Many areas of Computer Science use simple, abstract domains for both analytical and empirical studies. For example, an early AI study of planning and robotics (STRIPS) used a block world in which a robot arm performed tasks involving the manipulation of blocks.   
In this problem you will model a simple block world under certain rules and constraints. Rather than determine how to achieve a specified state, you will "program" a robotic arm to respond to a limited set of commands.   
The problem is to parse a series of commands that instruct a robot arm in how to manipulate blocks that lie on a flat table. Initially there are n blocks on the table (numbered from 0 to n-1) with block bi adjacent to block bi+1 for all 0 <= i < n-1 as shown in the diagram below:

The valid commands for the robot arm that manipulates blocks are:   
  
move a onto b   
where a and b are block numbers, puts block a onto block b after returning any blocks that are stacked on top of blocks a and b to their initial positions.   
  
  
move a over b   
where a and b are block numbers, puts block a onto the top of the stack containing block b, after returning any blocks that are stacked on top of block a to their initial positions.   
  
  
pile a onto b   
where a and b are block numbers, moves the pile of blocks consisting of block a, and any blocks that are stacked above block a, onto block b. All blocks on top of block b are moved to their initial positions prior to the pile taking place. The blocks stacked above block a retain their order when moved.   
  
  
pile a over b   
where a and b are block numbers, puts the pile of blocks consisting of block a, and any blocks that are stacked above block a, onto the top of the stack containing block b. The blocks stacked above block a retain their original order when moved.   
  
  
quit   
terminates manipulations in the block world.   
  
Any command in which a = b or in which a and b are in the same stack of blocks is an illegal command. All illegal commands should be ignored and should have no affect on the configuration of blocks.

Input

The input begins with an integer n on a line by itself representing the number of blocks in the block world. You may assume that 0 < n < 25.   
The number of blocks is followed by a sequence of block commands, one command per line. Your program should process all commands until the quit command is encountered.   
  
You may assume that all commands will be of the form specified above. There will be no syntactically incorrect commands.

Output

The output should consist of the final state of the blocks world. Each original block position numbered i ( 0 <= i < n where n is the number of blocks) should appear followed immediately by a colon. If there is at least a block on it, the colon must be followed by one space, followed by a list of blocks that appear stacked in that position with each block number separated from other block numbers by a space. Don't put any trailing spaces on a line.   
  
There should be one line of output for each block position (i.e., n lines of output where n is the integer on the first line of input).

Sample Input

10

move 9 onto 1

move 8 over 1

move 7 over 1

move 6 over 1

pile 8 over 6

pile 8 over 5

move 2 over 1

move 4 over 9

quit

Sample Output

0: 0

1: 1 9 2 4

2:

3: 3

4:

5: 5 8 7 6

6:

7:

8:

9:

最开始n个箱子在起始位置0~n-1

• 有一系列操作

• 　　move a onto b，将a,b上面的箱子放回初始位置，并将a放到b箱上。

• 　　move a over b ，将a上面的箱子放回初始位置，并将a放到b箱最上方。

• 　　pile a onto b ，将b上面的箱子放回初始位置，并将a和a上的箱子一起

放到b箱上。

• 　　pile a over b，将a和a上的箱子一起放到b箱最上方。

• 　　要求输出最后每个位置的箱子。

• n <= 25

题解：模拟题，有一个值得学习的小技巧，输入一共有4种指令，但是如果完全独立地处理各种指令，代码会变得冗长而且容易错，更好的方法是提取出指令之间的共同点，编写函数  
减少重复代码  
因为这个题木块的高度不确定，所以用vector很好

#include<cstdio>

#include<string>

#include<vector>

#include<iostream>

const int maxn=30;

using namespace std;

vector<int> pile[maxn];

string sa,sb;

int n;

void find\_block(int a,int &p,int &h)//找木块a所在的pile和height,以引用的形式返回调用者

{

for(p=0;p<n;p++)

for(h=0;h<pile[p].size();h++)

if(pile[p][h]==a)

return;

}

void pile\_clear(int p,int h)//把第p堆高度为h的木块上方的所有木块移回原位

{

for(int i=h+1;i<pile[p].size();i++)

{

int b=pile[p][i];

pile[b].push\_back(b);//把木块b放回原位

}

pile[p].resize(h+1);//pile只保留下标为0～h的元素

}

void pile\_move(int pa,int h,int pb)//把第p堆高度为h及其上方的木块整体移动到p2堆的顶部

{

for(int i=h;i<pile[pa].size();i++)

pile[pb].push\_back(pile[pa][i]);

pile[pa].resize(h);

}

int main()

{

int a,b;

cin>>n;

for(int i=0;i<n;i++)

pile[i].push\_back(i);

while(cin>>sa>>a>>sb>>b)

{

int pa,pb,ha,hb;

find\_block(a,pa,ha);

find\_block(b,pb,hb);

if(pa==pb) continue;//非法指令

if(sa=="move")

pile\_clear(pa,ha);

if(sb=="onto")

pile\_clear(pb,hb);

pile\_move(pa,ha,pb);

}

for(int i=0;i<n;i++)

{

cout<<i<<": ";

for(int j=0;j<pile[i].size();j++)

cout<<pile[i][j]<<' ';

cout<<endl;

}

return 0;

}