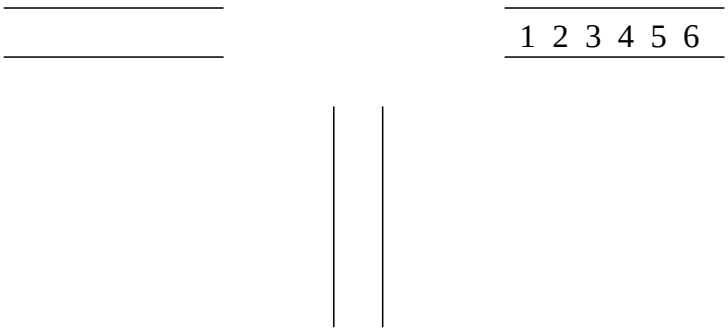
Assignment 1 Railroad/Stack Permutations Due: October 1

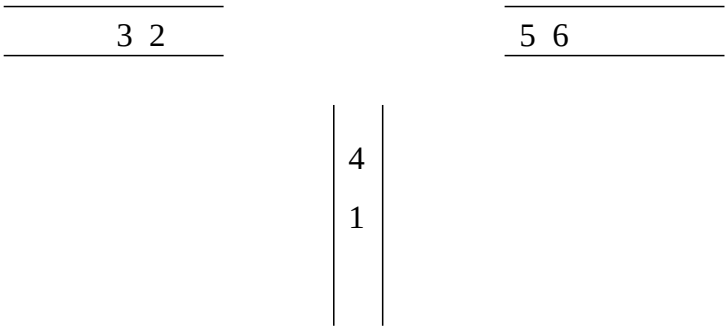
You have been hired by Fly-by-night, Inc. to write a computer program to manage their freight terminal. Pictured below is the terminus of a railroad line. The horizontal tracks are **one-way** right to left, but the vertical track is two-way. Initially the right track contains six cars numbered 1 through 6 as shown.



Suppose we wish to move the cars onto the left track so that they are arranged 3 2 4 1 6 5. If "R" means move a car off the right track and "L" means move a car onto the left track, then the following sequence of moves will do it.

R R R L L R L L R R L L

For example, after the first six of these moves the picture is:



On the other hand, the arrangement 3 1 2 4 6 5 is not obtainable from the initial position, for after "R R R L" there is no way to get car number "1" next on the left track.

The 3 tracks can be viewed as 3 stacks, say stkL, stkM, stkR (left, middle and right). The initial set-up amounts to making all three stacks empty and then pushing the numbers 6 through 1 onto SR. From that point on, an allowable move amounts to

R -- Pop(SR, i) then Push(SM, i) (move from right to middle)
L -- Pop(SM, i) then Push (SL,i) (move from middle to left)

Write a Java Class **Railroad.java that contains a method rrSwitch** that simulates the movement of the railroad cars. It will have as input **an array of six integers that gives the desired arrangements of cars on the left track** which are to be obtained from the initial position **1 2 3 4 5 6** on the right track. It will determine the sequence of moves that moves as many cars as possible to the left track in the desired arrangement. It will then return that sequence of moves along with the state of the three stacks when the program is done making progress toward the solution.

For example, the input: **3 2 4 1 6 5** would return the following array of four strings:

1. the smallest set of legal moves (letters R or L separated by spaces) that give the best achievable set of cars on the left track. That is, do **not** move any cars from Right track to the Middle track after it is clear that no more cars can be moved to the Left track.
2. The arrangement on the Left track -- digits separated by spaces.
3. The arrangement on the Middle track -- digits separated by spaces.
4. The arrangement on the Right track -- digits separated by spaces.

R R R L L R L L R R L L
3 2 4 1 6 5
empty string
empty string

the input: **3 1 2 4 6 5** would return the following as an array of four strings

R R R L
3
2 1
4 5 6

You are required to use stacks and the only allowable operations on the stacks are the ones from the stack interface described in lab. You must implement a class stack from scratch (i.e. using a linked implementation) and use it in your program. Your program **must** simulate the actual movement of the cars by pushing and popping the stacks as described above. Notice that after filling the cars on SR, the one-way restrictions say that SR can only be **popped** (followed by a push onto SM) and that SL can only be **pushed** (following a pop from SM).

Note that your program will have to detect immediately when it first becomes clear from the state of the stacks the situation where the arrangement is unobtainable.

NOTES:

- In lab on Tues. you will implement the Stack class that you will use in your program.
- By Wednesday at 6:00 p.m. you will submit a document showing the test cases you will use to test your program. (You do not have to provide test cases for the Stack class - only for the logic of the simulation.) In the same document you will show the **pseudo code for the logic of the simulation** not the Stack class. This must be typed and fit on a single page. The pseudo code must use indentation to emphasize the flow of control.
- Your program should assume that the input is a permutation of the digits 1 to 6 in some specified format.
- Make sure your program is as simple as possible. Even after you have it running correctly, think about how to make the design and implementation as understandable and simple as possible!