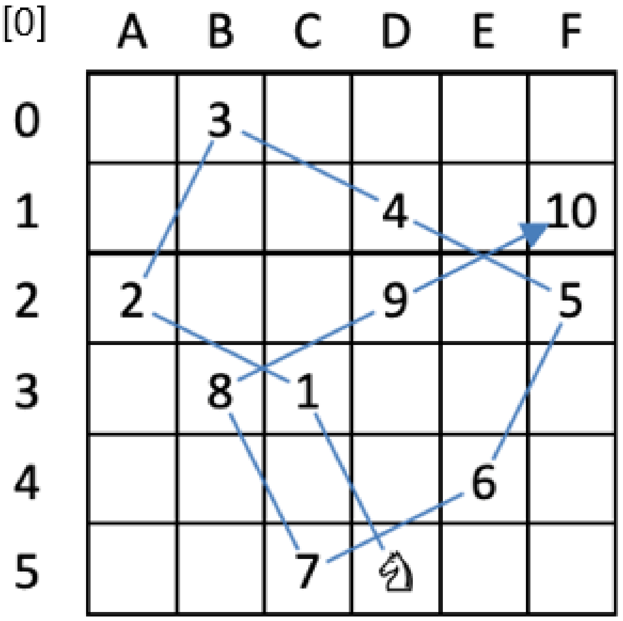
```
class KnightsPath {
public:
    const static int N = 6;
    KnightsPath(int r, int c);
    void print() const;
    int getSteps() const;
    bool isValid(int r, int c) const;
    bool hasMoreMoves() const;
    bool move(int r, int c);
private:
    int board[N][N];
    int currentR, currentC;
   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

Assignment 5: Program Structure

- **Class Specification: data**
  - **int board[N][N];**
  - An N×N 2-dimensional int-array.
  - Bounded by four corners.

board[0][0]

board[0][N-1]



board[N-1][0]

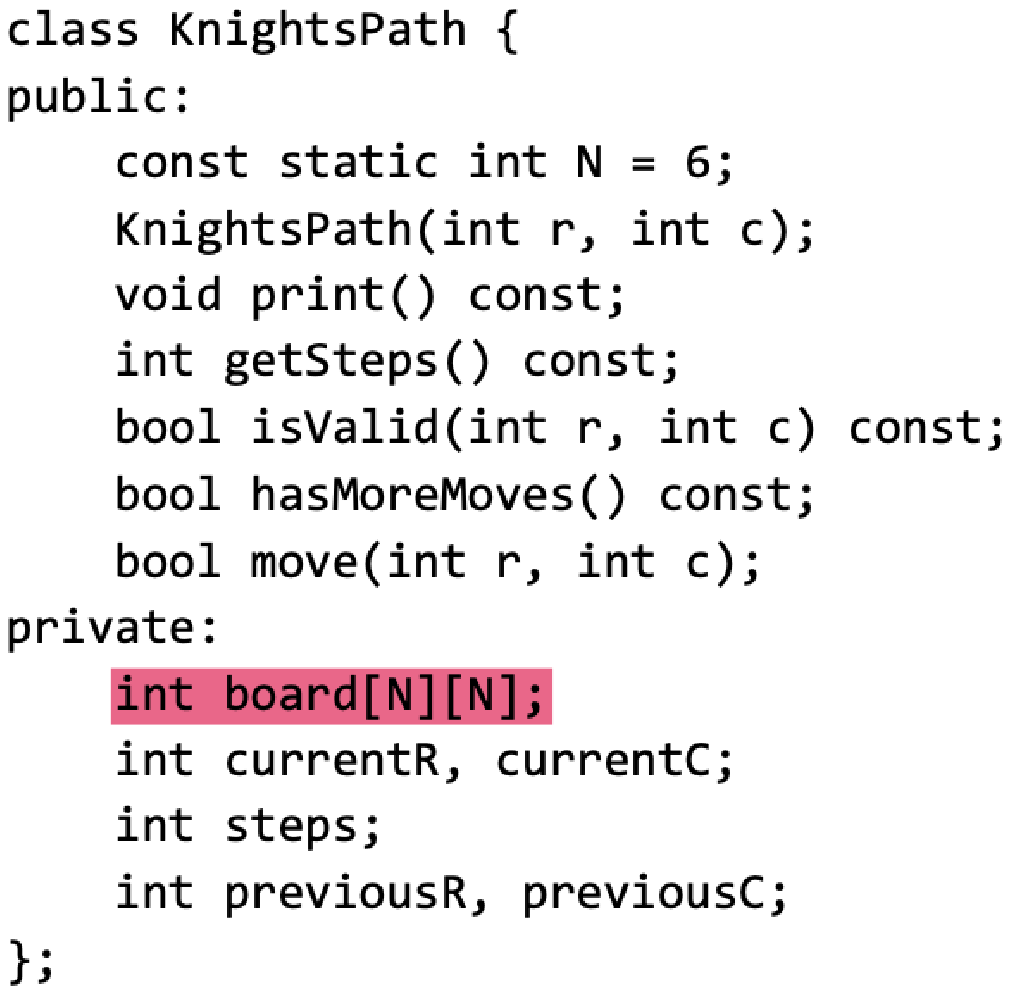
board[N-1][N-1]

```
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;
};
```

Assignment 5: Program Structure

➢ Class Specification: data

- **int board[N][N];**
- 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.

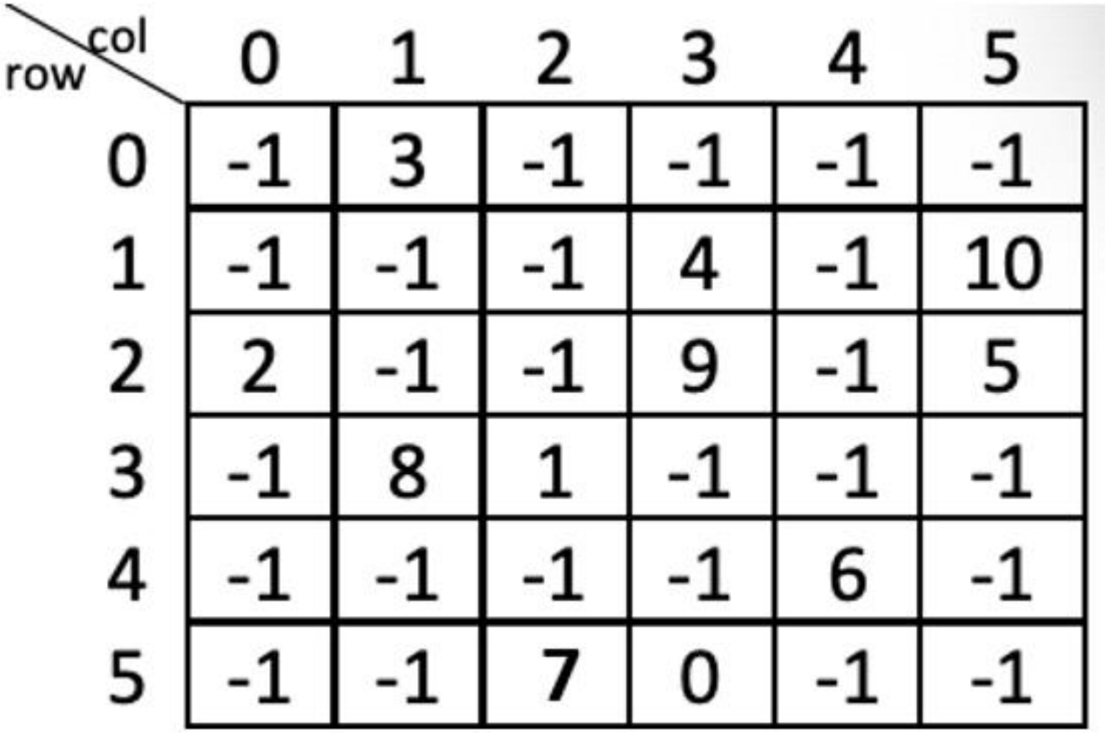


```
class KnightsPath {
public:
    const static int N = 6;
    KnightsPath(int r, int c);
    void print() const;
    int getSteps() const;
    bool isValid(int r, int c) const;
    bool hasMoreMoves() const;
    bool move(int r, int c);
private:
    int board[N][N];
    int currentR, currentC;
   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

Assignment 5: Program Structure

➢ Class Specification: data

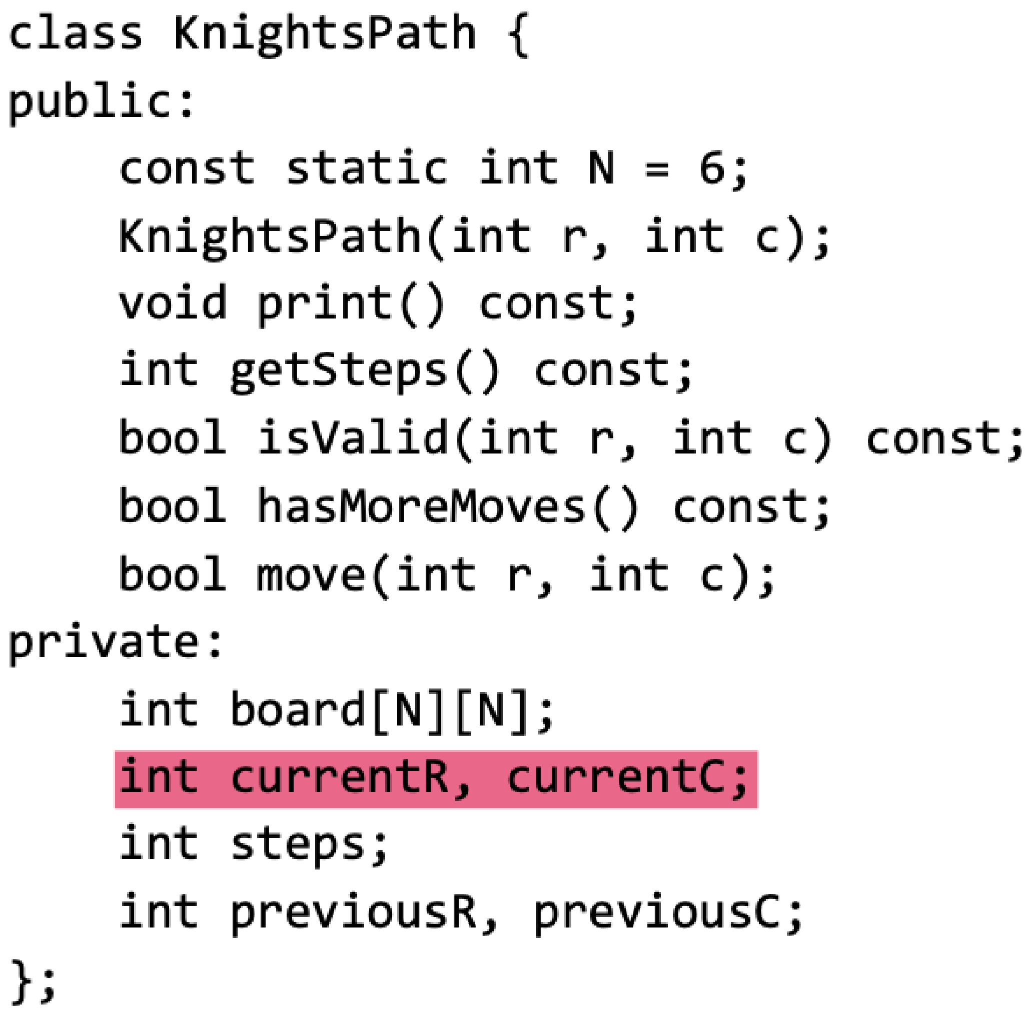
- **int board[N][N];**
- It stores the number of moves that the knight took to reach that position in a path.
- A value k means the knight reaches that position after k moves.
- A value 0 means the knight was at that position initially.
- A special value -1 means that position is not yet visited by the knight.



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

Assignment 5: Program Structure

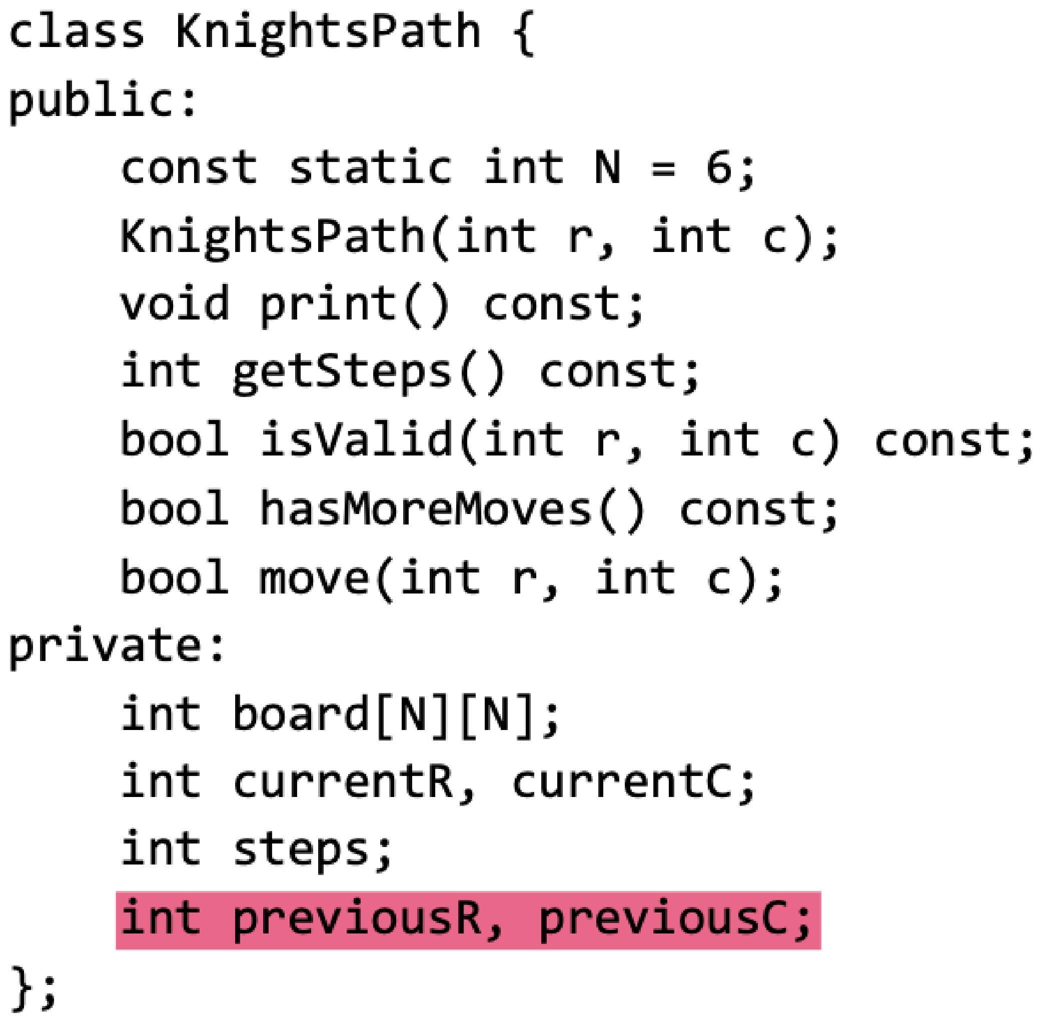
- **Class Specification: data**
  - **int currentR, currentC;**
  - The current position of the knight on the chessboard.
  - They store the row and column indices in board, respectively.



```
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;
};
```

Assignment 5: Program Structure

- **Class Specification: data**
  - **int previousR, previousC;**
  - The immediate last position of the knight on the chessboard.
  - The knight has just moved from row previousR, column previousC to row currentR, column currentC.

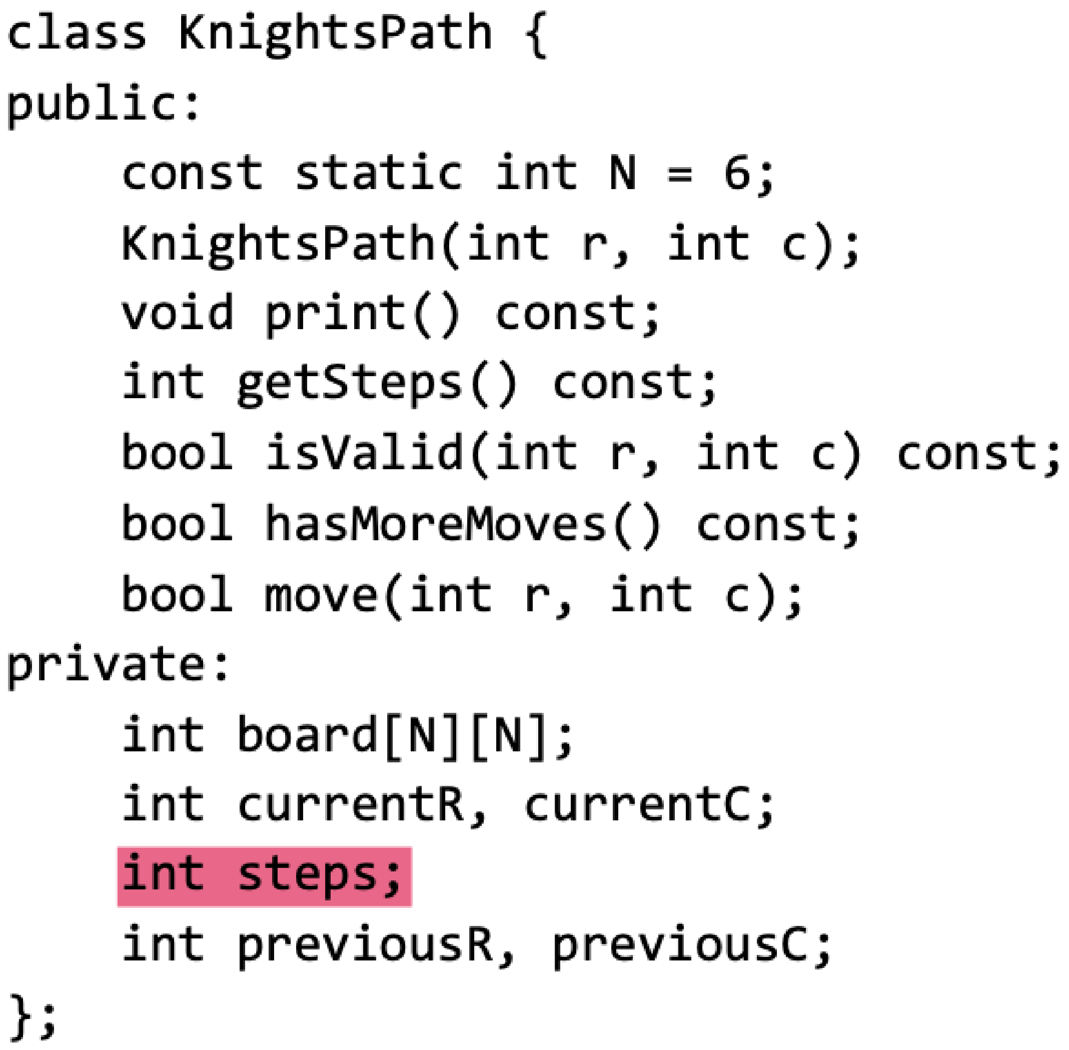


```
class KnightsPath {
public:
    const static int N = 6;
    KnightsPath(int r, int c);
    void print() const;
    int getSteps() const;
    bool isValid(int r, int c) const;
    bool hasMoreMoves() const;
    bool move(int r, int c);
private:
    int board[N][N];
    int currentR, currentC;
   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

Assignment 5: Program Structure

➢ Class Specification: data

- **int steps;**
- Stores the number of moves that the knight has already made since the beginning of the walk.

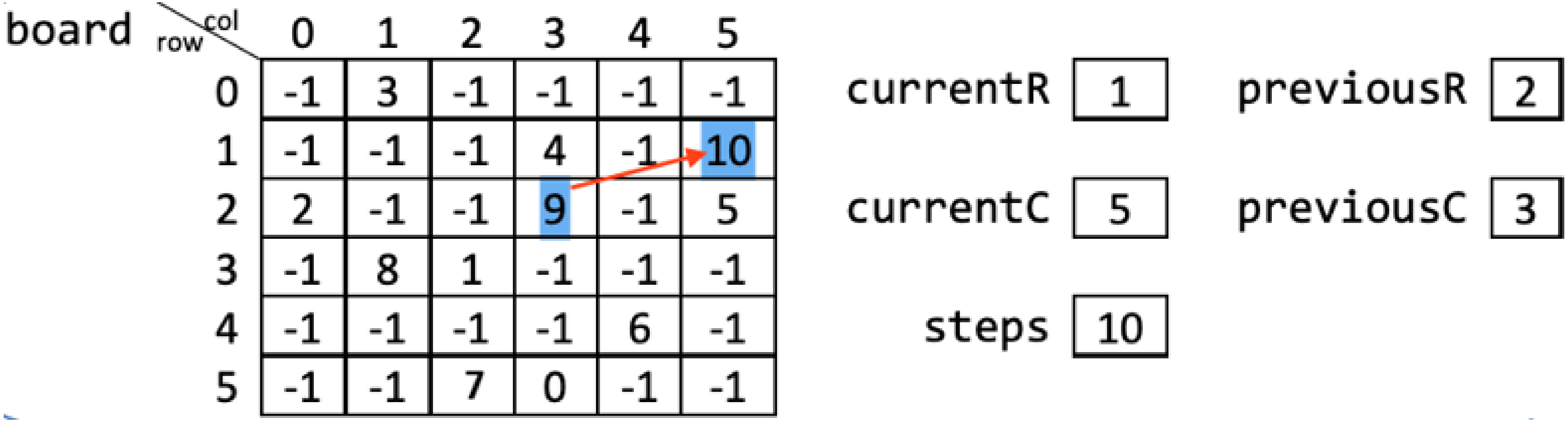


```
class KnightsPath {
public:
    const static int N = 6;
    KnightsPath(int r, int c);
    void print() const;
    int getSteps() const;
    bool isValid(int r, int c) const;
    bool hasMoreMoves() const;
    bool move(int r, int c);
private:
    int board[N][N];
    int currentR, currentC;
   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

Assignment 5: Program Structure

➢ Class Specification: data

- An instance of board and other parameters.
- board[1][5] is the current position with step of 10.
- board[2][3] is the previous position with step of 9.
- You cannot visit these values from outside.



board

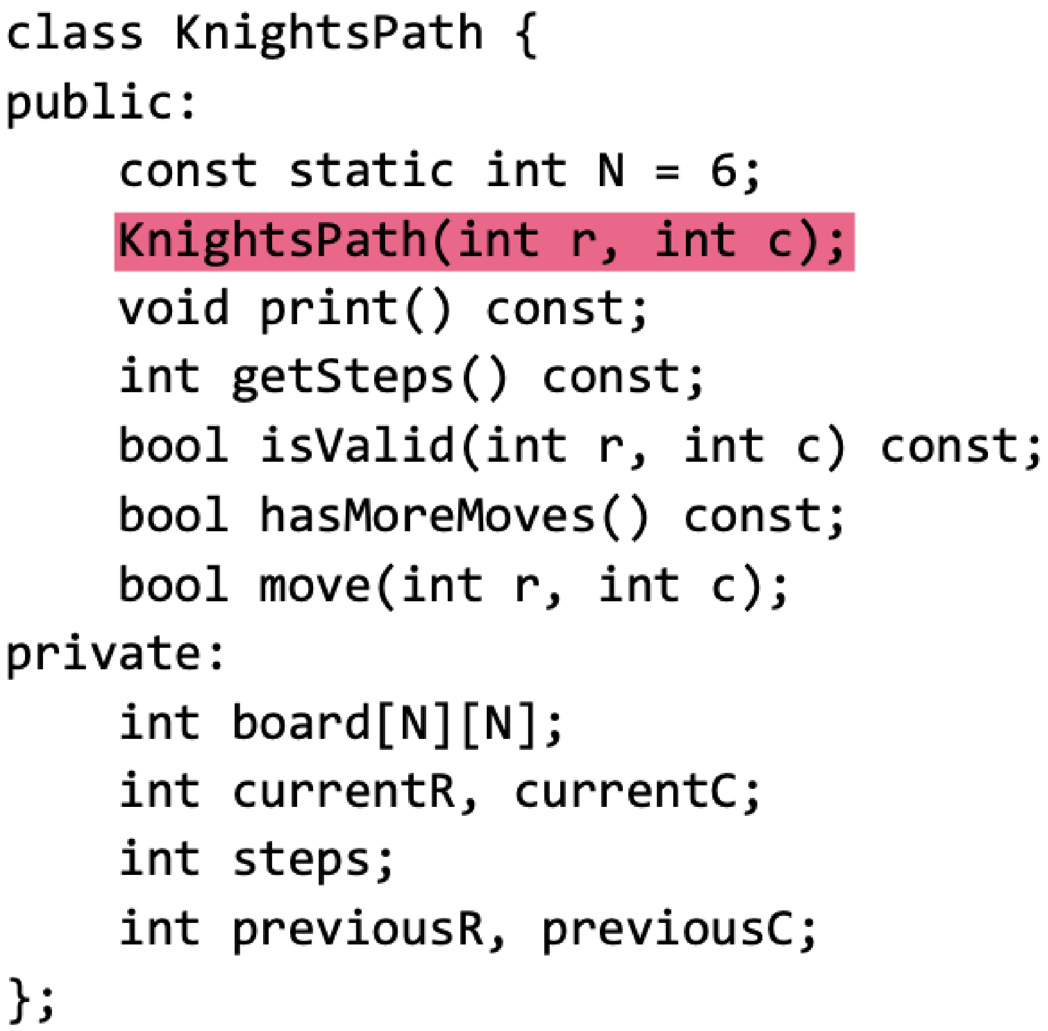
row \ col	0	1	2	3	4	5
0	-1	3	-1	-1	-1	-1
1	-1	-1	-1	4	-1	10
2	2	-1	-1	9	-1	5
3	-1	8	1	-1	-1	-1
4	-1	-1	-1	-1	6	-1
5	-1	-1	7	0	-1	-1

currentR	1	previousR	2
currentC	5	previousC	3
steps	10		

Assignment 5: Program Structure

➢ Class Specification: constructor

- **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 board shall be initialized to -1 (unvisited) except the starting position, which shall be initialized to 0.
- Declare a 2-D array and traverse it.
- Remember to assign value to each element you visit, otherwise it’ll be default value of the data type (0 for unsigned int8).

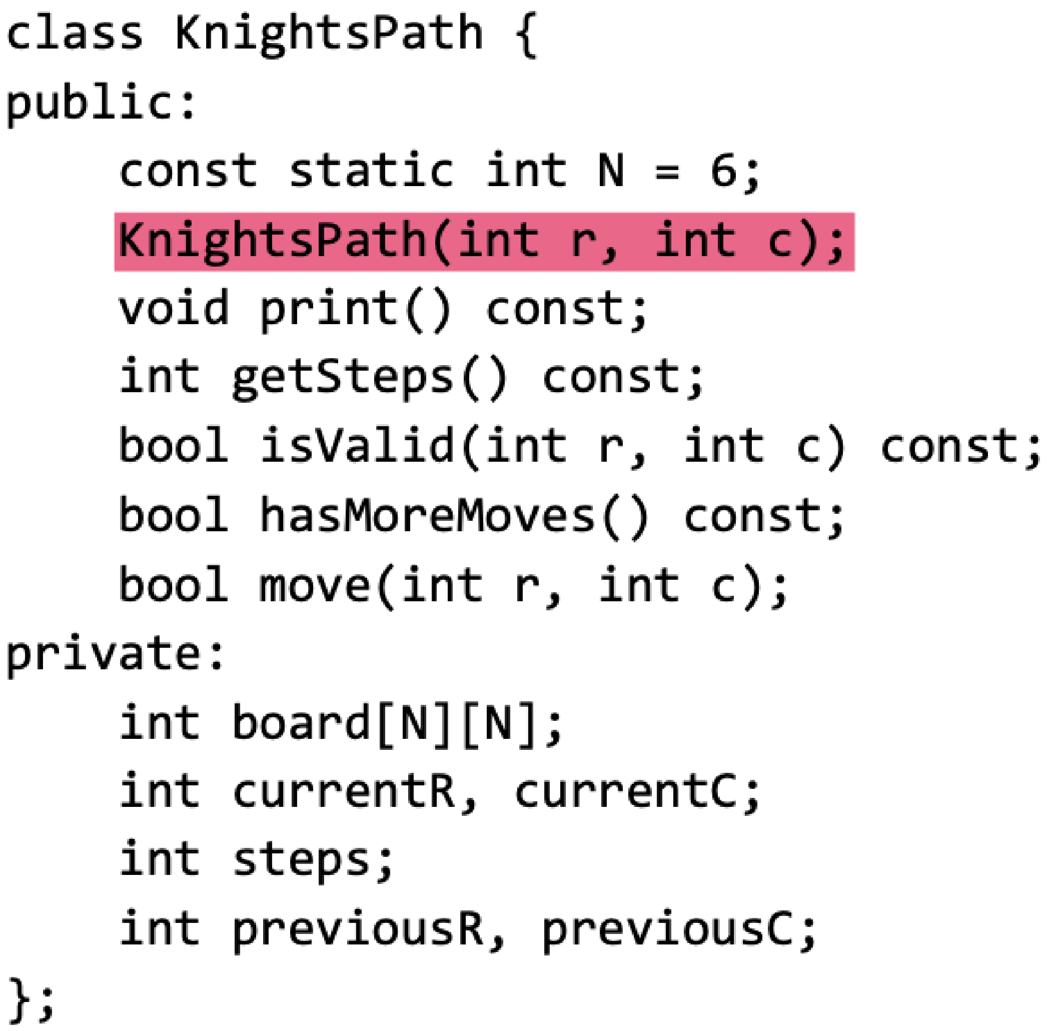


```
class KnightsPath {
public:
    const static int N = 6;
    KnightsPath(int r, int c);
    void print() const;
    int getSteps() const;
    bool isValid(int r, int c) const;
    bool hasMoreMoves() const;
    bool move(int r, int c);
private:
    int board[N][N];
    int currentR, currentC;
   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

Assignment 5: Program Structure

➢ Class Specification: constructor

- **KnightsPath(int r, int c);**
- The data members:
- (a) currentR and currentC shall be initialized using the parameters r and c;
- (b) steps shall be initialized to 0;
- (c) previousR and previousC shall be initialized to -1.

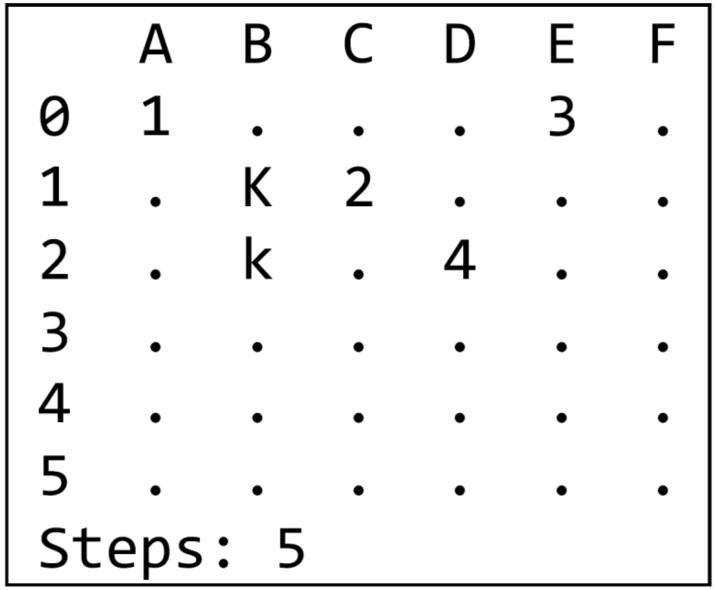


```
class KnightsPath {
public:
    const static int N = 6;
    KnightsPath(int r, int c);
    void print() const;
    int getSteps() const;
    bool isValid(int r, int c) const;
    bool hasMoreMoves() const;
    bool move(int r, int c);
private:
    int board[N][N];
    int currentR, currentC;
   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

Assignment 5: Program Structure

➢ Class Specification: function

- **void print() const;**
- Prints out the drunk knight’s path in the format shown below.



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

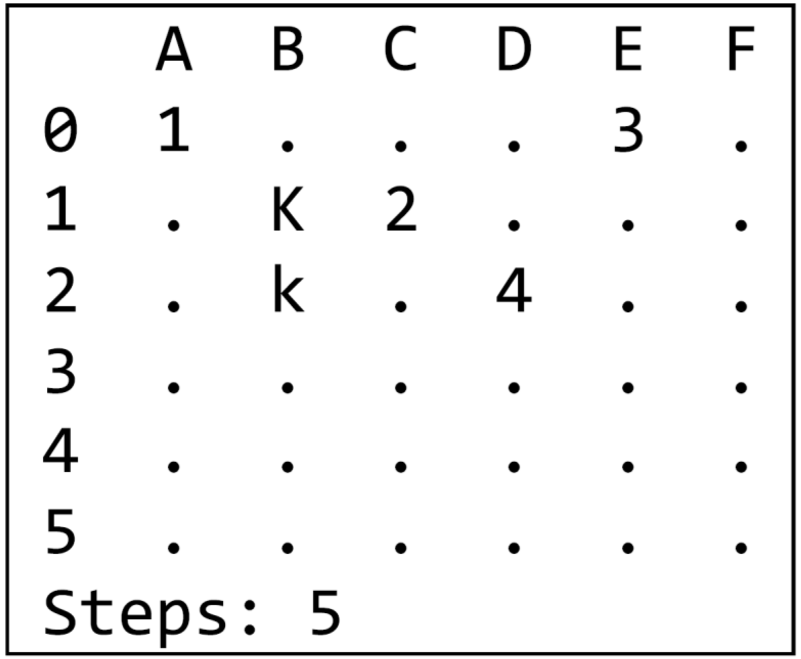
Figure 4: Printing Format of a Drunk Knight's Path

```
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;
};
```

Assignment 5: Program Structure

➢ Class Specification: function

- **void print() const;**
- 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 steps taken is printed below.



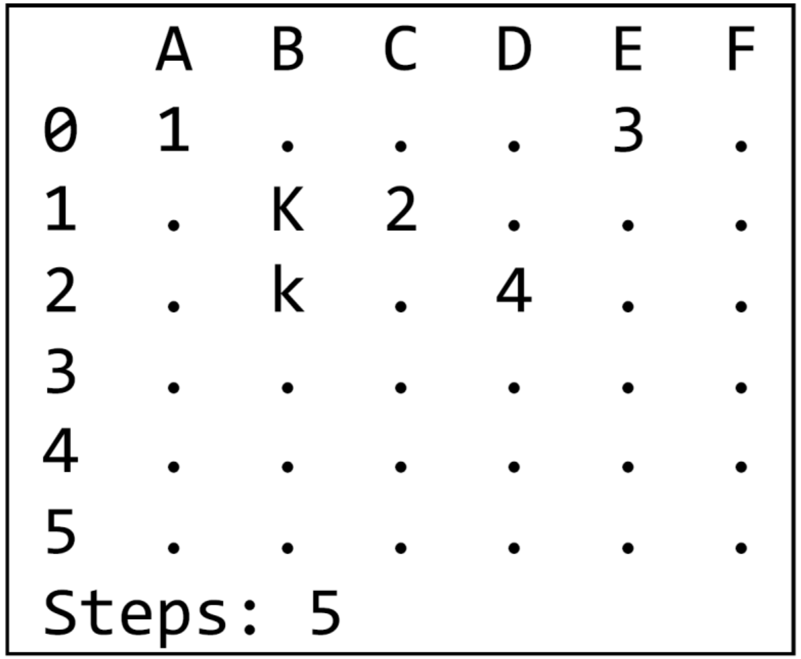
```
  A B C D E F
0 1 . . . 3 .
1 . K 2 . . .
2 . k . 4 . .
3 . . . . . .
4 . . . . . .
5 . . . . . .
Steps: 5
```

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

Assignment 5: Program Structure

➢ Class Specification: function

- **void print() const;**
- Column index:
  - Initialize a constant array of capital letters [‘A’, ‘B’, ‘C’……], so that you can visit the index letter via index number.
  - Type cast: Char a = ‘A’;
    - a + 0 -> 65
    - int(a) -> 65 (ASCII number if ‘A’)
    - a + 1 -> 66
    - char(a + 1) -> ‘B’



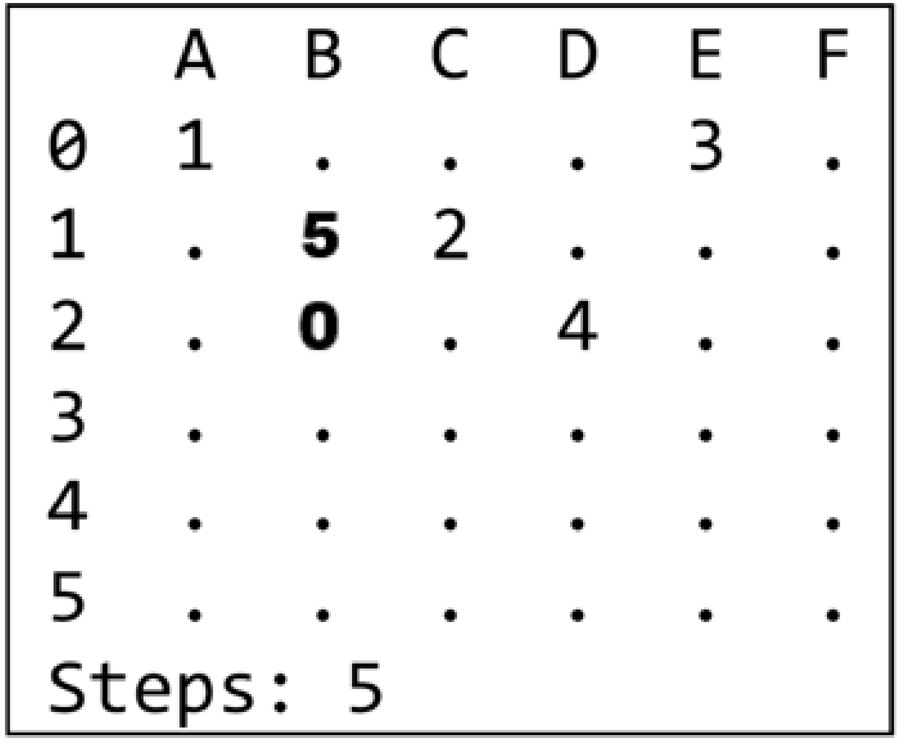
```
   A  B  C  D  E  F
0  1  .  .  .  3  .
1  .  K  2  .  .  .
2  .  k  .  4  .  .
3  .  .  .  .  .  .
4  .  .  .  .  .  .
5  .  .  .  .  .  .
Steps: 5
```

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

Assignment 5: Program Structure

➢ Class Specification: function

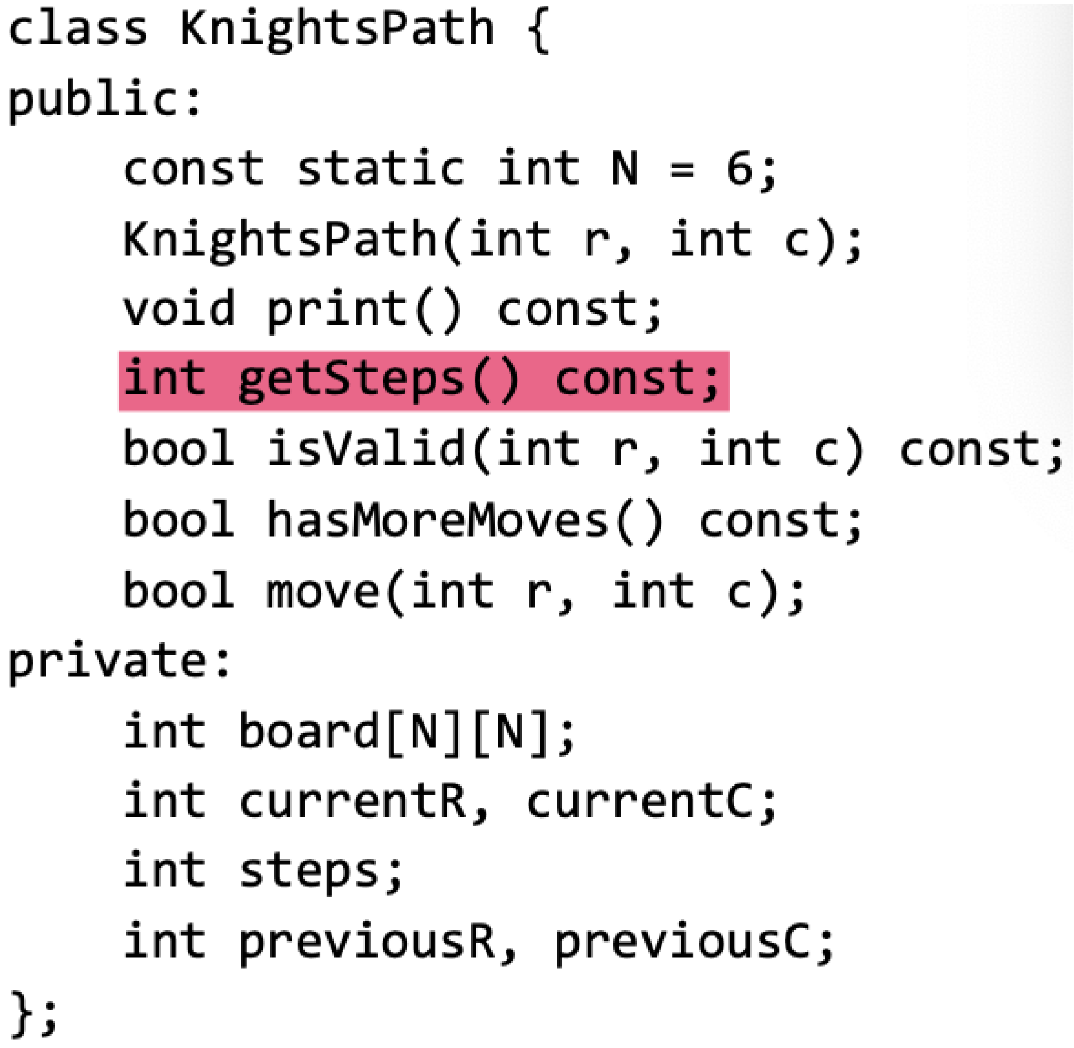
- **void print() const;**
- Currently at the starting position:
- E.g., the current position in board[N][N] is 5.
- The starting position in board[N][N] is 0.
- What if current position and starting position are one? board[r][c]=0 or 5?
- In order to print a lowercase ‘k’ instead of uppercase ‘K’:
  - Use a special number: e.g., board[r][c]=-2 ;
  - Just use 0 cover 5, board[r][c]=0;
  - Don’t use 5 cover 0. Otherwise, you won’t know whether it is starting position.



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

Assignment 5: Program Structure

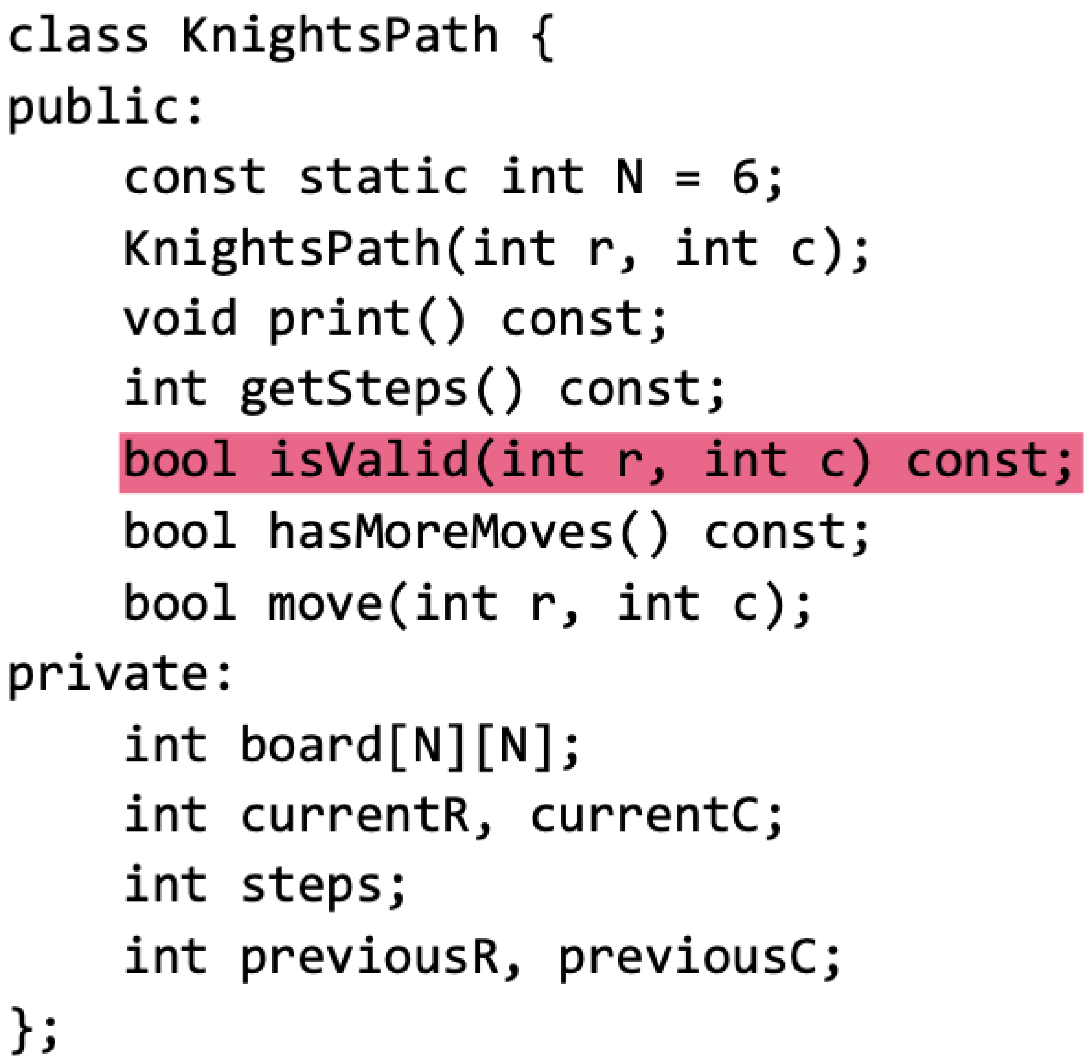
- **Class Specification: function**
  - **int getSteps () const;**
  - Returns the number of steps the knight has walked, that is, the data member steps.



```
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;
};
```

Assignment 5: Program Structure

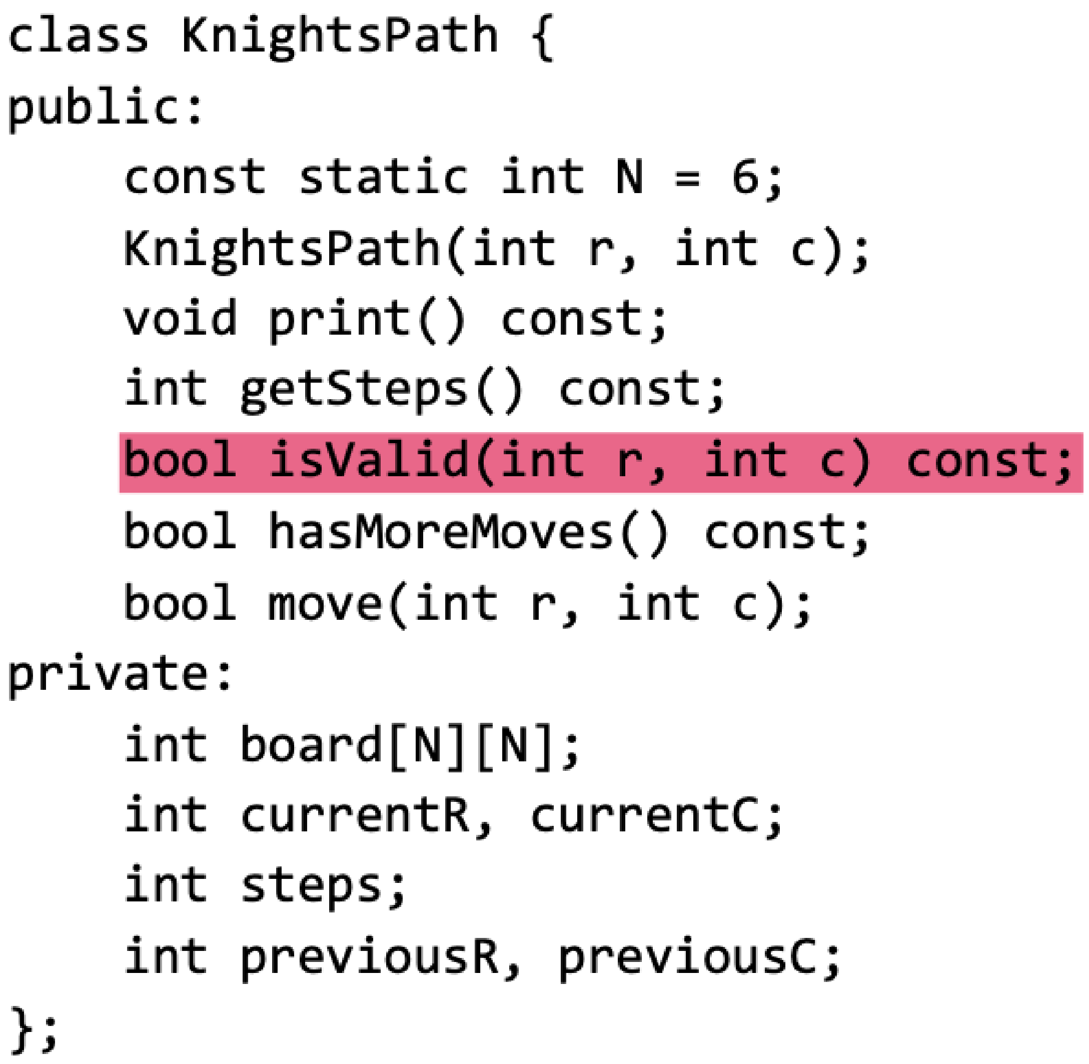
- **Class Specification: function**
  - **bool isValid(int r, int c) const;**
  - Checks whether the knight in the path can be moved from the current position (row currentR, column currentC) to the destination at row r, column c.
  - Note that the knight is not actually moved



```
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;
};
```

Assignment 5: Program Structure

- **Class Specification: function**
  - **bool isValid(int r, int c) const;**
  - Return True if all the following conditions are satisfied, otherwise False:
  - r and c form a proper position within the board.
  - The destination is an unvisited square;
  - The destination is 2H1V or 1H2V from the current position;
  - The destination is not at a back direction.

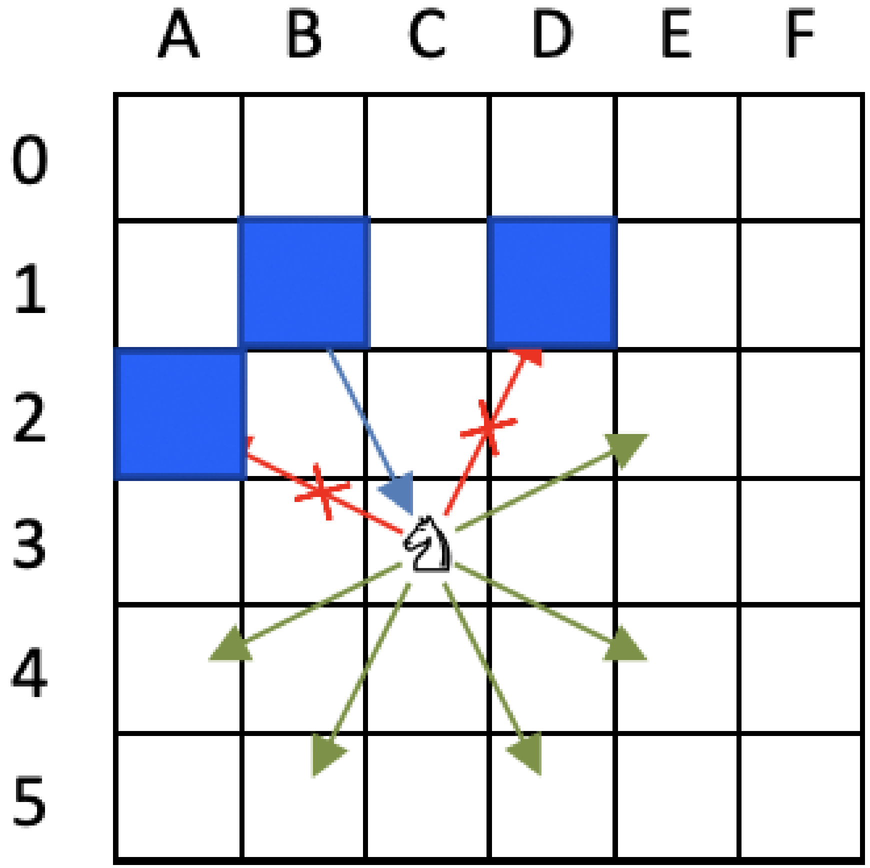


```
class KnightsPath {
public:
    const static int N = 6;
    KnightsPath(int r, int c);
    void print() const;
    int getSteps() const;
    bool isValid(int r, int c) const;
    bool hasMoreMoves() const;
    bool move(int r, int c);
private:
    int board[N][N];
    int currentR, currentC;
   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

Assignment 5: Program Structure

➢ Class Specification: function

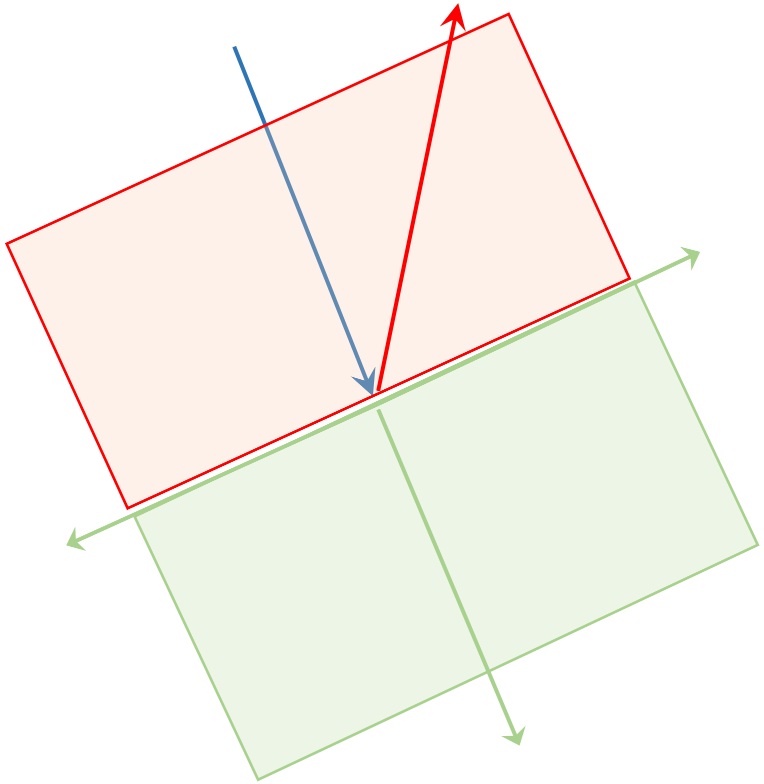
- **bool isValid(int r, int c) const;**
- Possible ways to determine whether a knight is turning back:
  - If the destination is too close to the previous position.
  - If the two vectors from current position to previous position and destination position form an acute angle (銳角, angle < 90°).



Assignment 5: Program Structure

➢ Class Specification: function

- **bool isValid(int r, int c) const;**
- Possible ways to determine whether a knight is turning back:
  - If the two vectors from current position to previous position and destination position form an acute angle (銳角, angle < 90°).
  - a・b > 0 if angle(a, b) < 90°.
- There's no such turning back constraint in the 1st step.



Assignment 5: Program Structure

➢ Class Specification: function

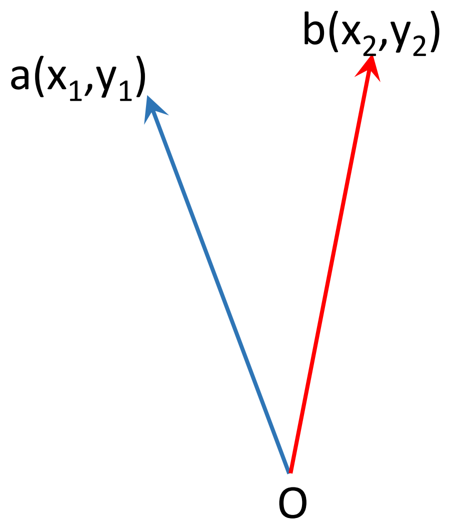
- **bool isValid(int r, int c) const;**
- a • b > 0 if angle(a, b) < 90°.
- Inner product: a • b = $x_1x_2 + y_1y_2$
- E.g., a=(-1,2), b=(1,2):

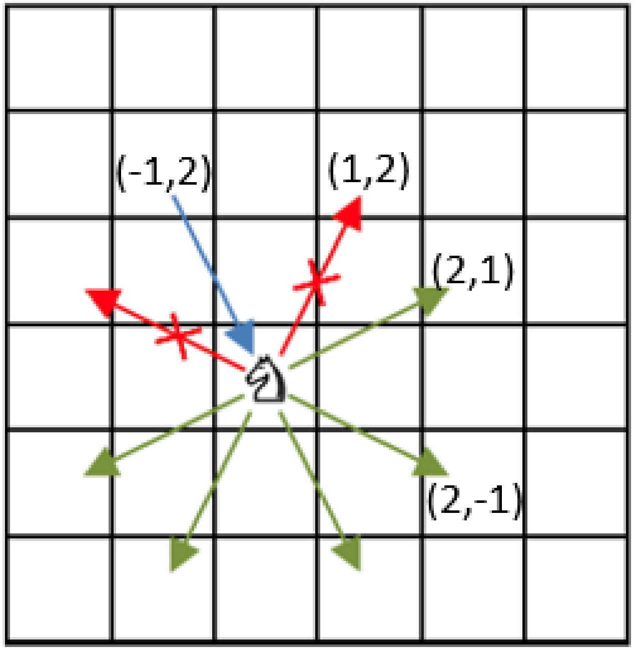
  a • b = 3
- c = (2,1), d=(2,-1)

  a • c = 0

  a • d = -4

- There's no such turning back constraint in the 1st step. (previousR and previousC are initialized to -1)

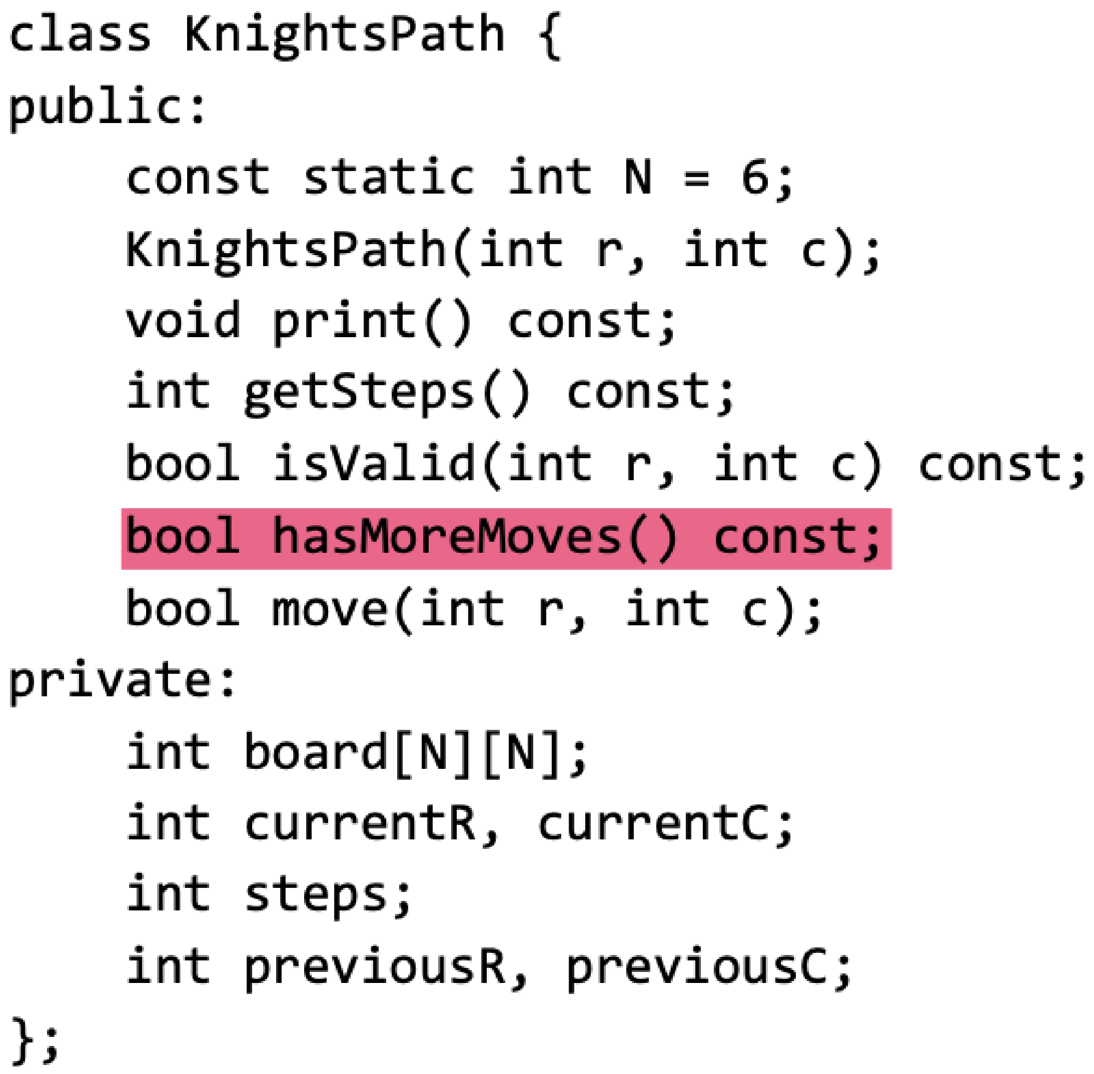




Assignment 5: Program Structure

➢ Class Specification: function

- **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.

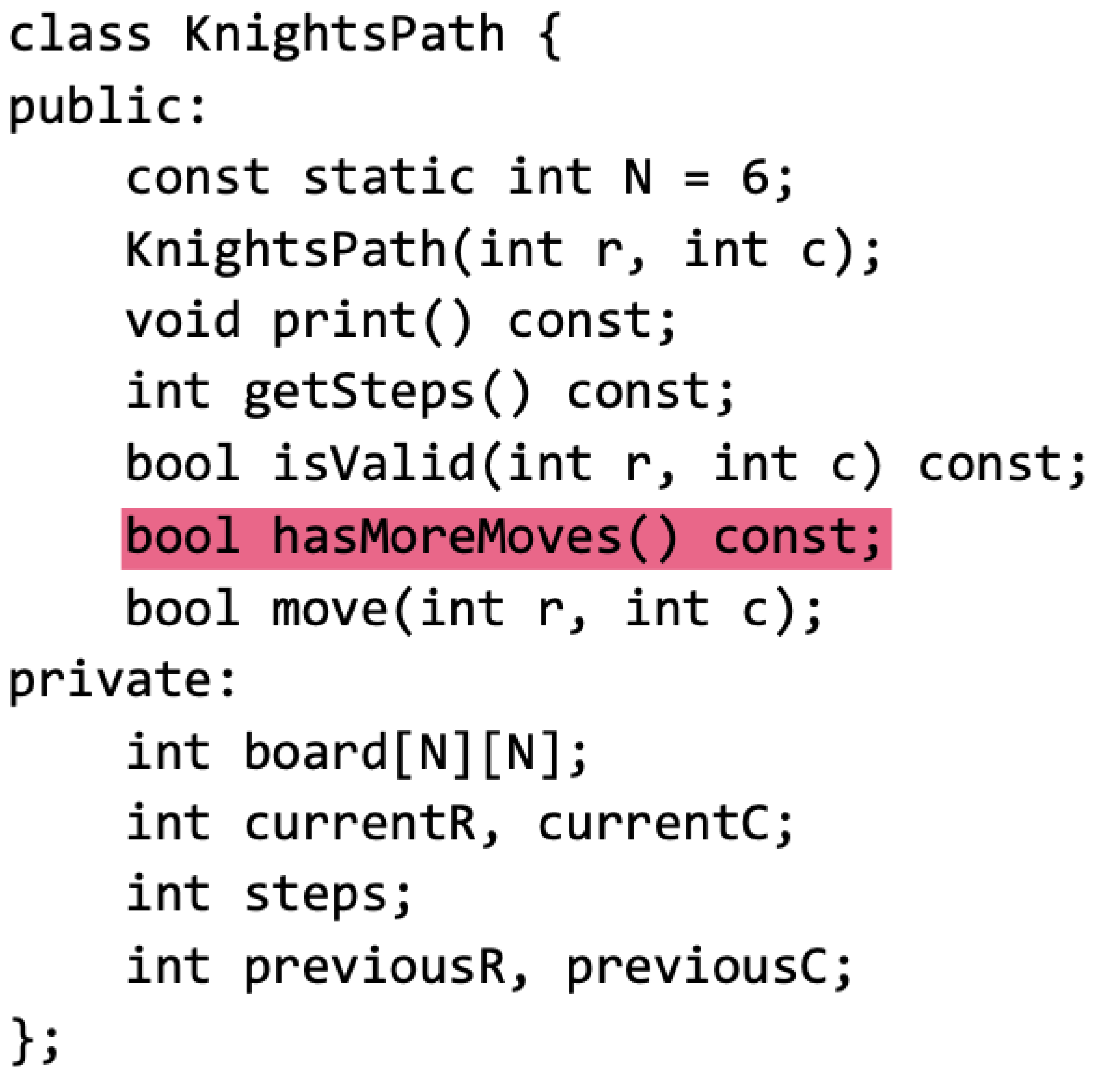


```
class KnightsPath {
public:
    const static int N = 6;
    KnightsPath(int r, int c);
    void print() const;
    int getSteps() const;
    bool isValid(int r, int c) const;
    bool hasMoreMoves() const;
    bool move(int r, int c);
private:
    int board[N][N];
    int currentR, currentC;
   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

Assignment 5: Program Structure

➢ Class Specification: function

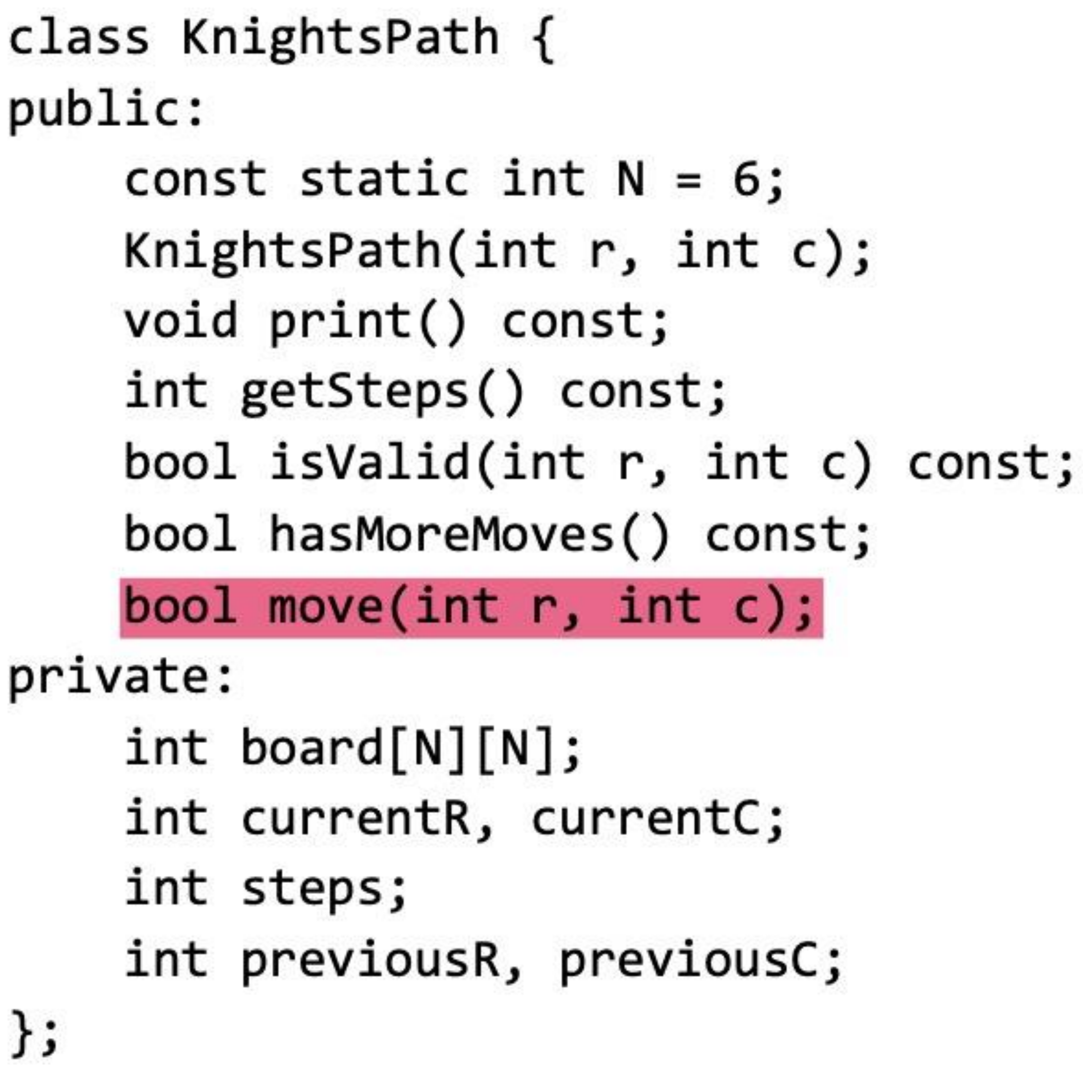
- **bool hasMoreMoves() const;**
- E.g.,
- isValid(...) is the function that check whether a position is valid for the next move.
- If all the possible moves are False by isValid(...), then there’s no more move.
- Check all the destinations with isValid(...).
  - Traverse the whole board;
  - Check positions around;
  - Check 8 L-shape positions.



```
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;
};
```

Assignment 5: Program Structure

- **Class Specification: function**
  - **bool move(int r, int c);**
  - Receive a position(r, c) given by user input and 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.

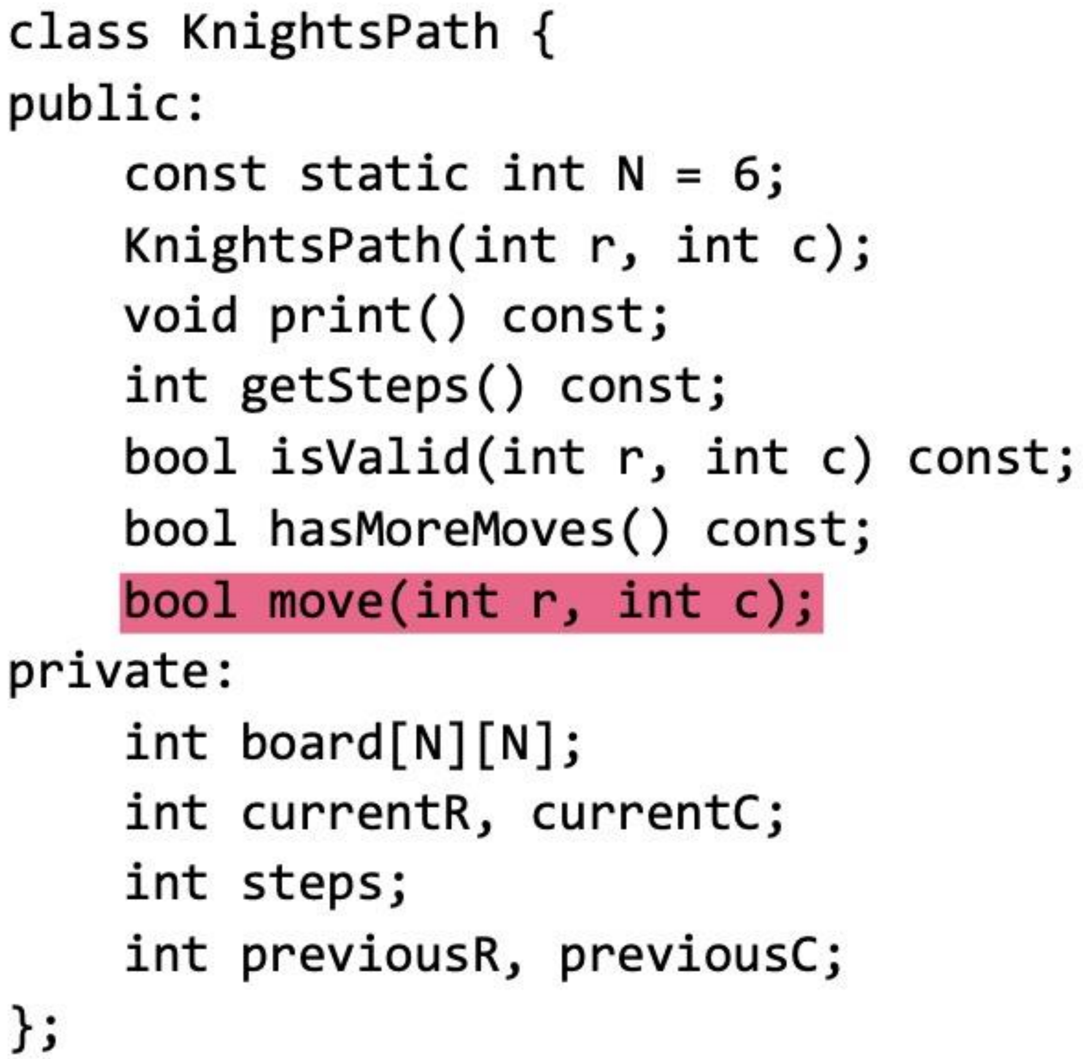


```
class KnightsPath {
public:
    const static int N = 6;
    KnightsPath(int r, int c);
    void print() const;
    int getSteps() const;
    bool isValid(int r, int c) const;
    bool hasMoreMoves() const;
    bool move(int r, int c);
private:
    int board[N][N];
    int currentR, currentC;
   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

Assignment 5: Program Structure

➢ Class Specification: function

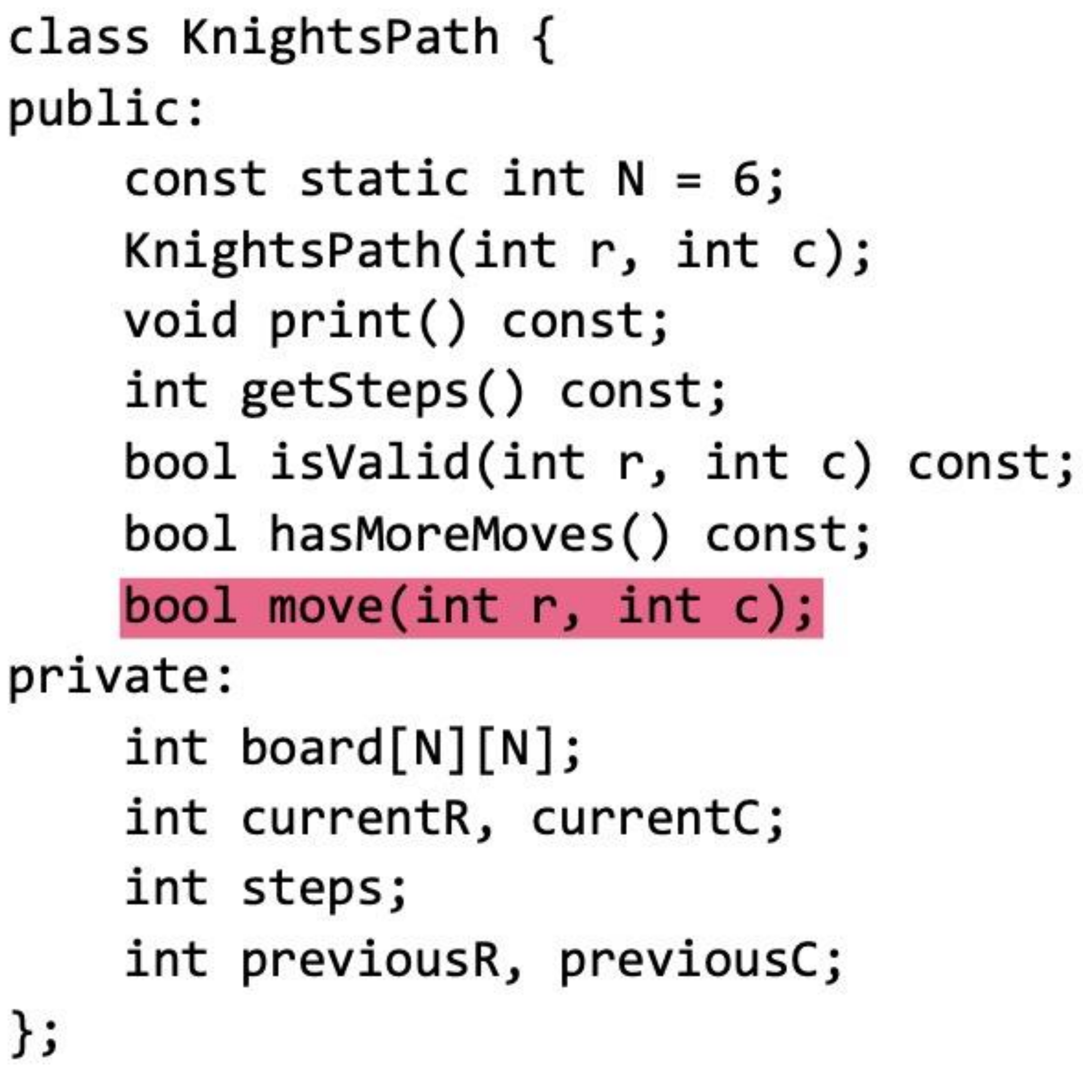
- **bool move(int r, int c);**
- If the destination forms a valid move, this member function shall update the data members board, currentR, currentC, steps, previousR, previousC and return true.
- Otherwise (that is, the move is not valid), this member function shall update nothing and return false.



```
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;
};
```

Assignment 5: Program Structure

- **Class Specification: function**
  - **bool move(int r, int c);**
  - E.g., when receiving an input (r,c), move will call isValid() function to validate the input.
  - If the input is valid, update the data members.



```
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;
};
```

Assignment 5: Program Structure (walk.cpp)

File Specification

- You shall write your program in two source files KnightsPath.cpp and walk.cpp.

- KnightsPath.cpp: implementation of the class KnightsPath.
- walk.cpp: a client program of class KnightsPath which performs the program flow

Assignment 5: Program Structure

➢ 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.

- Starts with user input of initial position.
- Create a KnightsPath object using the input position.
  - Ask for user input of next position. Check if it is valid.
    - Go to the next position and update KnightsPath instance;
    - Otherwise ask for user input of next position again;
  - Check if there remains any valid positions for next move;
  - Ask for user input of next position again.
- End if there’s no valid position left.

Assignment 5: Program Structure

➢ Client Program (walk.cpp)

- Starts with user input of initial position.
- Create a KnightsPath object using the input position.
  - Ask for user input of next position. Check if it is valid.
    - Go to the next position and update KnightsPath instance;
    - Otherwise ask for user input of next position again;
  - Check if there remains any valid positions for next move;
  - Ask for user input of next position again.
- End if there's no valid position left.

- You may call **KnightsPath(int r, int c);**
- Remember to validate the input. It's feasible to embed the validation in constructor **KnightsPath(int r, int c)** or outside.

Assignment 5: Program Structure

➢ Client Program (walk.cpp)

- Starts with user input of initial position.
- Create a KnightsPath object using the input position.
  - Ask for user input of next position. Check if it is valid.
    - Go to the next position and update KnightsPath instance;
    - Otherwise ask for user input of next position again;
  - Check if there remains any valid positions for next move;
  - Ask for user input of next position again.
- End if there's no valid position left.

- You may call **move(int r, int c);**
- As stated, **move()** function will check if the position is valid.

Assignment 5: Program Structure

➢ Client Program (walk.cpp)

- Starts with user input of initial position.
- Create a KnightsPath object using the input position.
  - Ask for user input of next position. Check if it is valid.
    - Go to the next position and update KnightsPath instance;
    - Otherwise ask for user input of next position again;
  - Check if there remains any valid positions for next move;
  - Ask for user input of next position again.
- End if there’s no valid position left.

- After a valid move is made, you may call **hasMoreMoves()** to check if there remains any valid destination left.
- Remember to print the board at each stage.

Assignment 5: Program Structure

➢ Client Program (walk.cpp)

- The most important part is isValid() function.
- Be very careful of the constraints and special cases.
  - r and c form a proper position within the board.
  - The destination is an unvisited square;
  - The destination is 2H1V or 1H2V from the current position;
  - The destination is not at a back direction. (except for 1st move)

Assignment 5: Program Structure

➢ Points to Note: KnightsPath is well wrapped

- 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.
- The KnightsPath class shall not contain any cout statements except in the print() member function (for printing the board).
- The **KnightsPath** class is a **Blackbox**, you can only call the public member functions.

Assignment 5: Program Structure

➢ Points to Note: KnightsPath is well wrapped

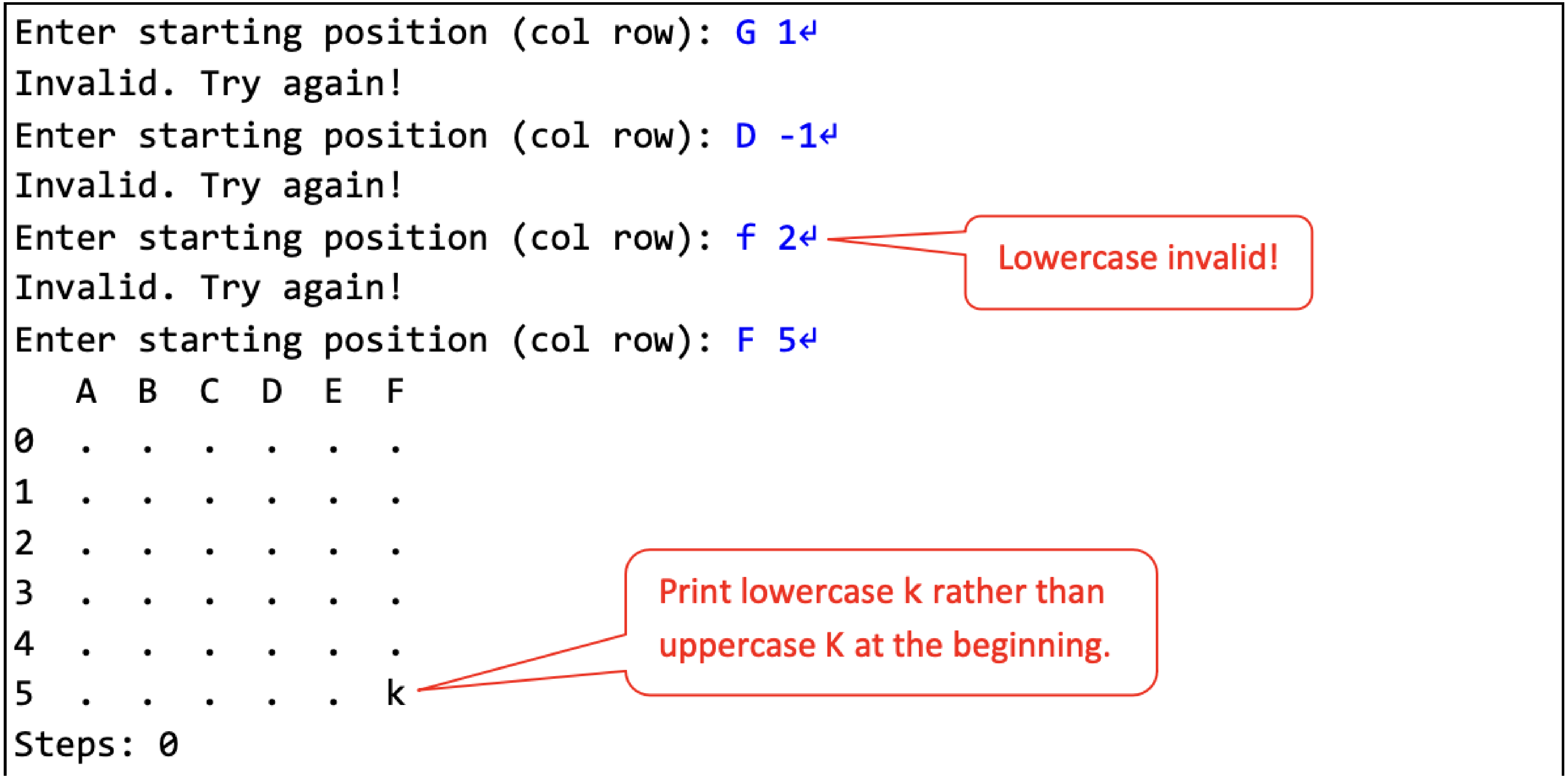
- It could be hard to debug within a well wrapped class,
- Because for each member function you want to check and each member data you want to inspect, you'll have to create an instance of object and call the public functions.
  - Use the debug mode in VS properly;
  - Write and debug the member functions separately outside the class could be easier.

Assignment 5: Program Structure

➢ Points to Note: User interface

- 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.
- You have cout statements in print() and walk.cpp only.

Assignment 5: Sample

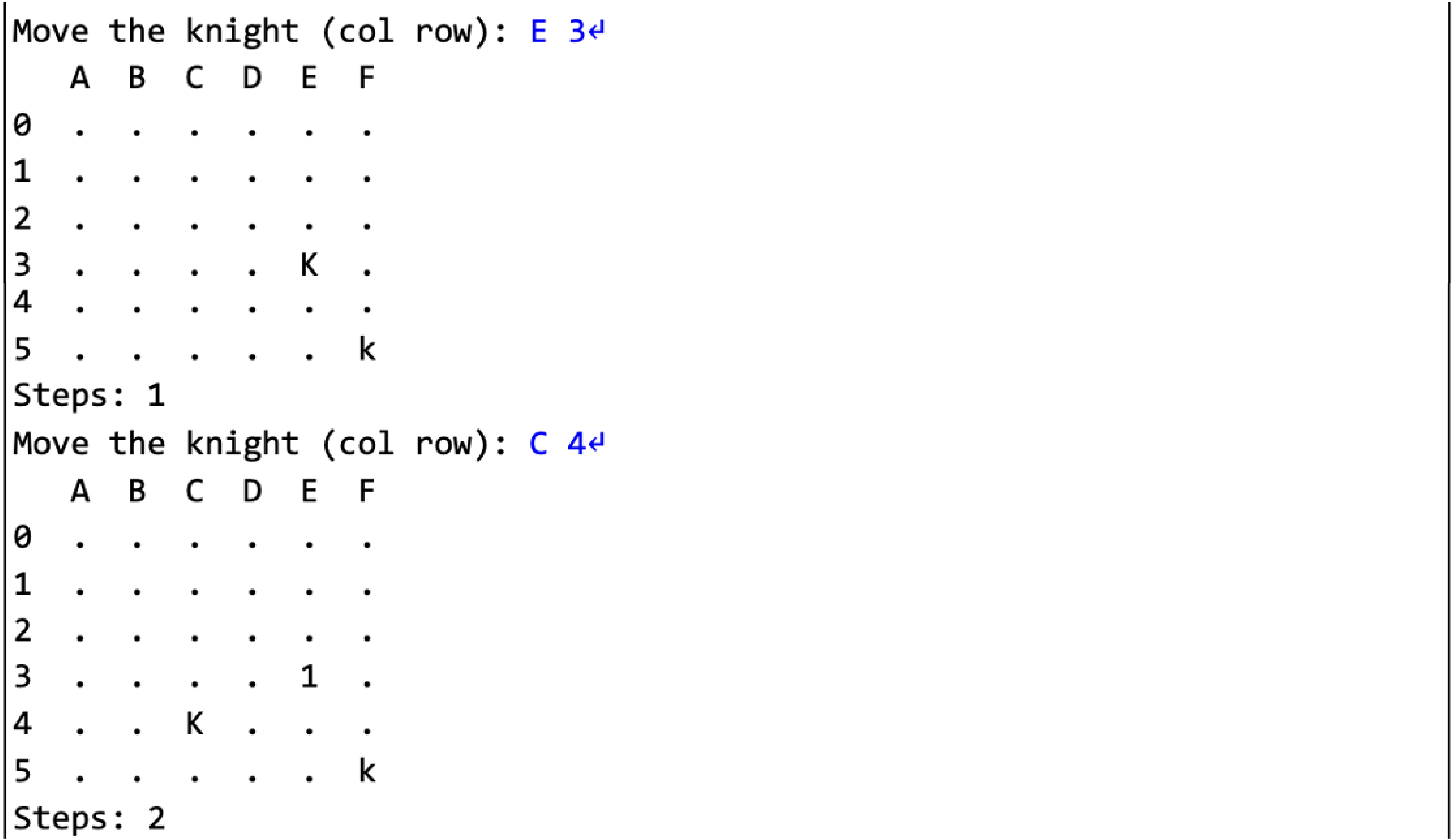


```
Enter starting position (col row): G 1↵
Invalid. Try again!
Enter starting position (col row): D -1↵
Invalid. Try again!
Enter starting position (col row): f 2↵
Invalid. Try again!
Enter starting position (col row): F 5↵
   A  B  C  D  E  F
0  .  .  .  .  .  .
1  .  .  .  .  .  .
2  .  .  .  .  .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

Lowercase invalid!

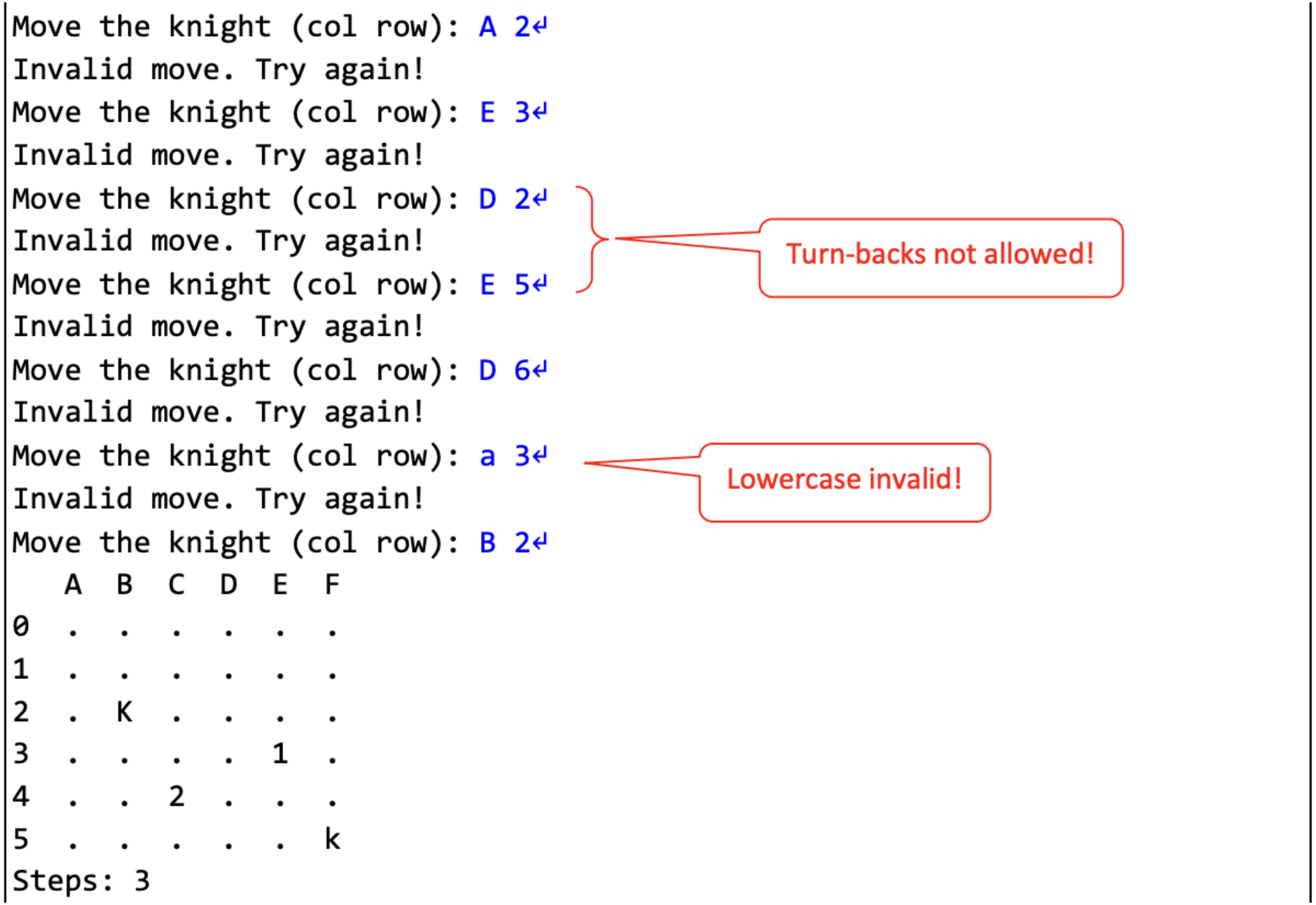
Print lowercase k rather than uppercase K at the beginning.

Assignment 5: Sample



```
Move the knight (col row): E 3↲
   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
```

Assignment 5: Sample

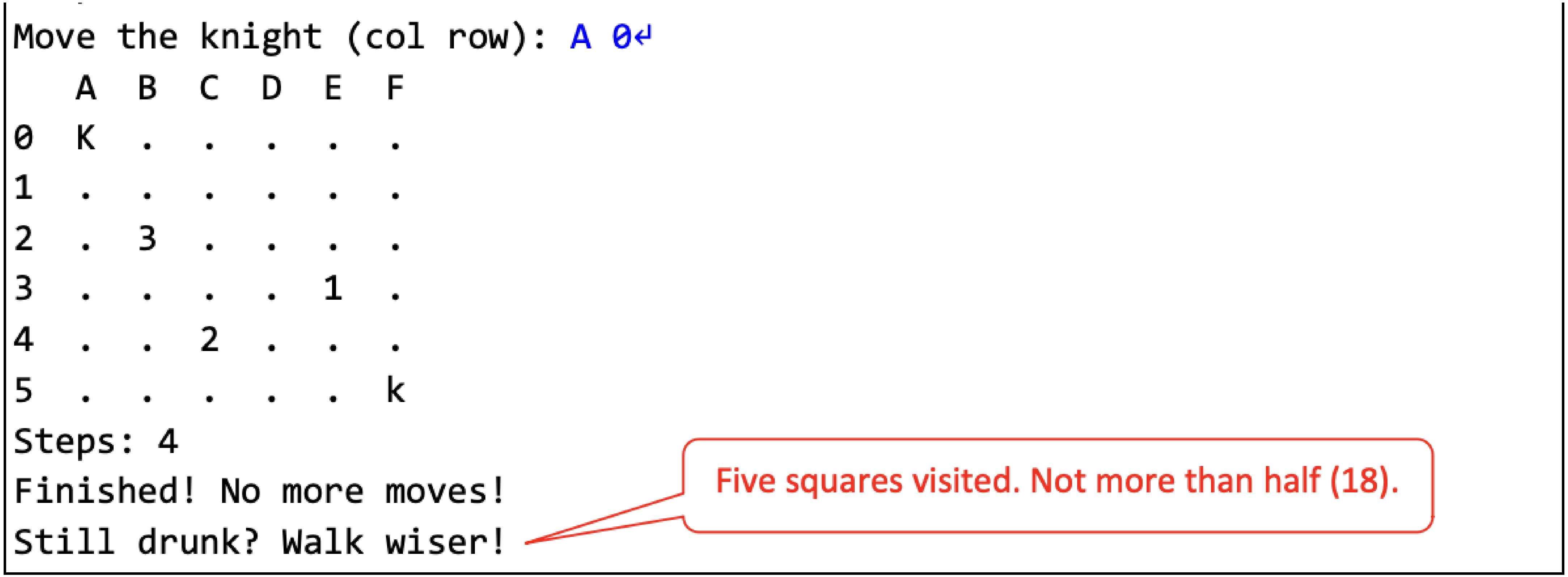


```
Move the knight (col row): A 2↵
Invalid move. Try again!
Move the knight (col row): E 3↵
Invalid move. Try again!
Move the knight (col row): D 2↵
Invalid move. Try again!
Move the knight (col row): E 5↵
Invalid move. Try again!
Move the knight (col row): D 6↵
Invalid move. Try again!
Move the knight (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
```

Turn-backs not allowed!

Lowercase invalid!

Assignment 5: Sample



```
Move the knight (col row): A 0↵
    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).