Implement a C program to solve the 15-puzzle problem using the A* search algorithm.

1. Objectives

- To gain more experience on using pointers and linked lists in C programs.
- To learn how to solve problems using state space search and A* search algorithm.

2. Background

A* search and 15-puzzle problem have been introduced in the class. For more information, please read the wiki page of 15-puzzle problem at https://en.wikipedia.org/wiki/15_puzzle, and the wiki page of A* search at https://en.wikipedia.org/wiki/A*_search_algorithm.

Solving a 15-puzzle problem needs to keep swapping the locations of the blank tile and a tile adjacent to it (i.e., above/below it or to the left/right of the blank title), such that in the end all the tiles are moved from their initial locations to their goal locations.

In the assignment, the blank title is labeled with number 0, and other titles are labeled with non-zero numbers from 1~15. The goal locations are as shown below, and the initial locations are provided through program arguments by listing tile indexes in a row-major order. Refer to page https://en.wikipedia.org/wiki/Row-_and_column-major_order, if you need to understand what row-major order is.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	0

For example, the command

```
./puzzle  2  3  0  4  1  6  7  8  5  9 10 12 13 14 11 15
```

is to solve a 15-puzzle problem, in which the tiles are initially placed as follows, and are to be moved to their goal locations shown above:

2	3	0	4
1	6	7	8
5	9	10	12
13	14	11	15

The execution of the program will print out the state transitions from the initial state to the goal state as a solution. For the above 15-puzzle problem, the following states will be printed out on the screen. Note that, to save space, this document uses two columns to show the output, with the column on the left followed by the column on the right and the line separating the two columns. Your program only needs to print the states in one column, and does not need to show the line.

```
Path (length=9):

 2  3  0  4
 1  6  7  8
 5  9 10 12
13 14 11 15

 2  0  3  4
 1  6  7  8
 5  9 10 12
13 14 11 15

 0  2  3  4
 1  6  7  8
 5  9 10 12
13 14 11 15

 1  2  3  4
 0  6  7  8
 5  9 10 12
13 14 11 15


 1  2  3  4
 5  6  7  8
 0  9 10 12
13 14 11 15

 1  2  3  4
 5  6  7  8
 9  0 10 12
13 14 11 15

 1  2  3  4
 5  6  7  8
 9 10  0 12
13 14 11 15

 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14  0 15

 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15  0
```

You may use the attached program ***GenGemPuzzle.c*** to generate initial states. The program starts from the goal state (i.e., all the tiles are at their goal locations) and moves tiles randomly. It takes an integer as its argument (example shown below), which is the number of moves it makes before it finishes.

```
./GenGemPuzzle 20
```

Usually, with a larger the argument, the program can generate an initial state that is more different is from the goal state; thus, when solving the problem, more moves are needed to move the tiles to their goal locations, and your program needs more time to finish.

3. Other Instructions on Programming

- The command line to run the program should have 16 ***unique*** integers between 0 and 15 (including 0 and 15) as tile indexes. The tile indexes are separated by spaces.

./puzzle index0 index1 … index15

- Your program needs to print out a sequence of states showing the movement the tiles, or text "*no solution*" if a solution cannot be found.
- If your program cannot print the state transitions in correct order (i.e., from the initial state to the goal state), it can still get partial credits if it can print the state transitions in reverse order (i.e., from the goal state to the initial state). But your program needs to print out an message as a note:

```
NOTE: the station transitions are in reverse order.
```

- You can improve the skeleton program attached with the assignment.
- The program will use intensively pointers and linked lists. Though you may print messages to stderr to help you trace its execution, gdb will be more helpful in debugging.

4. Instruction on Testing

Test 1. The program can solve the problem if the initial state matches the goal state. It should run with the following command and print out the goal state as the solution within 1 minute.

./puzzle 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0

Run the program with the arguments above and check the execution outputs.

Test 2. The program can solve a simple 15-puzzle problem. The initial state can be generated using command `GenGemPuzzle 4`. Run the program with the arguments corresponding to the initial state, and then check the execution outputs. The program should print out a correct solution.
Test 3. The program can solve a complex 15-puzzle problem. The initial state can be generated using command `GenGemPuzzle 40`. Run the program with the arguments corresponding to the initial state, and then check the execution outputs. The program should print out a correct solution.