A+ Computer Science

**Computer Science Competition**

Hands-On Programming Set

**I. General Notes**

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

**II. Point Values and Names of Problems**

|  |  |
| --- | --- |
| **Number** | **Name** |
| Problem 1 | Bookface |
| Problem 2 | Intensity |
| Problem 3 | Weights |
| Problem 4 | Sign |
| Problem 5 | Rack |
| Problem 6 | Volume |
| Problem 7 | Competition |
| Problem 8 | Planning |
| Problem 9 | Plate |
| Problem 10 | Security |
| Problem 11 | Scramble |
| Problem 12 | Incline |

**For more Computer Science practice tests and materials, go to www.apluscompsci.com**

**1. Bookface**

# Program Name: Bookface.java Input File: bookface.dat

You are now a software engineer at the new startup Bookface. Your job is to write a program to determine how many connections away from someone a use is. Given a list of all connections on Bookface, and the name of a user, output a list of everyone that is 1-3 connections away from them, sorted alphabetically by last name then first name.

**Input**

The first line will contain 2 integers, n and m, denoting the number of connections on Bookface, and the number of users for whom you need to find their connections. Each of the following n lines will contain 4 strings each: fn1, ln1, fn2, ln2, denoting the first name of the first user, the last name of the first user, the first name of the second user, and the last name of the second user, respectively. These pairs signify users that are connected, or 0 connections away from each other. The following m lines will each contain a first name last name pair, the name of the user whose connections we are looking for.

**Output**

For each user, output the name first on its own line, followed by a line containing the string "Users that are one connection away: ", followed by the list of all users separated by one connection from the given user, sorted by last name, in the format fn ln. The next line should contain the string "Users that are two connections away: ", followed by the list of all users separated by two connections from the given user, sorted by last name, in the format fn ln. The next line should contain the string "Users that are three connections away: ", followed by the list of all users separated by three connections from the given user, sorted by last name, in the format fn ln. If there are 0 users that are 1, 2, or 3 connections away from the given user, instead of the list of strings, output "No users are [one,two,three] connections away.", using the string version of the number of connections for which there are no users. So if there are no users 3 connections away from the given user, we use the string three in place of the [one,two,three].

**Example Input File**

8 2

Sam Armstrong Ben Davis

Ben Davis Wesley Chart

Sam Armstrong Danny Vunt

Ben Davis Ariel Hatley

Ariel Hatley David Smith

David Smith Johhny Blaze

Johnny Blaze Danny Vunt

Johhny Blaze Eric Chan

Ben Davis

Ariel Hatley

**Example Output to Screen**

Ben Davis

Users that are one connection away: David Smith, Danny Vunt

Users that are two connections away: Johhny Blaze, Johnny Blaze

Users that are three connections away: Eric Chan

Ariel Hatley

Users that are one connection away: Sam Armstrong, Johhny Blaze, Wesley Chart

Users that are two connections away: Eric Chan, Danny Vunt

Users that are three connections away: Johnny Blaze

**2. Intensity**

# Program Name: intensity.java Input File: intensity.dat

Anthony is trying to figure out if his current program he has set up for lifting is too much work or not. He needs to figure this out in order to make sure he has the most optimal program for him. He doesn’t want to get too exhausted and be unable to train at 100% intensity. So, he set a limit for himself. He will only do 5-6 lifts at the gym in total: less than five and he would be leaving fuel in the tank, and more than 6 and he is just adding “junk volume”. Ensure that on each day, Anthony is training to his utmost potential.

**Input**

The first line will contain a single integer d, the number of days that will be reviewed. For each day, there will be a list of exercises of unknown size, with each exercise separated by line. A line marked "END" signifies that a day has ended. Do not include this line as an exercise.

**Output**

For each day, output "just right" if the amount of exercises falls between 5 or 6, "needs more effort" if the amount of exercises is less than five, or "junk volume" if the amount of exercises is greater than 6.

**Example Input File**

3

Bench-Press

Pull-Ups

Dips

Weighted-Crunches

Seal-Rows

END

Incline-Bench

Incline-Curls

END

Larson-Press

Cable-Curls

Military-Press

Lat-Raises

Skiers

Leg-Raises

Skull-Crushers

END

**Example Output to Screen**

just right

needs more effort

junk volume

**3. Weights**

# Program Name: weights.java Input File: weights.dat

Anthony is a vet when it comes to lifting weights. The most glaring evidence of this (other than his succulent physique) is how he counts the weights that he lifts on a single exercise. For instance, if he is bench pressing 205 lbs, he thinks in his mind that he is lifting one 45, one 25, and one 10, even though in reality he is lifting the bar(which is 45 pounds), two 45 pound plates, two 25 pound plates, and two 10 pound weights, since there are weights on either side of the bar. The weight increments in his gym are 45, 25, 10, 5, and 2.5. Given a specific weight that Anthony lifts, calculate how he would think of the weight in his head.

**Input**

The first line will contain a single integer n that indicates the number of lines that follow. Each line will contain one integer d, which is the weight that Anthony is lifting.

**Output**

Output how Anthony thinks of the weight in his head, in the format shown below.

**Example Input File**

3

135

195

260

**Example Output to Screen**

1 45

1 45 1 25 1 5

2 45 1 10 1 5 1 2.5

**4. Sign**

# Program Name: sign.java Input File: sign.dat

Anthony has been making a lot of money at his current job over the last couple of years, and he has saved up enough to open a gym! He is trying to put up a sign through the glass of his gym, but in order for people on the outside to be able to read the sign, it must be printed in reverse. Help Anthony out by reversing his message for him.

**Input**

The first line will contain a single integer n that indicates the number of lines that follow. Each line will contain a message that Anthony wants to put up on the sign.

**Output**

Output the message in the way that Anthony desires.

**Example Input File**

3

WELCOME TO MY GYM!!

SIGN IN HERE

DO NOT DROP THE WEIGHTS

**Example Output to Screen**

!!MYG YM OT EMOCLEW

EREH NI NGIS

STHGIEW ETH PORD TON OD

**5. Rack**

# Program Name: rack.java Input File: rack.dat

Anthony is trying to do his squats for the day, so he is trying to find a squat rack that is not already in use. A squat rack is not in use when there is no one directly using it or standing around it, and there are no items left behind nearby it by someone who went to go use the restroom or something that is not currently there. Given the floor plan of the gym he is at right now, count how many open squat racks there are for Anthony to use.

**Input**

The first line will contain a single integer n that indicates the number of data sets that follow. Each data set will consist of a single line containing two integers, r and c, which represent the dimensions of the gym floor. The gym floor will always be rectangular in shape. There will then be r lines of data, each containing c characters. "." represent empty floor space that is not taken up by anything. "S" represents a squat rack that could potentially be used. If there is anything around the S that is not an empty space, Anthony will assume the squat rack is being used by someone and will not consider it an open rack, and there will never be two squat racks that are directly adjacent to each other.

**Output**

Output the number of squat racks that are open for use for Anthony.

**Example Input File**

3

3 3

...

.S.

...

6 10

....p.....

.S......S.

..l.......

...S...S..

..........

qro.....S.

5 5

.....

.S.S.

..o..

.S.S.

.....

**Example Output to Screen**

1

3

0

**6. Volume**

# Program Name: volume.java Input File: volume.dat

In bodybuilding and general periodization in strength and hypertrophy training, there is a very important concept known as volume. Similar to standard volume, lifting volume consists of three “dimensions”. In order to determine the total volume that an exercise will have, one will multiply the total amount of sets with the total amount of reps and the weight that is being used. Tracking volume over time is very important because it shows that one has grown and improved. Help track the volume of specific movements.

**Input**

The first line will contain a single integer n that indicates the number of data sets that follow. Each data set will contain three integers, s, r and w, which denote the amount of sets, reps, and weight of an exercise, respectively.

**Output**

Output the volume of the exercise.

**Example Input File**

4

2 12 110

4 8 205

10 10 315

2 3 600

**Example Output to Screen**

2640

6560

31500

3600

**7. Competition**

# Program Name: competition.java Input File: competition.dat

Anthony is now an experienced gym owner, and he is starting to branch out in his lifting interests. He has started to run local powerlifting competitions to promote the local talent that he sees in the area. To make the competitions more exciting, he takes the competitors, and in the order of their lifting total (combined squat, bench, and deadlift max), he wants to sort them such that the competitors are coming on stage in ascending order. Given an out of order list of competitor’s lifting totals, calculate the smallest amount of shifts that it would take to get the competitors in ascending order. A shift occurs when numbers are shifted over and a value is inserted in the space created. A shift might involve moving multiple values. A shift is not the same thing as a swap.

**Input**

The first line will contain a single integer n that indicates the number of data sets that follow. Each data set will contain an unknown amount of integers that represent the lifting totals of competitors.

**Output**

Output the least amount of shifts between competitors to get the entire group into ascending order.

**Example Input File**

3

1000 995 997 1280 1320

2000 1000

1495 1260 1140

**Example Output to Screen**

1

1

2

**Explanation:**

For the first test case above, 995-1320 are already sorted. By placing 1000 between 997 and 1280, it would only take one “shift” to get the entire list in order.

**8. Planning**

# Program Name: planning.java Input File: planning.dat

To ensure that Anthony sets up a balanced schedule for his competitions, he wants to ensure that he has his calendar set up correctly. He has a list of competition dates that he wants to use, but he wants to make sure everything is in the right order. Given a list of potential future competition dates, create a sorted list of competition dates that Anthony will actually end up using. If a potential competition date is on a weekday, Anthony will not include it. Other than that, Anthony wants the dates to be in order from soonest to latest.

**Input**

The first line will contain a single integer n that indicates the number of data sets that follow. Each data set will contain a single integer d that represents the number of potential lifting dates. The next d lines will contain a single date p given in the format "MM/DD/YYYY", which represents a potential competition lifting date.

**Output**

Output a potential list of competition dates from soonest to latest from when Anthony is making the list (assume Anthony is making the list on a day earlier than all of the dates given). Be sure to include an empty line between test cases.

**Example Input File**

3

4

06/26/2023

06/25/2023

06/24/2023

07/01/2023

7

07/15/2023

07/10/2023

07/11/2023

07/12/2023

07/13/2023

07/14/2023

07/09/2023

1

06/06/2006

**Example Output to Screen**

06/24/2023

06/25/2023

07/01/2023

07/09/2023

07/15/2023

**9. Plate**

# Program Name: plate.java Input File: none

With the introduction of powerlifting into Anthony’s gym, he decided to purchase a large collection of Eleiko powerlifting plates. Draw out one of them for him.

**Input**

none

**Output**

Output the Eleiko plate as shown below.

**Example Input File**

none

**Example Output to Screen**

####$$####

##OOOOOO##

#OoOOO$oO#

$ooooOooo#

Oooo$$oOO$

OoooO$oOO$

$oooooooo#

#oooOOOo$#

##Ooooo$##

###$O$####

**10. Security**

# Program Name: security.java Input File: security.dat

Since Anthony’s gym has gotten more successful, he has begun to fear the risk that some nefarious individual might try to hack into his database containing customer information. He wants to create a password that is secure enough for his business needs. For a password to be secure enough, it must contain all of these criteria:

The password must only contain alphanumeric characters, underscores, dashes, or question marks.

The password must contain at least one uppercase letter, one number, and one lowercase letter.

The password must be longer than eight characters.

If the password does not meet these criteria, it is not safe enough in Anthony’s mind to use in a professional setting. Help Anthony find passwords which might be of use to him.

**Input**

The first line will contain a single integer n that indicates the number of data sets that follow. Each data set will contain a single string s denoting a password that Anthony is considering. It COULD contain spaces, so be careful!

**Output**

Output "thumbs up" if the password is secure or "thumbs down" if the password is insecure.

**Example Input File**

3

MightyMouse12

My Guy

12345

**Example Output to Screen**

thumbs up

thumbs down

thumbs down

**11. Scramble**

# Program Name: scramble.java Input File: scramble.dat

Anthony is starting to try to spread his influence online, but unfortunately he doesn’t have the skills or the time to make his online presence himself. So instead, he is trying to hire a competent computer scientist to do it for him. To make sure the person he is hiring is of quality, Anthony set up this challenge for them to do during the interview.

Given a string s, generate all possible unique permutations of s and sort them such that they are in reverse alphabetical order. Output the first 5 in the sort. See if you are a viable candidate for Anthony’s team!

**Input**

The first line will contain a single integer n that indicates the number of data sets that follow. Each data set will contain a single string s that represents the string in the above challenge.

**Output**

On a single line, output the first 5 permutations of string s that are in the reverse alphabetical sort. If there are less than 5 permutations, output all of the permutations.

**Example Input File**

3

guard

told

hot

**Example Output to Screen**

urgda urgad urdga urdag uragd

told todl tlod tldo tdol

toh tho oth oht hto

**12. Incline**

# Program Name: incline.java Input File: incline.dat

Anthony is trying to help out his squat technique. He has really long femurs, so he is having trouble reaching full depth on the movement. Luckily, there is a solution for this. If Anthony implements inclines, he allows his ankles to have more dorsiflexion and thus he is able to go lower. Help him out by making some inclines for him.

**Input**

The first line will contain a single integer n that indicates the number of data sets that follow. Each data set will contain a single integer l, which represents the size of the incline to be printed.

**Output**

Output the incline pattern corresponding to the integer given. Be sure to include an empty line between test cases.

**Example Input File**

3

3

4

5

**Example Output to Screen**

#

##

###

#

##

###

####

#

##

###

####

#####