2016 Consortium for Computing Sciences in Colleges Programming Contest
Saturday, November 5th
University of North Carolina at Asheville
Asheville, NC

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 and good luck! ☺

Problem 1. Secret Club Names

Problem 2. Check Those Bills!

Problem 3. Pokémon Go

Problem 4. Flatland

Problem 5. Passcodes

Problem 6. The Bookmaker

Problem 7. Aerial Photography

Problem 8. Proper Subset Strings

Problem 1

Secret Club Name

BitCoin Asheville recently started using secret club names for their different chapters who meet throughout Buncombe County. Instead of relying on things like physical features for these names, they have decided to use a permutation of a member's first name in all uppercase. To create a club name, they will first reverse the order of all of the vowels in the member's first name, and then reverse the order of the consonants in the name. For example, for the first name WALDO, the vowels are A and O. Reversing their order gives WOLDA. The consonants W, L, and D are then reversed, giving the club name of DOLWA. You may assume a first name with non-letters will never be used.

BitCoin Asheville has asked for your help in creating club names. They consider the letters A, E, I, O, and U to be the only vowels. Write a program to take a list of member first names and output the corresponding club name for each.

Input

The input will consist of one or more names, one per line. Each name will consist of 1 to 80 uppercase letters. The last line of input will be a line with the word `LAST`. This line should not be processed.

Output

For each line of input generate a line with the original name, followed by a colon (‘`:`’) and the club name.

Sample Input

```
WALDO
MARYANNE
LINUS
MARGARET
GEORGE
SARAH
LAST
```

Output Corresponding to Sample Input

```
WALDO:DOLWA
MARYANNE:NENYARMA
LINUS:SUNIL
MARGARET:TERGARAM
GEORGE:GEORGE
SARAH:HARAS
```

Problem 2

Check Those Bills!

Your money may be worth more money than you think. Even a dollar bill could be worth enough to pay some of those monthly bills. The key is the eight-digit serial number, and whether it seems 'fancy' or special to collectors. When the redesigned $100 bill came out in October 2016, the one with the serial number 00000001 sold for $15,000 on eBay. But, history aside, it's mostly about number patterns. Even $1, $2 and $5 bills can be worth a lot more than face value. A check of eBay shows examples, like a $1 bill fetching $86 for having the serial number 67676767, or another selling for $666 for having the number 98765432.

There are eight patterns of those eight digits that could make you rich quickly. They are, in order of value, **solid** (every digit the same), such as 11111111; **ladder** (counting up or down), such as 12345678; **low**, 00000100 or lower; **high**, 99999900 or higher; **radar** (same backwards and forwards), such as 13466431; **repeater** (second half same as first half), such as 12791279; **seven in a row**, such as 33333335; **seven of a kind**, such as 35333333. Your job is to write a program to detect the presence of any of these special patterns in the serial number of a bill. Check each bill for the pattern of highest value. For example, 00000000 is worth more as a solid than it is as a low.

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. For each serial number being tested, the input will be a single line containing one valid 32-bit integer.

Output

For each serial number input, output it, followed by a colon (':') and the pattern of highest value detected. If no pattern is detected, output `no pattern`. Format output as shown below.

Sample Input

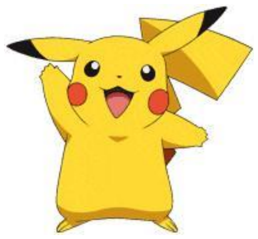
```
10
98765432
66666666
45555555
99909999
99999999
62936293
07344370
83398540
00000100
01234567
```

Output Corresponding to Sample Input

```
98765432:ladder
66666666:solid
45555555:seven in a row
99909999:seven of a kind
99999999:solid
62936293:repeater
07344370:radar
83398540:no pattern
00000100:low
01234567:ladder
```

Problem 3

Pokémon Go



Ash has been playing a lot of Pokémon Go lately. I mean, A LOT. He has become addicted to the gaming app that has taken the world by storm in 2016. While gaming, you've noticed that Ash is actually not very efficient, and wastes a lot of Poké Balls attempting to catch a Pokémon that he has already caught. You even saw him throw ten Poké Balls at a Zubat when there was a Mew 100 feet away! But I digress...

You have decided that you do not want to make that same mistake. After some googling you've found that each Poké Ball has a probability p of capturing a Pokémon that it is thrown at (given your throw is perfect). You are now wondering, how many Poké Balls on average will you need to catch k Pokémon?

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. The next n lines will each have an integer k ($1 \leq k \leq 5000$, the number of pokemon to catch, and p probability ($0.1 \leq p < 1.0$) that a Poké Ball will be successful at catching a Pokémon.

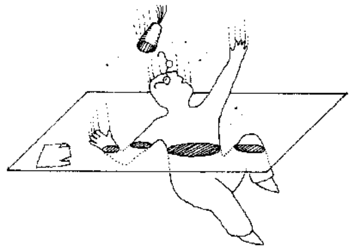
Output
For your output, you should print the average number of Poké Balls it will take to capture the specified number of Pokémon. If the average number of Pokémon is not an integer, you should take the next integer above that value.

Sample Input
```
2
1 0.5
3 0.25
```

Output Corresponding to Sample Input
```
2
12
```

Problem 4

Flatland



Sir Cull is a highly regarded figure in Flatland politics, but unfortunately faces a certain amount of social anxiety. You see, Sir Cull happens to be a circle and due to his rotund physique is not very nimble. Sometimes in his daily strolls he accidentally bumps into other citizens of Flatland. In particular, he has a nasty habit of inadvertently bumping squarely into his neighbor Mr. E. Lipse. Of course, Mr. Lipse, also a circle, is good-mannered and attempts to see the situation from Sir Cull's side. So, Mr. Lipse suggests using some math to predict their positions so that they no longer collide, which will surely round out Sir Cull's mood.

Here are the basics of the math: Sir Cull is a circle with radius `R1` that starts at a position (`x_c1`, `y_c1`) and moves continuously by (`vx_c1`, `vy_c1`) every second. Similarly, Mr. Lipse is a circle with radius `R2` that starts at position (`x_c2`, `y_c2`) and moves continuously by (`vx_c2`, `vy_c2`) every second. Given this information, Mr. Lipse claims that he can determine if they will ever collide. Can you do the same?

Input

The first line of input consists of a positive number $n < 10$ dictating the number of cases to follow. The next n lines will each have ten floating point numbers, dictating `R1`, `x_c1`, `y_c1`, `vx_c1`, `vy_c1`, `R2`, `x_c2`, `y_c2`, `vx_c2`, and `vy_c2` respectively. You can assume that Mr. Lipse and Sir Cull start far enough apart that they do not overlap in any way at the start, and that any intersection will occur after the start time. You may also assume all floating point values are between -10 and 10 exclusively.

Output

For each line of input you should simply print the time at which Sir Cull and Mr. E. Lipse would collide to three decimal places. If they will not collide, simply print `no collision`.

Sample Input

```
2
1.000 0.000 0.000 1.000 0.000 1.000 3.000 0.000 -1.000 0.000
1.000 0.000 0.000 1.000 0.000 1.000 3.000 0.000 2.000 0.000
```

Output Corresponding to Sample Input

```
0.500
no collision
```

Problem 5

Passcodes

Chip is fairly forgetful and has really been struggling to make a new passcode for his iPhone that is easy for him to remember. April suggests making passcodes that have particularly interesting properties. For instance, one of her suggestions is to make passwords such that every digit is relatively prime to the digit before it. This obviously excludes passcodes that have 0 or 1 in it, but still allow for quite a few passcodes.

As a quick reminder, two numbers are considered relatively prime if they share no common factors other than 1. For instance, 4 and 7 are relatively prime because the largest number that divides both is 1. 4 and 6, however, are not relatively prime because they are both divisible by 2. So a password that would conform to April's structure would be "2374". However, "2437" would not because 4 and 2 are not relatively prime.

Chip is now curious about the security of such a passcode. Since the relatively prime limits passcodes to only the digits 2 through 9, he's worried that there aren't very many passcodes available. Chip has asked you to calculate the number of passcodes available for a given passcode, provided that it has *k* digits.

Input

Input will start with a single integer n, ($1 \leq n \leq 25$), which will indicate the number of cases to follow. Following that are n lines, each with a single integer k, ($1 \leq k \leq 25$), which is the number of characters to allow in the passcode. You can assume there will be at most 25 cases, and the largest value for *k* will be 25.

Output

For each *k*, you should simply print the number of passcodes that adhere to April's rules.

Sample Input

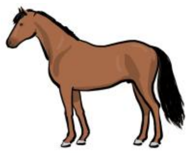
```
3
1
2
3
```

Output Corresponding to Sample Input

```
8
38
200
```

Problem 6

The Bookmaker



Declan's parents are horse trainers. So, naturally, he grew up around horses, and he always had a casual interest in them. However, Declan never liked getting too close to the creatures. He was not interested in riding horses, or training them, or mucking out after them. Instead, he was more interested in financial matters and betting. And he got his chance to test the waters when he got a part-time job with a bookmaking firm at a large racetrack.

And today is Declan's first day on the job. His boss has given him an assignment to test his skill with numbers. Declan needs to calculate the potential payout amount for each bet. He also needs to be able to estimate the firm's profit margin. These are necessary skills in order for Declan to succeed on the job. Let's write a computer program to help him.
The input to the program will be a list of bets. Each bet will have a dollar amount being wagered by the bettor, followed by the odds of the bettor winning. For each bet, the program needs to determine the payout. The payout is the amount of money the bettor will potentially win, based on the wagered amount and the odds. Finally, after processing all of the bets, the program will determine the bookmaker's expected profit, which is always nine percent (9%) of the total amount of money wagered by all bettors. There are three kinds of odds:

- Odds against (e.g. 10 to 1 against), which means the horse is not likely to win.
- Odds on (e.g. 4 to 1 on), which means that the horse is more than likely to win.
- Even money, which means there is a 50/50 chance of the horse winning.

Odds against are so common that the word "against" is omitted.

Input
The first line of input will indicate the total number of bets. The format of the first line is
```
<number> bets
```
where the `<number>` is a positive integer between 2 and 100, inclusive.

Each subsequent line of input will show one bet. The format of these input lines will follow one of the following three patterns:
```
$<number>, <number> to <number>
$<number>, <number> to <number> on
$<number>, even money
```

Each `<number>` will be a positive integer between 1 and 1000, inclusive. The first number on the line is the bet amount. The odds of winning the bet appear after the comma. The word "on" may optionally appear at the end of the line, and this indicates that the horse is more than likely to win. Both the bet amount and odds are used to determine how much money the bettor will win if the horse wins the race.

Output
Here is how to compute the payout amount. For each dollar wagered by the bettor, the payout will be:

- `$(1 + a/b)` if the odds are `a` to `b` against
- `$(1 + b/a)` if the odds are `a` to `b` on
- `$2` if the odds are even money

Your program should print the payout for each bet in the same order that the bets appear in the input. That is, there should be no attempt to sort the output. The payout of each bet should appear this way:

```
Bet <number> payout is $ <money>
```

The `<number>` is the sequential number of the bet, starting with 1. The `<money>` must be rounded to two decimal places. Also note that there needs to be a space on either side of the dollar sign. After processing all of the bets, the program should compute the bookmaker's expected profit, which is 9% of the sum of all of the bet amounts. This result should be written in a sentence as follows:

```
Expected bookmaker profit is $ <money>
```

Here, too, there should be a single space on either side of the dollar sign, and the amount of `<money>` must be rounded to two decimal places. Your output should look like the example output below.

Note that in this program, there is no information about which horses won. In other words, we are not concerned about which bets are successful. The program will calculate payout amounts assuming that all bets win. Of course, in real life, some bets will lose, and the bettor would not receive a payout in those situations, but that is beyond the scope of this program.

Sample Input

```
4 bets
$50, 15 to 8
$100, even money
$20, 7 to 4 on
$5, 100 to 1
```

Output Corresponding to Sample Input

```
Bet 1 payout is $ 143.75
Bet 2 payout is $ 200.00
Bet 3 payout is $ 31.43
Bet 4 payout is $ 505.00
Expected bookmaker profit is $ 15.75
```

Problem 7

Aerial Photography

A new state called Cumberland has been carved out of the frontier. It measures 600 miles from north to south, and 600 miles from west to east. Ariel Hawk, the first governor of the state, wants an aerial survey of Cumberland. She would like to have aerial photographs taken of every square mile of the state. However, there is not yet enough money in the state budget for such an appropriation. Therefore, the governor is accepting proposals for which part of the state should be photographed first, and she is soliciting proposals from the general public.

To make this a manageable task, Governor Hawk will accept proposals that identify rectangular regions of the state to be photographed. For the purpose of this land survey, the state of Cumberland is subdivided into 10,000 townships. Each township is a square measuring 6 miles on a side. The townships are numbered in the form `bb cc`, where `bb` and `cc` are two-digit numbers going from `00` to `99`. The individual numbers `bb` and `cc` function in a manner reminiscent of latitude and longitude, respectively. For example,

- The northwestern most township is numbered `00 00`
- The southeastern most township is numbered `99 99`
- The northeastern most township is numbered `00 99`
- The southwestern most township is numbered `99 00`

Each township is in turn subdivided into square sections measuring 1 mile on a side. There are thus 36 sections per township. The sections of a township are numbered from 01 to 36 according to the following scheme, where the 01 section is in the northeast corner and the 36 section is in the southeast corner:

06	05	04	03	02	01
07	08	09	10	11	12
18	17	16	15	14	13
19	20	21	22	23	24
30	29	28	27	26	25
31	32	33	34	35	36

Consequently, each 1-square-mile section of the state can be uniquely identified by a 6-digit code of the form `aa bb cc`, where `bb cc` is the township number and `aa` is the section number within the township. For example, the northwestern most section in all of Cumberland is numbered `06 00 00`, and the southwestern most section is numbered `31 99 00`.

A rectangular region of Cumberland is simple to define: we specify the northwestern most and southeastern most section numbers that are to be included in the aerial survey. Therefore, a rectangular region is specified by 12 digits of the form `aa bb cc dd ee ff`, where `aa bb cc` is the northwestern most section in the rectangle, and `dd ee ff` is the southeastern most section in the rectangle.

Your job is to assist Governor Hawk in evaluating the requests of rectangular regions. You will be presented with a list of rectangular regions, and you need to write a program that calculates the area, in square miles, of each region.

Input
The input to the program will have this form. The first line of the input will say:
`<n> regions`
where `<n>` is a positive integer less than 100. The next `n` lines of the input will each contain the 12-digit coordinates of a particular rectangular region of the state. The 12 digits will be presented in pairs, with a single space between each pair. There may be trailing spaces after the final pair of digits. In other words, each rectangular region will be specified as:
`aa bb cc dd ee ff` where the numbers `bb`, `cc`, `ee`, and `ff` are between `00` and `99` inclusive; and the numbers `aa` and `dd` are between 01 and 36 inclusive. The ordered triple `aa bb cc` refers to the northwestern most section of a rectangular region, and the ordered triple `dd ee ff` is the southeastern most section. Any number less than 10 will have a padded zero, so that all six numbers on an input line will have two digits. You may assume that the input is valid. In particular, you may assume that, within a given rectangular region, the second section given is not located either north of or west of the first section given. However, it is possible for a rectangular region to be only 1 mile wide in either or both dimensions. For example, `07 07 07 07 07 07`, is a square mile that does exist.

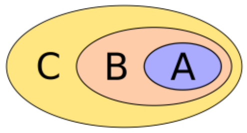
Output
Your output needs to be in this format: There should be *n* lines of output, one line per rectangular region. Give the area of each region in the form:
`Region <i> has <m> square miles.`
where `i` is a sequential integer from 1 to *n*, inclusive, and `m` is also an integer. If `m = 1`, then the word miles should be spelled in the singular, `mile`. The lines of the output should give the area of the rectangular regions in order from 1 to *n*. In other words, do not attempt to sort the output in any way.

Sample Input
```
5 regions
19 47 41 36 47 41
35 47 41 07 48 42
10 50 42 10 50 42
25 49 42 01 50 42
06 00 00 36 99 99
```

Output Corresponding to Sample Input
```
Region 1 has 18 square miles.
Region 2 has 9 square miles.
Region 3 has 1 square mile.
Region 4 has 3 square miles.
Region 5 has 360000 square miles.
```

Problem 8

Proper Subset Strings



A given **`string`**$_1$ is defined to be a *proper subset* of another **`string`**$_2$ if all three of the following conditions are true :

1) **`string`**$_1$ is a substring of **`string`**$_2$.
2) **`string`**$_1$ is either at the beginning of **`string`**$_2$ or it is immediately preceded by a space.
3) **`string`**$_1$ is either at the end of **`string`**$_2$ or it is immediately followed by a space.

For example, the string `"disk"` is a proper subset of the following strings.

- `"disk"`
- `"error on disk"`
- `"error on /dev/disk disk"`
- `"error on disk DSK1"`

The string `"disk"` is not a proper subset of the following strings.

- `"DISK"`
- `"error on disk3"`
- `"error on /dev/disk"`
- `"diskette"`

Your job is to write a program which checks to see if **`string`**$_1$ is a proper subset of **`string`**$_2$.

Input

The first line of input contains a single integer *n*, (1 ≤ *n* ≤ 1000), the number of test cases that follow. Each test case consists of two lines of input containing a string of length *n*, (1 ≤ *n* ≤ 80). The first line of the test case is **`string`**$_1$. The second line contains **`string`**$_2$. You may assume neither **`string`**$_1$ nor **`string`**$_2$ has a blank space at its beginning or ending.

Output

For each test case, output `"string`$_1$`" is a` or `is not` a proper subset of `"string`$_2$`".` `String`$_1$ and `string`$_2$ should be surrounded by quotation marks exactly as formatted below.

Sample Input

```
3
disk
error on disk
disk
error on /dev/disk disk
disk
error on /dev/disk
```

Output Corresponding to Sample Input

```
"disk" is a proper subset of "error on disk"
"disk" is a proper subset of "error on /dev/disk disk"
"disk" is not a proper subset of "error on /dev/disk"
```