Programming Assignment #2

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

As in PA1, 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 run in under 1 hour, no matter the input is.

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-strings)*, but now there is an additional property name (compared to PA1), *location,* and there may be more than one value string. The color property still takes a single value string (the color name), but the location property takes three strings, all of which should be numbers. The first is the x coordinate, the second is the y coordinate, and the last is the z coordinate (i.e. height off the table). In order to be legal, the height of a block must be (1) zero (i.e. on the table), (2) one greater than the height of another block at the same (x, y) location (i.e. it must be on another block), or (3) anything, if the block is currently grabbed (i.e. being held by the avatar). Also, no two blocks can occupy the same (x, y, z) location.

Relations are specified as in PA1: 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. There are two relation types: *on-top-of* and *side-by-side.*

Initial configuration files must contain the location property for every block in the file, and may not contain explicit relations. (Your program should infer relations where appropriate.) Target configurations may leave locations unspecified and may specify relations.

Unlike PA1, target files may also contain wildcards, of the syntactic form *wildcard#.* A wildcard can be bound to any block name, but must be consistent. In other words, if *wildcard8* appears in two relations (or a relation and a property), it must be bound to the same block in both cases. Since colors cannot be changed, a statement such as *(has wildcard2 color red)* means that wildcard2 can only be bound to a block that has the property red.

Your program produces a sequence of commands, but the commands have changed. One command is *(command grab block-id)*. The pre-conditions for this command are: No block is on top of block-id, and no other block is currently grabbed. The post-condition is that block-id is now grabbed. Another grab is *(command release block-id).* The pre-condition is that block-id is currently grabbed. The post-condition is that it is no longer grabbed. The third command is *(command carry block-id deltax deltay deltaz),* where deltax, deltay and deltaz are elements of {-1, 0, 1}. The pre-conditions for the actions are that block-id is currently grabbed, and there is no other block at position x+deltax, y+deltay, z+deltaz, where (x,y,z) is the location of block-id. The post-condition is the location of block-id is now (x+deltax, y+deltay, z+deltaz). The final command is *(command slide deltax deltay)*, where the deltas are from {-1,0,1}. The preconditions for slide are that location x+deltax, y+deltay, z is unoccupied, the height of block-id is 0 (i.e. its z location is 0), and no block is currently grabbed. The post-condition is that the location block-id is now (x+deltax, y+deltay, z), and the locations of any other block with the same x and y values are also incremented by deltax and deltaz (i.e. if blocks are stacked, the whole stack slides).

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.