University Interscholastic League

Computer Science Competition

2002 Regional 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 10.
2. Problems have different point values, depending on their difficulty.
3. Several problems have a reference to columns of an input line, such as column 1 or columns 1-3. In these cases column is referring to the character position on the input line. Column 1 refers to the first character position on the line, while columns 1-3 refer to the first three positions on the line.
4. 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.
5. Your program should not print extraneous output. Follow the form exactly as given in the problem.

II. Point Values and Names of Problems

Number	**Name**	**Point Value**
Problem 1	Let Your Fingers do the Talking	5
Problem 2	Well Written	5
Problem 3	Grandpa's Computer	6
Problem 4	It's Only Logical	4
Problem 5	What's in a Name?	6
Problem 6	A Little Light on the Subject	4
Problem 7	Calculate This	5
Problem 8	Make a Note of It	5
Problem 9	Truly Shocking	4
Problem 10	Which way did it go?	6
Total		**50**

Problem 1 | **Let Your Fingers do the Talking** | **5 Points**

Program Name: tenprint.cpp Input File: tenprint.dat

Forensic specialists will tell you that there is a limited set of general shapes that fingerprints follow. Moreover, the patterns are typically not the same on any given person's hand. Even if there are only 3 general shapes (although there are more in reality), you can see that the chances of 2 people having the exact same set of "tenprints" (10 fingers with one print each) is 1 in $3^{10} = 59049$. For the purposes of this problem, we will assume that there are just 3 print shapes.

Shape	Input File Character	Descriptions
Circular	O	Fingerprint pattern forms 1 or more concentric circles.
Left Slant	L	Fingerprint pattern forms "U" shape that opens away from the end of the finger and slants to the left.
Right Slant	R	Fingerprint pattern forms "U" shape that opens away from the end of the finger and slants to the right.
Unknown	?	The fingerprint shape cannot be determined.

Another interesting tidbit is that the specialist can usually determine which finger left the print. Therefore, by comparing known fingerprint shapes at a forensic site against a databank, an investigator can build a list of candidates while excluding most people in the databank. For example, suppose we took a set of prints from a site such that the tenprints are "`LOO?RLLR??`" where the question marks are prints that are unavailable. Then comparing the prints against those in the following databank, we can determine a list of suspects.

Name	Tenprints	Result of comparison
Buck_Wheat	`LOORRLLROO`	Candidate because he matches in all 7 positions collected by the investigator.
Darla_Rascal	`OROROLLROL`	Not a candidate because he does not match in 1 or more of the 7 positions.
Spanky_Boyd	`LOORLLLRLR`	Not a candidate because he does not match in 1 or more of the 7 positions.
Al_Falfa	`RRROLORLLR`	Not a candidate because he does not match in 1 or more of the 7 positions.
Froggy_Mann	`ROORRLLROO`	Not a candidate because he does not match in 1 or more of the 7 positions.
Weaser_Kidd	`RLLRLOLROO`	Not a candidate because he does not match in 1 or more of the 7 positions.
Butch_Bully	`LOOLRLLRLR`	Candidate because he matches in all 7 positions collected by the investigator.
T_Cherlady	`OORLLROLRO`	Not a candidate because he does not match in 1 or more of the 7 positions.
Stimey_Hatman	`LLORRLROLR`	Not a candidate because he does not match in 1 or more of the 7 positions.
Cotton_Hatman	`OORLLRORLO`	Not a candidate because he does not match in 1 or more of the 7 positions.

You can see from the above table, that Buck_Wheat and Butch_Bully are candidates while the others are not. You can eliminate the others because if even 1 position does not match, then they cannot be a suspect. Even Froggy Mann is eliminated by the single incorrect leading print.

Input

Input to your program will consist of a databank followed by a series of forensic tests. The first line of input to your program will contain a single integer ($1 \le C \le 100$) which indicates the number of entries in the databank. The next C lines will each contain exactly 1 databank entry. A databank entry consists of the person's tenprints in columns 1-10 (valid values are "O", "L", and "R" from the table above). There will be a single space followed by the person's name starting in column 12. Names are at most 30 characters long and consist of alphabetic and underscore characters only.

Starting on line C+2, you will find a series of forensic tests. Each forensic test will be found on a line by itself in columns 1-10. Valid characters are "O", "L", "R", and "?" from the table above.

Output

For each forensic test, your program should print a list of candidates to the screen. Candidates should be printed 1 per line. If no candidates are found in the databank, your program should print the message "No Suspects" on a line by itself starting in column 1. After each candidate list (or "No Suspects" message) from a forensic test, your program **must** print a blank line.

Example: Input File

```
10
LOORRLLROO Buck_Wheat
ORORОLLROL Darla_Rascal
LOORLLLRLR Spanky_Boyd
RRROLORLLR Al_Falfa
ROORRLLROO Froggy_Mann
RLLRLOLROO Weaser_Kidd
LOOLRLLRLR Butch_Bully
OORLLROLRO T_Cherlady
LLORRLROLR Stimey_Hatman
OORLLRORLO Cotton_Hatman
LOO?RLLR??
ROLRRLRRLR
???R???R??
```

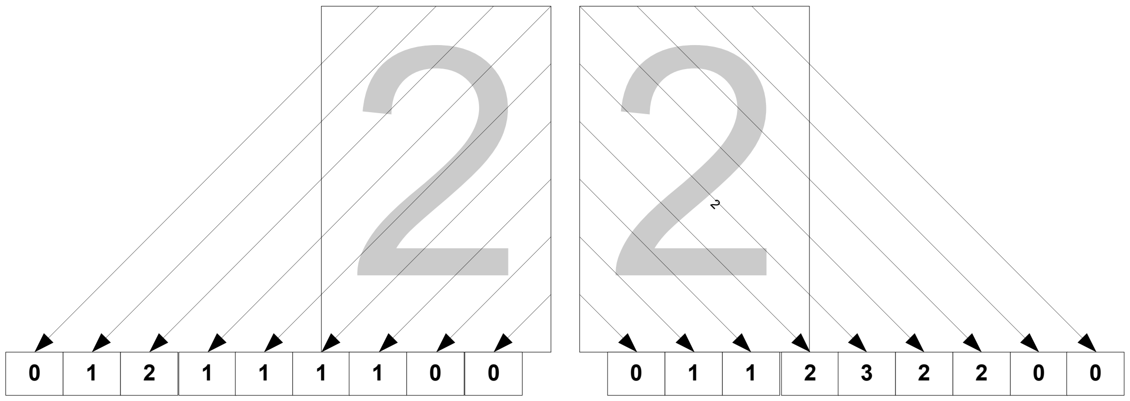
Output to screen

```
Buck_Wheat
Butch_Bully

No Suspects

Buck_Wheat
Darla_Rascal
Spanky_Boyd
Froggy_Mann
Weaser_Kidd
```

Problem 2 — Well Written — 5 Points

Program Name: written.cpp Input File: written.dat

The company that you work for is manufacturing a new character recognition system for a calculator. The user writes numbers on a pad and the samples are normalized to a specific size. Diagonal lines are then drawn in parallel as shown in the figure below and the number of times each line crosses the figure is registered. It is the series of counts that form the "recognition template" that can be used to recognize what digit was written on the pad.



See the table below for a complete list of the recognition templates for the digits 0-9.

When a user writes a character on the pad, the image of the character is normalized to the size of the template and the diagonal line samples are then taken and fed to your program. Your program is to then use the "least squares method" to determine which character was written on the pad. In the least squares method, each count in a sample is subtracted from the corresponding counts in the template and the differences is squared. The squares are then summed up to compute the "fit" for the template. The fit is computed for each template and the digit with the best "fit" (closest to zero) is determined to be the correct one.

For example, if we received an image characterized by the values 011221100,001133200, we could compute the following fit values. Complete work is shown for the first template and the others are left for you to work out.

Digit	Template	Fit Value
0	002222100,001222200	002222100,001222200 Template 011221100,001133200 image under consideration 011001000,000111000 value-to-value corresponding differences squared. 6 is determined to be the fit by summing up the differences squared.
1	001111100,000122100	7
2	012111100,011233300	6
3	011222100,001132200	2
4	000221100,001323110	9
5	012222100,001222110	7
6	002222100,001232200	5
7	011111000,001112210	9
8	002222100,001222200	6
9	002232100,001132200	5

Therefore, you can see that we would determine this character to be a "3" because the it has a fit of 2 (the lowest of any).

Input
Input to your program will consists of a series of keystrokes. Each keystroke will be on a line by itself and will start in column 1. Keystrokes are made up of 9 digits followed by a comma, followed by 9 more digits. The digits in the keystroke are limited to 0-9 and represent the number of times the corresponding diagonal crosses the image.

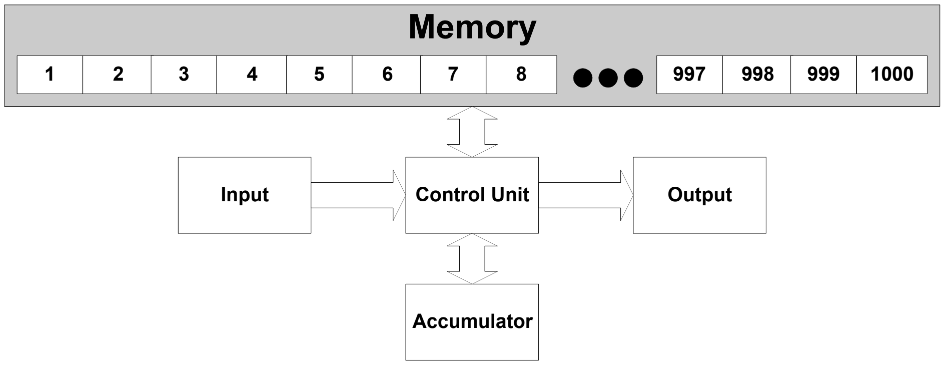
Output
For each image in the input file, your program should compute the "fit" for each digit (0-9) as in the table above. Your program should then print the digit with the best fit on a line by itself in column 1. If there is a tie by 2 or more digits for the best fit, this is considered to be an unrecognizable character and your program should print the word "beep" in columns 1-4 instead of the best fitting digit.

Example: Input File
```
011221100,001133200
001222000,011122100
000111000,001112210
```
Output to screen
```
3
beep
7
```

Problem 3	Grandpa's Computer	6 Points

Program Name: grandpa.cpp   **Input File: grandpa.dat**

The earliest computers had very simple architectures in which there was a simple accumulator on which all computations were performed. The computations (and memory) were very limited in nature. Your program will simulate the earliest versions of the assembly language operations for one of these computers. The following picture illustrates the architecture of the computer you will emulate.



The components of the computer are as follows.

- **Input** – The computer has an input mechanism which reads (from something called paper tape) a stream of sequential values. These values are your computer's program. (Hint: there will be no loops.)
- **Accumulator** –A single "memory" location called a register that is the destination of many operations including all mathematical operations.
- **Memory** – This computer has 1000 memory locations (pretty advanced, huh?) in which it can store 32-bit integers (hey, we can dream).
- **Output** – A sequential stream output to a paper tape that your program will simulate.
- **Control Unit** – The heart of the computer's processing capability. This unit reads your program from Input and executes the instructions it reads.

The computer can execute the following instructions:

Operation	**Instruction Format**	**Function**
Load a value from memory into the accumulator.	LOAD <mem>	ACC ← MEMORY[mem]
Store the value in the accumulator into memory.	STOR <mem>	MEMORY[mem] ← ACC
Data initialization of the accumulator.	DATA <val>	ACC ← <val>
Output the accumulator's contents	OUT	ACC → OUTPUT
Add memory contents to accumulator.	ADD <mem>	ACC ← ACC + MEMORY[mem]
Subtract memory contents from accumulator.	SUB <mem>	ACC ← ACC - MEMORY[mem]
Multiply memory contents by accumulator.	MUL <mem>	ACC ← ACC * MEMORY[mem]
Divide memory contents into accumulator.	DIV <mem>	ACC ← ACC / MEMORY[mem]
Modulus memory contents into accumulator.	MOD <mem>	ACC ← ACC % MEMORY[mem]

Some notes about the operation of the computer.

- All operations are based on 32-bit integers only and any division operation results in an integer result.
- Any divide-by-zero will result in a value of zero being stored in the accumulator. This applies to both the divide and modulus operations.
- The initial contents of the accumulator and all memory locations will be the value zero.
- The input stream is "preprocessed" to ensure that there are no invalid instructions or memory locations for those operations that involve a memory location.

Input
For this problem, your program should begin reading the contents of the input stream which will be stored one instruction per line. Each line of input will contain exactly one of the valid instructions from the table above. The instruction will start in column one. If the operation is the OUT operation, the operation will be immediately followed by the end-of-line character. All other operation will be followed by exactly on space character and then a single integer value and then the end-of-line character. If the operation calls for a memory location, the integer will be between 1 and 1000; otherwise, the integer will be a single signed 32-bit integer. There will be no invalid input in the stream.

Output
Output from your program will be the result of each OUT operation. Each OUT operation will print the integer value of the accumulator on a line by itself to the screen.

Example: Input File
```
DATA 100
STOR 3
DATA 3
STOR 4
LOAD 3
ADD 4
STOR 5
LOAD 3
SUB 4
STOR 6
LOAD 3
MUL 4
STOR 7
LOAD 3
DIV 4
STOR 8
LOAD 3
MOD 4
STOR 9
LOAD 3
OUT
LOAD 4
OUT
LOAD 5
OUT
LOAD 6
OUT
LOAD 7
OUT
LOAD 8
OUT
LOAD 9
OUT
```
Output to screen
```
100
3
103
97
300
33
1
```

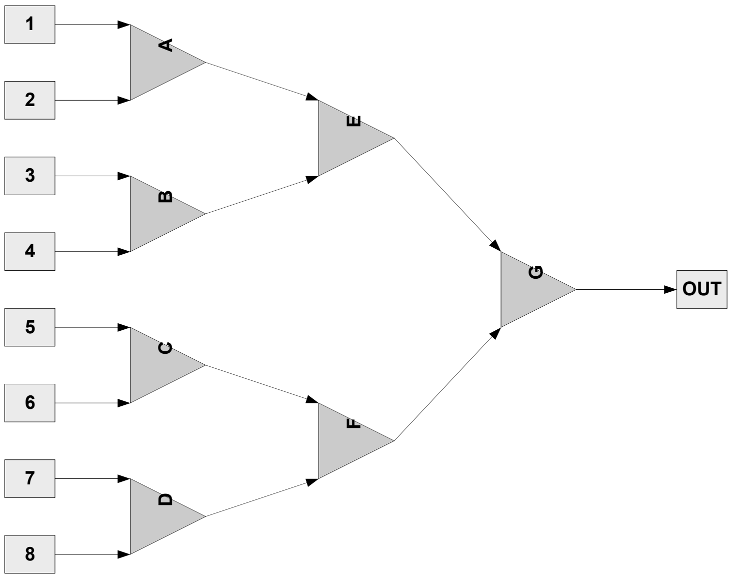
Problem 4

It's Only Logical

4 Points

Program Name: logic.cpp   Input File: logic.dat

For this problem, you will simulate a simple integrated circuit with 8 inputs and 7 operations. In the diagram below, you can see that your program will perform binary logic operations (denoted by the triangles) to compute a result. The 8 input values will each be a 0 or 1 and your program will compute the value to be output as a 0 or 1.



Valid logical operations to your program are as follows.

Operation	Symbol	Description
AND	*	Outputs a 1 if both inputs are 1; outputs a 0 otherwise
OR	+	Outputs a 0 if both inputs are 0; outputs a 1 otherwise
NAND	!	Outputs a 0 if both inputs are 1; outputs a 1 otherwise
NOR	-	Outputs a 1 if both inputs are 0; outputs a 0 otherwise
XOR	x	Outputs a 1 if the inputs are not the same; outputs a 0 otherwise.

Input

Input to your program is a series of IC tests. Each IC test consists of exactly 15 characters. Columns 1-8 will each contain a 0 or a 1 and represent the input values in positions 1-8 from the picture above. Columns 9-15 will each contain one of the symbols from the table above and represent the operations in positions A-G in the picture above.

Output

For each IC test, your program should compute the value represented by the position OUT and print this value (either a 0 or 1) on a line by itself to the screen.

Example: Input File

```
01010101*+!-x*+
01001101*+!-x*+
11000101*+!xxxx
10101010xxxxxxx
10101010!!!!!!!
```

Output to screen

```
1
0
1
0
1
```

Problem 5 | **What's in a Name?** | **6 Points**

Program Name: name.cpp **Input File: name.dat**

You've been hired by a junk mail company to mine intercepted e-mail for names and addresses of people. Most e-mail messages contain "signature" lines or "routing" lines or both. Your program will simply look for these lines based on the following titles: "Dr.", "Mr.", "Ms.", and "Mrs.". A set of lines of input are considered to be an address is if:

1. A line contains one of the titles starting in column 1 with up to 40 characters on the line.
2. There are 2 total lines immediately following the line with the title that each contain 1-40 characters.
3. There is a blank line following the three "address" lines.

Your program should identify and print the address lines from an input line based on the rules above. Remember that address lines can appear anywhere in the input file.

Input

Input to your program consists of a series of input lines each between 0 and 80 characters in length. Your program should read the input file and determine if an address has been found.

Output

Each time an address is found per the 3 rules above, your program should print the 3-line address followed by a single blank line to the screen.

Example: Input File

```
Mr. Michael Zimm
1234 Namovur Street
Asity, Texas 78923

This is a sample e-mail with 3 addresses in it.  However, your program
will only pick up on a couple of them.  For example, your program would
never be able to distinguish a line that had too many characters.  In fact,
Dr. Michaels, you cannot be assured that you will find all addresses no
matter how much you try.  Without the power of simple human intelligence,
I doubt if you will find even half of them.

Sincerely,

Mrs. Ima Hogg
2 Wiggs Way
Garland, Texas 77049
```

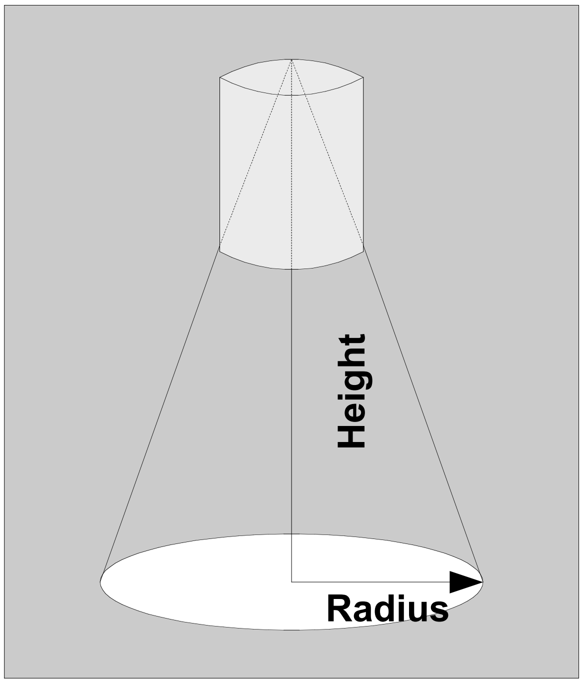
Output to screen

```
Mr. Michael Zimm
1234 Namovur Street
Asity, Texas 78923

Mrs. Ima Hogg
2 Wiggs Way
Garland, Texas 77049
```

Problem 6 | **A Little Light on the Subject** | **4 Points**

Program Name: light.cpp   Input File: light.dat

The Occupational Safety and Health Administration requires a minimum amount of light (given in lumens per square inch) in the workplace, that differs based on the type of work being done. You work for a company that produces high quality work lights that provide a uniform dispersal of light of within a cone using concave mirrors within a canister. The luminosity of the light can be computed based on the wattage of the light. One watt produces 14 lumens. Different models of the lights are available such that the dispersal ratio is different (dispersal ratio is given as Radius/Height to the mount on the light). See the figure below.



Your assignment is to determine what wattage lights are needed for a given workplace when you are given
1) the minimum lumens/square inch
2) a dispersal ratio from the type of the light, and
3) the height of the top of the light above the workspace.

Input

Input to your program consists of a series of product information requests. Each request can be found on a line by itself and consists of 3 floating point numbers starting in column 1 and separated by single spaces. The first value is the minimum lumens/square inch ($0.00<L\leq1000.00$); the second value is the dispersal ratio of the light model in question ($0.00<D\leq10.00$); and the third value is the height in inches the light will be mounted above the working surface ($0.00<H\leq250.00$). You can be certain that:

1. There will be no invalid or extraneous input.
2. All input values will have exactly 2 digits to the right of the decimal (even if both are zero).
3. There will be no leading zeroes on any input value except in the case that the zero is the only digit to the left of the decimal.

Output

For each product information request, you should compute the required wattage of the light to satisfy the minimum lumens/square inch requirement. You should print the required wattage as a floating point number with at least one digit to the left of the decimal and exactly 2 digits to the right of the decimal (rounded to the nearest 0.01). No output value should have leading zeroes unless the zero is the only digit to the left of the decimal.

Example: Input File

```
3.10 1.11 124.00
0.09 0.74 95.09
853.67 0.01 243.19
```

Output to screen

```
13178.71
100.00
1132.93
```

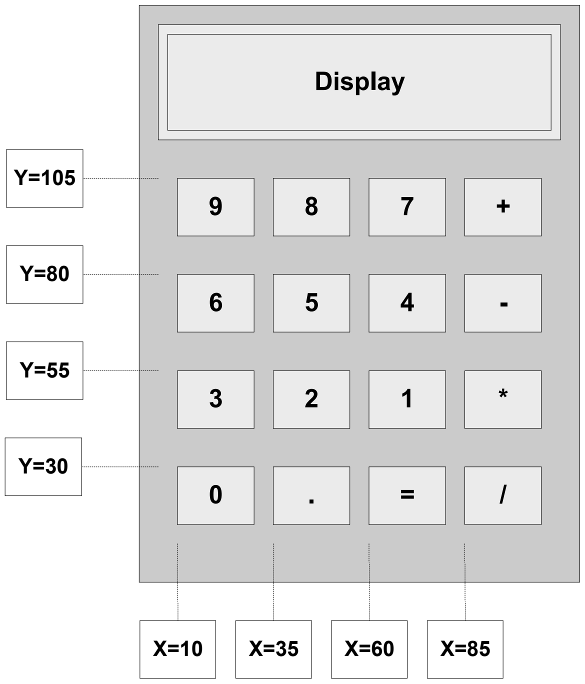
Problem 7

Calculate This

5 Points

Program Name: calc.cpp Input File: calc.dat

The company you work for is about to create a calculator for a touch screen. They need you to write the piece of software that, given the (x, y) coordinates of a screen touch, determine which button, if any, was touched. The figure below shows the layout of the screen.



The buttons on the calculator are 20 units wide (x-axis) and 15 units tall (y-axis). For example, the area of the "0" key extends from *X*=10 to *X*=30 between *Y*=15 and *Y*=30 inclusive.

Input

Input to your program will consist of a series of keystrokes each on a single line as an integer (*x, y*) coordinate pair. The (0≤*X*≤115) coordinate will be a single integer starting in column 1. There will be a single blank followed by the (0≤*Y*≤150) coordinate. There will be no extraneous characters or invalid/blank lines of input.

Output

For each keystroke, your program should determine the character touched and print it on a line by itself. If there is no key at the touched position, your program should print the word "Panel" on a line by itself starting in column 1.

Example: Input File

```
38 41
81 63
102 102
10 105
80 55
60 40
```

Output to screen

```
2
Panel
+
9
1
1
```

Problem 8

Make a Note of It

5 Points

Program Name: note.cpp   Input File: note.dat

Sometimes teachers intercept notes that you pass around in class. If you are lucky, they simply give them back to you when class is over. If you are unlucky, they read them aloud in class. For this problem, we will create a simple encryption method that may not deter the CIA, but will at least hinder a teacher.

The first step in defining your encryption is to establish the character set that your note can contain. In the table below, you can see the 56-character set. Note that the set starts numbering at zero and there are four punctuation characters. Messages and masks (described later) use the underscore (position 46) instead of the blank. Position 4 is the period; position 37 is the question mark; and position 53 is the exclamation.

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
L	x	D	n	.	v	p	J	S	G	P	Y	j	d	V	l	t	N	B	g	b	U	y	E	A	r	c	Z

28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55
R	e	K	k	o	Q	X	h	w	?	m	F	W	s	f	q	H	u	_	z	a	O	i	I	T	!	C	M

Messages are encrypted by shifting the characters of the message forward according to a "mask". For example, if the character "S" were to be encrypted with a mask of "B", we would count forward 18 (the value of "B") starting with "S" giving the value "c". To encrypt a message, we use a multi-character mask (called a key) that is repeated over and over. Each character of the message being encrypted is "masked" by the character in the key.

In the example encryption job below, you can see how the source message "Meet_at_locker_twelve!" would be encrypted using the key "Amber". The position of each character in the source message is added to the position of the corresponding character in the repeated key. You will note that the encryption of the first "M" uses "A" from the repeated key and that the sum of the two positions is computed as 55+24=79. When the resulting position exceeds 55, the character set is considered to wrap such that the "L" is at position 0 and at position 56. Therefore, the position 79 is interpreted to "E".

M	e	e	t	_	a	t	_	l	o	c	k	e	r	_	t	w	e	l	v	e	!
A	m	b	e	r	A	m	b	e	r	A	m	b	e	r	A	m	b	e	r	A	m
E	Y	O	u	l	t	C	P	H	x	i	d	O	C	l	b	B	O	H	K	!	h

With a little work, you can also figure out how to "decrypt" a message given the encrypted message and the key. In the case of a decryption job, an encrypted string is considered the source string.

Input

Input to your program is will consist of a series of encryption/decryption jobs each on a single line of input. Column 1 will contain either a "E" for encrypt or "D" for decrypt. Column 2 will contain a single space character (an actual space and not the underscore from the encryption character set). The source string will begin in column 3 and will be 1 to 50 characters in length and will be followed by a single space. Finally, the input line contains the "key" which is 1 to 10 characters in length and is followed by the end of line. You can be sure that there will be no invalid input (characters outside the above character set plus the space character) nor will there be any invalid formatting.

Output

For each source string, your program should encrypt or decrypt (depending on the job type) the source string and print the result on a line by itself to the screen.

Example: Input File

```
E Meet_at_locker_twelve! Amber
D SilNHofGv.?sNRnHXST Alfalfa
```

Output to screen

```
QVQgqFYvVlWNOg_R_A!eU
Where_is_Buckwheat?
```

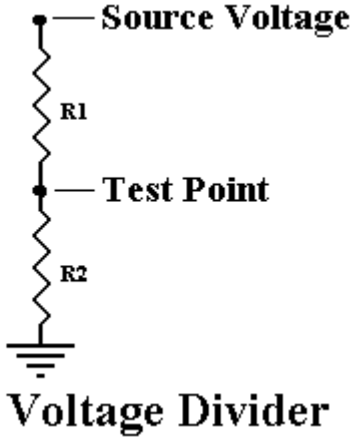
Problem 9

Truly Shocking

4 Points

Program Name: shocking.cpp   Input File: shocking.dat

One of the most fundamental principles of electrical engineering is the voltage divider. The voltage divider consists of two resistors connected in series as seen in the figure below.



In this program, you will be required to compute the voltage at the test point for voltage dividers. Some simple equations related to this problem are in the table below.

#	Equation	Explanation
1	$R_3 = R_1 + R_2$	The resistance of two resistors in series (R_3) is computed as the sum of the resistances (R_1 and R_2).
2	$V = I * R$	The voltage V across a resistor is equal to the current (I) times the resistance (R).
3	$TP = \frac{SV * R_2}{R_1 + R_2}$	Derived from #'s 1 and 2, the voltage at the test point of a voltage divider is based on a ratio of the resistance between the test point and ground versus the total resistance.

Input

Input to your program will consist of a series of voltage dividers each on a line by itself. Each line will contain 3 integers each separated by a space. The first two integers represent the resistance in ohms of the resistors #1 and #2 respectively ($0<R_1, R_2 \le 10000$). The third integer represents the source voltage ($0 < SV \le 1000$). There will be no invalid or extraneous input and no input values will have leading zeroes.

Output

For each input line, your program will produce a single line of output with the voltage of the test point starting in column 1. Output values are to be printed with at least one digit to the left of the decimal and exactly 3 digits to the right of the decimal. There should not be any leading zeroes to the left of the decimal unless the only digit is a zero. Your program should not contain any extraneous output or blank lines.

Example: Input File
```
1200 1800 124
800 75 120
2763 2351 10
6135 73 5
```
Output to screen
```
74.400
10.286
4.597
0.059
```

Problem 10	Which way did it go?	6 Points

Program Name: router.cpp **Input File: router.dat**

One of the most important functions in the I/T world today is the efficient routing of packets in networks. Typically, this routing occurs using a simple internet protocol (IP) address. IP addresses are 4 bytes long and are expressed as 4 numbers ranging 0-255 (a single unsigned byte is 0-255) separated by periods (example 9.87.65.210). When a router receives a packet, it uses the destination IP address from the packet to examine a routing table and determines the next "hop" for the packet.

A routing table contains ranges of IP addresses that are to be sent to each peer router or machine on the same subnet. Network topologies are typically constructed such that the whole ranges of IP addresses within the domain of a single byte can be routed to a single peer. This is possible by setting up "routing masks" that use wild cards to match any number in one or more of the IP bytes. For example, it is possible that all addresses starting with 125.14 are routed to a single peer. The entry in the routing table would contain wild cards in the routing range in an address that appears as "125.14.*.*". Your router will include this functionality and will search the routing table and select the first match it finds.

Rule #	Routing Mask	Routing Address	Comments
1	`226.76.*.*`	`226.76.27.1`	Anything starting with 226.76 will go here
2	`143.*.*.*`	`98.24.52.1`	Anything starting with 143 will go here
3	`115.74.*.*`	`115.74.*.1`	Anything starting with 115.74 will go to a router ending in 1 with the same third number (ie 115.74.99.16 will go to 115.74.99.1)
4	`225.8.15.*`	`225.8.*.1`	Although it is odd to have an absolute value (15) in the 3rd value of the mask and an wild card in the same position of the routing address, it is valid because it will always evaluate out to 15.
5	`225.*.*.*`	`225.14.9.1`	Anything that starts with 225 goes to 225.14.9.1
6	`9.*.*.*`	`9.1.1.1`	Anything that starts with a 9 goes to 9.1.1.1
7	`157.19.230.*`	`226.76.27.1`	Anything starting with 157.19.230 goes to 226.76.27.1
8	`27.18.43.18`	`27.18.43.18`	This guy must have done something bad if we are sending all of his stuff back to him.
9	`27.*.*.*`	`27.*.*.1`	Anything starting with a 27 will go to the same address with a 1 in the last position.
10	`187.32.42.*`	`226.75.18.1`	Anything starting with 187.32.42 goes to 226.75.18.1
11	`*.*.*.*`	`226.75.11.1`	Everything else goes to 226.75.11.1 since we don't know what to do with it. Most routing tables have this but it is not required.

You can see from the table above:

1. It is possible for an address to match more than one mask. In this case, your program should select the first table entry (top-to-bottom) that has a matching mask.
2. Wild cards can be used in any position in the mask but it does not make sense to use a wild card in one position and then use absolute values in the positions to the right. (For example, a mask with 225.*.19.17 does not make sense from a network topology point of view.) Therefore, you may assume that such a mask would be invalid and your program will not be subjected to invalid input.
3. It is possible for 1 or more routing addresses to contain a wild card. When this is the case, your program should use the value from the corresponding destination IP address position in constructing the routing address. For example, the address 27.189.212.47 would be routed to 27.189.212.1 per rule #9.

Input

Input to your program will consist of a routing table followed by a series of destination IP addresses. The first line of input will contain a single integer ($1 \leq R \leq 100$) indicating the number of rules in the routing table. The next R lines of input will each contain a routing rule. Routing rules will be of the form "*x.x.x.x y.y.y.y*" where *x.x.x.x* is the routing mask and *y.y.y.y* is the routing address and there is a single space separating them. In all cases, valid values of *x* and *y* are (0≤*x,y*≤255) and "*". All entries of the routing table will start in column 1 and there will be no other input on any of the lines describing the routing table.

The remaining lines of the input file will each contain a single destination IP address of the form "*z.z.z.z*" where the valid values are (0≤*z*≤255). Each destination IP address will start in column 1 and there will be no other input on any destination IP address line. You can also be assured that no value in the input file will contain leading zeroes unless zero is the only digit in the value.

Output

For each destination address in the input file, your program is to determine the routing address for the packet from the routing table and print the routing address on a line by itself. If no routing mask in the entire routing table matches the destination address, your program should print the message "Route to the bit bucket" on a line by itself. Your program should contain no other output, blank lines or extraneous characters.

Example: Input File

```
10
226.76.*.* 226.76.27.1
143.*.*.* 98.24.52.1
115.74.*.* 115.74.*.1
225.8.15.* 225.8.*.1
225.*.*.* 225.14.9.1
9.*.*.* 9.1.1.1
157.19.230.* 226.76.27.1
27.18.43.18 27.18.43.18
27.*.*.* 27.*.*.1
187.32.42.* 226.75.18.1
9.87.65.210
115.75.87.83
115.74.9.28
143.87.82.81
226.75.76.76
```

Output to screen

```
9.1.1.1
Route to the bit bucket
115.74.9.1
98.24.52.1
Route to the bit bucket
```