

# CS3610 Project 2

Dr. Jundong Liu

For this project, you will write a recursive backtracking program to solve the knight's tour problem as described in exercise 19 on page 393 of your textbook "Data Structures Using C++". You have been provided template code that you must modify to complete this assignment. Specifically, you will implement the following two functions:

```
void KnightsTour::move(int row, int col, int& m, int& num_tours) :
```

`move` is a recursive backtracking function that will print all solutions to the knight's tour problem on a chessboard starting from positions `row`, `col`. The total number of tours found is returned in the reference variable `num_tours`.

In this function, `m` is an integer that represents the current move of the tour (moves are labeled starting from 1). It is incremented at the beginning of each call to `move` to indicate at what point along the tour the knight reached the chessboard cell at indices `row`, `col`. In each call to `move`, you will record the value of `m` in the private member variable `board` at position `row`, `col`. Next, you will find all valid knight moves reachable from position `row`, `col` by calling the function `get_moves`. For each new move found from `get_moves`, recursively call `move` to find all remaining tours.

When `m` equals the total number of cells in the private member variable `board`, it means a tour has been completed. Every time a tour has completed, you will print `board` using the provided function `print`.

```
void KnightsTour::get_moves(  
    int row, int col,  
    int row_moves[], int col_moves[],  
    int& num_moves  
) :
```

`get_moves` is used to find all valid knight moves reachable from board indices `row`, `col`. An invalid move would be one that sends the knight off the edge of the board or to a position that has already been visited in the tour.

`row_moves` and `col_moves` are arrays used to store the indices of all new valid moves found. `num_moves` is used to indicate how many valid moves were found. In other words, `num_moves` records the sizes of `row_moves` and `col_moves`.

To ensure that everyone's output remains consistent with the test cases used in grading, check all valid moves in a **CLOCKWISE FASHION** starting from the move in which the knight travels two squares to the right and one square up (i.e., **clockwise, starting roughly at 2 o'clock**).

In this project, you are only required to find tours on a 5x5 chessboard starting at locations specified by the user. The driver program (**main.cpp**) to setup this chessboard and accept user input has already been provided. There is no need to modify it.

## Input

Find all knight's tours for a 5x5 chessboard starting from indices  $0 \leq r < 5$  and  $0 \leq c < 5$ .  $r$  and  $c$  are to be passed in as command line arguments. (A driver program has already been implemented for you).

## Output

Print the solution for every knight's tour found starting from position  $r$ ,  $c$  along with the total number of tours  $t$ . (Remember to call `print` inside `move`).

## Sample Test Cases

### Test Case 1

```
$ ./a.out 3 3
 23   4  11  16  25
 12  17  24   5  10
  3  22   9  18  15
  8  13  20   1   6
 21   2   7  14  19

 21   4  11  16  19
 12  17  20   5  10
  3  22   9  18  15
  8  13  24   1   6
 23   2   7  14  25

...

 21  14   3   8  23
  4   9  22  13   2
 17  20  15  24   7
 10   5  18   1  12
 19  16  11   6  25

Number of tours: 56
```

## Compile and implementation

A *Makefile* has been included in this assignment. The TA will use `g++` compiler to compile your program. For the implmentation, I strongly recommend that you implement the `get_moves()` and have it fully tested first, before you start to implement the `move()` function.

Your `move()` function is expected to follow a similar backtracking design to the  $n$ -queen puzzle solution covered in class. To help you get familiar with the design, I've

attached an implementation of the  $n$ -queen puzzle in this assignment, along with the lecture notes. You can compile and test the code with different  $ns$ . A good understanding of  $n$ -queen puzzle should be very helpful in designing your `move()` function.

## Turn In

Submit your source code through blackboard. If you have multiple files, package them into a zip file.

## Grading

**Total:** 100 pts.

- **10/100** - Code style, commenting, general readability.
- **10/100** - Compiles.
- **10/100** - Follows provided input and output format.
- **70/100** - Successful implementation of the knight's tour backtracking algorithm.