Assortment of Practice Problems

In general, these practice problems are a little more difficult.

Program #1: "Can't Fool the Folio"

A print shop has enlisted you to help them make some booklets.

Here is how booklets are made: an original manuscript consists of several 8½x11 pages, and we need to photocopy these pages onto folios. A folio is a double-sized and double-sided sheet of paper (11x17) where we have two normal-sized pages on each side. Thus, one folio can accommodate 4 pages of information for the booklet.

After the pages of the manuscript are copied onto folios, the next step is to collate the folios and staple them (if necessary) into a booklet. To automate the procedure of making folios, the print shop needs to know the sequence of manuscript pages that appear on each folio.

For example, for a 4-page booklet, one folio will suffice. One side of the folio will contain pages 2 and 3, and the other side will contain pages 1 and 4. When this booklet is open you'll see page 2 on the left and page 3 on the right. If you flip the booklet over, you'll see page 4 on the left and page 1 on the right. Booklets with 5 or more pages will require at least 2 or more folios, in which case the folios need to be stapled together.

Input:
The first line of input will give *n*, the number of booklets the print shop needs to make. The next *n* lines of input will contain the number of pages of each booklet we want.

Output:
For each booklet, you need to output the order in which the pages should appear in the folios. The sequence of folios is such that when the booklet is open halfway through the pages, we simply have a stack of unfolded folios. You should output the folios' page numbers starting with the folio on top of the stack, and ending with the folio on the bottom.

For each folio, give the four page numbers in the following order:

- *left* page on the folio
- *right* page on the folio
- the page that appears on the *left* if we flip the folio over
- the page that appears on the *right* if we flip the folio over

If the number of pages in the manuscript is not a multiple of 4, then some of the pages in the booklet will be blank. Represent these blanks in your output with a hyphen. You may assume that the number of pages in a booklet is at least 1. Separate your page numbers with a single space, and always print 2 spaces after a colon. In your output, capitalize the words Booklet and Folio.

Example Input:
```
2 booklets
8 pages
10 pages
```

Example Output corresponding to the above input:
```
Booklet 1:
Folio 1:  4 5 6 3
Folio 2:  2 7 8 1

Booklet 2:
Folio 1:  6 7 8 5
Folio 2:  4 9 10 3
Folio 3:  2 - - 1
```

Program #2

Wrapping the Christmas Presents

Christmas is coming! It's time to start wrapping those presents we've purchased so far. In this problem, you will write a program that determines how much wrapping paper we will need to wrap each Christmas present.

Wrapping paper isn't cheap these days. Therefore, we would like to use as little wrapping paper as necessary for each gift. For this problem, you may make the following assumptions:

- Each gift is in the shape of a 3-dimensional rectangular box.
- The gift will be wrapped with a single piece of wrapping paper.
- Each time we cut off a piece of wrapping paper, it will be a rectangular piece that uses the entire width of the roll.
- You only need enough wrapping paper to barely conceal the gift. In other words, you don't need to have overlap. Any overlap may be cut off and thrown away.
- It is possible that a gift will be too big to be wrapped, because of the insufficient width of the roll and the way we are wrapping the presents as described in this problem.

Donald Duck knows how to use scissors, but he can't remember exactly how to wrap the paper onto the present. Mickey tells him how:

First, place the gift on the piece of wrapping paper, so that its sides are parallel to the sides of the wrapping paper. Then, on each of the four sides of the box, spread the paper onto the box so that it is snug against the gift. Don't tear or twist the paper as you try to cover the gift. Continuing to spread the paper onto the top side of the gift, which should now be completely covered. Finally, apply tape, and you're done!

Input
In the input file, the first line will indicate the width of the roll of wrapping paper. This sentence will be of the form "`Paper is xxx inches wide.`" with `xxx` replaced by a number. The second line will indicate the number of gifts. Each remaining line of input will show the dimensions of the gift, with positive integers separated by `x`'s. The three dimensions will not be given in any particular order.

Output
Output should be formatted as shown below. Note that the word "inches" will always be plural. If it is impossible to wrap a present, then say "`Gift xxx cannot be wrapped.`" using the appropriate gift number.

Sample Input

```
Paper is 30 inches wide.
3 gifts
6x5x4
16x20x18
14x8x12
```

Output Corresponding to Sample Input

```
Gift 1 needs 9 inches.
Gift 2 cannot be wrapped.
Gift 3 needs 40 inches.
```

Program #3

The Finnish Chef

“The Muppet Show” is coming back to television, and you have been hired to help write scripts for the next episode. One of the main characters on the Muppet Show used to be the Swedish Chef, but he has retired. He has a younger cousin, the Finnish Chef, who will take over for him in the kitchen.

To add a Finnish flair to our recipes, you will need to be able to translate English into Mock Finnish. The Mock Finnish language has some features of the real Finnish language. Accomplish the English-to-Mock Finnish translation in these 2 steps.

Step #1: There are 5 prepositions that we are going to transform. Each time you encounter one of these prepositions, you need to delete the preposition, and then add a suffix to the first 2 words after the preposition. You may assume that neither of these next two words is another preposition. The appropriate suffix is selected from the following table. Assume that vowels are the letters ‘a’, ‘e’, ‘i', ‘o’, ‘u’ and ‘y’.

Preposition	Suffix to add if the word ends in a consonant:	Suffix to add if the word ends in a vowel:
in	“issa”	“ssa”
from	“ista”	“sta”
into	“oon”	Duplicate the vowel and add “n”. For example “spice” becomes “spiceen”
on	“illa”	“lla”
onto	“ille”	“lle”

Step #2: Since the Finnish language doesn’t use articles, delete all instances of the words “a” and “the” as well as their inflected forms.

Notes:
- You may assume that the output begins with a number, indicating how many sentences to translate.
- Each subsequent line of input contains a sentence. Your output should also have 1 sentence per line.
- A sentence may begin with a preposition.
- Sentences will end with a period. When you output your sentences, make sure you do not put any whitespace between the last word and the period.
- Be sure to capitalize the first word of the sentence.

Example Input:

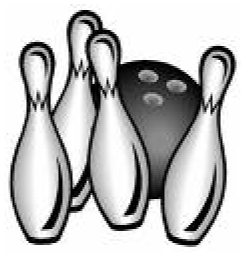
```
5
Put the turkey into the oven.
Now the turkey is in the oven.
Then take the turkey from the oven and put it onto the stove.
Now the turkey is ready to serve.
Yum yum yum, put the turkey into your mouth.
```

Example output corresponding to the above input:

```
Put turkey ovenoon.
Now turkey is ovenissa.
Then take turkey ovenista and put it stovelle.
Now turkey is ready to serve.
Yum yum yum, put turkey youroon mouthoon.
```

Program #4

Strike!



A group of friends decide to go out to a bowling alley. One person in the group is on crutches and cannot play, so he agrees to keep score for everybody. Unfortunately, he is unfamiliar with bowling score sheets, so he only counts and writes down how many pins fall for each roll of the game. We need to write a program that correctly calculates each player's total score.

Here is how scores in bowling are determined. A game consists of 10 frames. In each frame, you have up to two chances to roll the bowling ball to knock over all the pins. If you knock over all the pins on your first roll, this is called a “strike” and the frame immediately ends. Otherwise, on your second roll of the frame you try to knock over the remaining pins. If you manage to do this, this is called a “spare”. If you do not knock over all the pins on your two chances, then your score for the frame is simply the number of pins that did fall. Scoring for strikes and spares is a little more complicated.

If you bowl a strike, then your score for this frame is 10 plus the number of pins that fall on the next two rolls after the strike. (Note that these two subsequent rolls do not necessarily have to be in the same frame: it’s possible to bowl consecutive strikes.) If you bowl a spare, then your score for this frame is 10 plus the number of pins that fall on just the next roll. Thus, making strikes and spares causes some rolls to count more than once.

A game of bowling always consists of 10 frames, so the last frame becomes a special case. This frame may have up to 3 rolls. In the 10th frame, if you bowl a strike, you get 2 bonus rolls to finish out the game. If you bowl a spare in the last frame, you get 1 bonus roll to finish out the game. Note that if you happen to knock over all the pins on your third roll, although this may be a “strike” or a “spare”, the game must still end there. Only one strike or spare in the last frame can earn bonus points. For example, if you bowl two strikes and then a “4” in the last frame, your score for the frame is 10+10+4 = 24. Finally, note that if you get neither a strike nor a spare in the last frame, you do not get the 3rd roll.

You can earn up to 30 points per frame, so the highest possible score in bowling is 300. This is achieved by making 12 strikes, which includes 3 strikes in the last frame.

Input (strike.in):

The first line in this file will be a number *n*, representing the number of players. The next *n* lines contain the score data: each line will begin with a player’s name, which is followed by the number of pins knocked over by that player’s rolls. You may assume that the player’s name is a single word of not more than 14 characters. The numbers will be separated by 1 or more spaces. There may or may not be extra whitespace at the end of the line. You may also assume that the rolls have been tabulated correctly: there will not be any extra or missing values on each line.

Output (strike.out):

Print the list of players and their scores. The list must be sorted by descending order of score. The names should be left justified and the scores should be right justified. As you right justify the numbers, they need to be vertically aligned. Note that the names are up to 14 characters long and we want to leave at least 1 space between names and scores. Thus, the hundreds' digit of the score would appear in the 16th character on the line.

Example input file:

```
2
Emma     5 5  4 6  10   8 2  10   10   10   9 1  10   3 6
Max      1 2  3 4  5 4  3 2  1 0  6 4  0 10 2 8  5 4  9 1 10
```

Example output file:

```
Emma           201
Max             91
```

Program #5

<u>Pay to the Order Of</u>

Your friend runs a business, and has many bills to pay by check. It would be nice if there could be some way to help automate the process of writing out checks. Specifically, your friend would like a program that can convert an amount of money into words, such as "four and 56/100 dollars".

Input file(check.in):

The first line will contain a number *n*, which is the number of monetary values that need to be converted to text. The next *n* lines will each contain a floating number expressed to two decimal places. You may assume that each amount of money is greater than zero and less than 1 billion dollars.

Output file (check.out):

There will be *n* lines of output, one monetary value per line. The number of dollars must be spelled out in English. The general format of an output line is as follows:

<number of whole dollars in words> `and` *<integer cents>*`/100` *<*"dollar" *or "dollars" as appropriate>*

Use a hyphen for words like “twenty-one” and “ninety-nine”. The word “and” will only appear once on the line. There should only be a single space between each word in your output. The integer number of cents will be printed as an ordinary one or two digit integer. The last word on the line will be “dollar” if the amount of money is \$1.00 or less. For amounts of \$1.01 or more, use “dollars”. Use lowercase letters throughout.

Example input:

```
2
81.43
8.44
```

Example output:

```
eighty-one and 43/100 dollars
eight and 44/100 dollars
```

Program #6

Who Scored the Winning Goal?

In team sports, every player has an essential role to play. But when looking back on an exciting game, one of the greatest highlights is the goal that clinched the victory. In this problem, you will write a program that reads a list of goals scored during several different hockey games. For each game, you are to determine its final score and the name of the player responsible for scoring the winning goal.

Note that the “winning goal” is not necessarily the last goal scored in the game. We define the winning goal of a game as follows. If the winning team scored W goals, and the losing team scored L goals, then the winning goal is the (L+1)st goal scored by the winning team. For example, if Los Angeles beat Phoenix by a score of 6 to 3, then LA’s 4th goal is the winning goal. Also note that the order in which the goals are scored does not matter. Phoenix could have scored all of its goals before LA began scoring, or vice versa. The winning goal is always the (L+1)st goal scored by the winning team.

<u>Assumptions</u>: You may assume that there was at least one goal scored in each game, and that no game ended in a tie.

Input format: The first line is of the form

```
<n> games
```

and this indicates the number of games. Each game is introduced by a line of the form

```
Game <n> <team> vs <team>
```

Where <n> is a positive integer, and <team> is a 3-letter abbreviation. Do not assume that the order in which the teams are identified has any bearing on which team won the game.

The remaining lines of a game identify goal events. Each such line has this format:

```
<2 spaces> <team> (<player number>) <player name>
```

These lines will be indented by two spaces, and indicate the name (abbreviation) of the team that scored the goal, followed by the number and name of the scoring player. The player number is given inside parentheses. Note that the player's name may contain several words, but does not extend past the end of the line.

The end of input is signified by an asterisk given at the beginning of a line.

Output: For each game, your program needs to print a line of information, giving the game number, the score, and the name of the player who scored the winning goal. The format must be:

```
Game <n>, <win team> <win score> <lose team> <lose score>, winning goal by <player>
```

Note that your program must determine which team won the game, and output the winning team's abbreviation and score before those of the losing team. At the end of the line, print the player's name but not the player's number. Also note the commas required in the output format.

Example input:

```
2 games
Game 1 TOR vs TBL
  TOR (14) Matt Stajan
  TBL (16) Dixon Ward
  TBL (4) Vincent Lecavalier
Game 2 OTT vs MTL
  MTL (46) Andre Kostitsyn
  MTL (21) Chris Higgins
  MTL (46) Andre Kostitsyn
  OTT (20) Antoine Vermette
  MTL (51) Francis Bouillon
  MTL (79) Andrei Markov
  MTL (54) Mikhail Grabovski
  MTL (6) Tom Kostopoulos
  OTT (15) Dany Heatley
  OTT (28) Martin Lapointe
  OTT (15) Dany Heatley
  OTT (19) Jason Spezza
*
```

Example output corresponding to the example input given above:

```
Game 1, TBL 2 TOR 1, winning goal by Vincent Lecavalier
Game 2, MTL 7 OTT 5, winning goal by Mikhail Grabovski
```

Program #7

Crossing the Mississippi

The Mississippi River runs through the heartland of America. You and your friends are planning a road trip where you will visit several cities. During the trip, you will want to know how many times you will have to cross this mighty river. For example, if your trip contains 4 cities, it is possible that you could cross the river up to 3 times.

But you have not yet finalized your itinerary. You are considering several possible trips. A trip consists of a list of cities. Technically, the trip begins with the first city on the list, and ends with the last city listed. In particular, we do not count any travel from your home to the first city, or from the last city on a list back to your home.

For the purpose of this program, assume the following:

- All cities in Minnesota (MN), Iowa (IA), Missouri (MO), Arkansas (AR), and Louisiana (LA) are located west of the Mississippi River.
- All cities in Wisconsin (WI), Illinois (IL), Kentucky (KY), Tennessee (TN), and Mississippi (MS) are located east of the Mississippi River.
- Your road trip will only contain cities in these states on either side of the river.
- Within a trip, the cities must be visited in the order indicated.

Input format

The first line of the input will say `<t> trips`, where `t` is an integer and 2 ≤ t ≤ 200.

The rest of the input will itemize the individual trips. Each trip will begin with a line of this format:

```
Trip <i> has <c> cities.
```

where `c` is an integer and 2 ≤ c ≤ 200. The trip numbers are numbered consecutively from 1. The next `c` lines will each contain the name of a city. The format of a city name is:

```
<city name>, <state abbreviation>
```

Note that the city name could contain several words, but it will always be followed by a comma, a space and then the 2-letter state abbreviation. Both letters in this abbreviation will be capitalized. Blank space characters could follow the state abbreviation before the end of the line.

Output

Your program needs to indicate the number of river crossings on each trip, one trip per line of output. Each line of your output must be formatted like this:

```
Trip <i> crosses the Mississippi <n> times.
```

Where `i` is the trip number, and `n` is the number of river crossings your program has calculated. Note that if the number of river crossings is exactly 1, then the word "times" at the end of the sentence must be in the singular, "time."

Example input

```
2 trips
Trip 1 has 4 cities.
La Crosse, WI
Effingham, IL
St. Louis, MO
Jackson, TN
Trip 2 has 3 cities.
Mountain Home, AR
Rolla, MO
```

```
Davenport, IA
```

Example output corresponding to the example input given above

```
Trip 1 crosses the Mississippi 2 times.
Trip 2 crosses the Mississippi 0 times.
```

Program #8

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 unnecessary 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 concatenated 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 = 744. 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.

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 >= 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.

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.

Example input

```
4
682
390
81
207
```

Example output corresponding to the example 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
```

Program #9

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.

Please note the following assumptions:

The first line of input will indicate the number of calendars. In other words, this is the number of test cases.

Each subsequent line will contain a year number y, such that 1776 ≤ y ≤ 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.

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.

Throughout 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, it may help to 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.

Example input:

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

Example output corresponding to the example 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.
```

Program #10

Tangent Lines

Mary loves calculus, but there is one type of problem in particular that she would like to practice repeatedly to prepare for her next test. You will write a program to help her.

Consider the graph of $y = x^2$, which happens to be a parabola. Let the ordered pair (a, b) represent a point that lies "below" this parabola. In other words, assume that $a^2 > b$. Then, there exist two lines that pass through (a, b) that are each tangent to the graph of $y = x^2$. Your job is to compute the slopes of these two lines.

An instance of this problem is an ordered pair (a, b), where a and b are real numbers such that $-100 \le a \le 100$ and $-100 \le b \le 100$. You may assume that $a^2 > b$. For each instance, you are to calculate the slopes of the two tangent lines that pass through (a, b) and which are tangent to the graph of $y = x^2$.

The first line of the input will be the number of test cases. Each subsequent line contains a single ordered pair, including parentheses and a comma.

For each test case your program should print a line of this form:

```
Case <test case number>:  The slopes are <slope1> and <slope2>
```

The <test case number> is a positive integer starting with 1. The real numbers <slope1> and <slope2> should be printed to exactly 2 decimal places. The value of <slope1> should be greater than <slope2>. Be sure to print 2 spaces after the colon, and do not put a period at the end of the sentence.

Example Input:

```
3
(0, -9)
```

```
(-6, 1)
(8, 2)
```

Example output corresponding to the above input:

```
Case 1:  The slopes are 6.00 and -6.00
Case 2:  The slopes are -0.17 and -23.83
Case 3:  The slopes are 31.75 and 0.25
```

Program #11

Tournament Seeds

It's time for a big tennis tournament! Three tennis players from your college are competing in a national tournament. Before the first round starts, we are curious to find out who the opponents will be for each of your three players. This program will calculate tournament seeds for the first round.

Let N be the number of players in a tennis tournament. Let us assume that $8 \le N \le 256$. Also, assume that every player in the tournament is assigned a seed, which is a unique number between 1 and N, inclusive.

If N is exactly a power of 2, then it is easy to calculate the seed of a given player's opponent. For any two players who play against each other in their first round, the sum of their seeds is invariant. The value of this invariant is equal to the sum of the highest and lowest seeds. Since N is a power of 2, the highest seeded player is number 1, and the lowest seeded player is number N. Thus, the invariant equals N+1. So, if your seed is s, then the seed of your opponent is $(N + 1) – s$. For example, in a tournament of 16 players, the opponent of the #2 seed is $(16 + 1) – 2 = 15$, in other words the #15 seed.

However, what happens if N is not a power of 2? In this case, the top players get "byes" which means they do not play in the first round. They automatically qualify for the second round of the tournament. The players who do not get byes must play in the first round. The number of players who must play in the first round is 2^K, where K is an integer, and 2^K is the largest power of 2 less than N. Thus, the top $(N – 2^K)$ players get byes in the first round. For the other 2^K players, we can calculate seeds in a straightforward way. Again, for any two players in a matchup, the sum of their seeds is an invariant. And once again, this invariant is the sum of the highest and lowest seeds of the players who must play in the first round.

For example, if the tournament has 10 players, then the top 2 players receive byes. Seeds 3 through 10 must play in the first round. Thus, the invariant is $3 + 10 = 13$. Who is the opponent of the 3rd seed? Since the sum of the seeds is 13, the opponent's seed is $13 – 3 = 10$, which means the #10 seed.

An instance of this problem refers to a single tennis tournament and consists of 4 numbers. The first number is the number of players in the tournament. The remaining 3 numbers are the seeds of players from a particular college. Your program will calculate the seeds of the opponents of each of these 3 players, and indicate if in any case there is a bye instead of an opponent.

The first line of input is the number of test cases (i.e. the number of tennis tournaments). You may assume there will be no more than 100 tournaments. Each subsequent line of input contains the 4 numbers representing a test case.

In your output, you need to identify each tournament by number, starting with Tournament 1. Information about the seeds of a tournament must be indented 2 spaces, and must be in one of two possible forms, as appropriate:

```
  Seed <my seed number> gets a bye.
```

or

```
  Seed <my seed number> plays seed <opponent’s seed number>.
```

Be sure to conclude each sentence with a period.

Example Input:

```
2
48 11 30 48
10 3 4 5
```

Example output corresponding to the above input:

```
Tournament 1
  Seed 11 gets a bye.
  Seed 30 plays seed 35.
  Seed 48 plays seed 17.
Tournament 2
  Seed 3 plays seed 10.
  Seed 4 plays seed 9.
  Seed 5 plays seed 8.
```

Program #12

The Village Gong

In a certain remote village, monks announce the passage of time by striking a gong every ten minutes throughout the day and night. The gong is struck at exactly midnight, and at every ten minute interval after that, at 12:10:00, 12:20:00, etc.

Your job is to figure out how long it will be until the monks will next strike the gong.

An instance of this problem is a string of the form <h>:<m>:<s> that represents the current time in a 12-hour format, where h, m and s are integers and 1 <= h <= 12, 00 <= m <= 59, 00 <= s <= 59. For each instance, your program is to determine the integer number of seconds until the next strike of the gong. If the current time happens to be the precise moment when the gong is being struck, then output 0.

The first line of the input will be the number of test cases. On each subsequent line is a separate test case.

Example input:

```
4
8:11:47
12:05:00
7:19:59
4:50:00
```

Example output corresponding to the above input

```
493
300
1
0
```

Program #13

<u>Aerial Photography</u>

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 `aa bb`, where `aa` and `bb` are two-digit numbers going from 00 to 99. The individual numbers `aa` and `bb` function in a manner reminiscent of latitude and longitude, respectively. For example,

- The northwesternmost township is numbered 00 00.
- The southeasternmost township is numbered 99 99.
- The northeasternmost township is numbered 00 99.
- The southwesternmost 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 northwesternmost section in all of Cumberland is numbered 06 00 00, and the southwesternmost section is numbered 31 99 00.

A rectangular region of Cumberland is simple to define: we specify the northwesternmost and southeasternmost 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 northwesternmost section in the rectangle, and `dd ee ff` is the southeasternmost 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.

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 northwesternmost section of a rectangular region, and the ordered triple `dd ee ff` is the southeasternmost section. Any number less than 10 will have a padded zero, so that all six numbers on an input line will have 2 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.

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 instead be spelled in the singular, mile. Don't forget the period at the end of the sentence. 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.

Example 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
```

Example output corresponding to this 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.
```

Program #14

<u>The Bookmaker</u>

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.

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.

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 2 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 2 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.

Example input:

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

Example output corresponding to the above 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
```