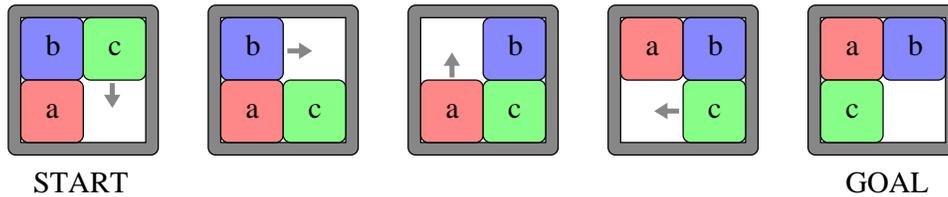


CSCI-1200 Computer Science II — Fall 2007

Homework 6 — Sliding Block Recursion

In this assignment you will write a recursive solver for simple two-dimensional *sliding block puzzles*. A sliding block puzzle consists of a frame and a set of rigid pieces that fit, but do not completely fill, the space inside the frame. The goal of the puzzle is to transform the pieces from an initial configuration to an end configuration by sliding the pieces within the free space of the frame. The pieces cannot be removed or lifted or allowed to overlap in any way. For example, the puzzle below can be solved in 4 moves:



For this assignment, we will restrict our puzzles to rectilinear frames with integer dimensions, so they may be represented with a 2D rectangular grid. Furthermore, the pieces will also have integer dimensions and right angles, allowing them to be described and stored on a 2D grid. However, the pieces are not necessarily rectangular; for example, we can have an 'L' shaped piece. In solving the puzzle, the pieces will move one piece at a time, one grid unit at a time, in one of the 2 axes (up, down, left, or right), and are not allowed to rotate. **Please read the entire handout before beginning your implementation.**

There are many excellent websites dedicated to this deceptively simple genre of mechanical puzzles. We encourage you to browse the web and play with some of the online versions of these puzzles. Some great examples include:

- <http://www.johnrausch.com/SlidingBlockPuzzles/>
- <http://www.puzzles.com/PuzzleLinks/SlidingBlockPuzzles.htm>

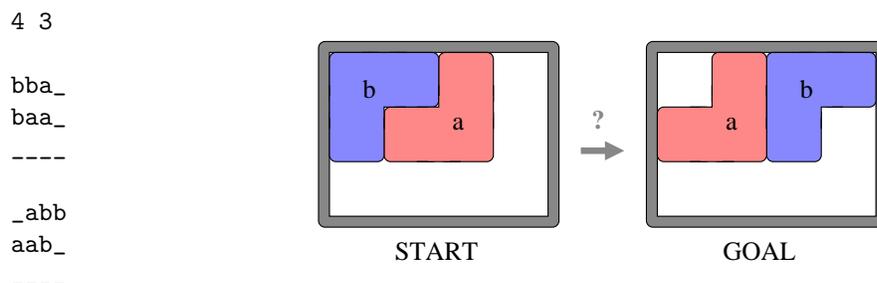
But as usual, you are not allowed to search for, study, or use in any way anyone else's code or pseudocode to complete this assignment!

Your Tasks

For this assignment you will write a C++ program that uses recursion to solve simple sliding block puzzles. Your program will read the starting and ending configurations from a text file that is specified on the command line. An optional argument, the maximum move search depth, may also be specified on the command line. Here are some sample command lines:

```
sliding_block.exe puzzle1.txt
sliding_block.exe puzzle2.txt -maximum_moves 10
```

The input file will contain the dimensions (width & height) of the puzzle frame, and then an ascii representation of the starting and ending configurations. Each piece will be identified with an alphabetic character. Empty space will be represented with the '_' character. For example, the data file on the left corresponds to the puzzle shown on the right:



If the puzzle can be solved, your program should output the details of the solution. If there are multiple solutions, your program should output a solution with the fewest moves. For example, the output for the first puzzle will print out like this:

```
Puzzle can be solved in 4 moves:
****
*bc*
*a_*
****
Move piece 'c' down
****
*b_*
*ac*
****
Move piece 'b' right
****
*_b*
*ac*
****
Move piece 'a' up
****
*ab*
*_c*
****
Move piece 'c' left
****
*ab*
*c_*
****
```

But not all puzzles can be solved! The second example puzzle is impossible. If the puzzle cannot be solved, your program should print out this message:

```
Puzzle cannot be solved.
```

If a maximum move search depth has been specified, and no solution with that number of moves or fewer exists, your program should print out a message like this:

```
Puzzle cannot be solved with <= 10 moves.
```

See the sample puzzles and their corresponding output on the course page.

Algorithm Analysis

This is a really hard problem! We don't expect your implementation to solve all sliding block puzzles. Your program will probably run slowly, hog lots of memory, and possibly crash on "moderate"-sized problems. In fact, you might be able to solve the problems faster by hand than with your program! We'll provide a range of test cases of increasing difficulty for you to throw at your program. It's OK if your program doesn't finish all of them.

In your `README.txt` file discuss the order notation of your program in terms of `w` and `h`, the dimensions of the frame; `p`, the number of pieces; `e`, the number of empty spaces in the frame; `m`, the number of moves in the best solution; and any other relevant variables. You should make up your own test cases and include them with your submission. Summarize the results of your testing, which test cases completed successfully and the approximate "wall clock time" for completion of each test. The UNIX/cygwin `time` command can be prepended to your command line to estimate the running time:

```
time sliding_block.exe puzzle1.txt
```

Additional Information

You must use recursion for this homework submission. You may use any of the data structures and techniques we have discussed so far in this course. Do all of your work in a new folder named `hw6` inside of your CSII homeworks directory. Use good coding style when you design and implement your program. Be sure to make up new test cases to fully test your program and don't forget to comment your code! Use the template `README.txt` to list your collaborators and any notes you want the grader to read. When you are finished please zip up your `hw6` folder exactly as instructed for the previous assignments and submit it through the course webpage.

Extra Credit Contest!

We will hold a contest to find the best sliding block puzzle solver program. You must use recursion for the homework assignment, but you are not required to do so for the contest. Entries must include a plaintext `README_contest.txt` file explaining their optimizations. Extra credit will be awarded for all entries that compile and run the basic test cases. Entries will be judged based on their performance (accuracy *and* speed) on a variety of test cases. The winners will be announced during lecture (date TBA) and the winners must be present to claim their prizes.

To enter the contest, prepare your entry in a folder named `hw6_contest` and submit it on the homework submission site. The usual homework late day policy applies to the non extra credit homework submission. All contest entries are due on Saturday October 27th by 11:59pm (this will not count against your late day total).