1. Armstrong Numbers

Program Name: Armstrong.java **Input File: armstrong.dat**

An Armstrong number, also known as a narcissistic number, is a number that is equal to the sum of its own digits each raised to the power of the number of digits.

For example, 153 is an Armstrong number:

- 153 has 3 digits.
- Raise each digit to the third power.
- Sum the results.
- $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$

Input

- The first line will contain a single integer `n` that indicates the number of lines that follow.
- Each of the following `n` lines will consist of a single integer.
- All integers will be greater than or equal to 0 and less than 200,000,000.

Output

For each integer input, print “`ARMSTRONG`“ if the integer is an Armstrong number or print “ `NOT AN ARMSTRONG`“ if the integer is not an Armstrong number.

Example Input File

```
5
153
222
10000
5
1634
```

Example Output to Screen

```
ARMSTRONG
NOT AN ARMSTRONG
NOT AN ARMSTRONG
ARMSTRONG
ARMSTRONG
```

2. Bingeaux was his name-eaux

Program Name: Bingeaux.java **Input File: bingeaux.dat**

As editor for a baby-naming dictionary, you wish to write a program that will determine variants for a given name. You know some common name variant rules:

Original name containing	**Replaced with**
`ph`	`f`
`ai` not at end of name	`ay`
any letter appearing exactly twice in a row (but not more than twice consecutively)	a single occurrence of that letter
`o` at end of name	`eaux`

Note that one and only one change is made to the original name to determine a variant. E.g., the variants of "jennifer" would be "jenifer" but not "jennipher" nor "jenipher". However, the name "jennny" or "jennnny" would have no variants, as there are 3 n's and 4 n's respectively that appear consecutively. The two variants of "kailail" are "kailayl" and "kaylail" in alphabetical order.

Input

- The first line will be a single integer `n` that indicates the number of data sets in the input.
- Each data set will consist of a single line containing a name of 1-20 lowercase letters.

Output

For each data set in the input, output the original name, followed by an alphabetized list of variants for the given name, with each name and name variant on a separate line.

Example Input File

```
9
joseph
kailail
hai
thibodo
jennifer
timothy
yarrghtheviking
yarrrghtheviking
jarrrghthemanofjello
```

Example Output to Screen

```
joseph
josef
kailail
kailayl
kaylail
hai
thibodo
thibodeaux
jennifer
jenifer
timothy
yarrghtheviking
yarghtheviking
yarrrghtheviking
jarrrghthemanofjello
jarrrghthemanofjelleaux
jarrrghthemanofjelo
```

3. Birdman

Program Name: Birdman.java   Input File: none

Cliff, a famous bird watcher, is better known as the Birdman. Birdman needs a logo for his bird-watching shirts. You are to write a program that will create this logo for his shirts.

Input
No input.

Output
Print this logo exactly as it appears below.

Example Input File
There is no input for this problem.

Example Output to Screen
```
************************
************************
************************
***                *****
***  ************** ****
***  *************** ***
***  *************** ***
***  *************** ***
***  *************** ***
***  *************** ***
***  *************** ***
***  *************** ***
***  *************** ***
***  ************** ****
***                *****
***  ************** ****
***  *************** ***
***  *************** ***
***  *************** ***
***  *************** ***
***  *************** ***
***  *************** ***
***  *************** ***
***  *************** ***
***  ************** ****
***  ************* *****
***               ******
************************
************************
************************
```

4. Convoy at Cowhouse Creek Bridge

Program Name: Convoy.java Input File: convoy.dat

In a remote area of Fort Hood, there is a single lane bridge that spans Cowhouse Creek. Besides being single lane, there is a load limit of 42 tons for this bridge. Frequently, a convoy of military vehicles has to cross the bridge. The bridge gate keeper must divide the convoy into groups of one or more vehicles so that the load limit for the bridge is not exceeded and that there are as few groups as possible. You may assume the bridge is long enough to hold an entire group at one time and all of one group will completely cross the bridge before another group is allowed to start crossing the bridge.

Input
The first line of input will contain a single integer `n` that indicates the number of convoys to cross the bridge. Each of the following `n` lines will contain an integer `v`, $1 \le v \le 10$, indicating the number of vehicles in the convoy followed by a space and the weights `w`, in tons, of the vehicles in the convoy. The weights, $1 \le w \le 42$, will be separated by a space. There will be no more than 10 vehicles in a convoy.

Output
For each convoy, you will print the minimum of groups necessary to get all vehicles in the convoy across the bridge.

Example Input File
```
3
10 15 5 2 6 8 15 10 40 3 6
10 5 3 6 8 9 10 3 14 40 35
8 21 30 20 17 16 27 8 42
```

Example Output to Screen
```
3
4
5
```

5. Emirp

Program Name: Emirp.java **Input File: emirp.dat**

An *emirp* number is a prime number that forms a different prime number when reversed. Palindromic prime numbers are excluded. You will be given a range of numbers and you are required to list all the *emirps* in that range (inclusive).

Input
The input is two or more lines with two positive integers `a` and `b`, `a` < `b`, separated by a single space on each line.

Output
For each line of input, you will print, on a single line and separated by a single space, the list of emirps in the given range in ascending order. Print “`NONE`“ if there are no *emirp*s in the given range.

Example Input File
```
10 20
10 40
18 29
```

Example Output to Screen
```
13 17
13 17 31 37
NONE
```

6. Full House

Program Name: FullHouse.java   Input File: fullhouse.dat

There are many games that are played with 5 standard six-sided dice with numbers one through six on a side. In a poker type game, one of the hands a player can have is a full house. A full house consists of three dice that have the same number and the remaining two dice that have a different same number. For example, `3 3 3 5 5` is a full house. Ties are broken first by numbers on the three matching dice, with one being the lowest and six being the highest. If the three dice match in two or more hands, the tie is broken by the two matching dice, again with one being the lowest die and six being the highest. If the tie is not broken by one of these two methods, a tie exists.

For the purpose of this program, you will be given the outcomes after each of three players has rolled their dice. You are to determine if any of the players has a full house and, if more than one player has a full house, which player or players have the highest hand.

Input
The first line of input will contain a single integer `n` that indicates the number of games to be played. For each game, there will be three lines, one for each player in the game, and each will have a player's first name followed by the five dice the player had after the third roll. There will be a single space following the player's name and the dice will each be separated by a space.

Output
For each game

- If there is exactly one person with a full house, you will print the name of that person.
- If there is more than one person with a full house, you will print the name of the person with the highest full house. For example, a roll of `5 5 5 2 2` would beat a person with a roll of `4 4 4 6 6`.
- If there is more than one person with the same three matching dice, you will check the two matching dice to see which hand is the winner and print the name of the winner. For example, a roll of `5 5 5 2 2` would beat a person with a roll of `5 5 5 1 1`.
- If there is more than one person with the same full house, there is a tie and you will print, on a single line, the word "`TIE`" followed by the names of the players who were tied in alphabetical order and separated by a space.
- If no hands contain a full house, you will print "`NO WINNER`".

Example Input File
```
4
MARY 4 4 4 4 3
JOAN 4 4 4 3 3
ART 3 3 3 2 1
RON 4 4 4 4 1
ALEX 4 3 3 1 4
CHEZ 2 6 6 6 3
AMY 4 5 4 4 5
REX 4 5 4 5 4
RICK 2 2 4 2 3
GEORGE 5 1 5 1 1
JOHN 1 5 1 5 1
DICK 1 1 2 2 2
```

Example Output to Screen
```
JOAN
NO WINNER
TIE AMY REX
DICK
```

7. Little Miss Muffet

Program Name: LittleMM.java   Input File: littlemm.dat

Write a program that will print all of the data in the data file.

Input
One or more lines of data.

Output
Print the input file exactly as it appears in the input file.

Example Input File
```
LITTLE MISS MUFFET
SAT ON HER TUFFET,
EATING HER CURDS AND WHEY.
ALONG CAME A SPIDER,
WHO SAT DOWN BESIDE HER
AND FRIGHTENED MISS MUFFET AWAY.
```

Example Output to Screen
```
LITTLE MISS MUFFET
SAT ON HER TUFFET,
EATING HER CURDS AND WHEY.
ALONG CAME A SPIDER,
WHO SAT DOWN BESIDE HER
AND FRIGHTENED MISS MUFFET AWAY.
```

8. Love

Program Name: Love.java   Input File: none

Cindy wants to print the statement "`CINDY LOVES THE UIL PROGRAMMING CONTEST`" twelve times. You are to write a program to do this for her.

Input
No input.

Output
Print the above statement twelve times exactly as it appears below.

Example Input File
There is no input for this problem.

Example Output to Screen
```
CINDY LOVES THE UIL PROGRAMMING CONTEST
CINDY LOVES THE UIL PROGRAMMING CONTEST
CINDY LOVES THE UIL PROGRAMMING CONTEST
CINDY LOVES THE UIL PROGRAMMING CONTEST
CINDY LOVES THE UIL PROGRAMMING CONTEST
CINDY LOVES THE UIL PROGRAMMING CONTEST
CINDY LOVES THE UIL PROGRAMMING CONTEST
CINDY LOVES THE UIL PROGRAMMING CONTEST
CINDY LOVES THE UIL PROGRAMMING CONTEST
CINDY LOVES THE UIL PROGRAMMING CONTEST
CINDY LOVES THE UIL PROGRAMMING CONTEST
CINDY LOVES THE UIL PROGRAMMING CONTEST
```

9. Odd Numbers

Program Name: OddNumbers.java **Input File: oddnumbers.dat**

The 3nd grade students in Mrs. Johnson's class are learning about odd and even numbers. To reinforce the concept, Mrs. Johnson had the students write the odd numbers an odd number of times per line like so:

```
1
3 5 7
9 11 13 15 17
19 21 23 25 27 29 31
```

Given the number of odd numbers in a line, you will write a program that will give the sum of the last 3 numbers of that line. For example, given the number 3, you know you will be looking at the second line above, since that is the line with 3 odd numbers. Your program adds those 3 numbers.

Input
The input is a sequence of lines, each having a single odd number `n`, 1 < `n` < 100, where `n` denotes how many odd numbers would be written on the line you are interested in.

Output
For each value of `n`, you will print the sum of the last 3 odd numbers in that line.

Example Input File
```
3
5
7
```

Example Output to Screen
```
15
45
87
```

10. Problems For You And Me

Program Name: Problems.java   Input File: problems.dat

You are spending Martin Luther King day at home with a friend, and you really want to go for a hike to enjoy the unseasonably warm January afternoon. However, before you can take your stroll, you need to come up with a good programming contest problem for one of the UIL hands-on contests.

You know that the key to a good set of problems is an equal number of hard, moderate, and easy problems. For this contest, there are already 11 written, and you have to come up with the 12th. Knowing the difficulty rating of all the existing problems, determine the appropriate difficulty rating for the final problem (or note that, given the existing problems, there is no way to arrive at a "good set").

Input
The first line will contain a single integer `n` that indicates the number of data sets in the input. Each data set will consist of a single line with 11 characters (no spaces) representing the difficulty levels of the 11 problems in the existing problem set you are working with:

H – hard
M – medium
E - easy

Output
For each data set in the input, output a single line. For data sets where it is possible to meet the criteria for a "good set", display either "`hard`", "`medium`", or "`easy`", indicating the difficulty of the problem you need to write. For other data sets, simply display "`Oh well, that's what I get for procrastinating`".

Example Input File
```
3
EEEEMMMMHHH
EEMMHHEMHMH
HHHHHHHHHHH
```

Example Output to Screen
```
hard
easy
Oh well, that's what I get for procrastinating
```

11. Rank Arrays

Program Name: Rank.java   Input File: rank.dat

The rank of an element in an array of integers can be determined by its position in the array and the value of the other elements in the array. The rank of an element `e` in an array of size `n` is determined by adding the position `p`, 1 <= `p` <= n, of the element in the array to the number of array elements to the left of `e` that are less than `e`.

For example, consider the array containing the integers: `6 9 3 8 2 10 1`. The rank of the target element 8 is 6 because its position is 4 and there are 2 elements in the array to the left of 8 that are less than 8.

You are to determine the rank of a target element in a given array.

Input
The first line of input will contain a single integer `n` that indicates the number of lines to follow. Each the following `n` lines will contain a list of `m` integers separated by a single space. The first `m-1` integers in the list are unique elements of an array. The last integer in the list is the target element. The target element will be contained in the array.

Output
For each array, you will print the rank of the target element.

Example Input File
```
3
6 9 3 8 2 10 1 8
14 -3 4 6 9 10 -2 7 0 9
5 11 21 8 7 -2 -7 -3 1 9 30 3 -5 4 -4 6 9
```

Example Output to Screen
```
6
8
17
```

12. Strong Passwords

Program Name: Strong.java   Input File: strong.dat

The technology department requires that all teacher passwords be "strong" passwords. For a password to be considered "strong", it must:

- Contain at least one uppercase letter or a special character
- Special characters are: `! @ # $ % ^ & * ( ) _ + =`
- Contain at least one digit: `0 1 2 3 4 5 6 7 8 9`
- Must be at least 8 characters long
- Can contain only upper case and lower case alphabetic characters, special characters as defined above, and digits

It is your job to write a program to see if a given password is a "strong" password.

Input
The first line of input will contain a single integer `n` that indicates the number of passwords to follow. Each of the following `n` lines will contain a single string of characters with no spaces that represents one password.

Output
For each password, on a single line print "`STRONG`" if the password meets the criteria listed above or print "`INVALID`" if it does not.

Example Input File
```
4
MARY_HADaLITTLElamb
MARY_HAD1LITTLElamb
{RogerRabbit2}
R0gerR@bbit
```

Example Output to Screen
```
INVALID
STRONG
INVALID
STRONG
```