E. Optimization (16 Mb, 5 sec)

Kirill is a hacker and he learns programming at the academy. He is fond of trying various pieces of destructive software against the LAN of his *alma mater*.
In order to make the talented one busy and give some rest to unfortunate system administrators, Kirill's teacher assigned him a challenging course project.
The goal of the project is to develop an algorithm to optimize programs written in assembly language of a simple CPU.
A program consists of a sequence of lines numbered 1 to ***n***. Program execution starts with the line number 1, and stops on line ***n***, which is guaranteed to contain the **RET** command. Program lines are executed sequentially, one after another. The order of execution may be altered only by special jump commands.
The CPU has four eight-bit registers labeled **A**, **B**, **C**, and **D**. Each register can store an integer from the range [0, 255]. Aside from the registers the CPU has two flags: zero flag **ZF** and carry flag **CF**. Possible flag values are 0 or 1. There is also a special command register **R**. This register stores the number of the line being executed.
The following table lists all commands supported by this CPU.

Mnemonic	**Command description**	**CPU actions**
MOV reg1, reg2	Move data	reg1 ← reg2
LD reg1, #const	Load a constant into a register	reg1 ← #const
ADD reg1, reg2	Add	CF ← (reg1 + reg2) div 256; reg1 ← (reg1 + reg2) mod 256; ZF ← [reg1 = 0]
ADC reg1, reg2	Add with carry	CF ← (reg1 + reg2 + CF) div 256; reg1 ← (reg1 + reg2 + CF) mod 256; ZF ← [reg1 = 0]
NEG reg	256 complement	reg ← 256 – reg; ZF ← [reg = 0]; CF ← 1
AND reg1, reg2 OR reg1, reg2 XOR reg1, reg2	Bitwise boolean operations "and", "or" and "exclusive or"	reg1 ← reg1 & reg2; (AND) reg1 ← reg1 | reg2; (OR) reg1 ← reg1 xor reg2; (XOR) ZF ← [reg1 = 0]; CF ← 0
INC reg	Increment register	reg ← reg +1; ZF ← [reg = 0]
DEC reg	Decrement register	reg ← reg – 1; ZF ← [reg = 0];
CLC	Clear carry flag	CF ← 0
STC	Set carry flag	CF ← 1

JUMP #xxx	Unconditional jump to line #xxx	R ← #xxx
JZ #xxx	Jump to line #xxx if zero flag (ZF) is set	if ZF = 1 then R ← #xxx
JNZ #xxx	Jump to line #xxx if zero flag (ZF) is not set	if ZF = 0 then R ← #xxx
JC #xxx	Jump to line #xxx if carry flag (CF) is set	if CF = 1 then R ← #xxx
JNC #xxx	Jump to line #xxx if carry flag (CF) is not set	if CF = 0 then R ← #xxx
RET	Finish execution	Stop

Character "←" in the table stands for assignment, "div" for integer division, "mod" for remainder of division. Expression "[*reg* = 0]" evaluates to 1 if register *reg* contains zero, and 1 otherwise.

The following actions are allowed for program optimization:

- change the order of lines, except for the last line, which must always contain the **RET** command;
- change type, condition and target line for jump commands;
- remove and/or add jumps of any kind if it does not influence the result of execution;
- remove lines that are never executed.

It is forbidden to add or remove other commands. The changes must not influence the program logic.

Write a program that will optimize an assembly language program by removing the largest possible number of jumps (JUMP, JZ, JNZ, JC, JNC) from it.

Limitations

Number of lines in the assembly language program, n: $1 \le n \le 100$.

It is guaranteed that no line of the input data is executed more than once.

Input

The input file consists of one or more lines. The file is ended by a line containing the **RET** command. Each line contains number i and a command. Number i is an integer without leading zeroes, separated from the command by exactly one space character. A command consists of a mnemonic (see table) and possible operands. The first operand (if applicable) is separated from the mnemonic by exactly one space character. The second operand (if applicable) is separated from the first one by a comma character, without spaces. Mnemonics and register names are written using only capital Latin letters. If an operand is an integer (line number for a jump, or constant for an **LD** command) then it is written as a decimal number.

The last line of the input file contains a single integer K, the limit of optimization (the output program must contain at most K jumps).

Output

The output file must contain the text of the program after optimization, written according to the rules given for the input file. The output program must contain no more then ***K*** jumps.

Sample Input 1	**Sample Output 1**
1 MOV A,B 2 JUMP 5 3 LD B,10 4 JUMP 6 5 JUMP 3 6 RET 0	1 MOV A,B 2 LD B,10 3 RET
Sample Input 2	**Sample Output 2**
1 ADD A,B 2 JZ 4 3 JUMP 5 4 ADD B,C 5 RET 1	1 ADD A,B 2 JNZ 4 3 ADD B,C 4 RET
Sample Input 3	**Sample Output 3**
1 ADD A,A 2 LD B,1 3 AND A,B 4 JZ 6 5 JUMP 8 6 LD C,5 7 JUMP 9 8 LD C,6 9 RET 0	1 ADD A,A 2 LD B,1 3 AND A,B 4 LD C,5 5 RET