January 2011

Computer Science Competition
Hands-On Programming Set

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Point Values and Names of Problems

Number	Name
Problem 1	The Onion
Problem 2	Amortization
Problem 3	City Paths
Problem 4	Floating Bases
Problem 5	Dishes
Problem 6	Tagging
Problem 7	PHP
Problem 8	Significant Digits
Problem 9	Connections
Problem 10	Expletives
Problem 11	Did you mean *recursion*?
Problem 12	The Onion 2

1. The Onion

Program Name: Onion.java

The Onion, "America's Finest News Source," is a 250 year old publication known for its stunning journalistic accuracy and integrity. Output the titles of some of their most popular articles.

Input
No input.

Output
Output the names of the articles, each on separate lines exactly as they appear below.

Example Output to Screen
```
NASA Baffled by Failure of Straw Shuttle
McDonald's Stock Slides as More Consumers Turn to Food
Thomas Edison Invents Marketing Other People's Ideas
Scientists Abandon AI Project after Seeing The Matrix
Bush On Economy: 'Saddam Must Be Overthrown'
```

2. Amortization

Program Name: Amortization.java **Input File: amortization.dat**

When a bank issues a loan, it charges interest, an additional fee for using the bank's money. The amount of interest that is ultimately paid to the bank depends on three variables – the principal (amount being borrowed), interest rate and the loan's term. For instance, a $100,000 loan at 6% over 30 years will actually cost the borrower $215,838.19. Every monthly payment made to the bank is broken up into two portions: an interest and principal payment. However, they are not in equal proportions. Initially, payments made to the bank consist almost entirely of interest. In other words, your principal balance does not really begin to decrease until years into the loan. Using the formulas below, determine the remaining principal of the loan after a certain number of payments.

```
C = Monthly Payment to the bank
L = Number of months of loan
N = Number of monthly payments made so far
P = Principal (Amount borrowed from the bank)
m = Monthly Interest Rate
R = Remaining balance after N payments
```

$$C = P\frac{m(1+m)^L}{(1+m)^L - 1}$$

$$R = (1+m)^N P - C\frac{(1+m)^N - 1}{m}$$

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will list L, N, P, and m as described above on a single line separated by spaces. L, N, and P will be integers. m will be a decimal truncated to four decimal places.

Output

Output R rounded to two decimal places with a leading dollar sign.

Example Input File

```
4
360 5 100000 .0050
240 239 1000000 .0300
564 324 1234567 .0083
12 12 10000 .0070
```

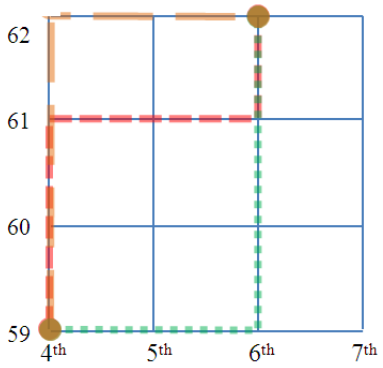
Example Output to Screen

```
$99497.24
$29150.41
$1074910.65
$0.00
```

3. City Paths

Program Name: City.java **Input File:city.dat**

A particular city that I fabricated for this problem uses a grid system to organize its streets and buildings. East to west streets are numbered, such as 59th or 62nd St, with higher numbered streets being further north. North to south streets are also numbered but are known as avenues, such as 4th Ave or 7th Ave, with higher numbered streets being further east. All blocks are the same size.



Clearly, the shortest distance between any two street corners can be found by adding the horizontal and vertical distances between the two corners. For instance, to get from 59th St. and 4th Ave. to 62nd St. and 6th Ave. would require moving 3 (62-59) streets north and 2 (6-4) streets east. However, there are many different paths of that distance. A few different paths are shown in the diagram to the right. Given a starting and end point, how many different optimal paths exist?

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will contain two locations separated by the word "to". Each location with consist of a numbered street and avenue as described above, separated by the word "and". The suffix after the number will be "st", "nd", "rd", or "th". The suffix will be followed by either "St" or "Ave".

The street (east to west) will always appear before the avenue (north to south). Two locations can be up to 14 blocks apart in both directions i.e. an end location can be up to 14 blocks north/south **and** up to 14 blocks east/west. The number of any address can range from 1 to 99. The starting and end points will not be the same.

Output

Output the number of optimal ways to get from the starting to the ending address.

Example Input File

```
4
59th St and 7th Ave to 62nd St and 5th Ave
15th St and 14th Ave to 1st St and 2nd Ave
61st St and 29th Ave to 53rd St and 32th Ave
99th St and 99th Ave to 99th St and 86th Ave
```

Example Output to Screen

```
10
9657700
165
1
```

4. Floating Bases

Program Name: Floating.java **Input File: floating.dat**

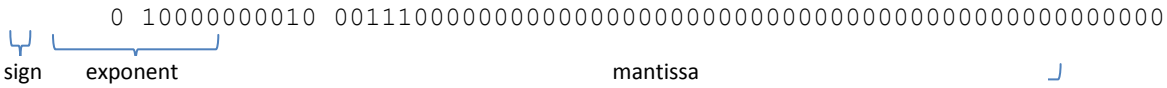
How can fractions be represented with 1s and 0s? With floating point numbers! Write a program to convert a 64 bit binary floating point number into a base 10 decimal. A description of floating points in Java follows.

Description

Like any base, binary numbers can have a decimal point: 1001.11_{bin} is a perfectly valid binary number. Each bit to the left of the decimal point stands for a power of 2 (2^0,2^1,2^2…). Similarly, each bit to the right stands for a negative power (2^{-1},2^{-2},2^{-3}). $1001.11_{bin} = 2^3+2^0+2^{-1}+2^{-2} = 9.75_{dec}$.

But what about the decimal point? It can't be represented as a 1 or 0. To solve this problem, all fractional binary numbers used by computers are represented in scientific notation: `mantissa x 2^exponent`. For example, 1001.11_{bin} would become $1.00111_{bin} \times 2^3$.

To create floating point numbers, the scientific notation is compressed into a block of bits. In Java, doubles are 64 bits long. Binary numbers stored in doubles take the following form:



Sign	A single bit that represents the sign of the number: 0 if a positive number, 1 if negative. In the example above, 0 tells us the number is positive.
Exponent	11 bits that represent the exponent-1023_{dec}. In the example above, $10000000010_{bin} = 1026_{dec}$. $1026_{dec} - 1023_{dec} = 3_{dec}$, representing 2^3. (The subtraction of 1023 allows for negative exponents)
Mantissa	The 52 bit mantissa represents the significant digits of the number. The first bit stands for 2^{-1}, the second 2^{-2}, and so on. Since scientific notation leaves only one option for the bit to the left of the binary point (1), it is implicitly added to the mantissa. In the example above, the mantissa is: $2^{-3} + 2^{-4} + 2^{-5} = .21875_{dec} + 1_{dec} = 1.21875_{dec}$.
Converting to a decimal	Plug each part into the scientific notation formula, `mantissa x 2^exponent`. $1.21875_{dec} \times 2^3 = 9.75_{dec}$.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will contain a 64 bit floating point binary number on a single line. The number will be between 1000000_{dec} and -1000000_{dec}.

Output

Output the floating point number as a decimal, as in "9.75". Round your answer to 2 decimal places.

Example Input File

```
3
01000000000010010010000111111011010101000100010000101101000110000
11000000000100001100110011001100110011001100110011001100110011001101
01000001011001101111000111011010110011111101110011000110001111111
```

Example Output to Screen

```
3.14
-4.20
12029654.50
```

5. Dishes

Program Name: Dishes.java **Input File: dishes.dat**

At a restaurant dishes are cleared and cleaned by bussers, people who assist the waiting staff. They do not do both jobs; a busser either clears or cleans. It is very important that a restaurant has an optimal balance. Too many clearers and the dishes never get cleaned – too many cleaners and dirty dishes remain on the tables. To make matters more complicated, all dishes do not take the same amount of time to clear and clean. Given a set of dishes, the time it takes to process each dish, and number of bussers, determine the optimal split of clearers and cleaners.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow.

- Each data set will start with an integer `t` representing the number of types of dishes. There will be no more than 26 types of dishes.
- The next line will list the individual dishes that need to be processed. Each dish is denoted as a capital letter, which represents its type.
- The next `t` lines will each list 2 integers: the number of seconds it takes for **one** busser to clear or clean that type of dish. For instance, `3 4` would mean it would take one busser 3 seconds to clear the dish or 4 minutes to clean the dish.
- The final line lists an integer `b` representing how many bussers are available to either clear or clean. At least 1 busser must be clearing and 1 must be cleaning. Therefore $b \geq 2$. **Multiple bussers can work on the same dish.** For example, if 2 bussers are cleaning, a dish that would normally take 2 seconds to clean only takes 1 second.

Output

Output the optimal number of bussers clearing and cleaning in the following form: "`x clearing y cleaning: time`". Round the time (in seconds) to two decimal places. There will only be 1 solution.

Example Input File

```
3
2 ABA
A 1 2
B 2 1
5
3 ABAC
A 1 2
B 2 1
C 8 2
10
3 ABABBABBABABABABBABCABABBBBAABCABBABCBACBC
A 7 4
B 6 5
C 2 8
8
```

Example Output to Screen

```
2 clearers 3 cleaners: 3.67
6 clearers 4 cleaners: 3.75
4 clearers 4 cleaners: 114.25
```

6. Tagging

Program Name: Tagging.java **Input File: tagging.dat**

Recently "tags" have become a popular method of organizing content, especially user generated content. For instance, a video on internet bullying might have the tags "backtraced", "cyberpolice", and "consequences". However, a problem with this method is that tags that differ only in punctuation, capitalization, or whitespace are not grouped by the application. Given a list of videos with tags, write a program to determine the popularity of tags that ignores these slight differences.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will contain a list of tags. Each tag will be surrounded in quotes. Tags will be separated by spaces.

Output

Output the tags in descending popularity. Print each tag on a separate line in the following format: `"tag_name" popularity`. If tags have the same popularity, sort them alphabetically in ascending order (digits before A before Z). Tag names should be lower case and free of punctuation and whitespace. These differences should be ignored when determining popularity.

Example Input File

```
6
"bite" "finger" "funny" "bit" "Charlie"
"Charlie" "unicorn" "candy" "funny"
"unicorn" "dentist" "nitrous oxide" "laughing gas!" "bite"
"nitrousoxide" "balloon" "funny" "laughing gas" "dance"
"lego" "8 bit" "stop motion" "unicorn" "dance" "funny"
"lego" "wii" "Mario" "funny" "candy"
```

Example Output to Screen

```
"funny" 5
"unicorn" 3
"bite" 2
"candy" 2
"charlie" 2
"dance" 2
"laughinggas" 2
"lego" 2
"nitrousoxide" 2
"8bit" 1
"balloon" 1
"bit" 1
"dentist" 1
"finger" 1
"mario" 1
"stopmotion" 1
"wii" 1
```

7. PHP

Program Name: Php.java **Input File: php.dat**

PHP, or Hypertext Preprocessor, is a scripting language designed to produce dynamic webpages. One of the most useful features of the language is that the programmer does not need to declare a variable type – a variable can hold any type of data, such as an integer, a decimal, or even a string. This feature is made possible by the interpreter, which automatically determines the variable type. Write an interpreter that will similarly determine a piece of data's type.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set can consist of letters, numbers, and periods.

Output

The most specific type of data – print out "integer", "decimal", or "string".
An integer is a number without a decimal point.
A decimal is a number with a decimal point.
A string is any data with letters or multiple periods.

Example Input File

```
4
314
3.14
FOURTHFRENCH
3.0.0
```

Example Output to Screen

```
integer
decimal
string
string
```

8. Significant Digits

Program Name: Significant.java **Input File: significant.dat**

Significant digits are meant to standardize error across figures. For instance, if I said that I was about 1.6 meters tall, I might be off by a few centimeters. However, if I said Jupiter's radius was about 71,000.0 km, I could be off by hundreds of meters. Using the rules below, count the number of significant digits in a number.

1. All non-zero numbers are significant (1-9)
2. Zeros between non zero digits are significant. 1000.1 has 5 significant figures.
3. Leading zeros are not significant. 0.000017 and 0001.0 only have 2 significant digits. However, 10.001 would have 5 significant digits due to the second rule.
4. Trailing zeros are significant. .17000 has 5 significant figures.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will contain a decimal number. There will never be integers, as the number of significant digits can be ambiguous.

Output

Print out how many significant figures each number has on a separate line.

Example Input File

```
3
3.14
1001.1
0.001202965400000
```

Example Output to Screen

```
3
5
13
```

9. Connections

Program Name: Connections.java **Input File: connections.dat**

A way of visualizing social connections is by representing them as an adjacency matrix. An adjacency matrix is a 2 dimensional 1 0 matrix – that is, consisting only of values 1 and 0. 1 is used to mark a connection. For instance, imagine 4 people, A, B, C and D. A knows B, C knows A, and D knows B. The matrix would look like the one shown below. Notice it is symmetrical along its top left to bottom right diagonal. Also notice that this diagonal is always 1s, since everyone knows themselves.

	A	B	C	D
A	1	1	1	0
B	1	1	0	1
C	1	0	1	0
D	0	1	0	1

Given a list of connections, create an adjacency matrix.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will start with 2 integers `x` and `y` denoting how many people are in the matrix and how many connections are being given. The next line will list `x` names. The first name corresponds with the first row and column in the matrix, the second corresponds with the second row and column, etc. The next `y` lines will list a connection. Each connection will consist of two names separated by a dash. **Self connections should be assumed – A person will always know himself.**

Output

Output the adjacency matrix. Print each row on a separate line, and separate each 1 and 0 with a space. Include an extra blank line between data sets.

Example Input File

```
2
4 3
A B C D
A-B
C-A
D-B
9 10
Xavier Louis George Brittney Doug Jason Sushmitha Petra Yinfei
Xavier-Louis
Louis-George
George-Xavier
Petra-Sushmitha
Brittney-Doug
Brittney-Xavier
Xavier-Petra
Jason-Yinfei
Jason-Louis
Xavier-Sushmitha
```

Example Output to Screen

```
1  1  1  0
1  1  0  1
1  0  1  0
0  1  0  1

1  1  1  1  0  0  1  1  0
1  1  1  0  0  1  0  0  0
1  1  1  0  0  0  0  0  0
1  0  0  1  1  0  0  0  0
0  0  0  1  1  0  0  0  0
0  1  0  0  0  1  0  0  1
1  0  0  0  0  0  1  1  0
1  0  0  0  0  0  1  1  0
0  0  0  0  0  1  0  0  1
```

10. Expletives

Program Name: Expletives.java **Input File: expletives.dat**

Expletives, or offensive words, are often referred to by replacing certain letters with random symbols. For instance, if "Java," was offensive, we could refer to it as "J@v@," although the symbols could change. Given a list of expletives, determine how many of them a censored word could refer to.

Input

The first line will contain two integers: the first represents how many expletives will be given; the second represents the number of data sets, `n`. The next line will list the expletives separated by spaces. The next `n` lines will contain a data set. Each data set will consist of a censored word. The symbols used to block out letters will be limited to "!@#$%^&*()" and "?".

Output

Print out how many expletives each word could refer to. For instance, "c$t" could refer to "cat" or "cut" as the "c" and "t" are in their correct positions, and the dollar sign blocks out some other character. However, "c$t" could not refer to "bat" because the first letters are different.

Example Input File

```
10 5
cat bat dat foo java lava do mom tom bomb
c@t
#@t
@@@
b0^b
%@v@
```

Example Output to Screen

```
1
3
6
0
2
```

11. Did you mean: *recursion*?

Program Name: Recursion.java **Input File: recursion.dat**

In Java, method calls are evaluated via a stack. A method call is pushed onto the stack, and if that method in turn calls other methods, those are also pushed onto the stack. Hopefully, the calls will eventually end. Each method call is then popped off the stack, evaluated, and a value may be returned to the calling method. Write a program to evaluate such method calls.

Input

- The first line will contain two integers: the first represents the number of methods; the second represents the number of method calls.
- For this program each method will follow a simple pattern designed to calculate an integer `x`. The pattern is as follows:
  `methodName(x) {if(x operator val)return other_val; else return other_method(val2 operator other_val2)}`
- `x` will always be calculated using integer arithmetic. Case validation operators will be limited to =,<,and >. = will be substituted for == to make things simpler – it will still be used to test for equality. The case will only ask for a strict comparison of two values; there will never be any complex evaluation necessary.
- Operators used in the else clause will be limited to addition and subtraction.
- Method names will never contain the substrings if, else, or return. They can however be comprised of letters, numbers, and underscores. Each name will be unique.
- Each method call will consist of a method name followed by an integer value enclosed in parentheses.

Output

For each data set, print out the stream of method calls separated by spaces. They should be in the order that they were called. Refer to each method by the name given in its signature. Do not include the method's parameters.
On the next line, print out the result of the method call.

Example Input File

```
6 3
a(x) {if(x=3) return 3; else return a(x+1)}
b(x) {if(x<5) return 1; else return c(x-2)}
c(x) {if(x>7) return -2; else return d(x-3)}
d(x) {if(x=0) return 3; else return d(x-2)}
alpha(x) {if(x<0) return 0; else return beta(x-15)}
beta(x) {if(x<10) return 5; else return alpha(x-5)}
a(0)
b(7)
beta(70)
```

Example Output to Screen

```
a a a a
3
b c d d
3
beta alpha beta alpha beta alpha beta alpha beta
5
```

12. The Onion 2

Program Name: Onion2.java

Print out the Onion's logo.

Input
None

Output
Output the logo below. *Psst…the sample output folder may be helpful…*

Output to Screen
```
....................N...
....................N.N.
....................NN..
.................NNNNNNN
..........NNNNNNNNNNN...
........NNNN.NNN....NN...
......NNN.NNN.....NNNN...
....NNN.NNN......NNNNN...
...NNN.NN......NNN..NN...
...NN.NN......NNN...NN...
..NN.NN......NNN...NNN...
..NNNN......NNN....NNN...
..NNNN......NN.....NNN...
..NNN......NNN....NNNN...
..NNN.....NNN....NNNNN...
...NN.....NN....NNN.NN...
...NN....NNN..NNNNN.NN...
....NN..NNNNNNNNNN.NN....
....NNN.NNNNNNNN..NNN....
....NNNNNN.......NNN.....
...NNNNNNNN...NNNNN......
...NNNNNNNNNNNNNNN.......
.....N.....NNNNN.........
.........................
```