2009 Consortium for Computing Sciences in Colleges Programming Contest

Saturday, November 14th

Roanoke College

Roanoke, VA

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. Each team will receive a small prize upon solving a problem correctly.

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.

Have Fun & Good Luck! ☺

Problem 1

Pay To the Order Of

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
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
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 and token 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.

Sample Input
```
2
81.43
8.44
```

Output Corresponding to Sample Input
```
eighty-one and 43/100 dollars
eight and 44/100 dollars
```

Problem 2

What Time Is It in Paris?

You have just gotten adjusted to our change to Eastern Standard Time here in Roanoke, and now it's time to pack your bags for a weeklong Thanksgiving break trip to Disneyland Paris. Before you head out for the City of Lights, you want to add an application to your phone that will let you quickly know what time it is in Roanoke while you are abroad, and vice versa.

Roanoke is six hours behind Paris, France. So, when it is 12:00 PM here, it is 6:00 PM in Paris. Your task is to write a program that will take as input a time in either Roanoke or Paris, and convert it to the current time in the other city.

Input

The first line will contain a number *n*, which is the number of lines of input which will follow. The next *n* lines will each contain a city (either `Paris` or `Roanoke` spelled exactly as shown). This is followed by an equal sign, and the current time in that city.

The time will be expressed in military time in the format *hh:mm* where there will always be two digits before & after a single colon. Military time is a synonym for 24-hour clock notation, which counts the hours of the day from 00 to 23. So, 1:00 PM in the afternoon is 13:00. There will never be any spaces on the input line. Hours will range from 00 through 23, and minutes will range from 00 through 59.

Output

There will be *n* lines of output. Each line will represent the corresponding time in the other city. Use the format exactly as shown below. Do not use military time in your output. Subtract 12 from all hours in the range from 13 to 23, and print `PM` following all afternoon & evening hours (12 through 23). Use `AM` following all morning hours (00 through 11 in military time), and convert 00 to 12. Do not display any leading zeros for the hour (e.g., use 7:05 not 07:05).

Sample Input

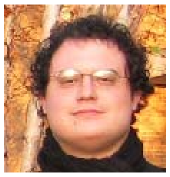
```
5
Paris=06:04
Paris=23:59
Roanoke=00:34
Roanoke=16:50
Roanoke=23:59
```

Output Corresponding to Sample Input

```
The time in Roanoke is now 12:04 AM
The time in Roanoke is now 5:59 PM
The time in Paris is now 6:34 AM
The time in Paris is now 10:50 PM
The time in Paris is now 5:59 AM
```

Problem 3

Cody's Code



Cody has made a reputation for himself as a writer of obscure and difficult-to-understand C++ source code. Secretly, however, Cody has come to realize the importance of good programming style, and that it is impossible to write good programs without adhering to programming conventions, such as meaningful variable naming, indenting, etc.

But he has his reputation to maintain! So what he would like you to do is to write a program that will take Cody's clearly written C++ source code and "Codyize" it for him. Below is a list of what he would like your program to do.

- Rename variables to make their purpose unclear. To make things easier for you, Cody will provide you with the list of variables to be renamed. Each variable should be renamed to have the name `cody` followed by a number starting with `0`. Therefore, the first variable on the list should be renamed `cody0`, the second `cody1`, etc. No original variable name will begin with `cody`. You also may assume that every variable name begins with an alphabetic character, and thereafter contains only alphabetic characters or digits.
- Replace all string and character constants with their hex escapes, e.g. turning string `"cust#1"` into `"\x63\x75\x73\x74\x23\x31"` and `'a'` into `'\x61'`. (You can assume that no string or character constants initially contain escape sequences.)
- Remove all comments. All comments will begin with `//`.
- Replace every consecutive sequence of $n > 0$ *whitespace characters* with a single blank. *Whitespace characters* are defined as either a blank or a tab. Do not remove any newline characters.
- Do not remove blank spaces from within a string literal, e.g., the string "The   blanks" with three blank spaces between the two words should become the string below. `“\x54\x68\x65\x20\x20\x20\x62\x6C\x61\x6E\x6B\x73”`

Input

The input will consist of a single test case. Input will begin with an integer *n*, denoting the number of variables to be renamed, followed by *n* lines each containing a variable name. The rest of the input is the source code.

Output

The output will convert the C++ source code given as input to a "codyized" version according to each of the five specifications above.

Sample Input

```
3
num
name
i
#include <iostream>
// This is a simple program
using namespace std;

int main()
{
    int i, num;
    string name;
    cout << "Enter your name: ";
    cin >> name;
    num = 0;
    for (i = 0; i < 10; i++) {
        num = num * 2;
    }
    cout << num << endl;
    return 0;
}
```

Output Corresponding to Sample Input

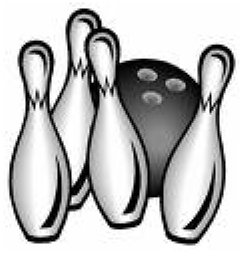
```
#include <iostream>

using namespace std;

int main()
{
 int cody2, cody0;
 string cody1;
 cout << "\x45\x6E\x74\x65\x72\x20\x79\x6F\x75\x72\x20\x6E\x61\x6D\x65\x3A\x20";
 cin >> cody1;
 cody0 = 0;
 for (cody2 = 0; cody2 < 10; cody2++) {
 cody0 = cody0 * 2;
 }
 cout << cody0 << endl;
 return 0;
}
```

Problem 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
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
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. 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.

Sample Input
```
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
```

Output Corresponding to Sample Input
```
Emma           201
Max             91
```

Problem 5

Defining Moment

As a homework assignment, you have been tasked with creating a program that provides the meanings for many different words. As you dislike the idea of writing a program that just prints definitions of words, you decide to write a program that can print definitions of many variations of just a handful of different root words. You do this by recognizing common prefixes and suffixes.

Since your program is smart enough to recognize up to one prefix and one suffix per word, it can process many forms of each word, significantly reducing the number of rote definitions required.

For this problem, you'll be writing the prefix/suffix processing portion of the program.

Valid prefixes and their meanings are:

anti<word>	**against <word>**
post<word>	**after <word>**
pre<word>	**before <word>**
re<word>	**<word> again**
un<word>	**not <word>**

Valid suffixes and their meanings are:

<word>er	**one who <word>s**
<word>ing	**to actively <word>**
<word>ize	**change into <word>**
<word>s	**multiple instances of <word>**
<word>tion	**the process of <word>ing**

Note that suffixes are tied more tightly to their root word and should therefore be expanded last. For example, the word "`unvaporize`" would be expanded through the following steps:

```
unvaporize
not vaporize
not change into vapor
```

Of course, the definitions are not exactly right, but how much polish does the professor expect for a single homework grade?

Input

Input to this problem will begin with a line containing a single integer *n* indicating the number of words to define. Each of the following *n* lines will contain a single word. You need to expand at most one prefix and one suffix, and each word is guaranteed to have a non-empty root (i.e., if the prefix and/or suffix are removed, a non-empty string will remain). Each word will be composed of no more than 100 printable characters.

Output

For each word in the input, output the expanded form of the word by replacing the prefix and/or suffix with its meaning.

Sample Input

```
6
vaporize
prewar
recooking
root
repopularize
uninforming
```

Output Corresponding to Sample Input

```
change into vapor
before war
to actively cook again
root
change into popular again
not to actively inform
```

Problem 6

The Lucky Lottery List

Lotto is a lottery, typically with an accumulating jackpot, in which participants play numbers of their choice in a random drawing. Lenny likes to play the lotto in Lincoln parish Louisiana. In the game, he picks a list of *n* numbers in the range from 1 to *m*. If his list matches the drawn list, he wins the big prize, a lifetime supply of large lemons.

Lenny has a scheme that he thinks is likely to be lucky. He likes to choose his list so that each number in it is at least twice as large as the one before it. So, for example, if *n* = 4 and *m* = 10, then the possible lucky lists Lenny could like are:

```
1 2 4 8
1 2 4 9
1 2 4 10
1 2 5 10
```

Thus Lenny has 4 lists to choose from.

Your job, given *n* and *m*, is to count how many lucky lists Lenny has to choose from.

Input
The first line of input is a single integer which is the number of data sets to follow. All data sets should be handled identically. The next lines, one per data set, contain two integers, *n* and *m*. You are guaranteed that *1 <= n <= 10* and *1 <= m <= 2000* and *n <= m*.

Output
For each data set, print a line like the following:
`Data set` *`i`*`:` *`n m number`*
where `i` is the data set number (beginning with 1), and `number` is the maximum number of lucky lists corresponding to the provided values of `n` and `m`.

Sample Input
```
3
4 10
8 42
1 335
```

Output Corresponding to Sample Input
```
Data set 1: 4 10 4
Data set 2: 8 42 0
Data set 3: 1 335 335
```

Problem 7

An Excel-lent Problem

A certain spreadsheet program labels the columns of a spreadsheet using letters. Column 1 is labeled as “A”, column 2 as “B”, …, column 26 as “Z”. When the number of columns is greater than 26, another letter is used. For example, column 27 is “AA”, column 28 is “AB” and column 52 is “AZ”. It follows that column 53 would be “BA” and so on. Similarly, when column “ZZ” is reached, the next column would be “AAA”, then “AAB” and so on.

The rows in the spreadsheet are labeled using the row number. Rows start at 1. The designation for a particular cell within the spreadsheet is created by combining the column label with the row label. For example, the upper-left most cell would be “A1”. The cell at column 55 row 23 would be “BC23”.

You will write a program that converts numeric row and column values into the spreadsheet designation.

Input
Input consists of lines of the form: *RnCm*. *n* represents the row number, 1 <= *n* <= 300,000,000. *m* represents the column number, 1 <= *m* <= 300,000,000. The values *n* and *m* define a single cell on the spreadsheet. Input terminates with the line: *R0C0* (that is, *n* and *m* are 0). There will be no leading zeroes or extra spaces in the input.

Output
For each line of input (except the terminating line), you will print out the spreadsheet designation for the specified cell as described above.

Sample Input
```
R1C1
R3C1
R1C3
R299999999C26
R52C52
R53C17576
R53C17602
R0C0
```

Output Corresponding to Sample Input
```
A1
A3
C1
Z299999999
AZ52
YYZ53
YZZ53
```

Problem 8

Correlation Analysis

In statistics, *correlation* indicates the strength and direction of a relationship between two random variables. A positive correlation tells us as one variable goes up, so does the other. For example, there is a strong positive correlation between hours studied and GPA. A negative correlation would tell us as one variable goes up, the other goes down. A nice example is outside temperatures and amount of heating bills. One of the easiest ways to measure correlation is with the Pearson correlation coefficient, **ρ**, given by the formula below. **ρ** will always range from a low of -1.0 up to a high of 1.0.

$$\rho = \frac{(\sum_{i=1}^{n} X_i Y_i) - n\overline{X}\overline{Y}}{(n-1)s_x s_y}$$

In the numerator, we have the sum of the product of each X & Y pair. We then subtract from this the product found when multiplying three values: number of pairs (n), average of the X values ($\overline{X}$), and average of the Y values ($\overline{Y}$). In the denominator, we have the product found when multiplying three values : number of pairs minus 1, standard deviation of the X values, and standard deviation of the Y values. Here is the formula for standard deviation.

$$s_x = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \overline{X})^2}{n-1}}$$

Now, let's take a simple example with three pairs. Suppose we have (1, 2), (2, 4), and (3, 6).

X	1	2	3
Y	2	4	6

$\sum_{i=1}^{3} X_i Y_i = (2 + 8 + 18) = 28$ $\quad \overline{X} = 2$ $\quad \overline{Y} = 4$ $\quad s_x = 1$ $\quad s_y = 2$

If we plug these values into our formula, we get a correlation of 1.0.

Input

The first line of input contains a positive integer *n* for the number of pairs. This is followed by *n* X values on line 1, and *n* Y values on line 2. All X & Y values will be floating point values separated by a single space. There will always be at least two pairs, but no more than 20. You may assume there will never be a standard deviation of zero.

Output

Print out your correlation between your two variables in the format shown below. Use two digits following your decimal point, and round up to the nearest hundredth.

Sample Input

```
10
3.07 3.20 2.80 4.00 3.80 3.75 2.25 1.87 3.55 3.34
87.0 72.0 58.0 98.0 95.0 80.0 82.0 72.0 93.0 79.0
```

Output Corresponding to Sample Input

```
The correlation between your X and Y variables is 0.59.
```