2014 Consortium for Computing Sciences in Colleges
Programming Contest
Saturday, November 8th
College of Charleston
Charleston, SC

There are eight (8) problems in this packet. Each team member should have a copy of the problems. These problems are NOT necessarily sorted by difficulty. You may solve them in any order.

Remember input/output for the contest will be from `stdin` to `stdout`. `stderr` will be ignored. Do not refer to or use external files in your source code. Extra white space at the end of lines is ignored, but extra white space at the beginning or within text on a line is not ignored. An extra blank line of output is ignored, but blank lines at the beginning or between lines of text are not ignored.

Have A Lot Of Fun & Good Luck! ☺

Problem 1. Royal Tweet

Problem 2. Cypher

Problem 3. Ken Ken

Problem 4. One Extra Digit

Problem 5. Recycle Calendar

Problem 6. Powerful Number Too

Problem 7. Eight Days a Week

Problem 8. Zeller's Congruence

Problem 1

Royal Tweet

On October 24th, 2014, Britain's Queen Elizabeth II dipped a toe into 21st-century communications when she posted her first tweet. Signing herself Elizabeth R., for regina or queen, she welcomed visitors to a new Information Age gallery focused on the evolution of modern communications at the Science Museum in London. The inaugural tweet (shown above) was posted to the official British monarchy Twitter feed, which has more than 700,000 followers.

Your job is to write a program to scan a given number of tweets for how many mention the at symbol to the British Monarchy, `@BritishMonarchy`. The username following the `@` symbol in Twitter is case insensitive.

Input

The first line of input contains a single integer n, ($1 \leq n \leq 1000$), which is the number of tweets that follow. Each tweet consists of a single line of input containing a string. Each string is of length n, ($1 \leq n \leq 140$). All tweets are separated by a single blank line.

Output

Output the total tweets that contain the substring `@BritishMonarchy` (regardless of case) exactly in the format below. A given tweet that mentions `@BritishMonarchy` more than once is legal and should not be double counted.

Sample Input

```
4
So excited that we can now tweet The Queen @bRitishMonaRchY!! :-)

@BritishMonarchy #Christmas is 6 weeks away!! #soClose

Russ Rose would like to wish you all a Happy #Thanksgiving. #SNL

@BritishMonarchy's great grandson, heir to throne of @BritishMonarchy
```

Output Corresponding to Sample Input

```
Total Tweets Containing @BritishMonarchy = 3
```

Problem 2

Cypher

In honor of the new movie *The Imitation Game* coming out this month on Alan Turning's World War II code-breaking, you've invented a new cypher for text. The trick is to simply slide your hands over a single key on the keyboard while typing. This results in the letter `q` becoming `w`, `l` becoming `;`, and so on. As an example, the text `hello world` would become `jr;;p ept;f`. Note that because the space bar is so wide, that when you slide your hands over, the spaces don't change. Your task is to implement a program to cypher and decipher text using this format.

Input

The first line of input contains a single integer n, ($1 \leq n \leq 1000$), which is the number of test cases that follow. Each test case consists of a single line of input containing a string. Each string is of length n, ($1 \leq n \leq 80$).

The first letter of the line is either `E` or `D`, meaning encrypt or decrypt the following text. The rest of the line is the text you should encrypt or decrypt (ignore the space after the `E` or `D`). All characters will be lowercase, and reasonable. This means that text that is to be encrypted will only have alphabetic characters, and text that is to be deciphered will have only alphabetic characters, excluding `q`, `a`, and `z` but potentially have `[`, `;`, and `,` since these decipher to `p`, `l`, and `m` respectively. Each line of input will always be 78 characters or less, excluding the `E` or `D` at the beginning of the line.

Output

For each case, you should print out the deciphered (or ciphered) text on a single line.

Sample Input

```
3
E hello world
D jr;;p ept;f
E asd
```

Output Corresponding to Sample Input

```
jr;;p ept;f
hello world
sdf
```

Problem 3

KenKen

KenKen is a Sudoku-like puzzle where constraints are put on groups of boxes, called cages, which require a certain mathematical property to be true. For instance, two adjacent boxes with the text `11+` at the top would require the contents of the two boxes to sum to 11. Most KenKen puzzlers, when dealing with a 6 x 6 puzzle would instantly jump to this square since there are really only two possibilities on this square: 5, 6 or 6, 5. Note that the order does matter. Furthermore, this would constrain other boxes on the row helping to further solve the puzzle.

Solving a KenKen puzzle is tough, but we're not going to do that. Instead, we want you to count the number of ways you could sum to a given number *n* on a given KenKen row, given all squares in the cage are adjacent and the cage has *k* spaces in it. Furthermore, you can assume the puzzle is $\subset$ by $\subset$ meaning that you can only use numbers up to and including $\subset$.

Note that all numbers used in the sum must be greater than or equal to 1 (like a Sudoku puzzle).

Input

The input will consist of a positive integer *M*, which is the number of cases to follow. The next *M* lines will contain three positive integers *n*, *k*, and *c* in that order each separated by a space. You are guaranteed that ($1 \leq n \leq 100$) , ($1 \leq k \leq 50$), and ($1 \leq c \leq 50$).

Output

Your output for each case should be the integers *n*, *k*, and *c* for that case each separated by a space followed by the number of possible orientations of numbers that meet those constraints.

Sample Input

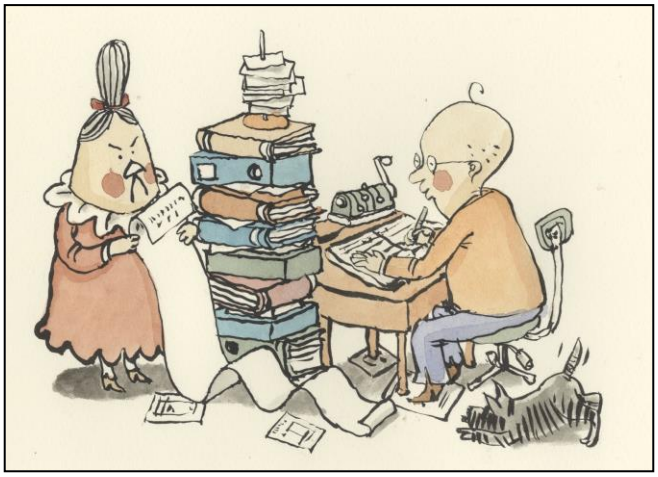
```
2
11 2 6
3 2 6
```

Output Corresponding to Sample Input

```
11 2 6 2
3 2 6 2
```

Problem 4

One Extra Digit



Bob is a great bookkeeper. But there is one mistake he occasionally makes. While adding up a long list of numbers, he sometimes types an extra stray digit in a two-digit number making it an erroneous three-digit number. For example, the number 82 might be mistyped as 862 due to the extra stray digit 6. As a result, his sums are noticeably a little too high. In the case of typing 862 instead of 82, his sum would be off by the difference, i.e. 862 – 82 = 780.

Let's write a program to help Bob find his error. That is to say, if Bob knows by how much his sum is off, there should be some way of figuring out which three digit number(s) could potentially be the two digit number that has an extra digit. Let's assume that when Bob computes a sum, he only makes this error once.

Definitions:

- *A two-digit number is an integer n where 10 ≤ n ≤ 99.*
- *A three-digit number is an integer n where 100 ≤ n ≤ 999.*

An instance of this problem will be a number *d*, representing the difference between the correct and incorrect sum. This number *d* will be a positive integer, and will equal $n_3 - n_2$, where n_2 is a two-digit number and n_3 is a three-digit number, and n_3 can be obtained from n_2 by the appending of an additional digit (0-9) either at the beginning, middle or end of n_2. For example, the digit 6 could be appended into the number 82 to produce the three possible numbers 682, 862 and 826 depending on whether the 6 is put at the beginning, middle or end.

An example instance of this problem is where $d = 780$. Then a possible value for n_3 is 826 and its corresponding value of n_2 would be 82. In this case the 6 was concatenated to the end of n_2 to produce n_3. Note that many possible values of n_3 could exist for a particular value of *d*. In other words, each problem instance could have several solutions. It's also possible to have no solution.

Input

Your program should be written so that it can process several instances (i.e. test cases) of this problem. The first line of the input will give you T, the number of test cases. Assume that $T \geq 1$. Each of the next T lines of the input will give a value for d. In your output, you need to identify the test case number, and all solutions of that test case. Print each solution to a test case on its own line. A solution will be specified in the form $n_3 - n_2 = d$. Print one space on either side of the minus sign, and one space on either side of the equals sign. Also format your solutions so that they are indented by two spaces. See the example I/O below.

Output

If there is more than one solution to a test case, then your solutions must be sorted in ascending order of n_3. Within a test case, do not print the same solution more than once. If a test case has no solution, then you should print "No solution for d = <value of d>" where the value of d appears after the equals sign. As with a bona fide solution, this statement should also be indented two spaces.

Sample Input

```
4
682
390
81
207
```

Output Corresponding to Sample Input

```
Test case 1
  757 - 75 = 682
Test case 2
  430 - 40 = 390
  431 - 41 = 390
  432 - 42 = 390
  433 - 43 = 390
  434 - 44 = 390
  435 - 45 = 390
  436 - 46 = 390
  437 - 47 = 390
  438 - 48 = 390
  439 - 49 = 390
Test case 3
  No solution for d = 81
Test case 4
  229 - 22 = 207
  230 - 23 = 207
```

Problem 5

Recycle Calendar

Well, 2014 will be a fond memory in a couple of months. But don't throw that 2014 wall calendar away! You will be able to use it again in some future year. How long do you have to wait? You will write a program to find out.

Input

The first line of input contains a single integer n, ($1 \leq n \leq 1000$), which is the number of calendars that follow. Each subsequent line will contain a year number y, such that $1776 \leq y \leq 2776$. Each case is to be solved independently. For each given value of y, your program needs to determine the smallest value of z where $z > y$ and the calendar for year z is the same as the calendar for year y.

Output

For each test case, your program must print a sentence of the form:

```
Calendar for y can next be reused in z.
```

Where `y` is the given year number and `z` is nearest year in the future that the calendar for year `y` can be used again. In this problem, we are assuming the conventional Gregorian calendar. In this system, there is a leap year every four years. However, if a year ends in 00, then it must also be divisible by 400 to be a leap year. For example, 1900 was not a leap year, but 2000 was.

To help you solve this problem, note that in a non-leap year, January 1 and December 31 fall on the same day of the week. In a leap year, December 31 falls on the next day of the week following the day of the week for January 1. For example, January 1, 2000 was on Saturday, and December 31, 2000 was on Sunday.

Sample Input

```
5
2011
2012
2013
2014
1988
```

Output Corresponding to Sample Input

```
Calendar for 2011 can next be reused in 2022.
Calendar for 2012 can next be reused in 2040.
Calendar for 2013 can next be reused in 2019.
Calendar for 2014 can next be reused in 2025.
Calendar for 1988 can next be reused in 2016.
```

Problem 6

Powerful Number Too

You are to efficiently compute the number of powerful numbers that are in some interval [a, b]. For purposes of this problem, we will use the following definition:

> A powerful number is a positive integer m such that for every prime number p dividing m, p^2 also divides m. Equivalently, a powerful number is the product of a square and a cube, that is, a number m of the form $m = a^2b^3$, where a and b are positive integers. Powerful numbers are also known as squareful, square-full, or 2-full. Paul Erdős and George Szekeres studied such numbers and Solomon W. Golomb named such numbers powerful.
>
> The following is a list of all the powerful numbers between 1 and 1000:
> 1, 4, 8, 9, 16, 25, 27, 32, 36, 49, 64, 72, 81, 100, 108, 121, 125, 128, 144, 169, 196, 200, 216, 225, 243, 256, 288, 289, 324, 343, 361, 392, 400, 432, 441, 484, 500, 512, 529, 576, 625, 648, 675, 676, 729, 784, 800, 841, 864, 900, 961, 968, 972, 1000

Given two integers a and b where $a \leq b$, you need to count the number of integers m such that m is powerful and $a \leq m \leq b$. For example, for the pair (4, 36), the count is 8. Note, some counts will be larger than 2^{31}.

Input

The input will consist of pairs of positive integers (a, b), one pair per line, where $0 < a \leq b < 2^{64}$ will be terminated by the pair `0 0` which should not be processed. For each pair, you need to print a statement containing the number of powerful numbers between a and b.

Output

For each pair, you need to print a statement containing the number of powerful numbers between a and b. Output should be formatted exactly as shown below.

Sample Input

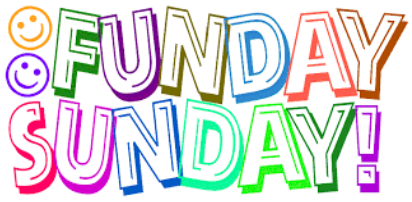
```
11 11
11 17
4 36
1 10000000000000000000
0 0
```

Output Corresponding to Sample Input

```
0 powerful number(s) are between 11 and 11
1 powerful number(s) are between 11 and 17
8 powerful number(s) are between 4 and 36
6869227848 powerful number(s) are between 1 and 10000000000000000000
```

Problem 7

Eight Days a Week



Beginning in 1582 with France, Italy, Portugal, Poland, and Spain, and finishing with Turkey in 1927, the world made the switch from the Julian calendar to the Gregorian calendar. The Julian calendar moved too slowly, introducing an error every 128 years. The Gregorian calendar only loses a day every 3236 years. The problem was too many leap years under the Julian calendar. Instead of every 4 years, the Gregorian calendar omitted century years, with the exception of century years divisible by 400. For example, 1900 was not a leap year, but 2000 was. This was a big improvement.

However, the other issue with the Gregorian calendar is that it is boring and the weekends are too short. Due to the recent phenomenon of "Sunday Funday", the aging but still awesome Sir Paul McCartney has decreed that beginning back with the year 2000, the world should have switched to the "Eight Days a Week" calendar, which includes a new day of the week between Sunday and Monday: Funday. This calendar is called the McCartneyian Calendar.

Your task is to read in a date string and report back which day of the week it falls on under the new Eight Days a Week system. It must work for any date from the year 0001 until 9999. For your reference, January 1, 2000 was a Saturday. Under the Gregorian system, January 3, 2000 would have been a Monday. But under the McCartneyian calendar, it is now Funday. All months have the regular numbers of days and leap years are still decided under Gregorian rules.

Input

The first line of input contains a single integer n, ($1 \leq n \leq 1000$), which is the number of test cases that follow. Each subsequent n lines will contain a date string in the form: `Month DD, YYYY`, where `Month` is one of the twelve months with a capital first letter and the entire month name spelled out with no abbreviation, `DD` is a valid two-digit date of the month as an integer type, and `YYYY` is a valid four-digit year as an integer type.

Output

On a separate line for each test case, output only the day of the week. Capitalize the first letter and spell the whole word: e.g. `Saturday`, `Sunday`, `Funday`, `Monday`, `Tuesday`, `Wednesday`, `Thursday`, `Friday`.

Sample Input

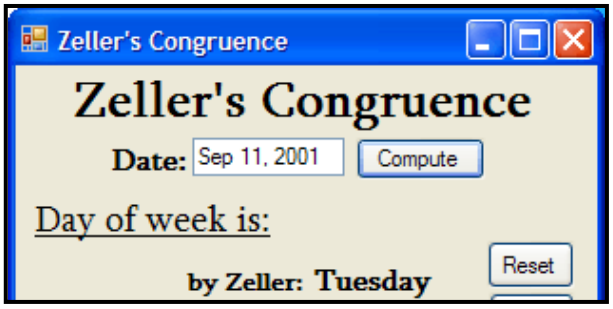
```
4
January 1, 2000
January 3, 2000
December 31, 1999
February 28, 2005
```

Output Corresponding to Sample Input

```
Saturday
Funday
Friday
Wednesday
```

Problem 8

Zeller's Congruence



Zeller's congruence is an algorithm devised by Christian Zeller to calculate the day of the week for any Gregorian calendar date. Zeller's congruence relies on the following quantities: `J` is the century (`19`, for example if the year input is `1998`), `K` is the year within the century (`98`, for example if the year input is `1998`), `m` is the month (where March is 3, April is 4, etc.), `q` is the day of the month. The day of the week is determined by the following formula:

```
h = (q+26(m+1)/10+K+K/4+J/4+5J) mod 7
```

where the results of all divisions are truncated. The value of `h` will lie between 0 (Saturday) and 6 (Friday). Note that the expression a `mod b` yields the remainder produced by `a ÷ b`.

Zeller's congruence assumes that January and February are treated as months `13` and `14` of the previous year; this affects the values `K` and `m`, and possibly the value of `J`. That is, January will never be a 1 and February a 2 (rather they are special cases where January will be 13 of the year prior and February 14 of the year prior). The tricky test case to watch out for is January 1, 2000 in which case you want to make sure your century and year is correct.

Input

The first line of input contains a single integer *n*, ($1 \leq n \leq 1000$), which is the number of test cases that follow. Each test case consists of a single line of input containing a string. Each string is of length *n*, ($1 \leq n \leq 80$). The string will have the form

```
month day, year
```

The month will be one of the twelve words January, February,… The letters in the month may be lowercase, uppercase, or any mixture of the two. The day is a one- or two-digit number. The year is a four-digit number. There may be any number of spaces as follows:

- before the month
- between the month & day
- between the day & year
- after the year

You may assume there is at least one space between the month and day and between the day and year, and you may assume that there is a comma after the day. You may also assume all input is valid.

Output

For each test case, generate one line of output with the converted date. The converted date must have the form:

```
dayOfWeek, month day, year
```

with one space between each of the four outputs. The first letter of the day of week and month must be uppercase, and the remaining letters must be lowercase.

Sample Input

```
5
November 8, 2014
October 31, 2013
OCTOBER     24,      2014
January 1, 2000
september 11, 2001
```

Output Corresponding to Sample Input

```
Saturday, November 8, 2014
Thursday, October 31, 2013
Friday, October 24, 2014
Saturday, January 1, 2000
Tuesday, September 11, 2001
```