University of Virginia

2011 Computer Science
High School Programming Contest

Welcome to the University of Virginia's first annual High School Programming Contest, sponsored by the student chapter of the Association for Computing Machinery at UVa. Before you begin, please take the time to read the following rules:

1. There are nine (9) problems in the packet, using letters A–I. These problems are NOT sorted by difficulty. When a team's solution is judged correct, the team will be awarded a balloon. The balloon colors are as follows:

Problem	Problem Name	Balloon Color
A	Geometric Inflation	Yellow
B	Public Transit	Blue
C	Password Validation	White
D	Arithmetic and Geometric Sums	Red
E	Robot in a Maze	Orange
F	Triangular Matrices	Green
G	Integer Flipping	Pink
H	Property Lines	Black
I	Lifeboat Balancing	Purple

2. Solutions for problems submitted for judging are called runs. Each run will be judged. The judges will respond to your submission with one of the following responses. In the event that more than one response is applicable, the judges may respond with any of the applicable responses.

Response	Explanation
Correct	Your submission has been judged correct.
Wrong Answer	Your submission generated output that is not correct or is incomplete.
Output Format Error	Your submission's output is not in the correct format or is misspelled.
Excessive Output	Your submission generated output in addition to or instead of what is required.
Compilation Error	Your submission failed to compile.
Run-Time Error	Your submission experienced a run-time error.
Time-Limit Exceeded	Your submission did not solve the judges' test data within 30 seconds.

3. A team's score is based on the number of problems they solve and penalty points, which reflect the amount of time and number of incorrect submissions made before the problem is solved. For each problem solved correctly, penalty points are charged equal to the time at which the problem was solved plus 20 minutes for each incorrect submission. No penalty points are added for problems that are never solved. Teams are ranked first by the number of problems solved and then by the fewest penalty points.

4. This problem set contains sample input and output for each problem. However, you may

be assured that the judges will test your submission against several other more complex datasets, which will not be revealed until after the contest. As a major challenge, you should design other input sets for yourself, so that you may fully test your program before submitting your run. Should you receive a judgment stating that your submission was incorrect, you should consider what other datasets you could design to further evaluate your program.

5. In the event that you feel a problem statement is ambiguous or incorrect, you may request a clarification. Read the problem carefully before requesting a clarification. If the judges believe that the problem statement is sufficiently clear, you will receive the response, "The problem statement is sufficient; no clarification is necessary." If you receive this response, you should read the problem description more carefully. If you still feel there is an ambiguity, you will have to be more specific or descriptive of the ambiguity you have found. If the problem statement is ambiguous in specifying the correct output for a particular input, please include that input data in the clarification request.

   You may not submit clarification requests asking for the correct output for inputs that you provide. Sample inputs may be useful in explaining the nature of a perceived ambiguity, e.g., "There is no statement about the desired order of outputs. Given the input: . . . , would both this: . . . and this: . . . be valid outputs?"

   If a clarification is issued during the contest, it will be broadcast to all teams.

6. Runs for each particular problem will be judged in the order they are received. However, it is possible that runs for different problems may be judged out of order. For example, you may submit a run for B followed by a run for C, but receive the response for C first. Do not request clarifications on when a response will be returned. If you have not received a response for a run within 30 minutes of submitting it, you may have a runner ask the site judge to determine the cause of the delay. Under no circumstances should you ever submit a clarification request about a submission for which you have not received a judgment.

   If, due to unforeseen circumstances, judging for one or more problems begins to lag more than 30 minutes behind submissions, a clarification announcement will be issued to all teams.

   This announcement will include a change to the 30 minute time period that teams are expected to wait before consulting the site judge.

7. The submission of abusive programs or clarification requests to the judges will be considered grounds for immediate disqualification.

8. All solutions must read from standard input and write to standard output. In C this is scanf/printf, in C++ this is cin/cout, and in Java this is System.in/System.out. The judges will ignore all output sent to standard error (cerr in C++ or System.err in Java). You may wish to use standard error to output debugging information. From your workstation you may test your program with an input file by redirecting input from a file:

   program < file.in

9. Every effort has been made to ensure that the compilers and run-time environments used by the judges are as similar as possible to those that you will use in developing your code.

With that said, some differences may exist. It is, in general, your responsibility to write your code in a portable manner compliant with the rules and standards of the programming language.

You should not rely on undocumented and non-standard behaviors.

One place where differences are likely to arise is in the size of the various numeric types. Many problems will specify minimum and maximum values for numeric inputs and outputs. You should write your code with the understanding that, on the judges' machines:

A C++ int, a C++ long, and a Java int are all 32-bits wide.
A C++ long long and a Java long are 64-bits wide.
A float in both languages is a 32-bit value capable of holding 6-7 decimal digits, though many library functions will be less accurate.
A double in both languages is a 64-bit value capable of holding 15-16 decimal digits, though many library functions will be less accurate.

The data types on your own machines may differ in size from these, but if you follow the guidelines above in choosing the types to hold your numbers, you can be assured that they will suffice to hold those values on the judges' machines.

10. All lines of program input will end with the appropriate line terminator (e.g., a linefeed on Unix/Linux systems, a carriage return-linefeed pair on Windows systems).

11. All input sets used by the judges will follow the input format specification found in the problem description. You do not need to test for input that violates the input format specified in the problem.

12. Unless otherwise specified, all lines of program output should be left justified, with no leading blank spaces prior to the first non-blank character on that line, should end with the appropriate line terminator (\n, endl, or println()), and should not contain any blank characters at the end of the line, between the final specified output and the line terminator, should not print extra lines of output, even if empty, that are not specifically required by the problem statement.

13. Unless otherwise specified, all numbers in your output should begin with the – (minus sign) if negative, followed immediately by 1 or more decimal digits. If it is a real number, then the decimal point should appear, followed by the appropriate number of decimal digits. For output of real numbers, the number of digits after the decimal point will be specified in the problem description (as the "precision").

    All real numbers printed to a given precision should be rounded to the nearest value. Rounding should be carried out so that trailing digits of 5 of higher are rounded up, trailing digits of 4 or less are rounded down. For example, if a precision of 2 decimal digits is requested, then 0.0152 would round to 0.02, but 0.0149 would round to 0.01.

    In simpler terms, neither scientific notation nor commas will be used for numbers, and you should ensure you use a printing technique that rounds to the appropriate precision.

14. If a problem specifies that an input is a floating point number, the input will be presented according to the rules stipulated above for output of real numbers, except that decimal points and the following digits may be omitted for numbers with no non-zero decimal

portion.

Scientific notation will not be used in input sets unless a problem explicitly allows it.

15. If you wish to print, you may do so to printer `cs_113`. Make sure your team's school is included in a comment on the first line of the printed output.
16. The reference material, scoreboard, sample input/output and a template for java file IO may be found at: http://plato/~hspc/

Good luck and HAVE FUN!!!

Problem A: Geometric Inflation

A very unusual model of inflation can be simulated by multiplying the cost of a given item 3 months ago with the cost of the item 2 months ago and dividing that quantity by the cost of the item 1 month ago. The costs will be given in dollars, with no more than 2 decimal places representing cents. No cost can include partial cents, so always round to the nearest whole cent for every month.

For example, suppose dumplings from the corner cost $5.23 in month 1, $5.50 in month 2, and $5.52 in month 3. It is now month 4 and we need to find out how much to charge considering inflation. We would multiply 5.23 by 5.50 to get 28.765. Note: no rounding takes place at this step. Then take that quantity and divide it by 5.52 to get 5.21.

Input

Each line will start with 3 decimal numbers, with 1 or 2 decimal places, representing the costs at month 1, 2, and 3. The next input on the line will be an integer, N, between 1 and 100 inclusively indicating for which month the cost is to be calculated. A line starting with -1 will mark the end of the test cases.

Output

For each test case (line of input) you should calculate the cost for that month and output your result as “Month N cost: $c”, where c is the cost in dollars. Each output should have exactly 2 decimal digits and be comma grouped. The cost for every month that you are requested to find will fit in a Java double.

Example

Sample Input

```
5.23 5.5 5.52 4
5.23 5.5 5.52 2
-1
```

Sample Output

```
Month 4 cost: $5.21
Month 2 cost: $5.50
```

Problem B: Public Transit

At UVa, engineers have a way of looking for more optimal solutions until it can be proven that no better solution can be found. To students from the College of Arts and Sciences nerd sniping[1] us engineers looking at bus maps; we need a solution that will find the shortest route from one stop number to another, faster than mere humans.

To achieve this you will be given a list of directional bus routes, the time it takes the bus to travel that route, and how many minutes from now the bus will arrive at that route. You will also be given the travel time (walking) to each bus stop. Your task is to find the most optimal (least time) chain of stops that will get you from your current location to your destination.

Input

The first line of input will be an integer W, indicating how many stops are within walking distance of your current location. The following W lines will represent the stops that are within walking distance of your location. Each of these lines will have a 5 digit integer stop identifier, followed by the number of minutes it will take you to reach that stop.

After the stops you can walk to, an integer, R, will be provided telling you how many routes will follow. Each of the next R lines will contain represent bus routes. Each route will have a 5 digit stop ID indicating where you can get on a bus, followed by a time of arrival (in minutes from the current time), followed by another 5 digit stop ID indicating where that route leads and finally an integer representing how many minutes it takes the bus to reach the destination from the time you get on. After the R routes a 5 digit stop ID will be given on a line by itself, this is your destination. Time starts at time equal to zero. Test cases will be repeated until a line with only 0.

Output

Print a single line of output including the number total duration of the trip and the stops to take in the order you take them. Use the form “ETA: *X* minute(s) Take stops: *Stop1 Stop2 StopN*” If the given ending destination is unreachable print “Unreachable”. If there are multiple ways to reach the destination you must print the one with the least number of stops.

[1] http://xkcd.com/356/ (Reproduced with permission on next page.)

Example

Input

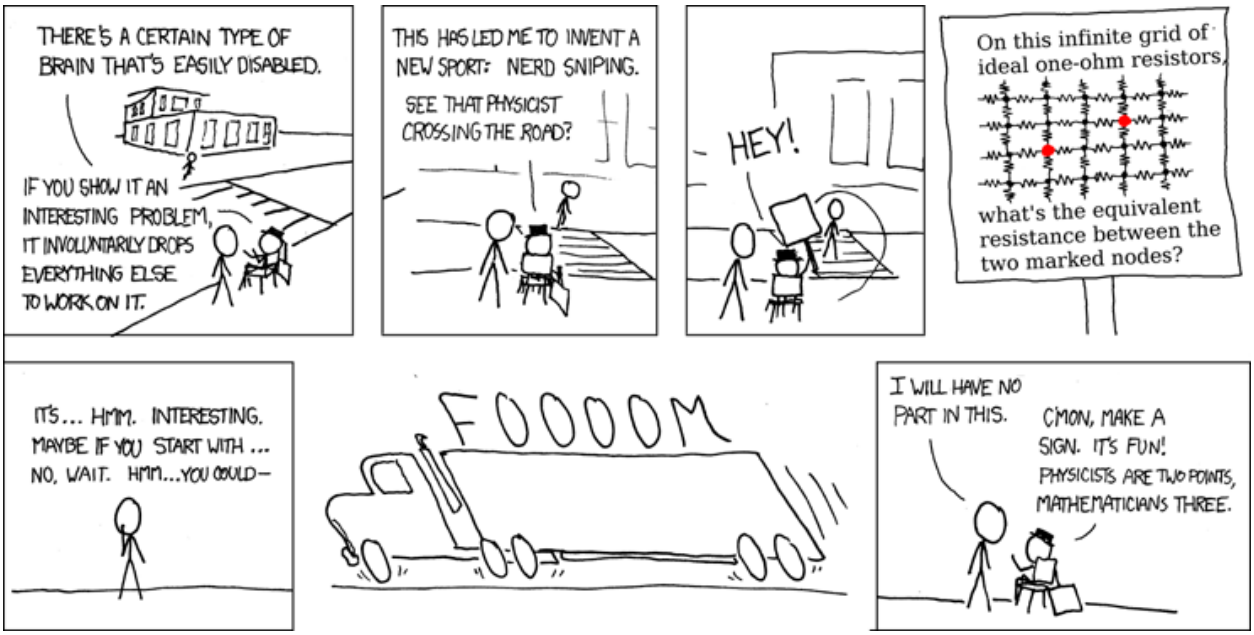
```
1
10000 2
7
10000 2 10001 5
10001 7 10002 3
10002 10 10003 2
10003 12 10004 7
10004 19 10005 10
10000 0 10005 5
10000 3 10004 16
10004
0
```

Output

```
ETA: 19 minute(s) Take stops: 10000 10004
```

XKCD Nerd Sniping

Note: This is completely irrelevant to the actual problem.



Problem C: Password Validation

A company has asked you to write a password checker for their online system. When new users enter their passwords, your program must determine whether or not it is a good password. A good password meets all of the following requirements.

- The password is between 9 and 20 characters in length inclusive (its length can be equal to 9 or 20)
- The password contains at least 2 lowercase letters
- The password contains at least 2 uppercase letters
- The password contains at least 1 number
- The password contains at least 2 non-alphanumeric characters (assume the only non-alphanumeric characters are : ! @ # $ % ^ & * . , ; / ?
- The password does not contain any 3 consecutive characters (case matters)
- Disregarding all non-alphanumeric characters and case, the subsequence of alphanumeric characters is not a palindrome (for example: "&Ra^#r" still counts as a palindrome)
- The password contains no subsequence of alphanumeric characters, disregarding case, either in forward or reverse order that is equal to any of the following words (for example: pKassWordL and KdRrowSmsap are both invalid passwords):

password
virginia
cavalier
code

Input

The first line of input will contain an integer N signifying the number of test cases. The following N lines will contain a string representing the password to be checked. The passwords will contain no spaces.

Output

For each test case, determine whether it meets the requirements for the passwords as stated above. If the test case is valid, print "Valid Password", if the test case is invalid, print "Invalid Password".

Example

Sample Input

```
3
SPring2011!!
pKassW&ordL*
123^45ABcde.
```

Sample Output

```
Valid Password
Invalid Password
Valid Password
```

Problem D: Arithmetic and Geometric Sums

An arithmetic series is defined as a series of numbers where each following number is an additive constant away from the previous number.

$$a_1 + a_2 + a_3 + \ldots + a_n = a_1 + (a_1 + d) + (a_1 + 2d) + \ldots + (a_1 + nd)$$

The sum of all values from a_1 to a_n is given by:

$$S_n = \frac{n[2a_1 + (n-1)d]}{2}$$

A geometric series is defined as a series of numbers where each following number is a multiplicative constant away from the previous number.

$$a_1 + a_2 + a_3 + \ldots + a_n = a_1 + (ra_1) + (r^2 a_1) + \ldots + (r^n a_1)$$

The sum of all values from a_1 to a_n is given by:

$$S_n = a_1 \frac{r^n - 1}{r - 1}$$

Given the first 3 numbers of either a geometric or arithmetic series, determine the sum to N terms. The first term is index 1, the sum to N terms should include the Nth term in the calculation.

Input

The input will be given by 2 lines for each data set. The first line will be N, the term to which the sum should be computed. The next line of data is the series of numbers. They will be integers with a space separating each one. The series will be either geometric or arithmetic. The end of the input will be signaled by an N value of zero.

Output

For each test case output S_n on its own line.

Example

Sample Input

```
4
9 27 81
36
29 94 159
0
```

Sample Output

```
360
41994
```

Problem E: Robot in a Maze

A robot is stuck in a maze with multiple exits and needs to find the shortest path out. The maze can be represented as a two dimensional grid where each space is either a wall, which the robot cannot go through, an exit, or one of the goals. The robot can only move one space at a time, and at each move it can only move either horizontally or vertically, not diagonally. It can exit the maze from any exit, but it needs to start at the specified place. What is the fewest number of moves the robot can make to exit from the maze?

Input

The input will be given first by a number that tells the number of different mazes in the input file. Following that, for each data set, there will be two integers, R and C. The first number, R, represents the number or rows in the maze, and the second number, C, representing the number of columns. The following R lines will each be a series of characters representing spaces on the grid for that row. An X will represent an obstacle, and O (the letter) will represent an open space that the robot can move. An S will represent the starting location of the robot. The goals in the maze will be given by the letter G. There may be more than one goal.

Output

The output should be if the robot can get out of the maze:

"Shortest Path: t"

Where t is an integer representing the length of the shortest path. If there is no way out of the maze, the output should be:

"No Exit"

Each line of output should be printed at the end of the program.

Example

Sample Input

```
3
7 8
XXXXXXXX
XSOOXOOX
XOXOOOOX
XOXXXOOX
XOXOXXOX
XOOOXOOX
XXXXXGXX
4 4
XGXX
XSOX
XOOX
XGGX
4 4
XXXX
XSOX
XXXX
XXGX
```

Sample Output

```
Shortest Path: 11
Shortest Path: 1
No Exit
```

Problem F: Triangular Matrices

Let triangular matrices be defined as equilateral triangles with numbers in the form:

$$\begin{matrix} a_1 \\ a_2 \ a_3 \\ a_4 \ a_5 \ a_6 \end{matrix}$$

Let triangular matrix addition be defined as:

$$\begin{matrix} a_1 \\ a_2 \ a_3 \\ a_4 \ a_5 \ a_6 \end{matrix}$$

$$+$$

$$\begin{matrix} b_1 \\ b_2 \ b_3 \\ b_4 \ b_5 \ b_6 \end{matrix}$$

$$\begin{matrix} a_1 + b_1 \\ a_2 + b_2 \ a_3 + b_3 \\ a_4 + b_4 \ a_5 + b_5 \ a_6 + b_6 \end{matrix}$$

Let triangular matrix multiplication be defined as:

$$\begin{matrix} a_1 \\ a_2 \ a_3 \end{matrix}$$

$$*$$

$$\begin{matrix} b_1 \\ b_2 \ b_3 \\ b_4 \ b_5 \ b_6 \end{matrix}$$

$$\begin{matrix} a_1b_1 + a_2b_2 + a_3b_3 \\ a_1b_2 + a_2b_4 + a_3b_5 \ a_1b_3 + a_2b_5 + a_3b_6 \end{matrix}$$

The multiplication operation is not commutative, and the first matrix, A, must be of size less than or equal to B. The elements in the resultant matrix will be determined by matching A with every triangle of size A within the matrix B. For each of these matchings, multiply the overlapping elements of A and B and then sum them. Each matching will have a corresponding place in the resulting matrix.

In the example above, the A matches to the sub matrices formed by: $\{b_1, b_2, b_3\}$, $\{b_2, b_4, b_5\}$, and $\{b_3, b_5, b_6\}$. So taking the element wise multiplication and then sum of each sub matrix with matrix A gives the values c = $a_1b_1 + a_2b_2 + a_3b_3$, d = $a_1b_2 + a_2b_4 + a_3b_5$, and e = $a_1b_3 + a_2b_5 + a_3b_6$. Since c corresponds to the top submatrix from B, it will be in the top position in the resulting matrix. d and e will be in the next row with d to the left of e because that is how the original corresponding submatrices of B were arranged.

The input will be given in postfix notation. For example, if you were given:

A B * A +

That would correspond to:

(A * B) + A

To solve this expression, start from the left side. First you get A, then B, then *. Which means A*B, we'll call this result C. Replace A B * with C, since that operation has been done. That leaves C A +, which means A + C, giving you the final result.

Input

Input sets will be given by an integer N, the number of matrices involved in the operation. For the next N lines, the matrices will be specified. A matrix will be specified by a string identifier K followed by an integer L, the length of each side of the matrix. The next L lines will contain the values in the matrix, each value separated by a space. The first line will only contain 1 value, the second line will contain 2 values, the third line will contain three values, and so on until the L^{th} line. The first line with one value signifies the top point of the triangle, and the L^{th} line of L values signifies the bottom of the triangle.

After the matrices are given, there will be an expression, with spaces between the operators and symbols that needs to be evaluated. The expression will contain matrices that are represented by their string identifiers. The end of input will be given by N = 0.

Output

The output should be given by the resultant triangular matrix. It should be output in the same way the matrices were input. The top value on the first line, the next lowest values in order on the second line, all the way to the bottom row. If the expression is impossible to evaluate, output: “Invalid expression”.

Example

Sample Input

```
2
A 2
1
2 2
B 3
1
3 1
2 2 1
A B * A +
0
```

Sample Output

```
10
13 9
```

Problem G: Integer Flipping

Aliens have finally visited Earth. They are in fact, similar to us in every single way except in the way that they represent integers in binary. We represent an integer in a computer as a 32-bit binary. These aliens represent numbers completely differently, but still as a 32-bit number.

Given an unsigned 32-bit number in our representation, determine how the aliens would represent it. To determine how the aliens would represent it:

1. Convert the number to its binary representation
2. Reverse the order of the bits

Finally, take the reversed order number and convert it back to a positive, unsigned decimal as we would represent it.

For example, if they were 4-bit numbers and you were given the value 4, you would convert it to binary and get 0100_2. Then flip it and get 0010_2. Finally convert it back to decimal to get the final answer of 2.

Input

Each line will contain a single integer on which to perform the integer flipping algorithm. The number will be in Earth's representation. A line starting with -1 will mark the end of the test cases.

Output

For each integer in Earth's integer representation, you should calculate the value of the aliens' "flipped" integer and print its value in decimal format. More specifically, take the input number, convert it into a 32-bit integer, flip this integer into the aliens' format, and then evaluate the 32-bit binary number as if it were still in earth's representation. Print this value in decimal format.

Example

Sample Input

```
2
1
536870912
-1
```

Sample Output

```
1073741824
2147483648
4
```

Problem H: Property Lines

A rectangular city recently decided that it needed to draw new property lines because the divides were too outdated. The mayor thought it would be a good idea to let each property owner redraw his or her own property lines. The only stipulation is that all property has to be drawn in rectangles. Unfortunately, this created a lot of conflicts. The city needs to find out what land is currently disputed and what land is unclaimed.

It is your job to figure out the amount of land area that is currently claimed by more than one property owner and what amount of land area has been claimed by no one. No land outside of the city will be claimed.

Input

The input will be given first by two floating point numbers, W and H, and an integer N in that order. The city is a rectangle in the x-y plane with length in the x direction given by W and length in the y direction given by H. The lower left corner of the city is situated at (x,y) = (0,0). N represents the number of residents making claim to some amount of property. The following N lines will contain property claims in the format : X Y LX LY, where all numbers are floating point numbers. The coordinates (X, Y) represent the lower left corner of the rectangular property claim. LX is the length in the x direction of the rectangular property claim and LY is the length in the y direction of the rectangular property claim. The end of the program will be given by W H N corresponding to 0 0 0.

Output

The output should be in the format:

Disputed: d
Claimed: c
Unclaimed: u

Where d is the amount of land that is claimed by 2 or more people, c is the amount of land claimed by 1 or more person, and u is the amount of land that is claimed by no one. The output should be rounded to 3 decimal places with no comma grouping.

Example

Sample Input

```
4.0 4.0 2
1.0 1.0 2.0 2.0
2.0 1.0 2.0 2.0
0 0 0
```

Sample Output

```
Disputed: 2.000
Claimed: 6.000
Unclaimed: 10.000
```

Problem I: Lifeboat Balancing

Six Virginia Tech engineers recently designed and built a large cruise ship. Because of its many obvious flaws, it has just begun to sink on its first voyage. There are two lifeboats on board, each of which can fit half of the passengers on board. The only problem is that the total weight of the people on each of the lifeboats needs to be as equal as possible so that neither lifeboat weighs too much and risks sinking. It is your job to determine how people should be distributed between lifeboats so that the weight is as equal as possible and there is an equal number of people on each boat. All passengers must make it onto a lifeboat.

Input

The first number in the input will be C, the number of test cases.

For each of the C following test cases, the first line will be an integer N, the total number of people that need to go on the lifeboats ($1 \leq N \leq 5000$). The following N lines will contain an integer weight, W, representing the weight of one passenger ($1 \leq W \leq 1000$).

Output

For each of the test cases, determine the sum of the weights of all the people on each lifeboat given the most equal distribution of weights. If the weights on each boat are not equal, then output the lower number first. Both weights should be output on the same line, separated by a space.

Example

Sample Input

```
2
4
20
10
15
10
5
17
13
25
4
6
```

Sample Output

```
25 30
31 34
```