(Factorials)!

Problem 1

Given a non-negative integer *n*, we define *n*! (pronounced *n* factorial) as follows:

$$n! = n * (n - 1) * (n - 2) * \ldots * 2 * 1.$$

By convention, 0! is defined to be 1. For this problem, we define *n*# to be the number of trailing zeros on *n*! Here are a few examples:

5! = 12**0**, and so 5# = 1.
2! = 2, and so 2# = 0.
16! = 20,922,789,888,**000**, and so 16# = 3.
174! = 6,425,425,663,347,064,733,166,342,506,526,881,458,348,150,508,
160,426,541,851,455,077,080,468,607,287,618,805,557,105,047,805,861,
775,775,912,692,278,116,502,462,953,528,378,524,937,389,131,268,196,
460,620,409,529,506,610,362,739,317,326,974,626,432,136,901,748,478,
076,969,280,322,196,922,086,635,340,638,923,811,068,556,323,532,935,
233,826,663,322,108,954,882,867,2**00,000,000,000,000,000,000,000,000,**
000,000,000,000,000, and so 174# = 41.

It is your task to compute the # function for various input values.

Details of the Input

The first line will contain the number *m* of cases to follow, and then each of the following *m* lines contain a non-negative integer *n* which is *no greater than 1000.*

Details of the Output

Each line of your output will look like

n: *n*#

for that particular value of *n*. There should be two (2) spaces between the colon and the value of *n*#. The value of *n*# is guaranteed to fit within the bounds of a standard integer. We do not, however, make any promises about the size of *n*!, so beware…

Sample Input	Sample Output
4	5: 1
5	2: 0
2	16: 3
16	174: 41
174	

Degree Days

Problem 2

A degree day is a meteorological statistic that has a wide array of applications. In short, a degree day indicates a deviation from a standard average temperature. It is a single index that can be used to determine how hot or cold it has been at a location over a period of time. Degree days are used by farmers to quantify the warmth and progress of a growing season, while energy companies use degree days to account properly for the effect of weather on energy consumption.

There are two types of degree days. A **cooling degree day** indicates how much warmer than normal it has been over a period of time while a **heating degree day** indicates how much cooler than normal it has been over time. A cooling degree day therefore is an index that can be used to estimate how much energy has been used for cooling purposes, while a heating degree day can show what amount of energy has been used for heating.

The most common number used for a "normal" temperature is **65° F**, and that is the temperature we will use for this problem. To calculate a degree day for a given day, calculate the average temperature for the day (the mean of the high and low temperature) and then calculate the difference between that day's average temperature and 65°. If that day's average was higher that 65°, you have found the day's cooling degree days. If that day's average temperature was lower than 65°, you have found the day's heating degree days. On any given day, heating degree days or cooling degree days can occur, **but not both**.

Daily averages that are calculated must be rounded **up** to the next integer value if the average is not an integer. Thus, a calculated average of 60.1° should be rounded up to 61°. Additionally, all degree days are positive integers – thus, any negative values calculated must be expressed as their absolute value.

Details of the Input

Your input will begin with a number $n < 30$ that indicates how many periods of data will follow. Then, n groups of data will follow. Each group will begin with an integer m that indicates how many days of data are in that data group. Following that integer will be m lines of data. Each line will contain a particular day's high temperature and then that day's lower temperature in degrees Fahrenheit.

Details of the Output

You are to calculate the total number of heating and cooling degree days for each period and output your results. **Be extremely cautious** with your output as proper grammar must be used.

Sample Input

```
3
3
75 55
73 59
57 45
4
78 55
86 67
65 45
41 32
1
48 32
```

Sample Output

```
In period 1 there were 14 heating degree days and 1 cooling degree day.
In period 2 there were 38 heating degree days and 14 cooling degree days.
In period 3 there were 25 heating degree days and 0 cooling degree days.
```

Budgeting with Debit Cards

Problem 3

Many teens have prepaid debit cards. College students have similar cards like Baldwin-Wallace's *Jacket Express Card* that are funded by parents, grandparents, other guardians, or unjustifiably high tuitions. Cards of this type have online statements that show credits and purchases. Most of these sites, however, do not have a purchase planner.

For this problem, write a program that will allow a student to plan a budget and check whether there are sufficient funds. If there are currently sufficient funds, the ending balance for the month should be printed to the screen. If not, output the number of months needed to save for the purchases and report what the ending balance will be after that number of months. If the purchase requires more than a student will have after 12 months pass, print a message stating that the purchase or purchases cannot be afforded.

Details of the Input

Input will begin with a number *n* denoting the number of students to plan purchases for. For each of the *n* cases, the first line will contain the starting balance, the monthly load increment, and the number *m* of planned purchases to follow. The monthly credit is applied at the end of each month. The next *m* lines contain the purchases which will each have a one-word description of what the money is for, and then the amount that will be spent.

Details of the Output

Output format of the output should follow the given examples. Be sure to use proper plural forms of words.

Sample Input

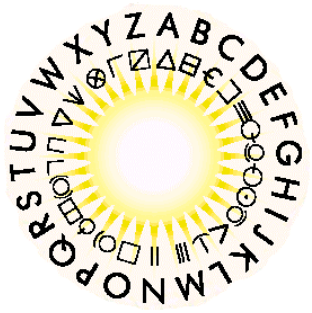
```
3
100.00 75.00 4
CDs 23.80
Clothes 40.00
Gasoline 25.00
Gifts 95.00
250.00 50.00 3
CDs 75.00
Clothes 100.00
Gasoline 40.00
200.00 40.00 5
CDs 38.40
Clothes 67.00
Gasoline 45.00
Gifts 84.50
Car 3000.00
```

Sample Output

```
After saving for 2 months, your balance will be $66.20.
Your balance will be $35.00.
You cannot afford these purchases in the next year.
```

Prime Hamming Code

Problem 4



One day while you were experimenting with your Ham radio when you came across a strange message. It was comprised completely of numbers. After you studied some of the code you noticed a pattern and eventually discovered a solution. Every lower case letter is represented by every other prime number starting with 2 and every other upper case letter is represented by every other prime number starting with 3. Also, any number that is not a prime represents a space. For example 2 = "a", 3 = "A", 5 = "b", 7 = "B", 11 = "c", 13 = "C", and 4,6,8 = " ".

Create a program that will decode messages sent in this prime encoding.

Details of the Input

The first line of input will be a single integer, *n*, that will tell you the number of encoded messages you are to decode. Each of the next *n* line will contain a single message. The lines will be comprised of a list of positive integers separated from each other by a single space. The last integer on the line will be followed by an end-of-line character.

Details of the Output

Output should be the decoded messages, one per line.

Sample Input

61 4 89 109 191 23 133 101 2 167 47
3 101 19 88 59 157 49 5 23 167 167 23 149 39 167 47 2 103 100 61 103 167 23 83

Sample Output

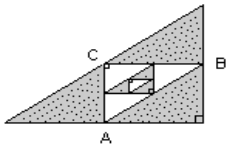
I Love Math
AMD is better than Intel

A Crisis of Geometric Proportions

Problem 5

Being a huge fan of the great mathematicians of the past, you are excited when you happen upon the ghost of Pythagoras on your way to school one day. Although Pythagoras is generally pretty adept at all things triangular, he was having a little difficulty with one particular problem that had been assigned as part of his trigonometry homework. He knows very well that when given two sides of a right triangle, the first leg squared plus the second leg squared equals the hypotenuse squared ($A^2+B^2=C^2$) and that the area of a triangle is the length of its base times the length of its height times 0.5 (the formula we normally see as $\frac{1}{2}bh$). He also knows that if each pair of midpoints of the sides of the triangle is connected, the resulting inscribed triangle is similar to the outer one. However, he is having trouble finding the area formed when the area of a right triangle is reduced by the area of the similar triangle inscribed inside of it, then increased by another similar triangle inscribed inside of the inner one, then reduced by a similar triangle inscribed inside of that one, and so on.

Gray = Included Area
White = Excluded Area



In order to help the poor ghost of Pythagoras, you decide to create a program that will compute several areas, each defined as above. Hurry, Pythagoras' trigonometry grade is resting on your shoulders!

Details of the Input

The first line of input to the program is a single integer, *n*, that tells how many areas will be computed. Each of the following *n* lines defines a single area in terms of seven integers separated by single spaces. The first six are the coordinates of the vertices of the outermost triangle (x_1 y_1 x_2 y_2 x_3 y_3) and the seventh is the total non-negative number of triangles to be inscribed which will not exceed 50. The program must first determine if the triangle defined by the three coordinates is a right triangle and, if so, calculate and output the area defined.

Details of the Output

If the triangle is not a right triangle, print the message seen below. If the triangle is right, then output "The defined area is *k* square meters." The value of *k* should be accurate and shown to 3 decimal places no matter what. Truncate (chop off) any further decimal places.

Sample Input

```
2
0 30 0 0 40 0 3
4 -3 4 5 -2 -3 4
```

Sample Output

```
The defined area is 478.125 square meters.
The given triangle is not a right triangle.
```

Loop Orientation

Problem 6

Your goal in this problem is to figure out the orientation of a loop in the xy plane. You will be given an *m* by *m* matrix of O's which contains in it a closed loop of X's. The bottommost row of the matrix represents y = 1, the next row up is y = 2, and so on. The leftmost column represents x = 1, the next column over is x = 2, and so on. You will then be given three points (*x*, *y*) and be asked to figure out whether the loop traveled in that order is of positive or negative orientation. Positive orientation means, in some respects, counterclockwise rotation. If you're traveling along a loop with positive orientation, then the INSIDE of the loop should always be to your left. Negative orientation means clockwise rotation. While traveling along the negatively oriented loop the INSIDE of the loop will always be to your right.

Details of the Input

The input will consist of multiple instances of the problem. For each instance, the first line will be an integer $m \geq 3$. A value of 0 for *m* signifies the end of the input. The following *m* lines each contain a string of length *m* of the characters 'X' and 'O' in such a way that the resulting *m* x *m* matrix forms a loop of X's. The following 3 lines contain, in order, 3 points visited during a trip around the loop in the form (*x*, *y*).

Details of the Output

For each instance of the problem, output "Negative" if the orientation of the loop was negative or "Positive" if the orientation of the loop is positive.

Sample Input

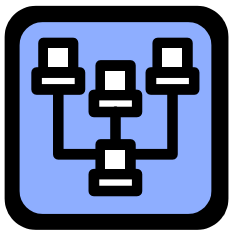
```
3
XXX
XOX
XXX
(1, 1)
(1, 3)
(3, 2)
5
OOXXX
XXXOX
XOOOX
XXOXX
OXXXO
(2, 1)
(1, 3)
(1, 2)
0
```

Sample Output

```
Negative
Positive
```

Disconnect Networks!!

Problem 7



Nate the network guy is responsible for ensuring that his company's intranet stays up and running. If his fellow employees experience problems sending Instant Messages to each other to make lunch plans, he's in big trouble. Suppose the company's network consists of n computers. Each computer has connections to one or more of the other computers in the intranet. There is only one way that a message between any two computers can be delivered within the intranet because there is no set of computers that are connected in a complete cycle.

Nate would like to determine which computers would have to be functioning to guarantee that a given communication between two people would be successful. It is your job to write a program that, given the number of computers and how each computer is connected to others, can report the desired information.

Details of the Input

The first line of input will be a single number, *n*, that is the number of computers in the network. The next *n* lines will each list the "name" of a computer (1 through *n*) followed by the number *m* of computers to which it is connected, and then followed by the *m* "names" of all the computers to which it is connected. Not all computers will be connected to the same number of other computers. The next line of input will contain an integer, *p*, which tells how many communications Nate would like to analyze. This is followed by *p* lines, each with two integers which represent the sending and receiving computers of the communication.

Details of the Output

Output of the program should be a list of the computers that must be used to pass the message between the communicating hosts. The order of the list should be from the computer that first receives the message from the sender to the computer that last passes the message to the ultimate receiver. Format your message as seen below. You are guaranteed to have at least one intermediate computer. Be sure, as always, to use proper grammar.

Sample input

```
6
1 3 2 5 6
2 3 1 3 4
3 1 2
4 1 2
```

5 1 1
6 1 1
1
4 6

Sample output
Computers 2, 1 must be operational for communication to succeed.