Computer Science Competition
Invitational A 2023
Programming Problem 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. Names of Problems

Number	Name
Problem 1	Abhinav
Problem 2	Alan
Problem 3	Bianca
Problem 4	Cho
Problem 5	Damian
Problem 6	Feng
Problem 7	Gael
Problem 8	Himanshu
Problem 9	Justin
Problem 10	Marek
Problem 11	Priyanka
Problem 12	Sunny

1. Abhinav

Program Name: Abhinav.java **Input File: none**

Abhinav is really excited about taking a programming course. He has tried following his teacher's examples but just cannot get his program to produce the exact output required by the assignment.

Can you help Abhinav with this problem?

Input: None

Output: The exact message as shown below.

Sample input: None

Sample output:

```
Ultimate Important Languages
U-I-L
UIL!
```

2. Alan

Program Name: Alan.java **Input File: alan.dat**

In Alan's Integrated Physics and Chemistry class, Alan's teacher has just introduced the concept that temperature is traditionally given in either Fahrenheit, Celsius, or Kelvin. Alan is familiar with both Fahrenheit and Celsius since he has spent time in both America and Europe, but the concept of temperatures in Kelvin is still a bit new and unfamiliar to him. In an attempt to better understand temperatures in Kelvin, Alan has the idea to write a program that will convert a given Fahrenheit temperature to Kelvin. Alan knows the formula for converting from Fahrenheit to Kelvin is:

$$K = 5/9(F - 32) + 273.15$$

Where K is Kelvin and F is Fahrenheit. Alan is having a bit of trouble with the math though, and would like your help completing the program. Think you can assist him with the task?

Input: Input will consist of an integer N, the number of test cases. The number of test cases will be in range [1,20]. Each subsequent line will contain an integer degree F, representing the temperature in Fahrenheit. F will be in range [-250,250].

Output: Each line of output must consist of "`F degrees Fahrenheit is equal to K Kelvin`", where F is the input temperature in degrees Fahrenheit, and K is the converted temperature in Kelvin using the formula above. Output should be formatted to two decimal places after the decimal point.

Sample input:

```
5
32
100
25
212
-5
```

Sample output:

```
32 degrees Fahrenheit is equal to 273.15 Kelvin
100 degrees Fahrenheit is equal to 310.93 Kelvin
25 degrees Fahrenheit is equal to 269.26 Kelvin
212 degrees Fahrenheit is equal to 373.15 Kelvin
-5 degrees Fahrenheit is equal to 252.59 Kelvin
```

3. Bianca

Program Name: Bianca.java **Input File: bianca.dat**

Your friend Bianca has Geometry homework due next period, and she forgot all about it! Quickly write a program to solve Pythagoras theorem before the homework is due.

Use the following theorem (you are given a and b, so solve for c):

$$A^2 + B^2 = C^2$$

Input: The first line will contain a single integer `n` ($0 < n < 100$) that indicates the number of data sets that follow. Each data set will consist of two integers A and B ($0 < A, B < 1000$), separated by a space, denoting the values A and B in the equation above.

Output: Output the value of C ($0 < C < 1000000$) given the A and B values from the input, rounded to 2 decimal places, as shown below.

Sample input:

```
4
3 4
12 14
5 12
5 8
```

Sample output:

```
5.00
18.44
13.00
9.43
```

4. Cho

Program Name: Cho.java **Input File: cho.dat**

You and your friend Cho have an internship at a data lab and they need you to go through sentences and find the letter that occurs the most in each one.

Input: The first line will contain a single integer `n` ($0 < n < 100$) that indicates the number of data sets that follow. Each data set will consist of one line of an unknown number of words(strings) separated by spaces, all appearing on the same line.

Output: Output the alphabetic (A-Za-z) character which occurred in the input line the most, do not include numbers, whitespace, special characters, or anything else that isn't a letter. If there is a tie, just choose the letter that comes first alphabetically, with the ENTIRE capital alphabet coming before the lowercase alphabet (basically sort by ascii values).

Sample input:

```
3
THE HOG GOES TO THE DOG TO CATH THE FROG IN THE BOG
WITCHES MADE MY SOUP, NOW I CAN FLY WHEEEEE
Qwertyuiolkjhgfdazxcvbnm
```

Sample output:

```
O
E
Q
```

5. Damian

Program Name: Damian.java **Input File: damian.dat**

Damian has always been intrigued by numbers. He is absolutely giddy about prime numbers, but honestly, he loves anything to do with factors. He was investigating numbers and he found one type of number that he really liked. These are what he calls "growing" numbers. That is when, looking at the digits from left to right, they increase.

Here are some examples: 123, 38, 1479, 5, 123456789

These are NOT growing numbers: 73, 762, 12557 (notice those two 5s)

Then Damian got an idea, he wants to find the "growing factors" of a number. These are factors that meet the qualifications of being growing. Write a program to help him out. You will make Damian's day.

Example: The factors of 1122 are: 1,2,3,6,11,17,22,33,34,51,66,102,187,374,561,1122
The growing factors of 1122 are: 1,2,3,6,17,34

Input: Input will consist of an integer N, the number of test cases. The number of test cases will be in range [1,20]. Each subsequent line will contain an integer in the range[1,1000000].

Output: Each line of output will be a line of "growing factors" listed in ascending order with a space between each number.

Sample input:

```
5
72
144
1331
621
554433
```

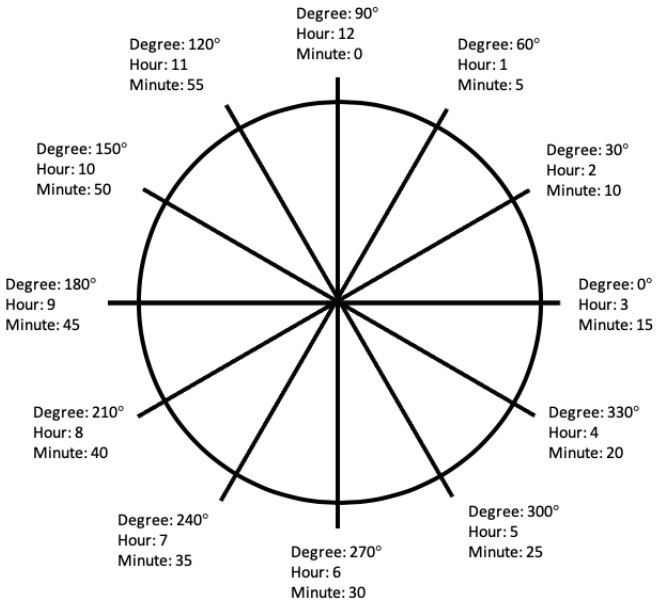
Sample output:

```
1 2 3 4 6 8 9 12 18 24 36
1 2 3 4 6 8 9 12 16 18 24 36 48
1
1 3 9 23 27 69
1 3 159
```

6. Feng

Program Name: Feng.java **Input File: feng.dat**

Feng just learned about the unit circle in her pre-calculus class. As she was leaving class, she realized that the hour and minute hands of an analog clock not only point to their respective hour and minute (representing the current time), but they each also point to a degree in range of [0,359] corresponding to the unit circle representation. (See below figure.) Feng would like your help writing a program that takes in the time, and emits two angles, one for hour, and one for minute, corresponding to the angle (in degrees) each hand would be pointing at.



Input: Input starts with a line containing an integer N, the number of test cases. N will be in range [1,20]. The following N lines contain the time, formatted as HH:MM. HH will be in range [1,12] and MM will be in range [0,59].

Output: For each test case, output the corresponding angle measures each of the two hands would be pointing to.

Hour angle (denoted as `H_Angle` below) and minute angle (denoted as `M_Angle`) will be guaranteed to be in range of [0,359]. Formatting for the output should be "`Hour: H_Angle Minute: M_Angle`"

Note: `H_Angle` and `M_Angle` should be outputted to 1 (one) decimal place.

Sample input:

```
6
08:25
09:18
12:05
03:00
03:01
03:15
```

Sample output:

```
Hour: 197.5 Minute: 300.0
Hour: 171.0 Minute: 342.0
Hour: 87.5 Minute: 60.0
Hour: 0.0 Minute: 90.0
Hour: 359.5 Minute: 84.0
Hour: 352.5 Minute: 0.0
```

7. Gael

Program Name: Gael.java **Input File: gael.dat**

You and your friend Gael have a new art project! You will be given a word and you must make a box design with it, as shown in the sample input and output.

Input: The first line will contain a single integer `n` ($0 < n < 100$) that indicates the number of data sets that follow. Each data set will consist of one word, on its own line, consisting of only capital letters.

Output: Output the shape created using the word given. It will be filled with spaces in the middle and the border will consist of the word multiple times. The corners should match up so that all the words are still spelled correctly, meaning certain words will have to be backwards. The two that will be backwards for this program are the word on the bottom of the box, and the word on the right side of the box. The necessary orientations of the 4 words in the design is as follows: The word on top of the box will go from left to right, the word on the left side of the box will go from top to bottom, the word on the bottom of the box will go from right to left, and the word on the right side of the box will go from bottom to top, see the sample output for clarification.

Sample input:

```
3
HELLO
WORLD
RACECAR
```

Sample output:

```
HELLO
E   L
L   L
L   E
OLLEH
WORLD
O   L
R   R
L   O
DLROW
RACECAR
A     A
C     C
E     E
C     C
A     A
RACECAR
```

8. Himanshu

Program Name: Himanshu.java **Input File: Himanshu.dat**

Himanshu's little brother loves to play the ancient game *Moksha Patam*. *Moksha Patam* originated in India in the 2nd century AD and is associated with traditional Hindu philosophy contrasting karma and kama, or destiny and desire. Today, however, the game is most known as either *Snakes and Ladders*, or *Chutes and Ladders*. Aside from its ancient roots, the game is a great tool for helping young children learn to count.

Moksha Patam is a turned based game played on a square grid with varied dimensions, in which players roll a single 6-sided die (1-6) to determine the number of moves the player is to move forward. After the end of a turn, a player may potentially land on either a "S#" signifying a snake (or chute) or an "L#" signifying a ladder. If a player lands on the lower-numbered end of a ladder, the player moves up to the ladder's higher-number square. If the player lands on the higher-numbered square of a snake, the player moves down to the snake's lower-numbered square. The player or players who reach the last square, or surpass the last square, is declared the winner. As this is a turned based game, all players are guaranteed to have the same number of turns, i.e. if Player 1 reaches or surpasses the last square, Player 2 is afforded one last move to try and reach or surpass the last square as well.

The below image shows an example *Moksha Patam* board with dimension 10 x 10:

```
99 98 S3 96 95 94 93 L3 91 90
80 81 82 83 84 85 86 87 88 89
79 S2 77 L3 75 74 73 72 71 70
60 61 62 63 64 65 66 67 68 69
59 58 57 S1 55 54 53 52 51 50
L2 41 42 43 44 45 46 S2 48 49
39 38 37 36 L1 34 33 32 31 30
20 21 22 23 24 25 26 27 28 29
L2 S1 17 16 15 14 13 12 S3 10
00 01 02 03 04 05 L1 07 08 09
```

All players start at the square with index `0`. For formatting purposes for the 10 x 10 board, `0` is denoted as `00` so that each column is straight. Assuming a player is currently at index `00` and roles a 6 from the die, the player would move to index `06`, but this index has `L1` on it denoting this as a ladder. Therefore, the player is then moved to index `35` which is the higher-number squared corresponding to `L1`. If the player were at index `91` and were to roll a 1, the player would move to index `92` and would remain here until the following turn, as this is the higher-numbered latter corresponding to `L3`. If the player was at index `54` and were to roll a 2, the player would move to index `55`. Index `55` however has `S1` denoting this as a snake (chute). Therefore, the player is moved down to index `18` as this is the lower-number square corresponding to `S1`. If a player were at index `43` and were to roll a 4, the player would move to index `47` and would remain here until the following turn, as this is the lower-numbered snake corresponding to `S2`. Although depicted in this board as being on separate rows, snakes and ladders do not necessarily have to be on the different rows. This means that a player could move horizontally only, with no vertical movement on the board.

Unfortunately, with UIL Computer Science season ramping up, Himanshu doesn't have as much time to play *Moksha Patam* with his little brother due to CS practice. In an effort to not only appease his little brother's thirst for *Moksha Patam*, but to get some extra coding practice in as well, Himansu comes up with the idea to write a program that will play *Moksha Patam* against his little brother. The game will only feature two players: Player 1 and Player 2. A series of moves for each player will be given corresponding to the dice roll for that term. As soon as a winner is found and both players have had equal number of turns, the play for the current game is terminated. Can you help Himanshu write such a program?

Input: Input starts with a line containing an integer N, the number of games to be played. N will be in range [1,20]. Each game starts first with an integer D for the dimension of the board. D will be in range [2,32]. Odd dimensions

~ Problem continues on next page ~

Himanshu, continued

for the square board are allowed. This means that the last square to be reach or surpassed is not guaranteed to be at the top left. For formatting purposes all indexes will be outputted to uniform column width, so that each column is straight up and down. The following D lines contain the board for that given game play. Boards use a back-and-forth track from the bottom of the playing area (guaranteed to be the bottom left corner) to the last square. Indexes will be incremented by 1. The board will also have scattered snakes (chutes) and ladders placed throughout the board. Snakes will be represented by `S##` and ladders will be represented by `L##`. Where `##` represents the number of the respective snake or ladder. `##` will be formatted with padded zeros to make sure the formatting for each column remains straight. For every snake or ladder present, it will be guaranteed to have a corresponding end point. It can also be guaranteed that a snake nor a ladder will be present at index `00` nor at the max index of $D^2 - 1$. Following the board are two lines. The first starting with "`P1:`" and the second starting with "`P2:`", each followed by the respected moves for the player. Moves will be in range [1,6] and separated by a comma. The number of moves present may include extra moves that won't be carried out, if a player or players, is found to have reached the max index or surpassed the max index. Last, a line of 36 dashes "`-`" is presented to separate the different game inputs.

Output: For each game played, you are to output either: `Game #CURRENT_GAME_NUMBER: Player 1 wins!`, `Game #CURRENT_GAME_NUMBER: Player 2 wins!`, `Game #CURRENT_GAME_NUMBER: Both players win!`, or `Game #CURRENT_GAME_NUMBER: Neither Player 1 or Player 2 won.` Remember, each of the two players are guaranteed to have the same number of moves, resulting in the potential for two winners.

Sample input: (the new line character is only present after all the moves for a given player have been given)

```
9
10
99 98 S3 96 95 94 93 L3 91 90
80 81 82 83 84 85 86 87 88 89
79 S2 77 L3 75 74 73 72 71 70
60 61 62 63 64 65 66 67 68 69
59 58 57 S1 55 54 53 52 51 50
L2 41 42 43 44 45 46 S2 48 49
39 38 37 36 L1 34 33 32 31 30
20 21 22 23 24 25 26 27 28 29
L2 S1 17 16 15 14 13 12 S3 10
00 01 02 03 04 05 L1 07 08 09
P1:1,4,3,1,5,6,1,5,1,3,6,1,4,5,1,6,5,1,1,2,5,2,1,4,1,3,6,3,1,3,6,1,4,5,1,6,5,1,1,2,6,5
,2,4,1,3,6,4,4,3,2,3,1,6,5,2,4,1,3
P2:6,6,1,6,2,3,3,5,1,5,1,1,1,4,3,6,4,4,3,2,3,1,6,5,2,4,1,3,6,4,4,3,2,3,1,6,5,2,4,1,3,5
,2,4,1,3,6,4,4,3,2,3,1,6,5,2,4,1,3
------------------------------------
9
99 98 S3 96 95 94 93 L3 91
80 81 82 83 84 85 86 87 88
79 S2 77 L3 75 74 73 72 71
60 61 62 63 64 65 66 67 68
59 58 57 S1 55 54 53 52 51
L2 41 42 43 44 45 46 S2 48
```

~ Sample input continues on next page ~

Himanshu, continued

```
39 38 37 36 L1 34 33 32 31
20 21 22 23 24 25 26 27 28
L2 S1 17 16 15 14 13 12 S3
P1:1,6,6,6,2
P2:1,4,3,1,5
-----------------------------------


10
99 98 S3 96 95 94 93 L3 91 90
80 81 82 83 84 85 86 87 88 89
79 S2 77 L3 75 74 73 72 71 70
60 61 62 63 64 65 66 67 68 69
59 58 57 S1 55 54 53 52 51 50
L2 41 42 43 44 45 46 S2 48 49
39 38 37 36 L1 34 33 32 31 30
20 21 22 23 24 25 26 27 28 29
L2 S1 17 16 15 14 13 12 S3 10
00 01 02 03 04 05 L1 07 08 09
P1:6,6,5,6,6,6,6,6,2,5
P2:6,6,5,6,6,6,6,6,2,1
-----------------------------------
10
99 98 S3 96 95 94 93 L3 91 90
80 81 82 83 84 85 86 87 88 89
79 S2 77 L3 75 74 73 72 71 70
60 61 62 63 64 65 66 67 68 69
59 58 57 S1 55 54 53 52 51 50
L2 41 42 43 44 45 46 S2 48 49
39 38 37 36 L1 34 33 32 31 30
20 21 22 23 24 25 26 27 28 29
L2 S1 17 16 15 14 13 12 S3 10
00 01 02 03 04 05 L1 07 08 09
P1:6,6,5,6,6,6,6,6,2,5
P2:6,6,5,6,6,6,6,6,2,5
-----------------------------------
3
06 L1 08
05 S1 03
00 L1 S1
P1:1,1
P2:2,2
-----------------------------------
```

~ Sample input continues on next page ~

Himanshu, continued

```
2
03 L1
00 L1
P1:1,1
P2:2,2
-----------------------------------
5
20 L6 L5 S5 L6
L5 L4 S4 S5 S3
S3 S4 L4 L3 S4
L3 S1 L1 L2 S2
00 L1 S1 L2 S2
P1:6,6,5,6,6,6,6,6,2,5
P2:6,6,5,6,6,6,6,6,2,5
-----------------------------------
11
S21 L08 S02 S16 S03 115 L06 S12 118 S10 120
L05 L17 S17 S01 L06 104 L12 S05 101 L11 S08
S15 089 L02 L07 092 S07 S09 S03 S14 L08 S11
087 S22 L12 L13 L19 082 081 S18 S20 078 077
S04 067 L04 L07 L15 071 L18 L19 S17 L17 076
L02 064 S18 062 S02 S07 L16 L14 S13 S15 S06
S14 L10 L13 S06 S16 L09 S01 051 S20 053 S12
043 L09 L11 L14 L16 S08 L03 036 L03 L18 L10
S19 L01 S10 S13 026 027 L01 029 030 031 032
S05 L04 S11 018 017 016 015 S04 S09 S21 011
000 S23 L05 003 004 005 S23 007 L15 S22 S19
P1:1,4,3,1,5,6,1,5,1,3,6,1,4,5,1,6,5,1,1,2,5,2,1,4,1,3,6,3,1,3,6,1,4,5,1,6,5,1,1,2,6,5
,2,4,1,3,6,4,4,3,2,3,1,6,5,2,4,1,3
P2:6,6,1,6,2,3,3,5,1,5,1,1,1,4,3,6,4,4,3,2,3,1,6,5,2,4,1,3,6,4,4,3,2,3,1,6,5,2,4,1,3,5
,2,4,1,3,6,4,4,3,2,3,1,6,5,2,4,1,3
-----------------------------------
11
S21 L08 S02 S16 S03 115 L06 S12 118 S10 120
L05 L17 S17 S01 L06 104 L12 S05 101 L11 S08
S15 089 L02 L07 092 S07 S09 S03 S14 L08 S11
087 S22 L12 L13 L19 082 081 S18 S20 078 077
S04 067 L04 L07 L15 071 L18 L19 S17 L17 076
L02 064 S18 062 S02 S07 L16 L14 S13 S15 S06
S14 L10 L13 S06 S16 L09 S01 051 S20 053 S12
043 L09 L11 L14 L16 S08 L03 036 L03 L18 L10
S19 L01 S10 S13 026 027 L01 029 030 031 032
```

~ Sample input and output continues on next page ~

Himanshu, continued

```
S05 L04 S11 018 017 016 015 S04 S09 S21 011
000 S23 L05 003 004 005 S23 007 L15 S22 S19
P1:1
P2:6
-----------------------------------
```

Sample output:

```
Game #1: Player 2 wins!
Game #2: Neither Player 1 or Player 2 won.
Game #3: Player 1 wins!
Game #4: Both players win!
Game #5: Player 1 wins!
Game #6: Both players win!
Game #7: Both players win!
Game #8: Player 2 wins!
Game #9: Neither Player 1 or Player 2 won.
```

9. Justin

Program Name: Justin.java **Input File: justin.dat**

Justin's parents are considering buying a new car and when that happens he will get the current family car. He will finally have a car of his own! He wants to help his parents by creating a report that summarizes the purchase details for each of the cars they are considering.

They will be using a car loan to pay for most of the purchase so the information needed to help them make a decision is the monthly payment, the final cost of the car after making the loan payments and the amount of interest that will have been paid. Justin has found the following formula for calculating the monthly payment.

Formula	**Legend**
$p = \frac{ar}{1-(1+r)^{-n}}$	a = amount of loan r = monthly interest rate n = number of payments p = monthly payment

The loan amount ***a*** is the purchase price less the down payment. The monthly interest rate ***r*** is 1/12 of the APR or annual percentage rate for the loan as a decimal value not a percentage. The number of monthly payments ***n*** and monthly payment amount ***p*** are as expected. The monthly payment must be rounded to nearest penny; otherwise, partial pennies add up to extra pennies after all payments.

For each car, calculate and display the monthly payment, final cost which is the total of all payments plus the down payment, and the amount of interest paid which is the total of payments less the purchase price.

Using first set of data as a sample:

Monthly payment is $587.07
Final cost is $37224.20 [587.07 * 60 + 2000.00]
Interest paid is $6236.79 [37224.20 – 30987.41]

Input: The first line contains an integer ***T*** which is the number of test cases with $3 \le T \le 20$. Each of ***T*** following lines are one complete test case and will contain four pieces of data:

Purchase price between and including $10,000.00 and $100,000.00
Down payment ***d*** with $\$0.00 \le d \le \$5000.00$
APR as a percentage with $1.0000\% \le APR \le 20.0000\%$
Number of monthly payments ***n*** with $12 \le n \le 96$

Output: One line for each test case containing 3 values: monthly payment, final cost, and total interest paid. Each value is rounded to 2 decimal places and has a leading $ with the value right-aligned in a field of 9 columns. One space separates the fields.

Sample input:

```
3
30987.41 2000 7.95 60
65432.78 3000 9.375 84
15937.07 1000 15.66 48
```

Sample output:

```
$   587.07 $ 37224.20 $  6236.79
$  1016.41 $ 88378.44 $ 22945.66
$   420.72 $ 21194.56 $  5257.49
```

10. Marek

Program Name: Marek.java **Input File: marek.dat**

Your parents have asked you to help your younger brother, Marek, with his homework. He gave you 10 bucks to do it for him while he goes to the movies with his friends. Write a computer program to do all his homework problems.

Input: The input will begin with a single integer, `t` (1 <= `t` <= 100) denoting the number of test cases to follow. Each of the `t` test cases will be composed a single line with an infix expression made up of integers and `+`, `-`, `*`, `/`, `(`, `)`. Use standard order of operations to evaluate the expressions, all integers and operators on the line will be separated by spaces.

Output: For each of the `t` test cases, print out the answer to the given infix expression. Use integer division, i.e. ½ = 0. If division by 0 ever occurs, print out "`Infinity.`"

Sample input:

```
4
1 + 2 - 1
4 * 7 - 3
3 / 2 + 1
4 + 2 * 3
```

Sample output:

```
2
25
2
10
```

11. Priyanka

Program Name: Priyanka.java **Input File: priyanka.dat**

Priyanka is an elementary school teacher working with her students on their spelling words. The students have already learned about vowels and consonants, and they are also able to alphabetize letters. Priyanka noticed one very special type of word that she thought was pretty cool. These are words with all of the vowels at the front of the word and all the consonants at the end. She also noticed that sometimes, the vowels were sorted within themselves and the consonants were sorted as well.

Then she discovered what she called "Woo-Hoo" words. In it, the vowels {A,E,I,O,U} are sorted in reverse order at the start of the word and the consonants were sorted in ascending order at the end of the word.

Here are some examples: OAKS, EAST, OOPS, EASY, A, OF

To have a little fun in class, any time one of her students spotted a spelling word like this, they would shout "WOO-HOO!!!" Thus they are called Woo-Hoo Words.

Please write a program to test a word to see if it is worthy for a shout of "WOO-HOO!!!"

Input: Input will consist of an integer N, the number of test cases. The number of test cases will be in range [1,20]. Each subsequent line will contain one string consisting only of upper-case letters with no spaces and no punctuation marks. Each string will have a length in the range [1,32]

Output: Each line of output will consist of the letters of the input string rearranged. The vowels {A,E,I,O,U} will come first in reverse alphabetical order followed immediately by the consonants in alphabetical order. There is not a space separating those two groups. If a word is a "woo-hoo" word, then "WOO-HOO" will be written before the output word with one space separating it from the word.

Sample input:

```
5
UNIVERSITY
HELLO
ABCDEFGHIJ
EACH
I
```

Sample output:

```
UIIENRSTVY
OEHLL
IEABCDFGHJ
WOO-HOO EACH
WOO-HOO I
```

12. Sunny

Program Name: Sunny.java **Input File: sunny.dat**

Sunny is only a freshman and has been really excited about taking UIL programming. She has been looking at the dual-credit courses that are available through her school and hopes to start college with enough credits to reduce her in-person college experience to 3 years. She discovered the Texas Common Course Numbering System (TCCNS) but the data she found was not very user-friendly. She needs your help to generated a sorted list of courses. (Note: the data here is less than 10% of the real data!)

Input: An unknown number of lines with each line containing a 9-character course code followed by a single space and the course title. There are no punctuation or special characters except for the dash in the course codes.

Output: Alphabetical (A…Z) list by course title with course codes following the titles in a set of parentheses, separated by a single space.

Sample input:

```
PHYS-2125 UNIVERSITY PHYSICS I LAB
MATH-1342 ELEMENTARY STATISTICAL METHODS
COSC-2325 COMPUTER ORGANIZATION
COSC-1336 PROGRAMMING FUNDAMENTALS I
PHYS-2325 UNIVERSITY PHYSICS I
COSC-1301 INTRODUCTION TO COMPUTING
COSC-1315 INTRODUCTION TO COMPUTER PROGRAMMING
COSC-1320 C PROGRAMMING
MATH-2313 CALCULUS I
COSC-1337 PROGRAMMING FUNDAMENTALS II
```

Sample output:

```
C PROGRAMMING (COSC-1320)
CALCULUS I (MATH-2313)
COMPUTER ORGANIZATION (COSC-2325)
ELEMENTARY STATISTICAL METHODS (MATH-1342)
INTRODUCTION TO COMPUTER PROGRAMMING (COSC-1315)
INTRODUCTION TO COMPUTING (COSC-1301)
PROGRAMMING FUNDAMENTALS I (COSC-1336)
PROGRAMMING FUNDAMENTALS II (COSC-1337)
UNIVERSITY PHYSICS I (PHYS-2325)
UNIVERSITY PHYSICS I LAB (PHYS-2125)
```