.----------------. .----------------. .----------------.

| .--------------. || .--------------. || .--------------. |

| | \_\_\_\_\_ | || | \_\_ | || | \_\_\_\_\_ | |

| | / \_\_\_ `. | || | / | | || | / \_\_\_ `. | |

| | |\_/\_\_\_) | | || | `| | | || | |\_/\_\_\_) | | |

| | .'\_\_\_\_.' | || | | | | || | .'\_\_\_\_.' | |

| | / /\_\_\_\_ | || | \_| |\_ | || | / /\_\_\_\_ | |

| | |\_\_\_\_\_\_\_| | || | |\_\_\_\_\_| | || | |\_\_\_\_\_\_\_| | |

| | | || | | || | | |

| '--------------' || '--------------' || '--------------' |

'----------------' '----------------' '----------------'

\_\_\_ \_\_ \_\_ \_

/ \_\_| \_ \_ \_\_ \_ \_ \_\_\_ \_\_\_\_\_\_ \ \ / /\_\_ \_\_\_ \_\_| |\_\_\_

| (\_| || | '\_ \ '\_/ -\_|\_-<\_-< \ \/\/ / \_ \/ \_ \/ \_` (\_-<

\\_\_\_\\_, | .\_\_/\_| \\_\_\_/\_\_/\_\_/ \\_/\\_/\\_\_\_/\\_\_\_/\\_\_,\_/\_\_/

\_\_\_ |\_\_/|\_| \_ \_\_\_ \_\_ \_ \_\_\_

| \ \_\_\_ \_\_ \_\_\_ \_ \_\_ | |\_\_ \_\_\_ \_ \_ |\_ ) \/ / \_ \

| |) / -\_) \_/ -\_) ' \| '\_ \/ -\_) '\_| / / () | \\_, /

|\_\_\_/\\_\_\_\\_\_\\_\_\_|\_|\_|\_|\_.\_\_/\\_\_\_|\_| /\_\_\_\\_\_/|\_|/\_/ \_ \_

/ \_\_| |\_ \_ \_(\_)\_\_| |\_ \_ \_\_ \_\_ \_ \_\_\_ / \_\_|\_\_\_ \_ \_| |\_ \_\_\_ \_\_| |\_

| (\_\_| ' \| '\_| (\_-< \_| ' \/ \_` (\_-< | (\_\_/ \_ \ ' \ \_/ -\_|\_-< \_|

\\_\_\_|\_||\_|\_| |\_/\_\_/\\_\_|\_|\_|\_\\_\_,\_/\_\_/ \\_\_\_\\_\_\_/\_||\_\\_\_\\_\_\_/\_\_/\\_\_|

|  |  |  |
| --- | --- | --- |
| Problem # | Problem Name | Solved? |
| 1 | Movie Marathon |  |
| 2 | Unity Update |  |
| 3 | Twelve Days of Christmas |  |
| 4 | Naughty List |  |
| 5 | Sandwiches |  |
| 6 | ChristMaths |  |
| 7 | Too Many Bens |  |
| 8 | Ocho |  |
| 9 | Messy Written |  |
| 10 | Like, um, uh |  |
| 11 | Yeet Theorem |  |
| 12 | There’s Always Tomorrow |  |
| 13 | Bodies |  |
| 14 | Simulator Accuracy |  |
| 15 | QuizBot |  |
| 16 | Portmanteau |  |
| 17 | Recruits |  |
| 18 | AP/K Level Basket Weaving |  |

**1. Movie Marathon**

# Program Name: Movie.java Input File: None

You are about to begin the annual Christmas Movie Marathon – 212 Christmas movies in the span of 25 days, giving you approximately 2 and a half hours for each movie. You’ve already made important preparations for your unnecessarily long movie marathon, but at 11:50 PM on November 30th, you realize you’ve forgotten one of the most important items: the Christmas tree. You believe in the power of the extra degree, and the Christmas tree is necessary to demonstrate your commitment. Time is racing against you… can you get yourself a tree?

**Input**

None.

**Output**

Print out this picture of a Christmas tree.

**Example Input File**

None

**Example Output to Screen**

\*

/|\

// \\

// \\

// o \\

// o \\

// o \\

//\_\_\_\_\_\_\_\_\_\_\_\\

|||

|||

|||

|||

---

**2. Unity Update**

# Program Name: Unity.java Input File: unity.dat

The season of Christmas is all around, and it is a merry jolly time for all students and teachers alike. During this most generous of seasons, the 7th period computer science class is trying to work on their soft skills by collaboratively singing a Christmas carol one line at a time. However, some students don’t understand when to ask a personal question and interrupt this fun holiday activity. If a student named Maxwell asks about a Unity Update when the Christmas carol is being painstakingly sung, the whole song is a failure and the whole class gets an F and a meter stick to their faces. If a student other than Maxwell asks about a Unity Update, the song goes on because they are not a repeat offender. If a line matches “Maxwell: Regarding the Unity update...”, the class has failed. Successfully singing this Christmas carol consists of not being interrupted by Maxwell asking about a Unity Update, resulting in the class getting an A.

**Input**

The first line of input will contain a single integer n that indicates the number of lines of the Christmas carol. Each line will consist of a name followed by a colon and a space, and the line of the Christmas carol that they sung.

**Output**

If the students were able to successfully sing the Christmas carol without being interrupted by Maxwell’s question about a Unity Update, print “Song passed! Everyone gets an A for Christmas.” If you were unable to sing the song successfully, print “Song failed. Everyone gets an F in their stocking.”

**Example Input File**

17

Alex: Dashing through the snow

Mihir: On a one horse open sleigh

Ricky: O'er the fields we go,

Maxwell: Laughing all the way

Zeki: Bells on bob tail ring,

Tristan: making spirits bright

Eyeoosh: What fun it is to laugh and sing

SkeletonKing: A sleighing song tonight

Ral: Oh, jingle bells, jingle bells

Steben: Jingle all the way

Maxwell: Regarding the Unity update...

Jah: Oh, what fun it is to ride

TTNorth: In a one horse open sleigh

TTSouth: Jingle bells, jingle bells

Chang: Jingle all the way

Glover: Oh, what fun it is to ride

Roneck: In a one horse open sleigh

**Example Output to Screen**

Song failed. Everyone gets an F in their stocking.

**3. Twelve Days of Christmas**

# Program Name: Twelve.java Input File: twelve.dat

After some coercive action, you have finally been able to complete the singing of several carols without interruption by a certain student. However, a new problem has arisen – some students have not studied their lines ahead of time for the next carol, the Twelve Days of Christmas. However, being the good CS student that you are, you realize that the lyrical structure of the Twelve Days of Christmas is very formulaic, so you decide to write a program to help everybody remember their lines. Given a list of items that your true love has given to you, print each one out in a way that will tell each student what to sing.

**Input**

The first line of input will contain a single integer n that indicates how many test cases to follow. The next n lines will each consist of a item.

**Output**

Print each item as is, but with “The next gift is” added at the beginning, and a period added at the end.

**Example Input File**

4

four calling birds

three french hens

two turtle doves

a partridge in a pear tree

**Example Output to Screen**

The next gift is four calling birds.

The next gift is three french hens.

The next gift is two turtle doves.

The next gift is a partridge in a pear tree.

**4. Naughty List**

# Program Name: Naughty.java Input File: naughty.dat

Sammy Claws, the eternal spirit of Christmas, has returned to give the good students their presents and the bad students their coal. Usually, this would not be a difficult task, since Sammy does the same thing every year and has a very efficient system in place. However, Sammy practiced a little too much computer science this year, and his vision has gotten a lot worse to the point where he sometimes cannot distinguish similar-looking names. This is a problem, because a lot of kids on his naughty list have similar names to the kids on his nice list, so he’s enlisted you to write a program to see if the names are actually right. Given the correct name and a list of other names that Sammy thinks are the same but may be slightly different, find the name that has the most correct characters in the same spot of the correct name. If no character is correctly placed, a failed message will be printed. If more than one name has the same number of characters in their correct places, the name that appeared first in the original dataset will be printed first. Print out each similar name, followed by an accuracy percentage that represents the percentage of letters that were in the correct position using this equation:

**Input**The first line will contain integer n*,* which represents the amount of test cases to follow. The next ndatasets will each start with integer s, followed by the correct string, and s-1 lines, each with a random permutation of the original string.

**Output**

Output the string that has the greatest number of corresponding letters, and on the next line, print “Accuracy Percentage: ’’, followed by the accuracy percentage calculated using the formula aforementioned and rounded to a whole number. If no strings were found to have even one character in the corresponding place, print “FAILED” on a new line instead of the corresponding string and for the accuracy percentage, print “NaN” instead of a percentage.

**Example Input File**

2

4

wildcat

wdatlci

awtcdil

wiadclt

5

squid

qdisu

qsidu

diqus

usqdi

**Example Output to Screen**

wiadclt

Accuracy Percentage: 71%

FAILED

Accuracy Percentage: NaN

**5. Sandwiches**

# Program Name: Sandwiches.java Input File: sandwiches.dat

Sammy Claws always keeps his reindeer fed. After all, to drive his operations, his reindeer have to be healthy and fit. The reindeer are very picky and only eat handmade sandwiches. However, Sammy has recently become more and more busy and is looking at outsourcing the job of feeding his reindeer, and he has outsourced it to you because it means he won’t have to pay taxes. To make a sandwich, you need bread, meat, and cheese. You went to the grocery store earlier this week and made a log of what items you bought as you were putting them in your refrigerator. Being an organized person, you separated each type of food into different drawers. However, this also means the first item you wrote down is at the very back of your refrigerator, and you don’t want to just go off the list and dig out everything from the back. So, you’ve decided to write a program to create sandwich combinations for you to eat. Items will not be put back into the refrigerator once you have eaten them for obvious reasons. If any of the three items are missing, then you cannot make a sandwich.

**Input**

The first line of input will contain a single integer n that indicates the number of things you bought. The next n lines will contain a log of all the items you bought, with the name, a dash, and then the type of food it is – bread, meat, or cheese. These lines are in the order you put the food into the refrigerator.

**Output**

For each possible sandwich combination, output the sandwiches in the order you will be eating them, and the sandwiches’ ingredients in the format bread, meat, cheese.

**Example Input File**

10

Whole wheat-bread

White-bread

Turkey-meat

American-cheese

Swiss-cheese

Ham-meat

Cheddar-cheese

Chicken-meat

Hamburger buns-bread

Patty-meat

**Example Output to Screen**

Hamburger buns, Patty, Cheddar

White, Chicken, Swiss

Whole wheat, Ham, American

**6. ChristMaths**

# Program Name: ChristMaths.java Input File: christmaths.dat

Sammy Claws is finally finished giving all his presents to the good children, and he’s ready to call it a night. Sammy is feeling extra generous this Christmas, however, and decides to give all the children on the naughty list a second chance. He wants to see if the children have learned anything in their math classes, so he decides to give them a basic arithmetic problem. Unfortunately, it turns out that Sammy dropped out of school a little too early in order to pursue the art of gift giving, so he has no concept of order of operations. Instead, Sammy does each operation in the order that it is given to him. Given a mathematical expression and a child’s answer to the expression, determine whether Santa will give them a present or not. Sammy will give the child a present if and only if the child’s answer to the expression is equal to what Sammy would have gotten. (Note: Sammy can use all 4 basic operators: +, -, \*, /. All division will be integer division.) All numbers in each mathematical expression will be single digits, although the answer may not be a single digit.

**Input**

The first line of input contains 1 integer n, the number of test cases to follow. The next n lines of input contain a mathematical expression and the child’s answer for the expression.

**Output**

Determine whether Santa will give the child a present. If Santa is willing to give the child a present, print out “Present for You”, and if he will not give the child a present, print out “Coal for You”.

**Example Input File**

3

3+5\*4–2 30

4-6\*2+2 4

6/2+5-1 7

**Example Output to Screen**

Present for You

Coal for You

Present for You

**7. Too Many Bens**

# Program Name: Bens.java Input File: bens.dat

Sammy Claws is trying to deliver presents to the CS kids, but there are too many Bens. Being human, Sammy Claws cannot hope to deliver a high-quality present to every Ben. Instead, he must prioritize the bens first based on their grade in AP/K Level Basket Weaving, then their vocal range, and finally the length of their femur. The Bens will get better presents based on the hierarchy below:

* The higher their grade in AP basket weaving, the better the present
* Then the larger their vocal range, the better present
* Then the longer their femur, the better the present

**Input**The first number is the total amount of cases. The next number, x, is the number of Bens in that sorting case. Next will be x number of sets of a String representing their names, a double representing their grades in AP basket weaving, 2 integers representing the lowest and the highest frequencies of their vocal range, where the difference is the range, and finally a double representing their femur length in inches.

**Output**The output should contain all Bens in ascending order based on the hierarchy above separated by commas and a space.

**Example Input File**2

3

Ben Armstrong 100.0 27 100 57.5

Ben Yonas 98.0 7 89 56.8

Ben Gonzales 98.00 0 56 57

6

Ben Armstrong 100.1 27 100 57.5

Ben Yonas 98.0 7 89 56.8

Ben Gonzales 78 0 56 57

Ben Davis 78 8 9 35

Ben Stiller 78 44 55 14

Ben Franklin 100 44 45 5000000

**Example Output to Screen**Ben Gonzales, Ben Yonas, Ben Armstrong

Ben Davis, Ben Stiller, Ben Gonzales, Ben Yonas, Ben Franklin, Ben Armstrong

**8. Ocho**

# Program Name: Ocho.java Input File: ocho.dat

After demonstrating your soft skills at the job interview, you’ve landed a job at Nine Oceans, a prestigious software development company. For some weird reason, your first task is to convers numbers written in Spanish and compare them to other numbers. While this is simple enough, your boss graduated from 8 Rivers Middle School and enforces a policy requiring 8 to be greater than everything (except for itself, in which it is equal to 8). Write a program to do your job for you.

**Input**

The first number t indicates the number of test cases that follow. For each test case, there are two all uppercase words in Spanish. Spanish numbers are as follows:

* 0 = CERO
* 1 = UNO
* 2 = DOS
* 3 = TRES
* 4 = CUATRO
* 5 = CINCO
* 6 = SEIS
* 7 = SIETE
* 8 = OCHO
* 9 = NUEVE
* 10 = DIEZ

**Output**

For each test case, output the first number followed by a greater than, equal to, or less than sign followed by the second number

**Example Input File**

4

UNO DOS

TRES UNO

DIEZ CERO

DIEZ OCHO

**Example Output to Screen**

1 < 2

3 > 1

10 > 0

10 < 8

**9. Messy Written**

# Program Name: Written.java Input File: written.dat

Written tests are hard to grade, and sometimes, something goes wrong. At the 8 Rivers Middle School Computer Science Competition, Ral received a 236 on the written test! The way written tests are scored is 6 points for every question correct, no points awarded for skipped questions, and -2 points for incorrect answers. There are always 40 questions on a written test, so some scores shouldn’t be possible. Write a program to determine whether a given written score is possible.

**Input**

The first line of input is n, the number of data sets to follow. The next n lines will be an integer within the range -80 to 240.

**Output**

For each test case, print “Possible” if the written score is possible or “Impossible” if the written score is impossible.

**Example Input File**

5

118

101

236

-12

86

**Example Output to Screen**

Possible

Impossible

Impossible

Possible

Possible

**10. Like, um, uh**

# Program Name: Like.java Input File: like.dat

Alex stutters when he talks too much, and fills up the empty space with words like like, um, and uh. Mr. A wants to help him improve his soft skills by overcoming his speech problem. Write a program to remove all filler words from a sentence.

**Input**

The first line of input will contain a single integer n that indicates how many test cases to follow. Each test case will contain a sentence on one line.

**Output**

For each sentence, output the sentence without any filler words.

**Example Input File**

7

What if like we uh wrote more contest problems about like stupid stuff

Your thing is like not working

He um didn't like do that

Can you like please help me solve this foobar problem

What about like the equation for the uh um left branches

I was made in his likeness

First in, first out

**Example Output to Screen**

What if we wrote more contest problems about stupid stuff

Your thing is not working

He didn’t do that

Can you please help me solve this foobar problem

What about the equation for the left branches

I was made in his likeness

First in, first out

**11. Yeet Theorem**

# Program Name: Yeet.java Input File: yeet.dat

Zeki is really convinced that Yeet Theorem really works, and decides to dedicate a whole line of research behind this theorem. Yeet Theorem states that to raise a number n to the power p, you can “yeet” the power in front of the number and concatenate them to get your resultant number. However, Zeki is really stupid and needs your help to both determine the result using Yeet Theorem and find out if it equals the real value of n raised to p.

**Input**

Given t describing the number of test cases. For each test case, you are given n and p (0 ≤ n, p ≤ 15).

**Output**

Return the value after yeeting the power and if the value is valid. If the value equals n^p, print “Yeet”. Otherwise, print “Get Yoted”.

**Example Input File**

3

5 2

3 3

15 0

**Example Output to Screen**

25 Yeet

33 Get Yoted

15 Get Yoted

**12. There’s Always Tomorrow**

# Program Name: Tomorrow.java Input File: tomorrow.dat

After having his business idea brutally rejected by Mark Cuban, Lil’ Willy Brazofuerte, world-renowned rapper, is feeling a little upset. As his best friend, you want to help him cope with his struggles, so you tell him that “there’s always tomorrow”. “But what is tomorrow?” Lil’ Willy B. asks you. Now, your task is to tell him.

**Input**

The input will begin with a single integer, t, denoting the number of test cases to follow. Each test case will contain a date formatted as MM DD YYYY.

**Output**

Output the date of exactly one day after the date given, taking into account leap years. Please output each date in the MM DD YYYY format and do not print blank lines between each test case.

**Example Input File**

3

12 31 6969

12 14 2019

02 28 2004

**Example Output to Screen**

01 01 6970

12 15 2019

02 29 2004

**13. Bodies**

# Program Name: Bodies.java Input File: bodies.dat

Mr. A just killed someone. He has tasked his butler, Tristan Weaslepopsicle, to dispose of the body at night. Mr. A has already identified possible sites to dispose of the body, but there are police officers who roam around, so Butler Weaslepopsicle must by very stealthy. Write a program to see if Tristan can dispose of the body without being caught by the police.

**Input**

The first line represents the number of data sets to follow. In each data set the first line contains the rows and columns of the map, respectively. the next “r” lines will contain the map. The “T” represents where Tristan starts. There can be 1 to 3 “S”’s, which represents possible sites to dispose of the body. There can be 0 to 3 “P”’s, which represent police officers. Tristan can travel in any square except one space around a police officer, including diagonals, or a wall, denoted by a “#”. You can assume Tristan’s starting position won’t be right next to a police officer, but a possible site could be right next to a police officer, making it automatically unaccessible.

**Output**

Always output “Bravo Six Going Dark...” to signify that Tristan has started his night time mission. If it is possible for Tristan to dispose of the body without getting caught, output “at the end of the tunnel is a light” on the same line. Otherwise output “Mission Failed. We’ll get em next time.”

**Example Input File**

2

6 7

.S.####

.#P..##

T.##P##

...#..S

#....##

#####S.

5 5

.....

....S

..P.#

..#.#

T...#

**Example Output to Screen**

Bravo Six Going Dark... Mission Failed. We'll get em next time.

Bravo Six Going Dark... at the end of the tunnel is a light.

**14. Simulator Accuracy (no good theme yet)**

# Program Name: Simulator.java Input File: simulator.dat

Felix is trying to build his own simulator to predict how a stock will do over a certain time period. He creates a program to that predicts what a stock’s price will be during the course of year. He wants to test the accuracy of his model, and he needs your help doing this. Given three trials simulated trials and a real stock price after a period of time, output the accuracy of his model in the form of a percentage.

**Input**

The first line of input will contain an integer n which is the number of test cases. The first three numbers in any following line are the three simulated trials. The next value is the price of the stock after a time period.

**Output**

The output should be the percent error between the three data trials and the actual stock price, formatted as a percentage with two decimal places. The equation for percent error is as follows:

**Example Input File**

4

110.89 111.61 111.98 129.30

209.69 210.68 212.69 257.24

188.01 187.07 188.54 234.01

1106.79 1112.62 1101.08 1291.01

**Example Output to Screen**

13.77%

17.97%

19.72%

14.27%

**15. QuizBot (no good theme yet)**

# Program Name: QuizBot.java Input File: quizbot.dat

Jonathan decided that it was a good idea to start to develop a quiz bot that could quiz his friends. However, he couldn’t figure out how to compare the answers that the players gave to the answers of the question. So, he’s forced to turn to you. Could you help Jonathan figure out how to analyze the answers and print out the winner in the quiz?

**Input**

The first line will have an integer n, the number of test cases to follow. Each test case consists of 3 lines: The first line is the two names of the players separated by a space. The second line contains the proper answers to the questions separated by a space and the last line is the answers that each player has put in player1 player2 order. There will always be two player answers for each question answer. The answers will always be in the same case.

**Output**

The output will be a single line that says who won the round in the format “(Winner) has won this round!”. If the round ends as a tie, output “(player1) and (player2) are tied this round!

**Example Input File**

2

JohnChris ChrisJohn

Five Gray Green Orange

Five Blue Blue Green Orange Green Orange Yellow

Marie Clarie

Five Four

Five Four Five Four

**Example Output to Screen**

JohnChris has won this round!

Marie and Claire are tied this round!

**16. Portmanteau (no good theme yet)**

# Program Name: Portmanteau.java Input File: portmanteau.dat

A portmanteau is a combination of two words into a single word. Examples include motel (motor + hotel), brunch (breakfast + lunch), and hangry (hungry + angry). Given a portmanteau and two words, determine if it is possible to form the portmanteau with said words. A portmanteau is said to be formable if a continuous stream of characters from the beginning of one word combined with a continuous stream of characters from the end of the other word makes the portmanteau. The streams must both be of nonzero length.

**Input**

The first line of input will contain a single integer n that indicates how many test cases to follow. Each test case will contain the portmanteau, then two words.

**Output**

If it is possible to create the portmanteau from the words, output “YES”. Otherwise, output “NO”.

**Example Input File**

4

motel motor hotel

brunch lunch breakfast

jangry hungry angry

whism antidisestablishmentarianism what

**Example Output to Screen**

YES

YES  
NO

YES

**17. Recruits (no good theme yet)**

# Program Name: Recruits.java Input File: recruits.dat

A team is a group of at most 3 people. You are in charge of handling the people who are interested in joining teams. Given a list of raw data consisting of recruits and a team they want to be on, sort them into respective teams and print them out in a more friendly format. Teams are created on a first come, first serve basis. If somebody wants to join a team that already has 3 people, they will not be able to join.

**Input**

The first line of input will contain a single integer n that indicates how many test cases to follow. The first line of each test case will be a single integer r that indicates the number of recruits that signed up. Each of the following lines consists of the recruit’s name, then a dash -, followed by their preferred team’s name. If a recruit does not have a preferred team, they will be placed with other team-less recruits and recruits who could not get into their preferred team.

**Output**

Output each team’s name followed by the members of the team, both in alphabetical order. Any recruits without a team will be placed a team named Untitled Team X, where X is a number that starts at 1 and increments for each untitled team that is created. The untitled teams will be sorted together with the titled teams in alphabetical order. Separate each team with a newline, and separate each test case with ----- (5 dashes). There will also be a newline both in front of and after each set of dashes.

**Example Input File**

2

8

Alex-Team A

Mihir-Team A

Zeki-Team B

Ashay-Team A

Sidh-Team A

Ronak-Team B

Tristan

Ral

6

ASDF-GHJK

QWERTY

2539

J-J

foo

bar

**Example Output to Screen**

Team A

Alex

Ashay

Mihir

Team B

Ronak

Zeki

Untitled Team 1

Ral

Sidh

Tristan

-----

GHJK

ASDF

J

J

Untitled Team 1

2539

foo

QWERTY

Untitled Team 2

bar

-----