# Assignment 4: 15-Puzzle

Due: 20:00, Wed 23 Nov 2016 Full marks: 100

## Introduction

The *15-puzzle* is a sliding puzzle played on a $4 \times 4$ square board of 15 tiles with one tile missing. The tiles are numbered 1 to 15, placed randomly in the board. The goal of the puzzle is to place the tiles in order (See Figure 1) by sliding the tiles one at a time. In this assignment, you will apply object-oriented programming to write a program to play 15-puzzles.

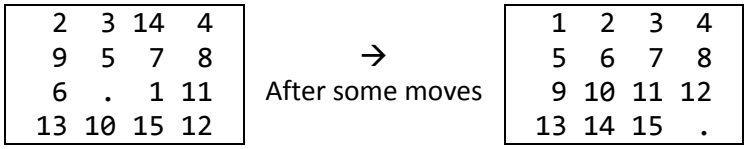


Figure 1: An Initial 15-Puzzle (Left) and A Solved 15-Puzzle (Right)

## Program Specification

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

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

```
class FifteenPuzzle {
public:
    FifteenPuzzle(unsigned s);
    bool move(char dir);
    void print() const;
    bool isSolved() const;
    int getSteps() const;
private:
    int board[4][4];
    int r, c, steps;
};
```

## Private Data Members

### `int board[4][4];`

The 15-puzzle is represented by a two-dimensional array of `int`, storing the numbers 1 to 15 on the tiles. The elements `board[0][0]` and `board[3][3]` denote the top-left and bottom-right corners of the puzzle respectively. The missing tile is denoted by the special value 16.

### `int r, c;`

The position of the missing tile in the 15-puzzle. They are the row and column indices respectively.

### `int steps;`

The number of steps a player has used during puzzle play.

## Public Constructor and Member Functions

### `FifteenPuzzle(unsigned s);`

This constructor creates a 15-puzzle object. The initial configuration of the puzzle is generated as follows.

1. Initialize data member `board` to the solved state. (That is, 1 to 16 from top-left to bottom-right, row by row.)
2. Use parameter `s` as the random seed to initialize the random number generator (by `srand(s)`).
3. Repeat the following 200 times:
   3.1. Generate four random numbers `i`, `j`, `p`, and `q` in order, all between 0 and 3 inclusively, using `rand() % 4`.
   3.2. Swap the contents of `board[i][j]` and `board[p][q]`. Note that it is possible that `[i][j]` and `[p][q]` are the same because of randomness. In such case, the swapping achieves no effect. Nonetheless, it is still counted as one of the 200 times.
4. Set data member `steps` to 0 and set `r` and `c` to the position of the missing tile accordingly.

Note that the above steps may generate an unsolvable 15-puzzle, but it does not matter.

### `bool move(char dir);`

This member function moves the missing tile of the 15-puzzle towards the direction specified in the parameter `dir`. We use the characters 'I', 'M', 'J', and 'K' (and also their lower case equivalents) to denote the up, down, left, and right directions respectively. (See the layout of keys I, M, J, and K on a keyboard.) If the missing tile is on a board boundary and you try to move it beyond the boundary, then no move would be made and the board remains unchanged. If the move is valid, then data members `board`, `steps`, `r`, and `c` have to be updated accordingly. This member function returns `true` if a valid move is made. It returns `false` if no move is made or `dir` is not any of the letters I, M, J, K, i, m, j, and k.

### `void print() const;`

Prints out the 15-puzzle and the number of steps that the player has used. Note that *each tile number is printed with a width of 3* (`setw(3)`), and *the missing tile should be printed as a dot '.' instead of 16*. The following is an example output of `print()`, in which ␣ denotes a space.

```
__1__2__3__4
__5__6__8_11
_10__7__._12
__9_13_14_15
Steps:_26
```

`bool isSolved() const;`

This member function returns `true` if the 15-puzzle is already solved, i.e., the tiles are in correct order; returns `false` otherwise.

`int getSteps() const;`

This member function returns the number of steps that a player has used for the 15-puzzle, i.e., the value of the data member `steps`.

## Client Program (`playgame.cpp`)

Your main program is a client of the `FifteenPuzzle` class; it performs the flow of the 15-puzzle.

1. The program starts by asking the user to enter an integer, which can be assumed to be *non-negative* and is used as a random seed. Then create a `FifteenPuzzle` object using the input seed.
2. Prompt the user to make a move. You can assume that *the user always enter a character for this input*.
3. An input here is valid only if it is 'I', 'M', 'J', or 'K', or their lower case equivalents. When the user enters an invalid input, the program gives a warning message and goes back to Step 2.
4. Move the missing tile of the puzzle towards the input direction.
5. If the puzzle is not yet solved after the move and fewer than 200 moves have been made, go back to Step 2.
6. Once the puzzle is solved, print the solved puzzle and the message "You finished in *XX* steps." where *XX* is the number of steps used.
7. Or if 200 steps have been made and the puzzle remains unsolved, print the unsolved puzzle and the message "You cannot finish in 200 steps.".

## Points to Notice

- Given the same random seed, you should create the same initial puzzle configuration. You can refer to the sample program for some sample initial configurations for some random seeds.
- You can assume that the initial puzzle (after randomization) is not solved yet.
- You are *not allowed* to modify the header file `FifteenPuzzle.h`.

# 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.

```
Enter seed: 123↲
  5  6 12  7
 13 15  2  .
 10  8  3  4
  9 14  1 11
Steps: 0
Move [IMJK]: L↲
Invalid. Try again!
Move [IMJK]: K↲
  5  6 12  7
 13 15  2  .
 10  8  3  4
  9 14  1 11
Steps: 0
Move [IMJK]: j↲
  5  6 12  7
 13 15  .  2
 10  8  3  4
  9 14  1 11
Steps: 1
Move [IMJK]: m↲
  5  6 12  7
 13 15  3  2
 10  8  .  4
  9 14  1 11
Steps: 2
Move [IMJK]: M↲
  5  6 12  7
 13 15  3  2
 10  8  1  4
  9 14  . 11
Steps: 3
Move [IMJK]: J↲
  5  6 12  7
 13 15  3  2
 10  8  1  4
  9  . 14 11
Steps: 4
Move [IMJK]: I↲
  5  6 12  7
 13 15  3  2
 10  .  1  4
  9  8 14 11
Steps: 5
```

*(Invalid input.)*

*(Moving the missing tile beyond boundary results in no change.)*

*(Lower case letters i/m/j/k are allowed.)*

```
Move [IMJK]: I↵
  5  6 12  7
 13  .  3  2
 10 15  1  4
  9  8 14 11
Steps: 6
Move [IMJK]: K↵
  5  6 12  7
 13  3  .  2
 10 15  1  4
  9  8 14 11
Steps: 7
     .
     .    (Inputs skipped to save space. Complete output available in Blackboard.)
     .
  1  2  3  4
  5  6  7  8
  9 10  . 11
 13 14 15 12
Steps: 86
Move [IMJK]: K↵
  1  2  3  4
  5  6  7  8
  9 10 11  .
 13 14 15 12
Steps: 87
Move [IMJK]: M↵
  1  2  3  4
  5  6  7  8
  9 10 11 12
 13 14 15  .
Steps: 88
You finished in 88 steps.
```

## Submission and Marking

- Your program file names should be `FifteenPuzzle.cpp` and `playgame.cpp`. Submit the *two files* in Blackboard (https://elearn.cuhk.edu.hk/). You do *not* have to submit `FifteenPuzzle.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 *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*.
- *Plagiarism is strictly monitored and heavily punished if proven.* Lending your work to others is subjected to the same penalty as the copier.