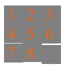
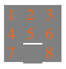
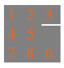
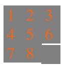
1. Week 7/8: The 8-tile Puzzle

1.1 The 8-Tile Puzzle

The Chinese 8-Tile Puzzle is a 3 × 3 board, with 8 numbered tiles in it, and a hole into which neighbouring tiles can move: 

After the next move the board could look like:


, or: 

In this assignment, you are going to read in a board and find the correct moves to return it to the 

state. To do this you will use a list of boards. The initial board is put into this list. Each board in the list is, in turn, read from the list and all possible moves from that board added into the list. The next board is taken, and all its resulting boards are added, and so on. One way of implementing this would be to use a queue. However, one problem with is that repeated boards may be put into the queue and 'cycles' occur. This soon creates an explosively large number of boards (several million). You can solve this by only adding a board into the queue if an identical one does not already exisit in the queue. A linear search is acceptable for this task. Each structure in the queue will contain (amongst other things) a board and a record of its parent board, i.e. the board that it was created from.

Be advised that a solution requiring as 'few' as 20 moves may take 10's of minutes to compute.

Exercise 1.1 Write a program that:

- Reads in a board from a file (`argv[1]`), and checks that it is valid.
- Prints out the correct solution and the number of moves required. Make sure that this is **not** in reverse order, but the solution from the starting board to the end.
- Use an array to form the queue - don't use any dynamic types (i.e. avoid malloc etc.). One big array of structures, each containing a board (etc.) is required.

Test your code on, amongst others, the following boards:

2		3
1	4	6
7	5	8

(5 moves)

5	1	3
2	7	6
	4	8

(10 moves)

2	5	8
4	3	1
	6	7

(20 moves)

■

70%

Exercise 1.2 when a 'winning' solution is possible, display an animated display of this with the aid of ncurses. Make the display more 'interesting' than simply a 3x3 grid - use different colours, bigger tiles, borders etc. ■

10%

Exercise 1.3 Rewrite your code, so that it makes no assumptions about the number of moves required. To do this, use a dynamic data structure (i.e. one that uses malloc etc.). ■

10%

When you submit the above exercises, make sure that the command:

```
gcc *.c -Wall -Wextra -Wfloat-equal -ansi -pedantic -lm -lncurses -O2 -o eighttile
```

compiles your code correctly, and that the file containing `main()` is called `eighttile.c`. Do not submit multiple versions of your code (e.g. `eighttile_part1.c`, `eighttile_part2.c` etc.), the first three parts of the assignment are additive - simply expand the functionality each time.

Exercise 1.4 Attempt an extension of your choosing This could be to do with speeding up your code, improving the search, or something else. ■

10%

If you do an extension, do not confuse me by putting it in the same place as the other three parts ! Instead, create a directory called `Extension` and put all relevant files inside this. This will include all necessary source files and a short plain-text file called `extension.txt` which explains what the extension is.