2004 Consortium for Computing Sciences in Colleges Programming Contest

Saturday, November 6th

Wofford College

Spartanburg, South Carolina

There are six (6) problems in this packet. Each team member should have a copy of the problems. These problems are NOT necessarily sorted by difficulty. Remember to name your source files appropriately before submission.

Have Fun & Good Luck! ☺

Problem Name	**Input File**	**Output File**
Congratulations, It's Twins!	`prog1_in.txt`	`prog1_out.txt`
The Laser Positioning System	`prog2_in.txt`	`prog2_out.txt`
Safe Cracker Code Cracker	`prog3_in.txt`	`prog3_out.txt`
Meet Me in Saint Louis	`prog4_in.txt`	`prog4_out.txt`
A Sweet Song is Taking Off	`prog5_in.txt`	`prog5_out.txt`
It's War!	`prog6_in.txt`	`prog6_out.txt`

Congratulations, It's Twins!

Input File: `prog1_in.txt` Output File: `prog1_out.txt`

A *prime* number is an integer greater than 1 such that the only integers that divide evenly into it are itself and 1. Thus some prime numbers are 2, 3, 5, 7, 11, 13, 17, 19, etc. It can fairly easily be proven that there must exist infinitely many prime numbers.

A *twin prime pair* is two prime numbers that differ by two, such as 11 and 13, or 29 and 31. The only twin prime pair between 4 and 10 is (5,7), but between 10 and 20 we have (11,13) and also (17,19). There are no twin prime pairs between 20 and 30.

It has been conjectured that there are infinitely many twin prime pairs (this is known as the "Twin Prime Conjecture"), but it has yet to be proven.

Your task is to write a program that counts the number of twin prime pairs between two given even positive integers.

Input
Input will consist of exactly 10 lines, each containing two even positive integers. No integer input will be larger than `2147483646`, and the difference between the two even positive integers is no larger than 500.

Output
There must be one line of output for each line of input as shown in the format below. Always output the smaller of the two inputs first.

Sample Input

```
4 10
10 20
20 30
10 30
100 200
2 4
122 134
200 400
4 2
98 2
```

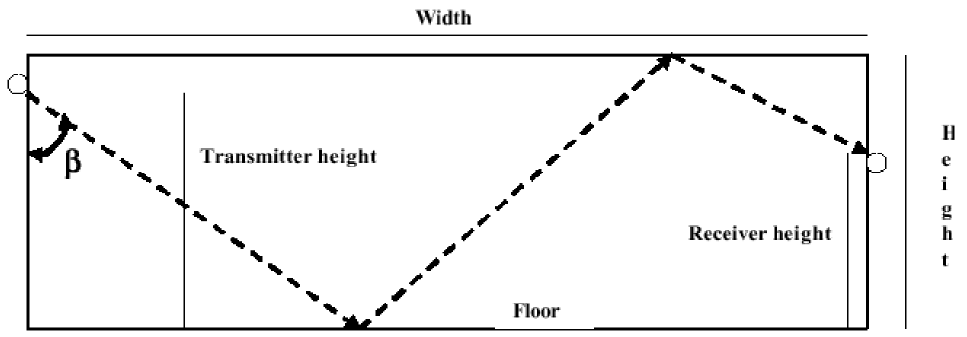
Output Corresponding to Sample Input

```
The number of twin prime pairs between 4 and 10 is 1.
The number of twin prime pairs between 10 and 20 is 2.
The number of twin prime pairs between 20 and 30 is 0.
The number of twin prime pairs between 10 and 30 is 2.
The number of twin prime pairs between 100 and 200 is 7.
The number of twin prime pairs between 2 and 4 is 0.
The number of twin prime pairs between 122 and 134 is 0.
The number of twin prime pairs between 200 and 400 is 6.
The number of twin prime pairs between 2 and 4 is 0.
The number of twin prime pairs between 2 and 98 is 8.
```

The Laser Positioning System

Input File: `prog2_in.txt` Output File : `prog2_out.txt`

The LaserOptics Company, a firm specializing in optics engineering, has contracted you to write a program that will calculate the angle with which a laser transmitter needs to be positioned for effective data transmission using fiber optics. The program is required to handle various transmitter and receiver elevations and also, multiple room dimensions. The laser transmitter will be positioned on the left side of the room whose walls are covered with glass panels. Due to power limitations, the transmitted light needs to be bounced off the walls exactly twice (one at the bottom and the other on top) before arriving at its target. The target is an optical receiver positioned at the right wall. Although it may not be obvious in the figure below, the glass walls are perfectly designed for light propagation such that the angle of incidence is equal to the angle of refraction. In addition, the transmitter will always be oriented to point to the floor and thus, you may assume that the first bounce of the light will always occur on the floor.



Input

Each line of input consists of two integer values representing the width and height of the room, respectively and two floating point values representing the positions (measured from the floor) of the transmitter and the receiver, respectively. The end of input, which should not be processed, is indicated by two zeroes.

Output

For each input line, output the data set number together with the input data as shown. This is followed on each line by a single tab, and then the calculated angular position (represented by **β** in the figure shown above) of the transmitter measured in degrees. You may assume that every input data set has a solution. The value of the angle must be shown precisely, rounded to the nearest tenth of a degree.

Sample Input

```
22 7 6.0 3.54
22 7 6.0 6.73
10 6 3.0 4.86
18 12 8.0 2.0
0 0
```

Output Corresponding to Sample Input

```
Data set # 1: 22 7 6.0 3.54   Laser Transmitter position: 53.2 degrees
Data set # 2: 22 7 6.0 6.73   Laser transmitter position: 58.9 degrees
Data set # 3: 10 6 3.0 4.86   Laser transmitter position: 44.6 degrees
Data set # 4: 18 12 8.0 2.0   Laser transmitter position: 31.0 degrees
```

Safe Cracker Code Cracker

Input File: `prog3_in.txt` Output File: `prog3_out.txt`

Safe crackers Sal and Roscoe think they are so smart. They have cracked the code on a number of safes with 4-directional combinations. For every safe they crack, they remember the combination by constructing an *n x n* table with cell entries representing numbers and directions on a safe. What they didn't count on is your genius in cracking their system.

For example, given this 4 *x* 4 table:

2D	1D	2L	2L
2R	2D	OPEN	1U
1R	1R	1R	1U
2U	1L	3U	1L

You have discovered that the 16-instruction sequence to open the safe is: `1L 3U 2L 2D 1R 1R 1R 1U 1U 2L 1D 2D 1L 2U 2R OPEN`. Having cracked their system, you can now write a program to discover the combination of any safe Sal and Roscoe have cracked.

Input

The first line of input indicates the number of test cases in the file. This is followed by each test case. Each test case begins with *n* (1 <= *n* <= 8) on a line by itself, followed by *n* rows, each containing *n* cell entries.

Output

For each test case, the output should consist of the *n x n* sequence of instructions to open the safe, all on one line, with one space between every pair of instructions. Every square must be visited.

Sample Input

```
2
4
2D 1D 2L 2L
2R 2D OPEN 1U
1R 1R 1R 1U
2U 1L 3U 1L
2
1R 1D
OPEN 1L
```

Output Corresponding to Sample Input

```
1L 3U 2L 2D 1R 1R 1R 1U 1U 2L 1D 2D 1L 2U 2R OPEN
1R 1D 1L OPEN
```

Meet Me in Saint Louis

Input File: `prog4_in.txt` Output File: `prog4_out.txt`

Ella Grace Bumgardner and her mother Allison Leigh frequently send each other e-mails. Ever wary of interceptions and wishing to keep their correspondence private, they encrypt their messages in two steps. After removing all nonalphabetic characters and converting all letters to upper case, they: 1) replace each letter by the letter *s* positions after it in the alphabet ($1 <= s <= 25$). We call this a *shift* by *s* and then, 2) divide the result of step 1 into groups of *m* letters and reverse the letters in each group ($5 <= m <= 20$). If the length of the message is not divisible by *m*, then the last *k* (less than *m*) letters are reversed. For example, suppose $s = 2$ and $m = 6$. If the plaintext were

```
Meet me in St. Louis, Louis.
```

after removing unwanted characters and changing to upper case we get

```
MEETMEINSTLOUISLOUIS
```

We will call this the *modified plaintext*. We then shift each letter by 2 (`Y` would be replaced with `A` and `Z` would be replaced by `B`, here), getting the intermediate result:

```
OGGVOGKPUVNQWKUNQWKU
```

And finally reverse every group of 6 letters:

```
GOVGGOQNVUPKWQNUKWUK
```

Note the last two letters made up the last reversed group. As is customary, we write the result in groups of 5 letters. So the ciphertext would be:

```
GOVGG OQNVU PKWQN UKWUK
```

Alas, it's not so hard to find the values for *s* and *m* when the ciphertext is intercepted. In fact it's even easier if you know a *crib*, which is a word in the modified plaintext. In the above example, `LOUIS` would be a crib. Your job here is to find *s* and *m* when presented with a ciphertext and a crib.

Input
Input will consist of multiple problem instances. The first line of input will contain a positive integer indicating the number of problem instances. The input for each problem will consist of multiple lines. The first line of input for a problem will contain the integer n ($20 <= n <= 500$) which is equal to the number of characters in the ciphertext. The following lines will contain the ciphertext, all upper case in groups of 5 letters separated by a single space. (The last group of letters may contain fewer than 5 letters.) There will be 10 groups of letters per line, except possibly for the last line of ciphertext. The input line following the last line of ciphertext will contain the crib; a single word consisting of between 4 and 10 (inclusive) upper case characters.

Output

Output will be two integers, `s` and `m` on a line, separated by a single space, indicating the encryption key that produces the crib, where `s` is the shift and `m` is the reversed group size. If there is more than one solution, output the one with smallest `s`. If there is more than one with the same `s`, output the one with smallest `m`. If no such `s` and `m` exist, output the message `Crib is not encrypted`.

Sample Input

```
```

Output Corresponding to Sample Input

```
```

A Sweet Song is Taking Off

Input File: `prog5_in.txt` Output File: `prog5_out.txt`

Delta Airlines has come a long way since their roots as the world's first aerial crop dusting corporation in Macon, GA in 1924. Since September 2001 though, they've faced some tough economic times and are now seriously faced with the prospect of bankruptcy. Last year, in an effort to try to get out of their economic bind, they began offering flyers competitive low-fare amenities through a brand new company unit named Song. Song is a no-frills airline with no first class service. But, it does offer free beverages, leather seats, and even satellite television at every seat! One can only imagine what the two brothers, Orville and Wilber Wright, would say if they could see how airplanes had changed in just one hundred years since they made that historic first flight at Kitty Hawk, NC in December 1903.

Your job is to write a data processing program for the Song Flight 4474 from Atlanta to Fort Lauderdale which began in early October. Flight 4474 is an Aerospatiale regional jet with 15 rows, 4 passengers to a row that takes off daily at 9:05 A.M. and arrives at 10:55 A.M. Rows are numbered 1 through 15, and individual seats on each row are labeled A, B, C, and D. Seats A and D provide a window, while B and C are on the aisle. Your program will read in data associated with a flight exactly one week from today. Seats at this date are now being taken on a space available basis, and are assigned starting from the first seat at the front of the plane. Passenger requests for up to four seats together on a row are given where available. Your program will read in all seats already assigned as well as the passenger information associated with each seat, and one or more batch reservations that have come in recently. Your program will print out the seat map showing all assigned and available seats after all new requests have been processed. This is followed by the results of each requested reservation.

Input
Your program will first read in all seats already assigned. All seat assignments that have been made are stored one per line in the following format using three tokens separated by a single tab. The first token, `xxx`, represents a three character seat number. The second token is a `name` made up of 1 more characters, and the third token is a unique three digit `id`. A seat assignment of `LAST` represents the end of reservations already made and should not be processed. This is followed by a single positive integer *m* representing the number of batch reservation requests that have come in for processing. Each batch reservation request $r_1\ r_2 ... r_m$ contains a single positive integer *n* ranging from 1 to 4 indicating the number of seats needed. This is followed by *n* lines with a `name` and `id` together on a single line separated by a tab.

Output
Include headings on the first two lines with the flight information shown. A blank line should appear before and after the seat map, as well as for the plane's aisle between rows B and C. Use `E` to indicate an empty seat and `X` a reserved seat. Use a tabular format exactly as shown below. This is followed by the results of each batch reservation request. If a request cannot be filled (i.e., the total number of seats requested is greater

than the number available on the plane), simply print `request denied - not enough seats available` and ignore all lines with a `name` and `id` in the input file for that reservation. (You could have a confirmed reservation for a small number of seats after a request for a larger number of seats on an almost full plane is denied.) Use the format shown below for confirming a reservation with a comma and space following each confirmed id and seat assigned in the order processed.

Sample Input

```
08A     Smith,C         532
08B     Swift,T         004
08C     Sumner,M        047
10A     Johnston,A      444
LAST
2
1
Willis,D        738
4
Brown,C         388
Barger,J        112
Thomas,S        476
Thomas,R        888
```

Output Corresponding to Sample Input

```
Song Flight 4474 from ATL to FLL
Departs 9:05AM 13NOV, Arrives 10:55AM 13NOV, 0 Stop

        01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
D        E  X  E  E  E  E  E  E  E  E  E  E  E  E  E
C        E  X  E  E  E  E  E  X  E  E  E  E  E  E  E

B        E  X  E  E  E  E  E  X  E  E  E  E  E  E  E
A        X  X  E  E  E  E  E  X  E  X  E  E  E  E  E

id 738 confirmed in 01A
id 388, id 112, id 476, id 888 confirmed in 02A, 02B, 02C, 02D
```

It's WAR!

Input File: `prog6_in.txt` Output File: `prog6_out.txt`

There are numerous classic card games everyone learns as a child. *Old Maid* and *Go Fish* rank among the most well known. Another classic is the game of *War*. It has very simple (and rather varied) rules. Here is an example of what the rules may be:

1. Deal out the cards evenly among the players. We start by dealing to player 1, and giving each player one card in turn. We don't give player 1 a second card until all players have had one card.
2. Each player turns over their top card and places it in the middle.
3. The player with the highest valued card wins both cards. (Aces considered high)
    a. If there is a tie, the tied players remove their next card and place it face down.
    b. Then they turn the next card over and place it on the top of that card.
    c. High card wins.
    d. Go to step 3a if there is still a tie.
    e. If one of the players runs out of cards, he loses the tie breaker and all of the cards he played during the tie breaker goes to the tie breaker winner.
    f. If all players involved in the tie run out of cards, they are all out of the game and the cards from the hand are discarded (i.e. nobody gets them).
4. Go to step 2

<u>The game is over when all players are out of the cards dealt them</u>. The victor is the player with the most cards.

Assume you are given a pre-shuffled deck of cards that are ready to be dealt as well as the number of players. Deal out the cards and play the game, then report the number of cards that each player won. The cards are to be dealt so that the first card a player receives is the last card the player uses. You can always assume there are more cards than players, and that once a card is won by someone it is out of the game.

Be warned, I have a very old deck of cards so some cards may be missing and sometimes I mix decks.

Input

The information can be read from the input file in the following format:

```
<# of games>
<# of players for a game>
<suit S, C, H, D><value 1-13 : 1 is ace,11,12,13 is jack, queen,
king>
…
<suit><value>
JOKER
```

The `JOKER` simply indicates the bottom of the deck and signifies the end of dealing. It is not used in the game.

Output
For output, print the number of the game followed by each of the player numbers and the number of cards won as shown in the sample output. A blank line should follow each game (including the last one). Players are numbered by the way the deck is dealt (e.g. player 1 is the player who receives the first card in the deal.)

Sample Input

```
2
2
S4
H1
C13
D10
C2
S11
H8
D8
JOKER
4
S3
C4
D10
C12
S5
C13
S4
H5
H7
S2
H3
D12
H1
C9
D1
S1
JOKER
```

Output Corresponding to Sample Input

```
Game 1
Player 1 : 6 cards
Player 2 : 2 cards

Game 2
Player 1 : 0 cards
Player 2 : 2 cards
Player 3 : 2 cards
Player 4 : 0 cards
```