The goal of this assignment is to write a program to solve the 8 puzzle problem. In order to achieve this, you need to implement the A* algorithm discussed in class. The program will take input for the start state and goal state configuration interactively from the user. Your program should internally maintain the following coordinate system as shown in Figure 1 for each of the 9 square tile positions (including the empty tile). The coordinate \((i,j)\) refers to the \(i\)th row and \(j\)th column, where each \(i\) and \(j\) can take three possible values: 0, 1, 2. For example, to get the input for the top left square of the start configuration from user, which is 4 in Figure 2, your program should say, something like,

```
Enter value for the tile located at (2, 0) coordinate position for start configuration: 
```

In return, user must enter 4 (to get a start configuration similar to the one shown in Figure 2). The empty tile should be represented by 0.

![Figure 1: 8 puzzle coordinate system](image1.png)

```
Figure 1: 8 puzzle coordinate system
```

```
Figure 2: Sample start and goal state
```

The evaluation function \(f\) that you need to use for A* algorithm will be of the following form for node/state \(n\):

\[
f(n) = g(n) + h(n)
\]

where \(g(n)\) is the cost to reach to state/node \(n\) starting from the start state. **Note that in each step the blank can be moved only one place which will cost** 1. The heuristic function \(h(n)\) on the other
hand guesses the cost to reach to the goal state from the current state/node $n$. For this programming assignment you need to use “Manhattan distance” to compute $h(n)$.

One way to compute Manhattan distance is as follows. For each coordinate position of your current state figure out what digit it contains. Next identify the coordinate of the same digit in goal state. Subtract one coordinate position from other and take the absolute sum. For example, suppose the current state is the start state. Consider the coordinate $(2,0)$ in current state. It contains the digit 4. Now in the goal state digit 4 is in coordinate $(1,1)$. So the resulting coordinate difference is $(1, 1) - (2, 0) = (-1, 1)$. Now take absolute value of $|(-1, 1)| = |-1| + |1| = 2$. What this says is that, since tiles can move only along the horizontal or vertical directions, the difference of tile positions of digit 4 between current state and goal state is 2. Now, in order to get the value of $h(n)$ for the current node, you need to do what we have just explained for each coordinate position of current state that doesn’t contain a blank and sum them up. If you do that, then you will see that the value of $h(n)$ for the current state, assuming current state is the start state is 6. So $f(n) = g(n) + h(n) = 0 + 6 = 6$, assuming $n$ is the current state.

Essentially, you need to implement the algorithm given in Figure 3.14, (page # 84) in your textbook, where path cost $\text{PATH-COST}$ for any node $n$ will be evaluation function $f(n)$ as we have described here.

**Input-Output :**

- As mentioned before your program should interactively ask for start and goal state configuration from user.
- Once you have found a solution you must show the tile and blank positions in each step. For example, say you start from the start state and then move the blank to the right in the next step and eventually reach goal state after say 15 steps etc.. Then your output should look like:

```
step 0:
  4 2 5
  3 0 1
  6 7 8
step 1:
  4 2 5
  3 1 0
  6 7 8
```

... 

```
step 15:
  0 1 2
  3 4 5
  6 7 8
```

- Since you will have parent information of each node, you can also print the output backward from goal state to start state for each step.

**Suggestion :**
In order to check if your program is working correctly, start with simple input first. For example, choose any arbitrary start configuration. Now move the empty tile say 3 steps from this configuration and use the resulting configuration as goal configuration. Clearly, there is an optimal solution at depth 3 from the root node. See if your program can find it. Once your program works for such simple examples, you can try and see if it works for little harder examples.
What to submit:

- All files that are necessary to run your program (you can write your program in any language you want).

- A README file that will explain very clearly how to compile/execute your program. **Without this README file your program will not be evaluated.** As a part of the README file, you should also provide a start and goal configuration of the 8-puzzle problem for which **your program works.** The grader will check your program using other random start and goal configurations as well.

- You can write the program in your favorite programming language. However, the only restriction is that, your program must compile/execute in department Linux environment. **It is your responsibility to make sure that whatever you submit do compile/execute in department Linux environment. Otherwise, you will receive zero.**

- **Extra credit:** You will receive extra credit if your program is generalized to solve the \((n^2 - 1)\)-puzzle problem. In this case, your program will seek an input \(n\) from the user. For example if \(n = 5\), your program will need to solve the 24-puzzle problem. In this case, both start and goal states will be specified by \(5 \times 5\) rectangular grids and the new coordinate system will also be represented by a \(5 \times 5\) rectangular grid, which is a simple extension of Figure 1.

Some points you will need to consider:

- Name and user ID (myWSU ID) must appear on top of each program that you submit

- You should be able to login ion to the LINUX-based systems within the department network using your myWSU ID and password.

- Use the handin command to submit your files:

Submission:

- You are to hand in your program files electronically. **YOU will be responsible for making sure that all necessary files are turned in on time and that your program works correctly on any of the LINUX-based systems within the department network.**

- Assignments after deadline (03/22/2016) will be accepted till a cutoff date (03/24/2016) after which they will be rejected. **Late submission (after 03/22/2016) penalties: 10 points each day.** Each assignment is due by 11:59PM on the specified day of deadline.

- **Hand ins:** Suppose you want to submit three files program1.c, program2.c and README. Submit your files using the following command:

  ```bash
  ~/cs771/bin/handin 1 prog1.c prog2.c README
  ```