

Assignment 1: Solvability of the NxN sliding tile puzzle




The sliding tile puzzle is a game that requires you to move tiles on a board. The board is $N \times N$, and there are $N-1$ tiles numbered from $1..N-1$ that occupy the board. There is hence 1 location on the board that is empty (referred to as a blank).

There is some (arbitrary) start configuration of the numbered tiles on the board. Starting with this configuration, the aim is to move tiles until some chosen goal configuration is reached, and to do this in the least possible number of moves. You may only move a tile into the blank if the tile neighbours the blank. Moves can be only be in the horizontal and vertical directions (not diagonal).

The sliding-tile puzzle also has other names, such as the 8 Puzzle (for the special case of a 3×3 board) or 15 Puzzle (a 4×4 board) and so on. Sometimes the name N Puzzle is used (indicating an $N \times N$ board).

You can play the game online at:

 [sliding-tile puzzle](#)

In this assignment you are asked to write a C program that determines whether a given puzzle is solvable. (Note that you do not have to actually solve the puzzle.)

There are some conditions that you should strictly adhere to:

1. the program reads text from *stdin* with a format described below (we will be auto-testing your program with our input so you must conform to this format)
2. your program should be able to handle any sized board, starting with 2×2
 - note that the size of the board is determined by the number of tiles on the input
3. if the input is correct and the goal configuration is:
 - reachable from the start configuration, your program should generate the output text *solvable* (to *stdout*)
 - not reachable from the start configuration, your program should generate the output text *unsolvable* (to *stdout*)

4. if the input is not correct, your program should generate an error message (to *stdout*)
5. if a system call fails in your program, the program should generate an error message (to *stderr*)
6. design and programming restrictions:
 - you are not allowed to use any arrays
 - you are not allowed to use linked lists/trees/graphs
 - you should use an ADT to represent the board and operations on the board

Input format

Two lines of text on *stdin* specifies the start and goal configurations, read from left to right, top to bottom. Each line consists of a sequence of integers, separated by any number (>0) of blanks and/or tabs, that represent the tile numbers, and a single letter *b* to represent the blank space on the board. These integers should of course be in the range $1..N-1$ where the board is of size $N \times N$. The first line specifies the start board, the second line the goal board. For example:

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

represents a sliding-tile puzzle on a 4×4 board with the tiles initially placed on the board as shown in the image at the top of page. The goal configuration has the tiles ordered row by row.

In the case of incorrect input

Checking the correctness of each configuration is vital. For example, an input line may not represent an $N \times N$ board, or the blank may be missing, or one or more of the tile numbers $1..N$ may be missing, or the 2 boards may not be the same size or the input contains something other than a number or *b*. There may be more possibilities.

If the configuration is erroneous, your program must generate an appropriate error message (to *stdout*). Note that it is possible to have more than one error, and in that case you only need to generate a single error message. For example, consider the configuration *1 2 b 1*. There are 2 errors in this configuration: the tile 3 is missing, and the tile 1 is duplicated. It does not matter which error is detected by your program, just as long as the error is correct. When an error is detected and reported, your program should exit gracefully, with status *EXIT_FAILURE*. The text you use in error messages should be informative, but please keep it brief.

In the case of correct input

If the input is correct, the program should write the following 3 lines to *stdout*:

- the text *start*: followed by the start configuration
- the text *goal*: followed by the goal configuration
- the text *solvable* or *unsolvable* as appropriate

The output for the 4×4 game above is for example:

```
start: 9 12 5 4 2 b 7 11 3 6 10 13 14 1 8 15
goal: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 b 15
solvable
```

The output for a game on a 2×2 board that happens to be unsolvable is:

```
start: 2 1 3 b
goal: 1 2 3 b
unsolvable
```

If the input is correct the program should exit with *EXIT_SUCCESS*.

Design

You should make an ADT to implement the puzzle. The *client*, which is the main program, calls functions in the ADT to read the input, check for correctness and determine solvability. The interface between the client and the ADT is a header file.

Marking

Marks will be deducted if you fail any of our tests for incorrect input, or incorrectly determine the solvability of the puzzle. Marks will also be deducted for poor design (e.g. not using an ADT), poor programming practice or violating any of the rules above.

The assignment is worth 10 marks.

Submission

You should submit exactly 4 files:

1. a *Makefile* that generates the executable *puzzle* (you should use the *dcc* compiler)
2. the C source code of the ADT (call it *boardADT.c*)
3. the header file of the ADT (*boardADT.h*)
4. the main program (*puzzle.c*)

To submit the assignment, use the command:

```
~cs9024/bin/classrun -give assign1 Makefile boardADT.c boardADT.h
puzzle.c
```