Programs executed concurrently on a uniprocessor system appear to be executed at the same time, but in reality the single CPU alternates between the programs, executing some number of instructions from each program before switching to the next.
在单处理器系统上并发执行的程序似乎同时执行，但实际上，单个 CPU 在程序之间交替，在切换到下一个程序之前执行来自每个程序的一些指令。

You are to simulate the concurrent execution（并行执行）of up to ten programs on such a system and determine the output that they will produce.
您将在这样的系统上模拟多达 10 个程序的并发执行，并确定它们将产生的输出。

The program that is currently being executed is said to be *running*, while all programs awaiting execution are said to be *ready*.
目前正在执行的程序据说正在运行，而所有等待执行的程序据说已经准备就绪。

A program consists of a sequence of no more than 25 statements, one per line, followed by an end statement. The statements available are listed below.
程序由不超过 25 个语句的序列组成，每行一个，然后是结束语句。现有的陈述如下。

Statement Type(语句类型)	Syntax(语法)
Assignment	*variable* = *constant*
Output	print *variable*
Begin Mutual Exclusion	lock
End Mutual Exclusion	unlock
Stop Execution	end

A *variable* is any single lowercase alphabetic char-acter and a *constant* is an unsigned decimal num-ber less than 100. There are only 26 variables in the computer system, and they are shared among the programs. Thus assignments to a variable in one program affect the value that might be printed by a different program. All variables are initially set to zero.
变量是任何一个小写字母 char-acter，常量是一个小于 100 的无符号小数 num-ber。计算机系统中只有 26 个变量，它们在程序中共享。因此，分配给一个程序中的变量会影响不同程序可能打印的值。所有变量最初都设置为零。

Each statement requires an integral number of time units to execute.
每个语句都需要一个整数的时间单位来执行。

The running program is permitted to continue executing instructions for a period of time called its *quantum*.
运行中的程序被允许在一段时间内继续执行指令，称为其量子。

When a program's time quantum expires, another ready program will be selected to run. Any instruction currently being executed when the time quantum expires will be allowed to complete.
当程序的时间量子过期时，将选择另一个就绪程序运行。当时间量子到期时，当前正在执行的任何指令都将被允许完成。

Programs are queued rst-in- rst-out for execution in a *ready queue*.
程序在就绪队列中排队等待执行

The initial order of the ready queue corresponds to the original order of the programs in the input file.
就绪队列的初始顺序与输入文件中程序的原始顺序相对应

This order can change, however, as a result of the execution of lock and unlock statements.
但是，由于执行了锁和解锁语句，这种顺序可能会发生变化。

The lock and unlock statements are used whenever a program wishes to claim mutually exclusive access to the variables it is manipulating.
无论何时，当一个程序希望要求对它正在操作的变量进行互斥访问时，都会使用锁和解锁语句。
These statements always occur in pairs, bracketing one or more other statements. A lock will always precede an unlock, and these statements will never be nested.
这些语句总是成对地出现，在一个或多个其他语句的括号中。锁将始终位于解锁之前，这些语句将永远不会嵌套。

Once a program successfully executes a lock statement, no other program may successfully execute a lock statement until the locking program runs and executes the corresponding unlock statement.
一旦程序成功执行锁语句，在锁定程序运行并执行相应的解锁语句之前，其他程序不得成功执行锁语句..

Should a running program attempt to execute a lock while one is already in effect, this program will be placed at the end of the *blocked queue*. Programs blocked in this fashion lose any of their current time quantum remaining. When an unlock is executed, any program at the head of the blocked queue is moved to the head of the ready queue.
如果一个正在运行的程序试图在一个已经生效的情况下执行一个锁，这个程序将被放置在阻塞队列的末尾。以这种方式被封锁的程序失去了当前剩余的任何时间量。当执行解锁时，阻塞队列首部的任何程序都会移动到就绪队列首部。

The first statement this program will execute when
it runs will be the lock statement that previously failed. Note that it is up to the programs involved to enforce the mutual exclusion protocol through correct usage of lock and unlock statements.
该程序在运行时将执行的第一个语句是以前失败的锁语句。请注意，这取决于所涉及的程序通过正确使用锁和解锁语句来强制执行互斥协议。

(A renegade program with no lock/unlock pair could alter any variables it wished, despite the proper use of lock/unlock by the other programs.)
(一个没有 锁/解锁成对 的变节程序可以改变它希望的任何变量，尽管其他程序正确地使用了锁/解锁。)

Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.
输入以一行上的一个正整数开始，它本身指示下面每个情况的数量，如下所述。这一行后面是一个空行，两个连续输入之间也有一个空行。

The first line of the input file consists of seven integers separated by spaces. These integers specify (in order): the number of programs which follow, the unit execution times for each of the five statements (in the order given above), and the number of time units comprising the time quantum. The remainder of the input consists of the programs, which are correctly formed from statements according to the rules described above
输入文件的第一行由由空格分隔的七个整数组成。这些整数指定（顺序）：跟随程序的数量，五个语句中每个语句的单位执行时间（按上面给出的顺序），以及组成时间量子的时间单位的数量。其余的输入是由程序组成的，这些程序是根据上述规则由语句正确形成的

All program statements begin in the first column of a line. Blanks appearing in a statement should be ignored. Associated with each program is an identification number based upon its location in the input data (the first program has ID = 1, the second has ID = 2, etc.).
所有程序语句都从一行的第一列开始。语句中出现的空格应该被忽略。与每个程序相关联的是基于其在输入数据中的位置的标识号（第一个程序具有 id=1，第二个程序具有 id=2 等）。

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.
对于每个测试用例，输出必须遵循下面的描述。两个连续情况的输出将用空白行分隔。

Your output will contain of the output generated by the print statements as they occur during the simulation. When a print statement is executed, your program should display the program ID, a colon, a space, and the value of the selected variable. Output from separate print statements should appear on separate lines.
您的输出将包含打印语句生成的输出，因为它们发生在模拟过程中。当执行打印语句时，程序应该显示程序 id、冒号、空格和所选变量的值。来自单独打印语句的输出应出现在单独的行上。

Sample Input

```
3 1 1 1 1 1 1
a = 4
print a
lock
b = 9
print b
unlock
print b
end
a = 3
print a
lock
b = 8
print b
unlock
print b
end
b = 5
a = 17
print a
print b
lock
b = 21
print b
unlock
print b
end
```

Sample Output

```
1:   3
2:   3
3:   17
3:   9
1:   9
1:   9
2:   8
2:   8
3:   21
3:   21
```