ACM International Collegiate Programming Contest

Sponsored by IBM

Rules:

1. There are **eight** problems to be completed in **five hours**.
2. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
3. The allowed programming languages are C, C++ and Java.
4. All programs will be re-compiled prior to testing with the judges' data.
5. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contestant officials (e.g., that might generate a security violation).
6. The input to all problems will consist of multiple test cases.
7. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
8. All communication with the judges will be handled by the PC^2 environment.
9. Judges' decisions are to be considered final. No cheating will be tolerated.

Problem A: CIVIC DILL MIX

Roman numerals are an ancient numbering system used extensively throughout Europe through the 13th century (where it was eventually replaced by our current positional system). Vestiges of this system still exist today on clock faces, building cornerstones, Super Bowls and Star Wars episodes. The system uses the following 7 symbols:

Symbol	I	V	X	L	C	D	M
Value	1	5	10	50	100	500	1000

Symbols I, X, C and M can be repeated as needed (though never more than three times for I, X and C), so that 3 is represented as III, 27 as XXVII and 4865 as MMMMDCCCLXV. The symbols are always written from the highest value to the lowest, but for one exception: if a lower symbol precedes a higher one, it is *subtracted* from the higher. Thus 4 is written not as IIII but as IV, and 900 is written as CM. The rules for this subtractive behavior are the following:

1. Only I, X and C can be subtracted.
2. These numbers can only appear once in their subtractive versions (e.g., you can't write 8 as IIX).
3. Each can only come before symbols that are no larger than 10 times their value. Thus we can not write IC for 99 or XD for 490 (these would be XCIX and CDXC, respectively). Note that the first two words in this problem title are invalid Roman numerals, but the third is fine.

Your task for this problem is simple: read in a set of Roman numeral values and output their sum as a Roman numeral.

Input

Input will consist of multiple test cases. Each test case starts with a positive integer n indicating the number of values to add. After this will come n values (potentially several on a line), all valid Roman numerals with whitespace only coming between values. A value of $n = 0$ will indicate end of input. All sums will be less than 5000.

Output

For each test case, output the case number and the sum, both as Roman numerals, using the format shown below. Case numbers should start at I.

Sample Input

```
2
XII MDL
4
I I I
I
0
```

Sample Output

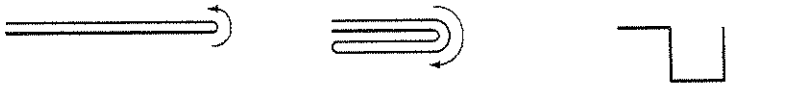
```
Case I: MDLXII
Case II: IV
```

Problem B: A Foldy but a Goody

Suppose you have a strip of paper and are given instructions to fold the paper in one of two ways: an upper fold, where the right end of the paper is brought over to the top of the left end; and a lower fold, where the right end of the paper is brought below the left end. The diagram below illustrates both types of folds.

Original strip Upper fold Lower fold

Now, after meticulously folding the strip several times, you are asked to unfold it by making a 90 degree angle at each crease. The example below shows the result of an upper fold, followed by a lower fold and then an unfolding.



Upper... followed by lower... and then unfolded

If the left end of the folded strip is placed at the origin (0,0) and the first right angle is at (1,0), it is natural to ask the questions: Where will the second right angle be located? The third right angle? Where will the other end of the strip be located? Well, that's for us to know and you to figure out.

Input

The input file will contain multiple test cases. The first line of the file will contain a single integer indicating the number of test cases. Each case will consist of a string of letters U and L indicating a series of upper and lower folds followed by an integer m. The length of the string will be between 1 and 30, inclusive. The value of m identifies a position on the paper. A value of $m = 0$ indicates the left end (at location $(0, 0)$). If there are n folds, then a value of $m = 2^n$ indicates the right end of the strip. Any value for m between these two extremes represents one of the right angles; $m = 1$ indicates the first right angle, and so on.

Output

For each test case, output a single line of the form `(x,y)` indicating the location of the right angle (or end point) specified by the problem. You should assume that if there are n folds in the test case, the length of the string is 2^n so that the distance between creases is 1 unit long.

Sample Input

```
3
UL 4
UL 3
LLUL 13
```

Sample Output

```
(2,0)
(2,-1)
(1,-2)
```

Problem C: Give Me an E

Everyone knows that the letter "E" is the most frequent letter in the English language. In fact, there are one hundred sixteen E's on this very page ... no, make that one hundred twenty one. Indeed, when spelling out integers it is interesting to see which ones do NOT use the letter "E". For example 6030 (six thousand thirty) doesn't. Nor does 4002064 (four million two thousand sixty four).

It turns out that 6030 is the 64th positive integer that does not use an "E" when spelled out and 4002064 is the 838th such number. Your job is to find the n-th such number.

Note: 1,001,001,001,001,001,001,001,001,000 is "one octillion, one septillion, one sextillion, one quintillion, one quadrillion, one trillion, one billion, one million, one thousand". (Whew!)

Input

The input file will consist of multiple test cases. Each input case will consist of one positive integer n (less than 2^{31}) on a line. A 0 indicates end-of-input. (There will be no commas in the input.)

Output

For each input n you will print, with appropriate commas, the n-th positive integer whose spelling does not use an "E". You may assume that all answers are less than 10^{28}.

Sample Input

```
1
10
838
0
```

Sample Output

```
2
44
4,002,064
```

Problem D: Lemmings, Lemmings Everywhere. But Not For Long.

On an $n \times m$ board there is a lemming on each square. Every second, the lemmings try to move either north, south, east or west, according to rules which are explained below. To determine which direction to move in, each lemming has an agenda, which is an ordering of the four possible directions (for example, one possible agenda might be NWES). The rules for lemming movement are the following:

1. Initially each lemming sets its direction of movement **D** to the first direction in its agenda.
2. At each time step, each lemming tries to move in its direction **D**. Three things can happen to lemming L:
   (a) If L's current direction **D** causes it to move off the board, then the world has one less lemming in it. Otherwise, L's target destination will be to another square.
   (b) If L's target square is empty or about to become empty as a result of another lemming leaving it, and no other lemming wants to move to the same square, then L moves into its target square. In this case, the lemming will use the same direction **D** in the next time step.
   (c) Otherwise, if another lemming is trying to move into L's target square, or if the target square contains a lemming which cannot move, then L stays put. In this case, it will update its **D** by going to the next direction in its agenda (wrapping around to the beginning if necessary).

Two lemmings which want to exchange squares can do so, unless of course some other lemming is trying to move into one of their two squares (in which case all three of the lemmings would stay in their current squares). Lemmings being lemmings, they continue to move until all of them have moved off the board. Your job is to determine how long that takes.

Input

Input will consist of multiple test cases. Each test case will consist of multiple lines. The first line will contain two positive integers n m, specifying the number of rows and columns in the board. The maximum value for each of these is 100. The board is situated so that square $(0, 0)$ is in the southwest corner, and square $(0, m-1)$ is in the southeast corner. Following this will be several rows containing the agendas for the nm lemmings. Each agenda will be a permutation of the string `NESW`. There will be 16 agendas on each line (except perhaps the last), with a single space between each. The agendas are assigned row-wise to the lemmings, so that the first agenda is associated with the lemming on square $(0, 0)$, the second with the lemming on square $(0, 1)$, and so on. The last case will be followed by the line 0 0 which will terminate input.

Output

For each test case, output a single line containing the case number (using the format shown below) followed by the number of steps it takes until the last lemming(s) falls off the board. Use only single spaces to separate items on a line.

(over)

Sample Input

```
2 2
ENWS WSNE NESW WENS
2 2
ENWS WSNE NESW SWEN
0 0
```

Sample Output

```
Case 1: 2
Case 2: 3
```

Problem E: Once Around the Lock

Most of us at one time or another have used a circular combination lock (think back to those glorious days in high school and your gym locker). Most combination locks consist of a dial with the numbers 0 through n-1 printed on it in clockwise order. The dial can be turned either clockwise or counterclockwise, bringing one of the numbers to the top of the dial (if 0 is at the top of the dial, a turn of 1 in the counterclockwise direction would bring 1 to the top). Each lock has a three number code (x, y, z) and can only be opened after the following series of steps:

1. The lock dial must first be spun clockwise at least one full rotation, ending with the number x at the top (with no intervening counterclockwise turns). Note this could be accomplished with consecutive clockwise turns.
2. The lock must be turned counterclockwise until the number y appears at the top for the second time. Note this could be accomplished with consecutive counterclockwise turns (but no intervening clockwise turns).
3. The lock must then be turned clockwise until the number z appears on top, without going more than one full rotation. Note this could be accomplished with consecutive clockwise turns (but no intervening counterclockwise turns).

Any rotation after this last step will cause the lock to be closed again.

For this problem, you will be given a lock and a series of turns and you must determine at the end whether or not the lock is open. You should assume prior to the first turn that the lock has just been closed, and the dial spun counterclockwise until 0 is on top.

Input

Input will consist of multiple test cases. The first line of each test case will contain four integers n x y z, indicating the number of digits on the lock's dial and the three-number combination (x, y and z will all be different and $n \leq 1000$). The next line(s) will consist of a series of dial rotations of the form d s, where d is either `C` or `CC` (for clockwise or counterclockwise) and s (> 0) indicates how many numbers to spin through at the top of the dial. For example, if $n = 50$ and the current number on top of the dial is 4, the rotation `CC 6` would bring the number 10 to the top, while a rotation of `C 6` would bring 48 to the top. The series of dial rotations may extend over multiple lines, ending with the character `?`. A line with a single 0 on it will follow the last test case.

Output

For each problem instance, output a single line containing either the word `Open` or `Closed`, prefaced by the test case number as shown in the sample output.

(over)

Sample Input

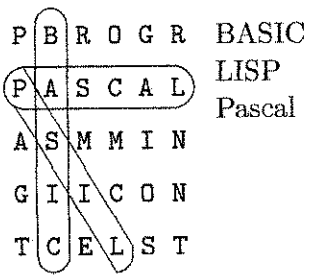
```
60 6 1 58
C 114 CC 115 C 3 ?
60 6 1 58
C 54 CC 115 C 3 ?
60 6 1 58
C 54 C 60 CC 115 C 3 ?
0
```

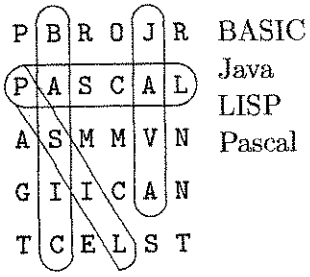
Sample Output

```
Case 1: Open
Case 2: Closed
Case 3: Open
```

Problem F: A Puzzling Problem

Anna Graham is a puzzle maker who prides herself in the quality and complexity of her work. She makes puzzles of all kinds - crosswords, logic problems, acrostics, and word search puzzles, to name but a few. For each puzzle, she has developed a set of rules which she constrains herself to follow. For word search puzzles, she insists not only that all the words be connected to one another (as in most word search puzzles), but also that removing any word from the word list will not cause one or more words to become disconnected from the rest. (Two words are connected if they contain a common letter in the grid.) The example word search puzzle on the left satisfies this condition, but the one on the right does not (removing the word Pascal from the word list disconnects Java from the other two words).





Your job is to write a program that checks to see if Anna's word search problems are up to snuff.

Input

Input will consist of multiple test cases. The first line of each test case contains 3 integers n m l, where n and m are the number of rows and columns in the puzzle and l is the number of words. Following this are n lines containing m uppercase characters each (the puzzle) followed by l lines containing one word each (the word list, in mixed case). Each word in the word list will appear in the puzzle exactly once. There will be no more than 100 rows and 100 columns in the puzzle and no more than 100 words to search for. There will be no spaces in the input words.

Output

For each problem instance, output the word `Yes` or `No` depending on whether the puzzle satisfies Anna's constraints.

Sample Input

```
5 6 3
PBROGR
PASCAL
ASMMIN
GIICON
TCELST
BASIC
LISP
Pascal
5 6 4
PBROJR
PASCAL
```

```
ASMMVN
GIICAN
TCELST
BASIC
Java
LISP
Pascal
0 0 0
```

Sample Output

```
Yes
No
```

Bonus Puzzle (For those of you with a little extra time.)

```
A D N E R H E B E T W E S T M I N S T E R T H
Y E G R A V E M E U D R U P O B E R L I N D E
E C M I C H I G A N W A T E R L O O I G I N T
L G E M R C U N T D I V E L L I V R A D E C A
L E U A H R W S E I E B T T Y S N T H R E S T
A R E I I I B N V Y O R K O S A A G A L N O S
V K G M N T O A I W H F T R G G G Z A E R I T
D A R D O M U D L E I H T O N I A D E R N B H
N R S O A Y E I O D O A T N N N K U D D B I G
A O L R N N N R G R W S I T N A Q E I S E O I
R A I R C G C E H G E I P O O W I A N N G N R
G H P O G I N H H N S I N F T V N T B H E A W
M S P R T U N S O G L R D W E A O N O T S G M
U A E N F D A C L R E I R U A L D I R F L I W
G E R O L B U E I V Y L H M E L I A O N C H D
N T Y I H O O Q T N A U L D G E L H Y H T C H
I E R H E T Y M U M N A O A A Y Y A I T E I B
K W O O S T E R N E W A H S N A F G C D O M R
S U C P I A T N A B S I T M C M A S T E R N O
U G K H O W L A U R E N T I A N O T E L R A C
M A S H L A N D Y T I C E V O R G L E J P B K
```

ALMA	DAYTON	MICHIGAN	SHERIDAN
AKRON	DUQUESNE	MT VERNON NAZARENE	SLIPPERY ROCK
ALLEGHENY	EDINBORO	MUSKINGUM	SPRING ARBOR
ASHLAND	E(astern) MICHIGAN	N(orthern) MICHIGAN	THIEL
BALDWIN-WALLACE	FANSHAWE	NOTRE DAME	TOLEDO
BEHREND	GRAND VALLEY	OBERLIN	TORONTO
BOWLING GREEN	GROVE CITY	OHIO N(orthern)	WATERLOO
BROCK	HIRAM	OHIO WESLEYAN	WESTMINSTER
CARLETON	IIT	OLIVET	WILFRID LAURIER
CEDARVILLE	INDIANA	OTTAWA	WINDSOR
CINCINNATI	LAURENTIAN	PITT	WOOSTER
C(entral) MICHIGAN	MARIETTA	PURDUE	WRIGHT STATE
CMU	MCMASTER	QUEENS	YORK
CONESTOGA	MIAMI	SAGINAW VALLEY	

Problem G: And Now, a Remainder from Our Sponsor

IBM has decided that all messages sent to and from teams competing in the ACM programming contest should be encoded. They have decided that instead of sending the letters of a message, they will transmit their remainders relative to some secret keys which are four, two-digit integers that are pairwise relatively prime. For example, consider the message `"THE CAT IN THE HAT"`. The letters of this message are first converted into numeric equivalents, where A=01, B=02, ..., Z=26 and a blank=27. Each group of 3 letters is then combined to create a 6 digit number. (If the last group does not contain 3 letters it is padded on the right with blanks and then transformed into a 6 digit number.) For example

`THE CAT IN THE HAT` $\rightarrow$ `200805 270301 202709 142720 080527 080120`

Each six-digit integer is then encoded by replacing it with the remainders modulo the secret keys as follows: Each remainder should be padded with leading 0's, if necessary, to make it two digits long. After this, the remainders are concatenated together and then any leading 0's are removed. For example, if the secret keys are 34, 81, 65, and 43, then the first integer 200805 would have remainders 1, 6, 20 and 38. Following the rules above, these combine to get the encoding 1062038. The entire sample message above would be encoded as

`1062038 1043103 1473907 22794503 15135731 16114011`

Input

The input consists of multiple test cases. The first line of input consists of a single positive integer n indicating the number of test cases. The next $2n$ lines of the input consist of the test cases. The first line of each test case contains a positive integer (< 50) giving the number of groups in the encoded message. The second line of each test case consists of the four keys followed by the encoded message. Each message group is separated with a space.

Output

For each test case write the decoded message. You should not print any trailing blanks.

Sample Input

```
2
6
34 81 65 43 1062038 1043103 1473907 22794503 15135731 16114011
3
20 31 53 39 5184133 14080210 7090922
```

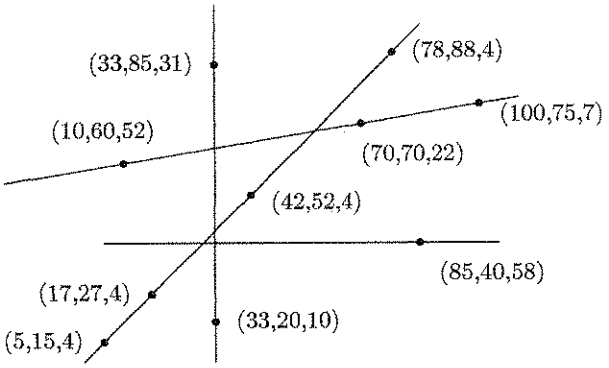
Sample Output

```
THE CAT IN THE HAT
THE END
```

Problem H: Target Practice

There are many forms of contests where the contestants (shooters) try to hit targets, either moving or still. In this version there are a number of small balloons sitting on the tops of poles that are in turn stuck in the ground at various points in a large field. These poles are not all the same height. The shooter circles the field and fires at the balloons, the goal being to burst all the balloons with as few shots as possible. Since the balloons offer almost no resistance to a bullet, the bullet will pass right through and possibly hit one or more other balloons. So, by judiciously taking shots, the shooter might need only a very few shots to hit all the targets (provided the shooter is a good marksman, which we will assume is the case).

For example, the following field of 10 targets can be covered in only four shots, as shown. (The first two numbers at each position indicate the position of the balloon, and the third number the height.)



Your job is to determine the fewest number of shots necessary to hit all the targets in a given field.

Input

There will be multiple test cases. Each test case will consist of an integer n (≤ 50) indicating the number of target positions to follow. A value of $n = 0$ indicates end of input. There will follow n integer triples, x y h, indicating a balloon at position (x, y) in the field at height h. (There will be at most one balloon at any position (x, y).) All integers are greater than 0 and no greater than 100. Furthermore assume that the shooter can take shots from anywhere on the field at any height. For simplification, assume here that the balloons are points and that the bullets can pass through the poles on which the balloons are perched.

Output

Each test case should produce one line of output of the form:

```
Target set k can be cleared using only s shots.
```

where `k` is the number of the test case, starting at 1, and the value of `s` is the minimum number of shots needed to hit all the targets in the set.

Sample Input

```
10
5 15 4 10 60 52 17 27 4 33 20 10 33 85 31 42 52 4 70 70 22
78 88 4 85 40 58 100 75 7
9
5 15 4 10 60 52 17 27 4 33 20 10 33 85 31 42 52 4 70 70 22
78 88 4 100 75 7
0
```

Sample Output

```
Target set 1 can be cleared using only 4 shots.
Target set 2 can be cleared using only 3 shots.
```