Programming Assignment #1

General Instructions

You are to write a program that performs the task outlined below. You may write your code in Python, Java or C++. Other programming languages may be acceptable; check with the instructor before beginning work in any other language. All work must be your own, and may not have been previously submitted for any other class.

To submit your work, create a *ReadMe.txt* file that clearly explains how to compile and run your code. Archive your source files, data files (if any) and the ReadMe.txt file into a *tar* or *zip* file, and email that file to CS540@cs.colostate.edu. If the TA cannot make your program compile and run based solely on the instructions in the ReadMe file, you will not receive full credit. The amount of the point deduction will depend on how often he has to contact you and how difficult it is for him to get it to run. If the TA cannot get your program to run even after trying to contact you, you will receive a grade of 0 for the assignment.

The Performance Task

You will write a program that reads two files: the first contains the initial state for a virtual "blocks world", and the second contains a goal state. Your program should output (to the terminal) a sequence of commands (one per line) that will change the state of the blocks world from the initial state to the goal state. Your program should create the sequence of commands using A* search, as defined in Russell & Norvig (and CS440).

The blocks world consists (at least conceptually) of a set of blocks on a table. A blocks world configuration specifies a set of blocks, their properties, and (some of) their relations to each other. In particular, every block is denoted by a symbolic identifier of the form *block#,* where # is any positive integer. Different configurations will have different numbers of blocks, and the numbers in the identifiers need not be sequential. The goal and target configurations must contain the same blocks.

Properties of blocks are specified by statements of the form *(*has *block-id property-name value-string).* "Has" is a fixed symbol denoting that the statement is a property, not a relation. *Block-id* is an identifier of the form *Block#* as discussed above, and specifies which block has the property. *Property-name* is an identifier specifying the property. For this assignment, there is only one property: *color.* But future assignments may add additional properties, such as *size, shape* or *location*. *Value-string* is a string that contains the value of the property. In the case of color, it is simply a string interpreted as the name of a color, such as red, green, blue, yellow, cyan, etc. Any string is acceptable (although I don't know what color *foo* denotes). For example, a file might begin with the statement *(has block3 color red),* in which case the configuration contains a block called *block3* that is red.

Relations are specified by statements of the form *(*is *block-id block-id relation-name).* For example, a file might contain the statement *(is block3 block4 on-top-of).* This says that block3 is on top of block4. For this assignment, we have two relations: *on-top-of* and *side-by-side.*

An example configuration file might begin with:
(has block3 color red)
(is block3 block4 on-top-of)
(is block5 block4 side-by-side)

Note that not all properties need be specified. We know that block3 is red. We do not know the color of the other two blocks, and that is OK. There are, however, some physical constraints. The first is that if block1 and block5 are side-by-side and block2 is on top of block1 and block6 is on top of block5, then block2 and block6 are also side-by-side. Alternatively, if block1 and block5 are not side-by-side, then neither are block2 and block6. The second is that blocks can only be side by side if they are the same *height.* What is height? If a block is not on top of any other block, its height is 0. If a block is on top of another block, its height is one more than the height of the block it is on top of. But note that height is not an explicit property or relation given in the configuration file; it must be inferred. For the purposes of this assignment, you may assume that the initial and goal configurations are legal.

Note too that not all relations are given in the configuration. If the configuration file in the example above says the block1 and block5 are side-by-side, it doesn't need to say that for block2 and block6. You should be able to infer that.

Your program produces a sequence of commands. There are only two types of commands at the moment, each with pre-conditions and post-conditions. The first command is *(*command *slide-to block-id block-id).* The slide-to command is how you make two blocks side-by-side. Its preconditions are (1) that both blocks have height 0 and (2) the second block has fewer than four other blocks side-by-side with it. The post-conditions are (1) the first block is side-by-side with the second block and (2) the first block is no longer side by side with any block except the second. Note that on-top-of relations are unaffected.

The second command is (command *stack block-id block-id-or-table)*. This is how blocks are stacked. The preconditions are that neither block has any other block on top of it. The post conditions are that the first block is now on-top-of the second block, and the first block is no longer on-top-of any other block. *Block-id-or-table* is either a block-id or the special symbol *table*, meaning that the block is put on the table and is not on-top-of or side-by-side with any other block.

Grading

We will test your program on multiple test cases (pairs of initial/goal configurations). On each test, 75% of the points will be awarded for correct solutions, while 25% of the points are determined by the quality of the solution. (Although points may also be deducted if the program is hard to compile or run, see the general instructions section above.)