UTD High School Programming Contest 10-29-2016

- **Time Allowed: three hours.**
- **Each team must only use one of UTD's computers.**
- **Novice and Advanced category students may attempt to solve any of the problems.**
- **Answer the questions in any order.**
- **Use only Java 1.8, minGW g++, or MS Visual Studio 14 C/C++**
- **Your program source code must be contained in one source file.**
- **Do not use the "package" construct in your Java code.**
- **Your programs must read from a named file and output to System.out (cout for C/C++ programmers.)**
- **Do not access the web.**
- **Do not use any recording device containing code, other than UTD's computer.**
- **Your solutions must be entirely yours, typed by you during the time allowed for the contest.**
- **As soon as you have solved a problem, submit ONLY the source file via your `PC^2` client.**

Scoring

Equal points will be awarded to each question, even though they may not be equally difficult.

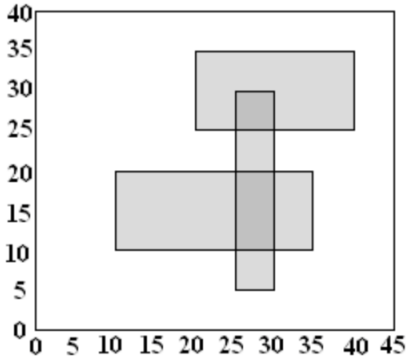
In order to break ties, we will use penalty points. The penalty points for a question will be zero if the question is not answered correctly. If a correct submission occurs for a question at time T minutes, then T penalty points will be added, plus 20 penalty points for each incorrect submission for that question.

The top ranked three teams, irrespective of category (Novice or Advanced) will be deemed 1st, 2nd and 3rd placed winners of the Advanced Contest. This rule in effect promotes any Novice teams within this group to Advanced Status. Those teams cannot also be place winners in the Novice Contest.

Of the remaining teams, the top three ranked Novice teams will be deemed 1st, 2nd and 3rd placed winners of the Novice Contest.

A Bulletins

Data File: A.txt
Time allowed = 10 seconds



The School Computer Club has just been given custody of a number of school bulletin boards. Several members agreed to clear off the old posters. They found posters plastered many levels deep. They made a bet about how much area was left clear, what was the greatest depth of posters on top of each other, and how much of the area was covered to this greatest depth. To determine each bet's winner, they made very accurate measurements of all the poster positions as they removed them. Because of the large number of posters, they now need a program to do the calculations. That is your job.

A simple illustration is shown above: a bulletin board 45 units wide by 40 high, with three posters, one with corners at coordinates (10, 10) and (35, 20), another with corners at (20, 25) and (40, 35), and the last with corners at (25, 5) and (30, 30). The total area not covered by any poster is 1300. The maximum number of posters on top of each other is 2. The total area covered by exactly 2 posters is 75.

Input:
The input will consist of one to twenty data sets, followed by a line containing only 0. On each line the data will consist of blank separated nonnegative integers.

The first line of a dataset contains integers n w h, where n is the number of posters on the bulletin board, w and h are the width and height of the bulletin board.

Constraints are $0 < n \leq 100$; $0 < w \leq 50000$; $0 < h \leq 40000$.

The dataset ends with n lines, each describing the location of one poster. Each poster is rectangular and has horizontal and vertical sides. The x and y coordinates are measured from one corner of the bulletin board. Each line contains four numbers x_l y_l x_h and y_h, where x_l and y_l, are the lowest values of the x and y coordinates in one corner of the poster and x_h and y_h are the highest values in the diagonally opposite corner. Each poster fits on the bulletin board, so $0 \leq x_l < x_h \leq w$, and $0 \leq y_l < y_h \leq h$.

Output:
There is one line of output for each data set containing three integers, the total area of the

bulletin board that is not covered by any poster, the maximum depth of posters on top of each other, and the total area covered this maximum number of times.

Sample Input	Sample Output
3 45 40 10 10 35 20 20 25 40 35 25 5 30 30 1 20 30 5 5 15 25 2 2000 1000 0 0 1000 1000 1000 0 2000 1000 3 10 10 0 0 10 10 0 0 10 10 0 0 10 10 0	1300 2 75 400 1 200 0 1 2000000 0 3 100

B Modulus

Data File: B.txt
Time allowed = 10 seconds

Given a positive integer X in radix (base) B compute X mod B-1.

For example:

$$7829_{10} \ mod \ 9 = 8$$
$$3777777777777773_8 \ mod \ 7 = 6$$
$$123456_7 \ mod \ 6 = 3$$

Input:
The first line of the input contains a single integer $P, (1 \leq P \leq 1000)$ giving the number of datasets to follow. Each dataset should be processed independently.

Each dataset consists of a single line of input containing three space-separated values. The first value is a decimal integer giving the dataset number, beginning at 1. The second value is the base or radix of the third number, $B(2 \leq B \leq 62)$, given as a decimal number. The third number is an unsigned integer, D, given in base B.

The digits of D will be selected from the first B characters of the string:

"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"

If $B = 4$, the digit set of D will be $\{0, 1, 2, 3\}$
If $B = 16$, the digit set for D will be $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$, where A represents 10_{10}, B represents 11_{10}, etc.

D will be no longer than 100 digits.

Output: For each dataset, output one line containing the dataset number followed by the value of D mod $(B - 1)$, presented in base B.

Sample Input	Sample Output
4	1 8
1 10 7829	2 3
2 7 123456	3 1
3 6 432504023545112	4 G
4 18 7AGH31FEH	

C Queens

Data File: C.txt
Time allowed = 10 seconds

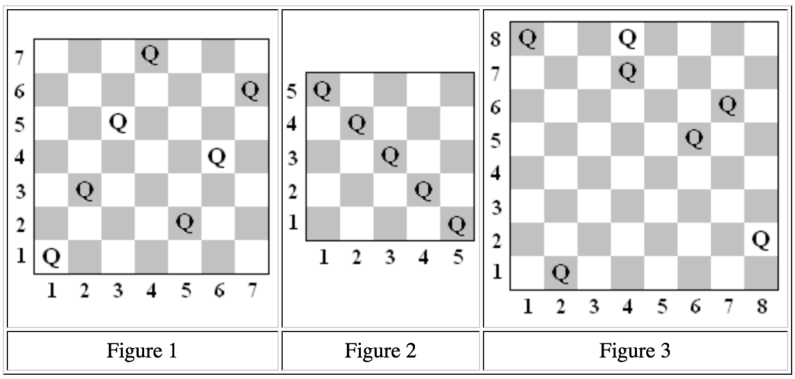


Figure 1 | Figure 2 | Figure 3

Two queens on a chessboard collide if they lie on the same row, column or diagonal, and there is no piece between them. Various sized square boards and numbers of queens will be considered. For example, Figure 1, with a 7×7 board, contains 7 queens with no collisions. In Figure 2 there is a 5×5 board with 5 queens and 4 collisions. In Figure 3, a traditional 8×8 board, there are 7 queens and 5 collisions.

On an $n \times n$ board, queen positions are given in Cartesian coordinates (x, y) where x is a column number in [1, n], and y is a row number in [1, n]. Queens at distinct positions (x1, y1) and (x2, y2) lie on the same diagonal if (x1- x2) and (y1- y2) have the same magnitude. They lie on the same row or column if x1= x2 or y1= y2, respectively. In each of these cases the queens have a collision if there is no other queen directly between them on the same diagonal, row, or column, respectively. For example, in Figure 2, the collisions are between the queens at (5, 1) and (4, 2), (4, 2) and (3, 3), (3, 3) and (2, 4), and finally (2, 4) and (1, 5). In Figure 3, the collisions are between the queens at (1, 8) and (4, 8), (4, 8) and (4, 7), (4, 7) and (6, 5), (7, 6) and (6, 5), and finally (6, 5) and (2, 1). Your task is to count queen collisions. In many situations there are a number of queens in a regular pattern. For instance in Figure 1 there are 4 queens in a line at (1,1), (2, 3), (3, 5), and (4, 7). Each of these queens after the first at (1, 1) is one to the right and 2 up from the previous one. Three queens starting at (5, 2) follow a similar pattern. Noting these patterns can allow the positions of a large number of queens to be stated succinctly.

Input:
The input will consist of one to twenty data sets, followed by a line containing only 0.

The first line of a dataset contains space-separated positive integers n and g, where n indicates an $n \times n$ board size, and g is the number of linear patterns of queens to be described, where $n < 30000$, and $g < 250$. The next g lines each contain five space-separated integers, $k\ x\ y\ s\ t$, representing a linear pattern of k queens at locations $(x + i * s, y + i * t)$, for $i = 0, 1, ..., k - 1$. The value of k is positive. If k is 1, then the values of s and t are irrelevant, and they will be given as 0. All queen positions will be on the board. The total number of queen positions

among all the linear patterns will be no more than n, and all these queen positions will be distinct.

Output:
There is one line of output for each data set, containing only the number of collisions between the queens.

The sample input data set corresponds to the configuration in the Figures. Take some care with your algorithm, or else your solution may take too long.

Sample Input	Sample Output
7 2 4 1 1 1 2 3 5 2 1 2 5 1 5 5 1 -1 1 8 3 1 2 1 0 0 3 1 8 3 -1 3 4 8 2 -3 0	0 4 5

D Crush 'Em

Data File = D.txt
Time limit = 10 seconds

This single-player game has a board with 12 rows and 10 columns, as shown in Figure 1. We label the rows **1** through **12**, starting at the bottom, and the columns **a** through **j**, starting at the left. Each grid location can either have a colored circle or be empty. (We use uppercase characters to denote distinct colors, for example with **B=blue**, **R=red**, and **Y=yellow**.) On each turn, the player clicks on a circle. The computer determines the largest "cluster" to which that circle belongs, where a cluster is defined to include the initial circle, any of its immediate horizontal and vertical neighbors with matching color, those circles' neighbors with matching colors, and so forth. For example, if a user were to click on the blue circle at cell (**h10**) in Figure 1, its cluster consists of those cells shown with empty circles in Figure 2.

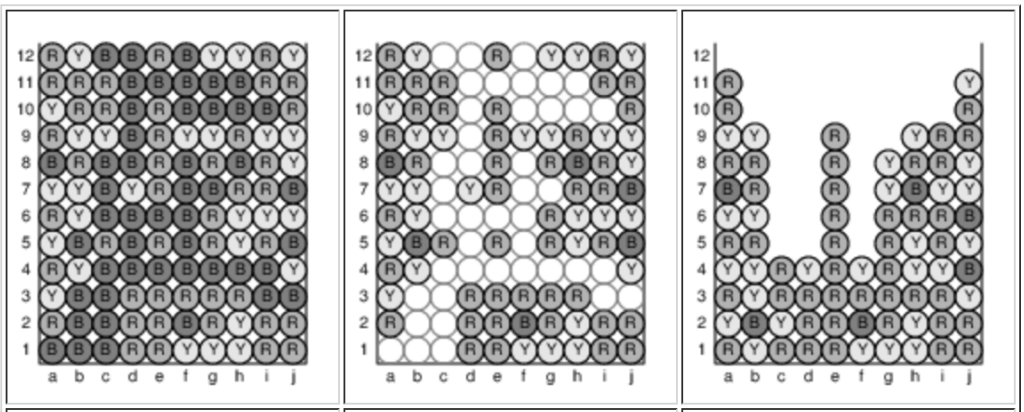


Figure 1 | Figure 2 | Figure 3

Processing a click on cell h10.

The player's turn is processed as follows. If the indicated grid cell belongs to a cluster of only one or two circles (or if there is no circle at that cell), the turn is wasted. Otherwise, with a cluster of 3 or more circles, all circles in the cluster are removed from the board. Remaining circles are then compacted as follows:

1 Circles fall vertically, to fill in any holes in their column.

2 If one or more columns have become empty, all remaining columns slide leftward (with each nonempty column remaining intact), such that they are packed against the left edge of the board.

For example, Figure 3 shows the board after the cluster of Figure 2 was removed following the click on (**h10**).

As another example, Figure 4 below, portrays the processing of a subsequent click on cell (**j1**). During that turn, column (**e**) becomes empty, and the resulting columns (**f**) through (**j**) slide to become columns (**e**) through (**i**), respectively. Figure 5 provides one further example in which several columns are compacted.

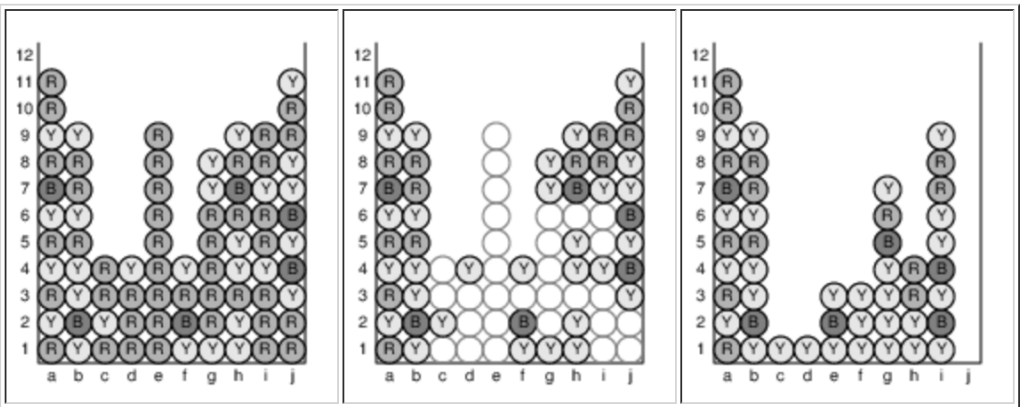


Figure 4: Processing a click on cell j1.

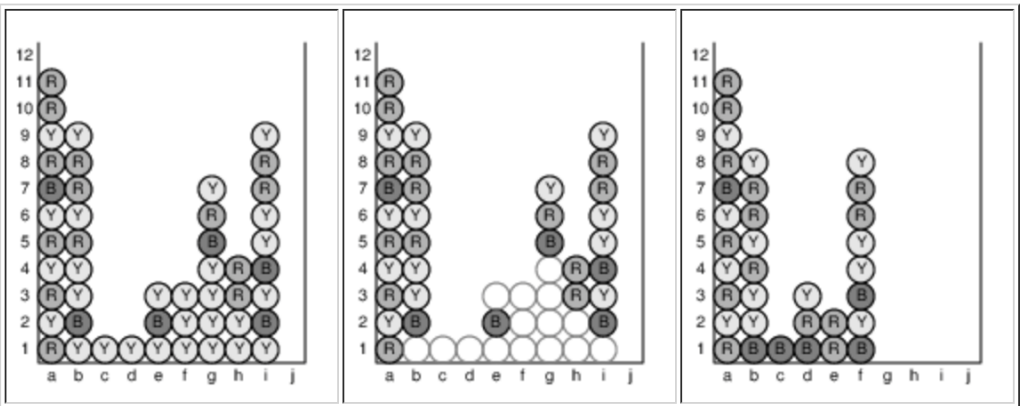


Figure 5: Processing a click on cell g2.

Input:
The input will consist of multiple games, each played with a new board. For each game, the input begins with a number T that denotes the number of turns that the player will be making, with $1 \leq T \leq 20$. Following that will be an initial board configuration, which always has 12 rows and 10 columns per row, with uppercase letters used to denote distinct colors. There will never be empty cells within the initial board. Following the presentation of the initial board will be T additional lines of input, each designating a cell of the grid; we rely on the coordinate system illustrated in the above figures, with a lowercase letter, from **a** to **j**, denoting a column and a number from **1** to **12** that denotes a row. We note that if a player clicks on a grid cell that does not currently have any circle, that turn is simply wasted.

The end of the entire input will be designated by a line with the number **0**.

Output:
For each game, output a single line designating the the number of circles that remain on the board after all of the player's turns are processed.

Sample Input	Sample Output
3 RYBBRBYYRY RRRBBBBBRR YRRBRBBBBR RYYBRYYRYY BRBBRBRBRY YYBYRBBRRB RYBBBBRYYY YBRBRBRYRB RYBBBBBBBY YBBRRRRRBB RBBRRBRYRR BBBRRYYYRR h 10 j 1 g 2 3 YYYYYBBBBB YYYYYBBBBB YYYYYBBBBB YYYYYBBBBB YYYYYBBBBB YYYYYBBBBB YYYYYBBBBB YYYYYBBBBB YYYYYBBBBB YYYYYBBBBB YYBYYBBBBB YYBYYBBBBB c 2 c 12 g 1 2 YYYYYBBBBB YYYYYBBBBB YYYYYBBBBB YYYYYBBBBB YYYYYBBBBB YYYYYBBBBB YYYYYBBBBB YYYYYBBBBB YYYYYBBBBB YYYYYBBBBB YYBYYBBBBB YYBYYBBBBB g 1 c 12 0	33 62 2

E Towers

Data File: E.txt
Time allowed = 10 seconds

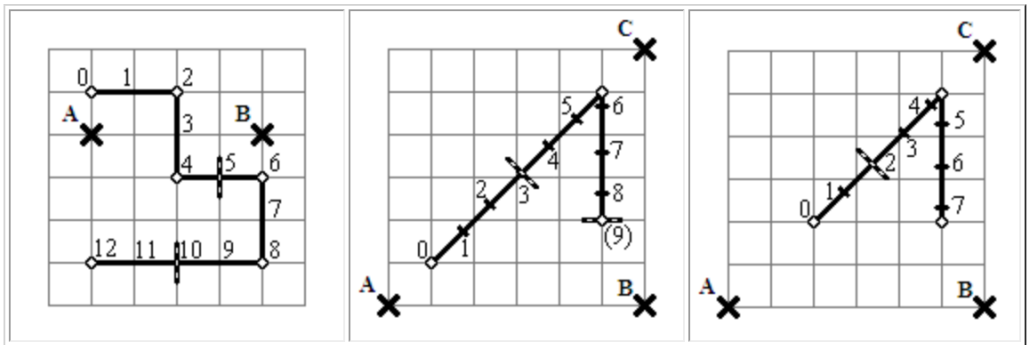
Cell phones generally provide service by connecting to a nearby cellular tower. At any given time there may be several towers in range. The cell phone, however, will only connect to the one with the best signal strength. The purpose of this problem is to track a traveler carrying a cell phone. At each mile marker along a road, note which tower the cell phone is using, and report when it is different from the previous marker.

The position of towers will be specified as $X-Y$ coordinates in miles relative to some arbitrary origin. The traveler will travel down a road composed of straight line segments laid end to end. The road will not intersect itself. Assume that there are markers occurring every mile along the road, with mile marker zero being at the starting point. If the road ends at least 0.5 miles past the last mile marker, the end of the road is labeled with the next mile. For instance if the road is 8.6 miles long, the endpoint is labeled as mile 9. If the road is 8.2 miles long, regular mile marker 8 is the last.

If d is the distance to a tower with power p, the signal strength for the tower will be calculated as p/d^2, rounded to the nearest integer. A tower will never be placed at the position of a mile marker.

Consider the examples shown in the Figures below. Each shows a road and labeled mile markers and cell towers.

In Figure 1, where the segments of the road all follow the background grid, mile markers come at intersections in the grid. The cell towers A and B are at (1, 4) and (5, 4). Both have power 1,000. At mile markers 0 and 1 the strength of A is greater. At mile markers 2 - 4 the strength of A and B are equal. In such cases you are to note the cell tower with the label closer to the beginning of the alphabet, so it continues to be A. At mile markers 5 - 9 B is stronger. At mile 10 the towers are of equal strength, but the one with first letter is reported, A. Tower A remains strongest to the end. The long crossbars show the mile markers where the reported strongest tower is different than at the previous marker.



The example in Figure 2 has three towers: A at (0, 0) and C at (6, 6), both with power 1,000, while tower B at (6, 0) has a power of 600. The mile markers are at the tick marks. They show

the traveler's progression along the road. The mile markers are no longer at grid intersections because of the angled road. Initially tower A is the strongest. Tower C is strongest at mile markers 3 - 8. The end of the road at (5, 2) is more than a half mile from mile 8, so it is labeled as mile 9. At the endpoint, B is strongest, unlike at mile 8, so a report is made for the endpoint.

Figure 3 is similar to Figure 2, except it shifts the beginning of the road and changes the power of cell tower B to 300. Tower A starts as the strongest, and then at mile markers 2 - 7 tower C is strongest. The endpoint at (5, 2) is not considered, since it is less than .5 miles from mile marker 7, even though tower B is strongest at this endpoint.

Input:
The input consists of at least one data set, followed by a line containing only 0.

The first line of a data set contains two space separated integers T R, where

T is the number of towers, $1 \leq T \leq 10$
R is the number of line segments which comprise the road, $1 \leq R \leq 10$
The next T lines each contain three space separated integers representing the X-coordinate, Y-coordinate, and power of one tower, respectively. The towers are implicitly labeled 'A', 'B', 'C', and so on.

The next line contains $2(R + 1)$ integers which are the coordinates for the $R + 1$ points that define the road. The road starts on the first point and moves in straight line segments through all R remaining points.

All coordinates will be integers between 0 and 100, inclusive. No two coordinate pairs are equal. The power for a tower will be an integer between 1 and 1,000,000, inclusive.

Output:
There is one line of output for each road in the data set. The line consists of ordered pairs separated by a single space. The first element of a pair is a number representing a mile marker. The second element of the pair is a letter corresponding to the tower with the strongest signal. There are entries for mile 0 and every mile marker where the strongest tower recorded is different than at the previous mile marker. The ordered pairs are surrounded by parentheses, and the elements are separated by a comma, with no whitespace inside the ordered pair.

The sample input data corresponds to the Figures.

Sample Input	Sample Output
2 5 1 4 1000 5 4 1000 1 5 3 5 3 3 5 3 5 1 1 1 3 2 0 0 1000 6 0 600 6 6 1000 1 1 5 5 5 2 3 2 0 0 1000 6 0 300 6 6 1000 2 2 5 5 5 2 0	(0,A) (5,B) (10,A) (0,A) (3,C) (9,B) (0,A) (2,C)

F We Have Parity

Data File: F.txt
Time allowed = 10 seconds

A bit string has odd parity if the number of 1's is odd. A bit string has even parity if the number of 1's is even. Zero is considered to be an even number, so a bit string with no 1's has even parity. Note that the number of 0's does not affect the parity of a bit string.

Input:
The input consists of one or more strings, each on a line by itself, followed by a line containing only "#" that signals the end of the input. Each string contains 1 to 31 bits followed by either a lowercase letter 'e' or a lowercase letter 'o'.

Output:
Each line of output must look just like the corresponding line of input, except that the letter at the end is replaced by the correct bit so that the entire bit string has even parity (if the letter was 'e') or odd parity (if the letter was 'o').

Sample Input	Sample Output
101e	1010
010010o	0100101
1e	11
000e	0000
110100101o	1101001010
#	

G Steganography

Data File: G.txt
Time allowed = 10 seconds

In cryptography, the goal is to encrypt a message so that, even if the the message is intercepted, only the intended recipient can decrypt it. In steganography, which literally means "hidden writing", the goal is to hide the fact that a message has even been sent. It has been in use since 440 BC. Historical methods of steganography include invisible inks and tatooing messages on messengers where they can't easily be seen. A modern method is to encode a message using the least-significant bits of the RGB color values of pixels in a digital image.

For this problem you will uncover messages hidden in plain text. The spaces within the text encode bits; an odd number of consecutive spaces encodes a 0 and an even number of consecutive spaces encodes a 1. The four texts in the example input below (terminated by asterisks) encode the following bit strings: 11111, 000010001101101, 01, and 000100010100010011111. Each group of five consecutive bits represents a binary number in the range 0-31, which is converted to a character according to the table below. If the last group contains fewer than five bits, it is padded on the right with 0's.

" " (space)	0
"A" - "Z"	1 - 26
" ' " (apostrophe)	27
"," (comma)	28
"-" (hyphen)	29
"." (period)	30
"?" (question mark)	31

The first message is $11111_2 = 31_{10}$ = "?". The second message is $(00001, 00011, 01101)_2 = (1, 3, 13)_{10}$ = "ACM". The third message is $01\underline{000}_2 = 8_{10}$ = "H", where the underlined 0's are padding bits. The fourth message is $(00010, 00101, 00010, 01111, 1\underline{0000})_2 = (2, 5, 2, 15, 16)_{10}$ = "BEBOP".

Input:
The input consists of one or more texts. Each text contains one or more lines, each of length at most 80 characters, followed by a line containing only "*" (an asterisk) that signals the end of the text. A line containing only "#" signals the end of the input. In addition to spaces, text lines may contain any ASCII letters, digits, or punctuation, except for "*" and "#", which are used only as sentinels.

Output:
For each input text, output the hidden message on a line by itself. Hidden messages will be 164 characters long.

Note: There may be consecutive spaces within an input line. There will be no spaces at the beginning or end of a line.

Sample Input	Sample Output
Programmer, I would like to see a question mark. * Behold, there is more to me than you might think when you read me the first time. * Symbol for hydrogen? * A B C D E F G H I J K L M N O P Q R S T U V * #	? ACM H BEBOP

H Tiling a Rectangle

Data File = H.txt
Time limit = 10 seconds

You are given the dimensions, *length, width*, of a rectangular floor. Square tiles, all of the same size, are to be used to tile the floor. Find the size of the largest tile that can be used to cover the floor completely without any tiles overlapping and without cutting any tiles. Assume that the tiles are made of marble so that no grout is needed between the tiles.

Input:
The input file will contain up to 20 test cases. Each test case will comprise two integers, X, Y, $0 < X, Y < 100000$, space separated, on a line by itself. The input file will be terminated by End of File.

Output:
For each test case output one line of text containing the length of the side of the largest square tile that can be used to tile the rectangle.

Sample Input	Sample Output
53 17	1
21 24	3
39 130	13
30 16	2

I Gnomes

Data File: I.txt
Time allowed = 10 seconds

In the book *All Creatures of Mythology*, gnomes are kind, bearded creatures, while goblins tend to be bossy and simple-minded. The goblins like to harass the gnomes by making them line up in groups of three, ordered by the length of their beards. The gnomes, being of different physical heights, vary their arrangements to confuse the goblins. Therefore, the goblins must actually measure the beards in centimeters to see if everyone is lined up in order.

Your task is to write a program to assist the goblins in determining whether or not the gnomes are lined up properly, either from shortest to longest beard or from longest to shortest.

Input:
The input starts with line containing a single integer N, $0 < N < 30$, which is the number of groups to process. Following this are N lines, each containing three distinct positive integers less than 100.

Output:
There is a title line, then one line per set of beard lengths. See the sample output for capitalization and punctuation.

Sample Input	Sample Output
3	Gnomes:
40 62 77	Ordered
88 62 77	Unordered
91 33 18	Ordered

J Barrel Roping

Data File: J.txt
Time allowed = 10 seconds

The Oil Drum Company makes steel drums that are used for oil transportation. Each drum is cylindrical in shape, 4 ft high, with a diameter of 3 ft. In order to keep the barrels from moving around during delivery, they are grouped into clusters of size N, where $N \in [2, 12]$ and a rope is tied around each cluster. The manager needs to know the length of rope needed for each cluster size.

Here is what we found during a day at the library:
In two dimensional Euclidean space, Joseph Louis Lagrange proved in 1773 that the highest-density lattice arrangement of circles is the hexagonal packing arrangement, in which the centers of the circles are arranged in a hexagonal lattice (staggered rows, like a honeycomb), and each circle is surrounded by 6 other circles. The density of this arrangement is:

$$\eta = \frac{\pi}{2\sqrt{3}}$$

This density equation applies to an infinite array of equal diameter circles, and is therefore not much value in our problem. The manager decides, however, to use hexagonal packing within each cluster size, as illustrated in Figure 1.

What is the length of the shortest rope needed to tie around each cluster size? Add 6 inches for knots and *round each answer up to the nearest foot.*

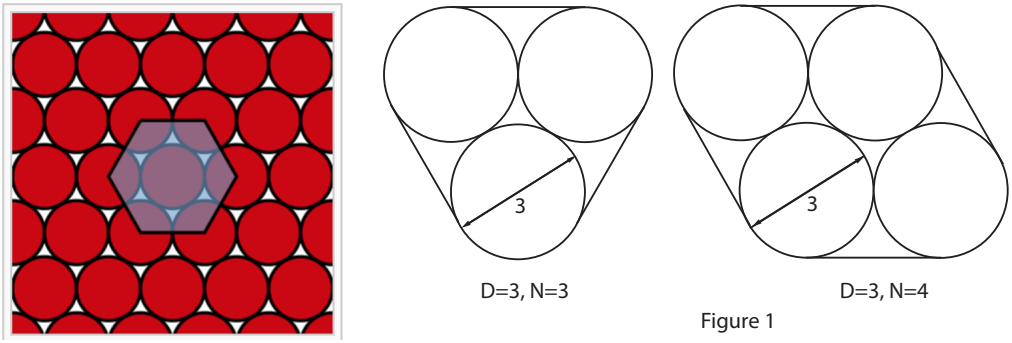


Figure 1

Input:
There is no input data file for this problem.

Output:
Output one line of text for each value of $N \in [2, 12]$ in that order Each line should contain the value of N followed by a space, then the rope length for that cluster size.

First Two Lines of Output
2 16
3 19