ACM Programming Contest
Dixie State University
February 23, 2013

Do NOT turn this page until

10:00 am Mountain Standard Time!

Rules

1. Each team consists of up to three students.
2. Each team may use one computer.
3. Teams may use any printed references.
4. Teams may not use any electronic aids, including the internet. The help system built in to your programming environment is okay.
5. High school teams and college teams compete and are judged separately (unless otherwise requested).
6. All solution must be written in C++ or Java. Python is also permitted for high school and middle school teams, and for college teams where no member has completed CS 3005 or any other course that requires C++.
7. The team with the most correct solutions wins.
8. Ties will be broken using a time score:
   (a) Each time a team submits a correct solution, the number of minutes that have elapsed since the beginning of the competition is added to the time score.
   (b) For each incorrect solution submitted, a 20 minute penalty is added to the time score, but only if the team eventually submits a correct solution to that problem.
   (c) Multiple penalties will be added for multiple incorrect solutions to the same problem.
9. The input for each problem comes from standard in. This goes by the names `cin` and `STDIN` in C++, `System.in` in Java, and `sys.stdin` in Python. Other methods may also exist for receiving input from standard in for each language.
10. The output for each problem should be sent to standard out. This goes by the names `cout` and `STDOUT` in C++, `System.out` in Java, and `sys.stdout` in Python. Other methods may also exist for sending output to standard out for each language.
11. The output of submitted solutions must exactly match the output of the reference solution, down the last character. Whitespace differences matter. Any other output, including debugging output, may cause an otherwise correct solution to be marked as incorrect. Each problem statement with example input and output shows exactly where newline characters are placed and where spaces are appropriate.
12. Solutions have a 10 second time limit. Any solution that runs longer than that will be considered incorrect.
13. If you need to handle last-minute registration details, please arrive at 9:00 am. Otherwise, please arrive at 9:30 am to find a computer, complete the sample problem, and sign in to the judging system.
14. The contest begins at 10:00 am and ends at 3:00 pm.

Contents

1 Packing Boxes

You just got a job with Amazon! Unfortunately, on your first day of work, you discover that you were hired to pack boxes in one of their warehouses instead of programming their cloud computing engine like you had hoped.

You decide that any job is better than no job, so you set to work. You take your place next to a conveyor belt, where items that customers have purchased travel down the line to you. Your job is to pack each item in a box, then hand it off to someone else to label it and send it to UPS for delivery.

The items that you pack come in all sizes, and you are required to pack each one in the smallest box that it will fit in. You are given a list of box sizes (length, width, and height in inches), and a list of items to pack (again, you are given the length, width, and height for each).

For each item, you must find the smallest box that is big enough to fit the item. You can rotate the item to make it fit, but only at right angles, i.e., you should not stuff an item diagonally into a box.

The list of box sizes is not given in any particular order. You must order them from smallest to largest yourself. You should size them according to their total surface area.

Note: The ¶ symbol in the examples below represents a newline character.

General Format

```
numberOfBoxSizes¶
length width height¶
length width height¶
etc...
numberOfItems¶
length width height¶
length width height¶
etc...
```

The input is given in the format outlined above. The output should be one line per item. Each line should give the size of the item followed by the size of the box you will pack it in. Each size should be given exactly as it was provided in the input.

Sample Input

```
3¶
5 7 2.5¶
9 12 3¶
11.5 17.5 4¶
5¶
6 6 2.3¶
7 1 4.3¶
3.2 10.3 8.6¶
1 1.1 1¶
2.5 7 5¶
```

Sample Output

```
6 6 2.3 9 12 3¶
7 1 4.3 5 7 2.5¶
3.2 10.3 8.6 11.5 17.5 4¶
1 1.1 1 5 7 2.5¶
2.5 7 5 5 7 2.5¶
```

2 Nicknames

You are the football coach at a local high school, and you are looking over the roster of players for the upcoming season. You decide to assign each player a short nickname so that you can bark orders at them as quickly as possible.

To keep things orderly, you decide to base their nicknames on their real names as much as possible. Being a coach, you decide to throw out their first names and just focus on last names.

You look down the roster, and decide that a player's nickname will be as many letters as necessary from his/her last name to make the nickname unique. For example, if you have these four last names:

- Davidson
- Davis
- Dixon
- Smith

Their nicknames would be:

- David
- Davis
- Di
- S

In each case, you took as many letters (in order, starting from the front) of each name to make the nickname unique. For "Smith", this could be a single letter because there were no other names starting with "S". "Dixon" just needed two letters to distinguish it from "Davidson" and "Davis", but those two needed five letters each to avoid ambiguity.

You are pretty proud of your system, until you notice that it fails in a few cases. For example, you may have two players with the same last name, or you may have a player whose last name is a prefix of another player's last name, e.g., "David" and "Davidson". To solve this problem, you decide to kick those people off the team. They sound like troublemakers. As a result, you may assume that each name has a longest unique prefix.

Your input will consist of a list of last names, one per line, given in sorted order. Your output should consist of one line per name, with the original name and the nickname separated by a space.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
Adams¶
Andersen¶
Anderson¶
Carson¶
Carter¶
Carville¶
Cooper¶
Coply¶
Smith¶
Smythe¶
Sorensen¶
Sorenson¶
Wynn¶
```

Sample Output

```
Adams Ad¶
Andersen Anderse¶
Anderson Anderso¶
Carson Cars¶
Carter Cart¶
Carville Carv¶
Cooper Coo¶
Coply Cop¶
Smith Smi¶
Smythe Smy¶
Sorensen Sorense¶
Sorenson Sorenso¶
Wynn W¶
```

3 Shopping Trip

You just received lots of money from a long lost relative who suddenly passed away and apparently left everything she had to you! In honor of the deceased, you decide to go on a grand shopping trip and spend that money on items she would no doubt agree you look great in. But you have to be a little careful, because you cannot put more items in your shopping cart than you can afford to buy—that would be embarrassing. But you also want to spend as much as you possibly can. (Mom said to go shopping just once, and save the rest for college.)

When you get to Big Designer Clothes Inc., you are thrilled beyond description when you discover that *everything* is on sale. Some items are 10% off, some 20%, and so forth. Some are even 90% off! That means you can buy more, but it also makes the problem of calculating the total cost that much more difficult.

Luckily, you never go anywhere without your laptop, so you whip it out and quickly write a program that will calculate the Grand Total for you. Since you may be running several scenarios before making your final purchase, you create your code so that it reads data in a particular format from a file. The first line contains the sales tax. The rest of the lines contain the regular price of an item, followed by the percent discount.

Note: The ¶ symbol in the examples below represents a newline character.

General Format

```
salesTaxPercent¶
regularPrice    percentOff¶
regularPrice    percentOff¶
regularPrice    percentOff¶
regularPrice    percentOff¶
regularPrice    percentOff¶
regularPrice    percentOff¶
etc...
```

Sample Input

```
8.25¶
20.00   10¶
40.00   20¶
30.00   10¶
```

Sample Output

```
83¶
```

For the above example, your code would determine that 10 percent off of $20 leaves $18, 20 percent off $40 leaves $32, and 10 percent off $30 leaves 27. The sum of the items is $77.00. Adding in 8.25% sales tax makes the grand total 83.3525. To keep your answer simple, always round *down* to the nearest dollar. So for the case above, your program should print just 83.

Note that in Python, you can round down with:

```
x2 = int(x)
```

In C++, you can round down with:

```
int x2 = (int)x;
```

4 Evil Overlord Cypher

You have been imprisoned by an evil, but stupid alien overlord. You told them it was a trap. It's not your fault. IT'S NOT YOUR FAULT.

You can pass notes containing information to other prisoners to coordinate your escape. You want to use an algorithm that can be easily deciphered by your fellow prisoners, in between torture sessions of forced *Buffy the Vampire Slayer* marathons. But you also want the notes to remain unreadable to evil alien overlord and his minions should the notes be discovered.

Therefore, you choose to implement a simple Caesar cypher given the following rule:

When the characters in the document are sorted by frequency, then by ASCII code (case sensitive), each character is replaced by the character in the same position in the reversely sorted set.

Given an arbitrary body of text as input, produce the appropriate output based on the cypher.

The first line of the input will contain a count of all the remaining lines, the remaining lines are all part of the text to be encrypted.

Note: The ¶ symbol in the examples below represents a newline character.

Also note that the newlines in the input text are treated as any other character and are encoded with the rest of the text. Note in particular that the last line of output may not end with a newline character.

Sample Input 1

```
1¶
Aliens are dumb¶
```

Sample Output 1

```
 mn¶
ibAud¶
Aralse
```

Sample Input 2

```
4¶
Lorem ipsum dolor sit amet, consectetur adipiscing elit.¶
Pellentesque neque enim, luctus ac venenatis et, euismod¶
eu mauris. Pellentesque vehicula laoreet nisl, a placerat¶
metus adipiscing eu.¶
```

Sample Output 2

```
eroLphPm.gphcrqroh.Pvh¶
pLvnhdr,.LdvLvgoh¶
cPmP.dP,uhLqPvsaiLqqL,vL.lgLh,LlgLhL,Ppnhqgdvg.h¶
edhtL,L,¶
vP.hLvnhLgP.prcaLghp¶
goP.shiLqqL,vL.lgLhtL Pdgq¶
hq¶
roLLvh,P.qnh¶
hmq¶
dLo¶
vapLvg.h¶
cPmP.dP,uhLgsa
```

5 Point Defense

During a battle, a computer virus attack from enemy spacecraft disabled your anti-missile point defense programs and destroyed the backups. You can feel smug that your last readiness report included a note that there should be offline backups of all critical programs on board. However, before you can feel truly self-superior, you have to survive the afternoon, and it is becoming increasingly less likely as the enemy ships seem to have other ideas. Under fire, you must re-implement the important features of the point defense programs.

The program must determine if each missile will hit the ship within a tolerance of 0.000000001 units. Then in what order the enemy missiles will hit your ship. Given this list, it must produce firing solutions for your point defense guns in the order that the missiles will impact.

The input will consist of a set of missile position points and velocity vectors, relative to your own position and velocity vector.

The format of each line of input will be:

$$(x, y, z)\ [v_x, v_y, v_z]$$

The output will consist of a unit vector (a vector with magnitude 1) of the direction to fire the point defense gun for each missile in the order the enemy missiles will hit the ship (one per line):

$$[v_x, v_y, v_z]$$

You may assume that no two missiles will hit at exactly the same instant. All output should be rounded and displayed with three digits of precision after the decimal point.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input 1

```
(0.0,0.0,10.0) [0.0,0.0,-1.0]¶
(0.0,0.0,-9.0) [0.0,0.0,1.0]¶
(0.0,5.0,0.0) [0.0,1.0,0.0]¶
```

Sample Output 1

```
[0.000,0.000,-1.000]¶
[0.000,0.000,1.000]¶
```

Sample Input 2

```
(10.0,10.0,10.0) [-5.0,-5.0,-5.0]¶
(0.0,0.0,10.0) [0.0,0.0,-10.0]¶
```

Sample Output 2

```
[0.000,0.000,1.000]¶
[0.577,0.577,0.577]¶
```

6 Weight Watchers

Weight Watchers is a national organization that has helped countless people lose weight. Aside from weekly meetings and weigh-ins, which help provide motivation, Weight Watchers uses their own system for counting food intake. Instead of simply using Calories, they use Points, where every 50 calories equals 1 Point. However, every gram of fat counts as 4 additional calories, and every gram of fiber subtracts 7 calories.

Suppose you ate something with a label of 220 calories, 6 grams of fat, and 2 grams of fiber. The adjusted calories would be $220 + 6 \times 4 - 2 \times 7$, or 230. From there, you would round up or down to the nearest 50. Everything from 176 to 225 rounds to 200, or 4 points. Everything from 226 to 275 rounds to 250, or 5 points. There is one exception to that general rounding formula—everything from 1 to 75 rounds to 50, or 1 point. That is, you can't eat something for 1 to 25 calories and have it count for 0 points. It counts for 1. Of course, zero calories counts for zero points.

Weight Watchers brags that using Points is somehow easier than counting calories. All Weight Watchers products list on the label how many points they are, which is easy enough, and Weight Watchers publishes big books for looking up the number of points for many products that are not their brand. But for those times when the Point information just isn't available, Weight Watchers sells a calculator that does the above calculations.

You, being a computer wiz, refuse to buy their silly little calculator, and decide to write your own program to calculate Points.

Note: The ¶ symbol in the examples below represents a newline character.

General Format

```
Calories GramsOfFat GramsOfFiber¶
Calories GramsOfFat GramsOfFiber¶
Calories GramsOfFat GramsOfFiber¶
Calories GramsOfFat GramsOfFiber¶
etc...
```

Sample Input

```
220 6 2¶
220 6 3¶
```

Sample Output

```
5¶
4¶
```

7 Luhn Algorithm

The input consists of a bunch of lines of sample credit card numbers, each followed by a newline. Your task is to write a program that validates each credit card number according to the Luhn algorithm, which will be explained below. You should output the same number followed by a comma, and the string "TRUE" if the number is valid or "FALSE" if the number is not valid.

The formula for validating a credit card, the Luhn algorithm, can be implemented as follows:

1. Sum the digits in odd-numbered positions (counting from the rightmost side of the number)

2. Double each digit in an even-numbered position (again, counting from the rightmost side of the number), sum the digits of the resulting values (note: not the values themselves)

3. Add the sums from steps one and two

4. If that total is evenly divisible by 10 (no remainder) the card number is considered valid

Note that valid cards must also have a valid prefix taken from this list:

```
51, 52, 53, 54, 55, 4
```

We are also assuming that all cards numbers in the input file are a length of 16.

Example

Given an example (valid) card number of 4652360088404638.

1. Odd digits: $8 + 6 + 0 + 8 + 0 + 6 + 2 + 6 = 36$

2.

Even digits	3	4	4	8	0	3	5	4
Doubled	6	8	8	16	0	6	10	8
Sum of digits	$6 + 8 + 8 + 1 + 6 + 0 + 6 + 1 + 0 + 8 = 44$							

3. Total of sums from steps 1 and 2: $36 + 44 = 80$

4. $80 \% 10 = 0$

The remainder at the end was zero, so the number is considered valid.

Example

Given an example (invalid) card number of 5370182444652350.

1. Odd digits: $0 + 3 + 5 + 4 + 4 + 8 + 0 + 3 = 25$
2.

Even digits	5	2	6	4	2	1	7	5
Doubled	10	4	12	8	4	2	14	10

   Sum of digits $1 + 0 + 4 + 1 + 2 + 8 + 4 + 2 + 1 + 4 + 1 + 0 = 28$
3. Total of sums from steps 1 and 2: $25 + 28 = 53$
4. $53 \% 10 = 3$

The remainder at the end was 3, so the number is not considered valid.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
4652360088404638¶
5370182444652350¶
```

Sample Output

```
4652360088404638,TRUE¶
5370182444652350,FALSE¶
```

8 Bounding Primes

In this problem you will read in a list of numbers, and for each number you will output three things:

1. The largest prime number less than the input number
2. The input number
3. The smallest prime number greater than the input number.

The numbers in the input file will range from 4 to 32000.

A prime number is defined to be a number that is evenly divisible only by itself and 1.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
4¶
32000¶
123¶
756¶
10000¶
20000¶
15000¶
1500¶
150¶
15¶
17¶
```

Sample Output

```
3 4 5¶
31991 32000 32003¶
113 123 127¶
751 756 757¶
9973 10000 10007¶
19997 20000 20011¶
14983 15000 15013¶
1499 1500 1511¶
149 150 151¶
13 15 17¶
13 17 19¶
```

9 Vowels

For each line of text in the input, print the count of the number of vowels and the total number of characters (not including the newline at the end) separated by a forward slash.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
The rain in Spain falls mainly in the plain.¶
AEIOU¶
Asterisks¶
```

Sample Output

```
13/44¶
5/5¶
3/9¶
```

10 Binary Coded Decimal

In this problem you will need to tell what time it is using a binary coded decimal (BCD) clock.

The clock consists of 13 LED lights. Lights 1 and 2 tell you the first digit of the hour. Lights 3–6 tell you the second digit of the hour. Lights 7–9 tell you the first digit of the minute. Lights 10–13 tell you the second digit for the minute. The clock is in 12 hour format. A time of 13:12 is an error.

For example, if lights 1, 3, 7, 10 and 12 are on the time would be 11:15. Light 1 on means the first digit of the hour is 1. Light 3 on means the second digit of the hour is 1. Light 7 means the first digit of the minute is 1. Lights 10 and 12 means that the second minute digit is $4 + 1$ or 5.

Read the input and then output the BCD time and the decimal time as illustrated below. A “1” means the light is on. A “0” means the light is off. The lights are encoded from left to right. The first digit is light 1, the fifth digit is light 5. If there is an invalid time (e.g. 11:76) then output “error”.

Please note how the output is formatted and spaces; take particular note of how inputs that are too long or too short are handled.

Illustration for the BCD clock

	6@		13@	-8
	5@	9@	12@	-4
2@	4@	8@	11@	-2
1@	3@	7@	10@	-1
Hours		Minutes		

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
1000000000000¶
1000000000010¶
0100001000100¶
0010000010010¶
1100011000011¶
100000000000¶
10101010101010101¶
1111111111111¶
```

Sample Output

```
1000000000000  -> 10:00¶
1000000000010  -> 10:04¶
0100001000100  -> error¶
0010000010010  -> 01:44¶
1100011000011  -> error¶
100000000000   -> error¶
10101010101010 -> error¶
1111111111111  -> error¶
```

11 Detection

You are part of a secret military organization dedicated to protecting Earth from invasion by evil aliens. A means of detecting the aliens has been devised that utilizes a network of detectors. Due to budget limitations, only some of the detectors can be powered at any given time. Luckily, the detectors are built in such a way that each detector can cover the area of adjacent detectors in the network. Therefore, when one node is powered, it covers all of its connected nodes.

Given the shape of the network, determine the minimum cost to power and cover the entire network

Each line of input will contain an integer ID for a node followed by the integer cost to power the node followed the IDs of all the nodes connected to it. There will be at most 16 nodes.

The output consists of the count of nodes required to power the system followed by the minimum cost. If there is a tie in cost, the solution with the fewest nodes should be selected.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
0 2 1 2 3 4 5 6¶
1 10 0 2¶
2 13 0 1¶
3 11 0 4¶
4 10 0 3 5¶
5 7 0 4¶
6 3 0 7¶
7 10 6¶
```

Sample Output

```
2 5¶
```

12 Traveling Salesperson

In this problem to will need to maximize the total sales for a salesman. The salesman sells computers so naturally the first city is city 0. There are five cities labeled 0, 1, 2, 3 and 4. The salesman starts at city 0 and visits each city exactly once. The sales volume that the salesman generates is in the matrix. For example a salesman going from city 0 to city 1 makes $999.

If a salesman is in city 1, the salesman has a choice for going to city 0 for a profit of $1 or city 2 for $9999 of city 3 for $999 or city 4 for $999. The salesman can not go to city 1 again.

Remember the salesman must start at city 0 and visit each and every city exactly 1 time.

In the output you should show the best route and the total profit for that route.

Example

				To		
		0	1	2	3	4
From	0	0	999	3	6	5
	1	1	4	9999	999	9
	2	999	0	0	9	7
	3	3	0	9	999	8
	4	4	5	999	0	5

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
0 999 3 6 5¶
1 4 9999 999 9¶
999 0 0 9 7¶
3 0 9 999 8¶
4 5 999 0 5¶
```

Sample Output

```
0 1 2 3 4 = 11019¶
```

13 River Floods

The flow of water in a river can be measured in cubic meters per second. A small river like the St. George's Virgin River might average 5 cubic meters per second. A large river, like South America's Amazon might average 200,000 cubic meters per second.

The amount of water that is expected to pass a location in the river can be found by multiplying the average flow rate by the amount of time of the measurement. For example, if the average flow rate of a river is 100 cubic meters per second, and the observer measures the amount of water for 60 seconds, then the expected amount would be 6,000 cubic meters.

We will define low water flow when the amount of water observed is less than or equal to half of the expected amount. We define high water flow when the amount of water observed is greater than or equal to twice the expected amount.

Each case in the input will be a single line with 3 integers: the observed amount of water (in cubic meters), the length of time (in seconds) that the water was measured, and the normal water flow (in cubic meters per second).

Your program must output "low", "normal", or "high" for each case.

The input ends with a line containing three "0"s. Your program must not produce output for this line.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
3031 60 101¶
3000 60 100¶
12000 60 100¶
11999 60 100¶
0 0 0¶
```

Sample Output

```
normal¶
low¶
high¶
normal¶
```

14 Book Worm

A fan has meticulously collected all of the volumes of a book series and stored them in order on a shelf, with the first volume on the left, and the last volume on the right. A finicky book worm starts at the first page of the first volume, then proceeds to eat a straight line to the last page of the last volume. It has to eat through any covers that are in the way. Your program must tell how far the worm had to eat.

The input file consists of two lines per problem. A file may have up to 1000 problems. The first line of a problem contains two numbers, C and N. $1 \leq C \leq 5$ is the thickness of all of the covers in this collector's set. $2 \leq N \leq 26$ is the number of volumes in the set. The second line is a space separated list of N numbers, the thicknesses of the volumes, in order. All numbers are integers. The thicknesses are in millimeters. The input file ends with a line that has $C = 0$ and $N = 0$.

Note: The ¶ symbol in the examples below represents a newline character.

Sample Input

```
4 21¶
39 30 34 10 50 17 24 30 38 50 19 19 21 34 39 50 30 16 33 40 42¶
3 13¶
14 31 17 45 42 12 50 30 10 12 40 45 28¶
2 14¶
16 31 29 12 46 19 27 14 30 18 19 23 28 34¶
4 6¶
30 49 30 21 27 45¶
2 20¶
10 36 40 32 12 26 28 29 35 42 40 30 11 21 12 38 16 21 23 21¶
0 0¶
```

Sample Output

```
744¶
406¶
348¶
167¶
568¶
```