Code Quest Problem Packet

2023 Annual International Contest

Saturday, April 22, 2023 – North America & Europe

Saturday, April 29, 2023 – Australia & Pacific

LOCKHEED MARTIN

Table of Contents

Frequently Asked Questions

How does the contest work?

To solve each problem, your team will need to write a computer program that reads input from the standard input channel and prints the expected output to the console. Each problem describes the format of the input and the expected format for the output. When you have finished your program, you will submit the source code for your program to the contest website. The website will compile and run your code, and you will be notified if your answer is correct or incorrect.

Who is judging our answers?

We have a team of Lockheed Martin employees responsible for judging the contest, however most of the judging is done automatically by the contest website. The contest website will compile and run your code, then compare your program's output to the expected official output. If the outputs match exactly, your team will be given credit for answering the problem correctly.

How is each problem scored?

Each problem is assigned a point value based on the difficulty of the problem. When the website runs your program, it will compare your program's output to the expected judging output. If the outputs match exactly, you will be given the points for the problem. There is no partial credit; your outputs must match *exactly*. If you are being told your answer is incorrect and you are sure it's not, double check the formatting of your output, and make sure you don't have any trailing whitespace or other unexpected characters.

We don't understand the problem. How can we get help?

If you are having trouble understanding a problem, you can submit questions to the problems team through the contest website. While we cannot give hints about how to solve a problem, we may be able to clarify points that are unclear. If the problems team notices an error with a problem during the contest, we will send out a notification to all teams as soon as possible.

Our program works with the sample input/output, but it keeps getting marked as incorrect! Why?

Please note that the official inputs and outputs used to judge your answers are MUCH larger than the sample inputs and outputs provided to you. These inputs and outputs cover a wider range of test cases. The problem description will describe the limits of these inputs and outputs, but your program must be able to accept and handle any test case that falls within those limits. All inputs and outputs have been thoroughly tested by our problems team, and do not contain any invalid inputs.

Can we access the internet during the competition?

Please see the section regarding Internet Access later in this packet. Certain websites should not be accessed and may be blocked if at an on-site location.

We can't figure out why our answer is incorrect. What are we doing wrong?

Common errors may include:

- Incorrect formatting - Double check the sample output in the problem and make sure your program has the correct output format.
- Incorrect rounding - See the next section for information on rounding decimals.
- Invalid numbers - 0 (or 0.0, 0.00, etc.) is NOT a negative number. 0 may be an acceptable answer, but -0 is not.
- Extra characters - Make sure there is no extra whitespace at the end of any line of your output. Trailing spaces are not a part of any problem's output.
- Decimal format - We use the period (.) as the decimal mark for all numbers.

If these tips don't help, feel free to submit a question to the problems team through the contest website. We cannot give hints about how to solve problems, but may be able to provide more information about why your answers are being returned as incorrect.

I get an error when submitting my solution.

When submitting a solution, only select the source code for your program (depending on your language, this may include .java, .cs .cpp, or .py files). Make sure to submit all files that are required to compile and run your program. Finally, make sure that the names of the files do not contain spaces or other non-alphanumeric characters (e.g. "Prob01.java" is ok, but "Prob 01.java" and "Bob'sSolution.java" are not).

Can I get solutions to the problems after the contest?

Yes! Please ask your coach to email Code Quest® Problems Lead Brett Reynolds at brett.w.reynolds@lmco.com .

How are ties broken?

At the end of the contest, teams will be ranked based on the number of points they earned from correct answers during the contest. If there is a tie for the top three positions in either division, ties will be broken as follows:

1. Fewest problems solved (this indicates more difficult problems were solved)
2. Fewest incorrect answers (this indicates they had fewer mistakes)
3. First team to submit their last correct response (this indicates they worked faster)

Please note that these tiebreaker methods may not be fully reflected on the contest website's live scoreboard. Additionally, the contest scoreboard will "freeze" 30 minutes before the end of the contest, so keep working as hard as you can!

Mathematical Information

Rounding

Some problems will ask you to round numbers. All problems use the "half up" method of rounding unless otherwise stated in the problem description. Most likely, this is the sort of rounding you learned in school, but some programming languages use different rounding methods by default. **Unless you are certain you know how your programming language handles rounding, we recommend writing your own code for rounding numbers based on the information provided in this section.**

With "half up" rounding, numbers are rounded to the nearest integer. For example:

- 1.49 rounds down to 1
- 1.51 rounds up to 2

The "half up" term means that when a number is exactly in the middle, it rounds to the number with the greatest absolute value (the one farthest from 0). For example:

- 1.5 rounds up to 2
- -1.5 rounds down to -2

Rounding errors are a common mistake; if a problem requires rounding and the contest website keeps saying your program is incorrect, double check the rounding!

Trigonometry

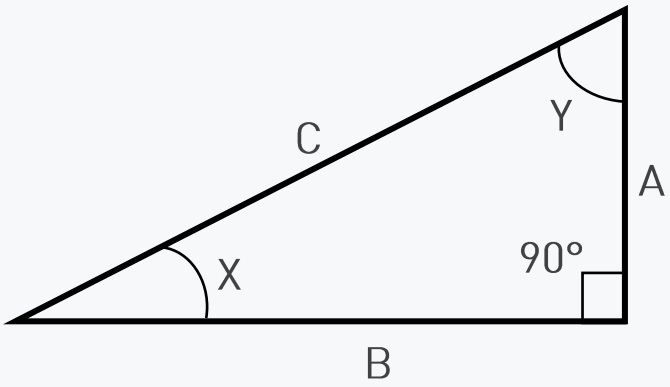
Some problems may require the use of trigonometric functions, which are summarized below. Most programming languages provide built-in functions for $\sin X$, $\cos X$, and $\tan X$; consult your language's documentation for full details. Unless otherwise stated in a problem description, it is *strongly recommended* that you use your language's built-in value for pi (π) whenever necessary.

$\sin X = \frac{A}{C} \quad \cos X = \frac{B}{C} \quad \tan X = \frac{A}{B} = \frac{\sin X}{\cos X}$

$X + Y = 90°$

$A^2 + B^2 = C^2$

$\frac{degrees * \pi}{180} = radians$



US ASCII Table

The inputs for all Code Quest® problems make use of printable US ASCII characters. Non-printable or control characters will not be used in any problem unless explicitly noted otherwise within the problem description. In some cases, you may be asked to convert characters to or from their numeric equivalents, shown in the table below.

Binary	Decimal	Character	Binary	Decimal	Character	Binary	Decimal	Character
0100000	32	(space)	1000000	64	@	1100000	96	`
0100001	33	!	1000001	65	A	1100001	97	a
0100010	34	"	1000010	66	B	1100010	98	b
0100011	35	#	1000011	67	C	1100011	99	c
0100100	36	$	1000100	68	D	1100100	100	d
0100101	37	%	1000101	69	E	1100101	101	e
0100110	38	&	1000110	70	F	1100110	102	f
0100111	39	'	1000111	71	G	1100111	103	g
0101000	40	(	1001000	72	H	1101000	104	h
0101001	41	)	1001001	73	I	1101001	105	i
0101010	42	*	1001010	74	J	1101010	106	j
0101011	43	+	1001011	75	K	1101011	107	k
0101100	44	,	1001100	76	L	1101100	108	l
0101101	45	-	1001101	77	M	1101101	109	m
0101110	46	.	1001110	78	N	1101110	110	n
0101111	47	/	1001111	79	O	1101111	111	o
0110000	48	0	1010000	80	P	1110000	112	p
0110001	49	1	1010001	81	Q	1110001	113	q
0110010	50	2	1010010	82	R	1110010	114	r
0110011	51	3	1010011	83	S	1110011	115	s
0110100	52	4	1010100	84	T	1110100	116	t
0110101	53	5	1010101	85	U	1110101	117	u
0110110	54	6	1010110	86	V	1110110	118	v
0110111	55	7	1010111	87	W	1110111	119	w
0111000	56	8	1011000	88	X	1111000	120	x
0111001	57	9	1011001	89	Y	1111001	121	y
0111010	58	:	1011010	90	Z	1111010	122	z
0111011	59	;	1011011	91	[	1111011	123	{
0111100	60	<	1011100	92	\	1111100	124	|
0111101	61	=	1011101	93	]	1111101	125	}
0111110	62	>	1011110	94	^	1111110	126	~
0111111	63	?	1011111	95	_			

Terminology

Throughout this packet, we will describe the inputs and outputs your programs will receive. To avoid confusion, certain terms will be used to define various properties of these inputs and outputs. These terms are defined below.

- An **integer** is any whole number; that is, a number with no decimal or fractional component: -5, 0, 5, and 123456789 are all integers.
- A **decimal number** is any number that is not an integer. These numbers will contain a decimal point and at least one digit after the decimal point. -1.52, 0.0, and 3.14159 are all decimal numbers.
- **Decimal places** refer to the number of digits in a decimal number following the decimal point. Unless otherwise specified in a problem description, decimal numbers may contain any number of decimal places greater or equal to 1.
- A **hexadecimal number** or **string** consists of a series of one or more characters including the digits 0-9 and/or the uppercase letters A, B, C, D, E, and/or F. Lowercase letters are not used for hexadecimal values in this contest.
- **Positive numbers** are those numbers strictly greater than 0. 1 is the smallest positive integer; 0.000000000001 is a very small positive decimal number.
- **Non-positive numbers** are all numbers that are not positive; that is, all numbers less than or equal to 0.
- **Negative numbers** are those numbers strictly less than 0. -1 is the greatest negative integer; -0.000000000001 is a very large negative decimal number.
- **Non-negative numbers** are all numbers that are not negative; that is, all numbers greater than or equal to 0.
- **Inclusive** indicates that the range defined by a given value (or values) includes that/those value(s). For example, the range "1 to 3 inclusive" contains the numbers 1, 2, and 3.
- **Exclusive** indicates that the range defined by a given value (or values) does not include that/those values. For example, the range "0 to 4 exclusive" includes the numbers 1, 2, and 3; 0 and 4 are not included.
- **Date and time formats** are expressed using letters in place of numbers:
    - **HH** indicates the hours, written with two digits (with a leading zero if needed). The problem description will specify if 12- or 24-hour formats should be used.
    - **MM** indicates the minutes for times or the month for dates. In both cases, the number is written with two digits (with a leading zero if needed; January is 01).
    - **YY** or **YYYY** is the year, written with two or four digits (with a leading zero if needed).
    - **DD** is the date of the month, written with two digits (with a leading zero if needed).

Internet Access

Code Quest began allowing internet access during the competition in 2021; this was largely done out of necessity, as our competition had moved to a virtual setting in response to the global COVID-19 pandemic. However, we have made the decision to continue to allow internet access during the competition this year, even as most of our contest locations move back to being on-site and in-person.

Please note that if you are at an on-site location, you will be connecting to Lockheed Martin's guest network, and certain websites may be blocked. The Code Quest team does not have the ability to unblock any websites during the competition; if you require access to a blocked website, please contact the judging team for assistance. The judging team can also assist you with connecting to this network in the first place. You should have received a username and password as part of your registration materials when you arrived; these will differ from the username and password you will need to log into the contest website.

While you will have the ability to access the internet during the contest, we ask and require that you do so responsibly. Please remember that Code Quest is a competition meant to test your team's programming skills, not those of random strangers on the internet. However, we understand that referring to documentation is a necessary part of software development. During the contest, we encourage you to make use of your programming language's official documentation, or basic tutorial websites that explain how to use key features of your programming language in general terms (such as W3Schools or official tutorials provided by Oracle, Microsoft, etc.). Our volunteers use these websites as well when we test the problems we present to you.

Please note that accessing these websites is against Code Quest rules:

- Stack Overflow or similar websites that allow users to ask and receive answers to specific programming questions
- Gitlab or similar websites that allow storing and sharing of code snippets – this includes access to your own accounts, as using pre-written code is also against Code Quest rules
- ChatGPT or other websites that provide the ability to generate solutions to problems (ChatGPT should be blocked if you are at an on-site location, but this includes any means by which to access ChatGPT or similar services through other websites which are not blocked)

All code that you submit during the contest must be your team's original work, created after the start of the contest. While we will not actively search for instances of dishonest behavior, our judging teams will be monitoring the competition closely, and any clear instance of dishonest behavior will be addressed, by means up to and including disqualification from the competition.

Thank you for your understanding and cooperation with helping us to maintain a fair and entertaining event. Best of luck today!

Problem 01: Finding Nimo

Points: 5

Author: Shelly Adamie, Fort Worth, Texas, United States

Problem Background

Your friend was clowning around earlier and stuck a program on your computer that keeps adding the name "Nimo" at random in your emails. You don't want to confuse everyone you're talking to, and with the amount of work you've got, it's important you just keep typing. Unfortunately, you can't figure out how your "friend" installed his little virus, so you think you need to write a program to find all of the Nimos he's added.

Problem Description

Your program will integrate with the email application on your computer to prevent any embarrassing messages from being sent out. It will scan each sentence in your email and report the position of the word "Nimo" within that sentence.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include a single line with a sentence that includes the word "Nimo" exactly once. Each line will consist only of letters and spaces.

```
3
I found Nimo
Nimo found me
I think poor old Nimo is lost
```

Sample Output

For each test case, your program must print the position of the word "Nimo" within the given sentence, where the first word is in position 1.

```
3
1
5
```

Problem 02: The Bottom Line

Points: 5

Author: Holly Norton, Fort Worth, Texas, United States

Problem Background

Competition is key in business! The bottom line is, if sales are up, companies are happy. The top two aircraft companies have just reported their sales for the year. Who will come out on top?

Problem Description

Your program should compare the number of aircraft sold by each company and determine who sold the most aircraft. Your program will be given two numbers, representing the number of aircraft sold by Cassowary Craft and Lead Balloons Ltd, respectively. It should then print a report comparing the sales between the two companies.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include a single line, containing two positive integers, representing:

- The number of aircraft sold by Cassowary Craft
- The number of aircraft sold by Lead Balloons Ltd

```
3
22 9
6 14
12 12
```

Sample Output

For each test case, your program must print a single line of text, as follows:

- If Cassowary Craft sold more aircraft, print "Cassowary Craft sold X more aircraft"
- If Lead Balloons Ltd sold more aircraft, print "Lead Balloons Ltd sold X more aircraft"
- If both companies sold the same number of aircraft, print "Cassowary Craft and Lead Balloons Ltd sold the same number of aircraft"

In both of the first two cases, X should be a positive integer indicating the difference in sales between the companies.

```
Cassowary Craft sold 13 more aircraft
Lead Balloons Ltd sold 8 more aircraft
Cassowary Craft and Lead Balloons Ltd sold the same number of aircraft
```

Problem 03: Pennies Add Up

Points: 10

Author: Ryan Regensburger, Huntsville, Alabama, United States

Problem Background

They say a picture is worth a thousand words, but what are words worth? Perhaps they're worth the sum of their letters, but then what is each letter worth?

You may have received a school assignment once to find words that were worth one dollar (100 cents), where each letter in the word was worth a certain number of cents. It's surprisingly difficult, so let's write a computer program to handle the problem for us.

Problem Description

Your program will be given a list of words and the value of the letter A in cents. Each successive letter in the alphabet will be worth one cent more, up to a maximum of 26 cents. The letter after the one worth $0.26 will wrap back around to be worth a single cent. For example, if A is worth 5 cents, B will be worth 6 cents, C will be worth 7 cents, and so on; V will be worth 26 cents, and W will be worth 1 cent.

For each word in the given list, you'll need to total up the values of the letters in that word to determine their value. If the word is worth exactly $1.00, it's a winner!

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include:

- A line containing two positive integer values, separated by spaces:
  - The value of the letter A, in cents. This value will be between 1 and 26, inclusive.
  - **W**, the number of words listed for this test case
- **W** lines, each containing a single word in uppercase letters

```
2
1 4
FIRST
SECOND
THIRD
PROBLEMS
7 3
WINNER
CHICKEN
DINNER
```

Sample Output

For each test case, your program must print one line for each word in the test case worth exactly $1.00, containing the following:

- The word "WINNER"
- A space
- An integer showing the value of the letter A in cents
- A colon (:)
- A space
- The $1.00 word in uppercase letters

Words should be printed in the order presented in the input.

```
WINNER 1: PROBLEMS
WINNER 7: DINNER
```

Problem 04: Make a Budget

Points: 10

Author: Matt Marzin, King of Prussia, Pennsylvania, United States

Problem Background

Adulting can be a difficult and grueling task. One of those complicated parts is managing what you can and should spend money on. You've got the basic expenses: gas for your car, a cell phone plan, meals, and rent. If you want anything extra like a new car, new computer, or an exotic vacation, you've got to save up money. The best way to manage all this is to create a budget!

Problem Description

For this problem, you'll be creating a program that can help you stick to your budget by telling you if you should or shouldn't go through with a purchase. Your budget will be organized into a series of categories, each of which will have a starting balance. You can think of these as separate bank accounts that can't be overdrawn.

As you spend (or plan to spend) money, you'll enter transactions against your budget. Expenses like going to a restaurant or paying the rent should subtract money from the relevant category. If that category doesn't have enough money left to handle the transaction, it shouldn't deduct anything but instead tell you not to make the purchase. Some transactions might increase the amount of money in a category, like getting a paycheck or an extra $100 from your grandparents for your birthday. These transactions can be added without validation.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include:

- A line containing two positive integers separated by spaces, C and T, representing the number of categories in your budget and the total number of transactions you're planning for.
- C lines listing the categories in your budget. Each line will contain a string of upper and lowercase letters (representing the name of the category), a space, then a non-negative integer representing the initial balance of that category.
- T lines listing the transactions you are planning for your budget. Each line will contain the name of a category (from amongst those previously listed), a space, a plus or minus sign (indicating if the transaction is income or an expense), a space, and a positive integer (representing the amount of money to add or subtract from the category)

```
2
3 5
Restaurants 25
NewCellPhone 80
Groceries 100
Groceries + 20
NewCellPhone - 800
Groceries - 80
Restaurants - 20
Restaurants - 20
3 8
Restaurants 25
NewCellPhone 80
Groceries 100
NewCellPhone + 400
NewCellPhone - 800
Groceries - 80
NewCellPhone + 200
Restaurants - 25
NewCellPhone + 200
Restaurants - 50
NewCellPhone - 800
```

Sample Output

For each test case, your program must print one line for each transaction, containing the word "YES" if the transaction was accepted, or "NO" if it was rejected. Transactions should be rejected whenever there is an attempt to remove more money from a category than the category currently contains.

```
YES
NO
YES
YES
NO
YES
NO
YES
YES
YES
YES
NO
YES
```

Problem 05: Detecting Multipaction

Points: 15

Author: Sowmya Chandrasekaran, Sunnyvale, California, United States

Problem Background

Multipaction is a phenomenon experienced by radio frequency (RF) devices exposed to the vacuum of space. Free electrons floating around can be accelerated by the energy in the RF field. When these electrons hit a metallic surface in the device, they can release additional electrons. These electrons go on to impact more surfaces, creating an exponentially growing electron cascade that can damage or destroy the device. Luckily, it's possible to detect when multipaction events occur in order to protect the integrity of these devices.

Problem Description

Lockheed Martin Space Systems is working on a new communications satellite system that includes an array of RF antennas. Your customer is concerned about the risk of multipaction events, and has asked your team to develop a system to automatically disable an antenna for a short period when it detects a multipaction event.

While most transmitters and antennas are intended to operate on a particular frequency, other frequencies - known as harmonics - can interfere with that signal. To combat this, your satellites will listen on multiple frequencies in order to eliminate this interference. Fortunately, this also gives you the information you need to identify multipaction events.

You've noticed during testing that a multipaction event results in abnormally high readings in both the phase null and third harmonic channels, between 60% and 85% of their maximum power output. Your team lead believes that a subroutine monitoring these two channels could give enough warning about a multipaction event to shut down the antennas before they can overload. Your team needs to develop this subroutine and test it before it's loaded onto the satellites. If, for a given time index, both channels show a power reading between .6 and .85 of their maximum outputs (inclusive), that indicates an event which should be reported.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include two lines of text, containing the readings obtained from the phase null and third harmonic channels, respectively. Each line will consist of a list of decimal values between 0 and 1 exclusive, separated by spaces. Both lines will contain the same number of values.

```
3
.3 .61 .4 .15 .81 .47 .98
.2 .64 .7 .36 .63 .71 .09
.45 .53 .59 .13 .21 .78 .34 .78 .91
.87 .71 .32 .33 .58 .61 .79 .86 .62
.5 .71 .42 .36 .49 .82 .6 .21
.67 .41 .76 .83 .85 .12 .51 .92
```

Sample Output

For each test case, your program must print a single line containing a sentence summarizing the findings of your subroutine, as follows:

- If no multipaction events were detected, print "No multipaction events detected."
- If one multipaction event was detected, print "A multipaction event was detected at time index X.", replacing X with the index number of the detected event.
- If more than one multipaction event was detected, print "N multipaction events were detected at time indices: X.", replacing N with the total number of detected events, and X with a space-delimited list of the index numbers of detected events in increasing order.

Index numbers correspond to the position of the values within the provided arrays of data. The first item in each list has an index number of 0.

```
2 multipaction events were detected at time indices: 1 4.
A multipaction event was detected at time index 5.
No multipaction events detected.
```

Problem 06: Illegal Input

Points: 15

Author: Javier Jimenez, Marietta, Georgia, United States

Problem Background

Validating input from users is an important part of maintaining the security of any application. Illegal inputs could cause any number of problems; usually, these are fairly minor and would only impact the person who provided the input. However, if the user has malicious intentions, they can design input specifically designed to attack your application or the underlying system. A recently discovered vulnerability with the popular Log4J library allowed exactly this; by entering a specifically formatted string, an attacker could hijack servers by making them run any command they wanted.

Problem Description

Your team is working for Lockheed Martin's Corporate Information Security division. In the aftermath of the Log4J vulnerability, your team has been asked to design a common utility program that can be incorporated into existing Lockheed Martin programs to protect against a number of various code injection attacks. Your program will be given a string input by a user and must scan it for a series of common attack vectors, listed below. For each of these vectors, the phrase **<ANY>** represents any text string, of any length; other characters are part of the attack vector.

- `'; <ANY> --`
- `' OR 1=1` *(case-insensitive)*
- `${<ANY>}`
- `$(<ANY>)`
- `&& sudo`
- `&& su -`
- `;;`
- `<script` *(case-insensitive)*
- `%s`
- `%x`
- `%n`

If any of these phrases are identified anywhere within the input string, you should reject the input so the application does not process it any further and fall victim to a possible attack. If the string is free of attacks, you can return the original string to allow the application to continue normally.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include a single line containing a user-input string of text. Input may contain any printable ASCII character.

```
4
This is valid input.
This is'; DROP TABLE scoreboard; --not valid
Perhaps you could " && sudo make-me-a-sandwich
This looks suspicious $[jndi:sudo shutdown --now] but it's not a threat.
```

Sample Output

For each test case, your program must print a single line containing:

- The word "REJECTED" if the input appears to contain an attack vector listed above, or
- The original input string, otherwise

```
This is valid input.
REJECTED
REJECTED
This looks suspicious $[jndi:sudo shutdown --now] but it's not a threat.
```

Problem 07: Ship's Health Summary

Points: 20

Author: Jacob Starratt, Dartmouth, Nova Scotia, Canada

Problem Background

A warship requires constant maintenance to ensure that everything is functioning correctly, particularly since the lives of hundreds of sailors could depend on the ship's ability to perform both in and out of combat. With potentially hundreds of distinct systems onboard a ship, however, it can be very hard to keep track of everything and determine just how capable the ship actually is. A group of admirals has approached Lockheed Martin asking for a means of identifying a ship's overall "health" at a glance to help them determine deployment and maintenance schedules.

Problem Description

Your team has been tasked with developing a prototype of this system, to be deployed on a Littoral Combat Ship currently under construction. Each of the ship's systems will be given a priority level - one of LOW, MEDIUM, or HIGH - and a health score, ranging from 0 to 10 (where 0 means the system is completely destroyed and 10 means it's in perfect condition).

Priority	Weight
LOW	1
MEDIUM	2
HIGH	3

Your system must report an overall health score for the ship by calculating a weighted average of the health of all reported systems. A system's priority level determines its weighting in this calculation, as shown in the table at left. If you're not familiar with weighted averages, consider a ship with just two systems, with health scores of 3 and 7. Calculating the normal (unweighted) average of these scores yields (3 + 7) / 2 = 5. However, if the system with a score of 3 has a LOW priority and the one with a score of 7 has a MEDIUM priority, the weighted average becomes 5.67. The MEDIUM priority system counts twice, so the calculation for the total becomes (3 + 7 + 7). It's then divided by 3, the sum of the weights (1 and 2).

Once the weighted average is calculated, convert it to a scale of 0 to 100 by multiplying by 10 and rounding to the nearest integer. This number will represent the health score of the entire ship.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include:

- A line containing a positive integer, X, indicating the number of systems on the ship
- X lines showing the status of each system, each containing a priority level (one of LOW, MEDIUM, or HIGH), a space, and a health score: a positive integer between 0 and 10 inclusive.

```
3
2
LOW 3
MEDIUM 7
5
LOW 8
LOW 9
MEDIUM 2
HIGH 5
HIGH 4
3
LOW 10
MEDIUM 10
HIGH 2
```

Sample Output

For each test case, your program must print the ship's overall health score as a positive integer between 0 and 100 inclusive.

```
57
48
60
```

Problem 08: Ellipses...

Points: 25

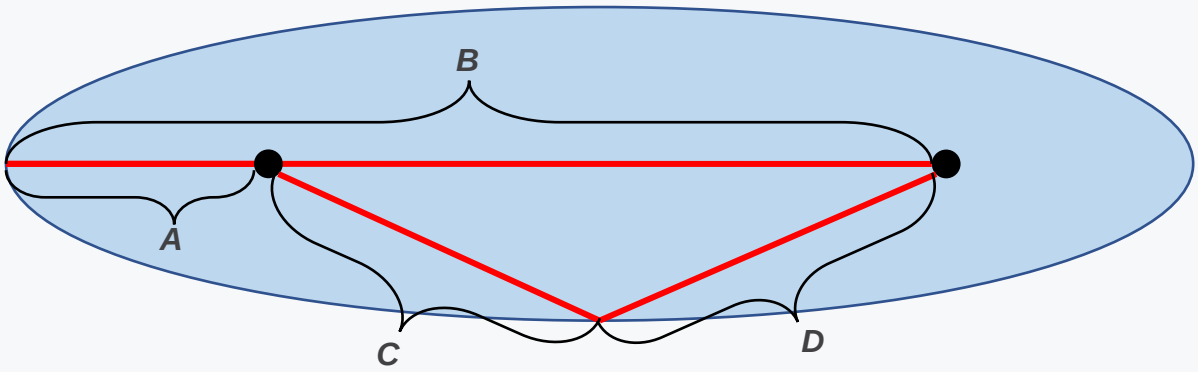
Author: David Van Brackle, Marietta, Georgia, United States

Problem Background

Engineers at Lockheed Martin Missiles & Fire Control are conducting a series of test simulations against a prototype weapon system. The weapon's range is simulated by a long ellipse in the direction the weapon is facing. The weapon should be able to identify if it's able to reach a given target; that is, if the target is actually within the weapon's range or not. Your team has been asked to assist with one of these tests.

Problem Description

An ellipse is similar to a circle that's been squashed so it's much wider in one direction. Ellipses are defined by two points, called foci. For each point on the ellipse, the sum of the distances between that point and each of the two foci is the same. In the example below, $A + B = C + D$.



Given the coordinates of the two foci forming the ellipse that represents the weapon system's range and the maximum width of that ellipse, you'll need to determine which points are inside the ellipse, and thus within the weapon system's functional range. No point involved in the test will be within 0.001 units of the edge of the ellipse.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include:

- A line containing six values, separated by spaces:
  - X1, a number representing the X-coordinate of the ellipse's first focus point
  - Y1, a number representing the Y-coordinate of the ellipse's first focus point

    - X2, a number representing the X-coordinate of the ellipse's second focus point
    - Y2, a number representing the Y-coordinate of the ellipse's second focus point
    - W, a number representing the maximum width of the ellipse
    - N, a positive integer representing the number of test points
- N lines, each containing two numbers separated by spaces, representing the X and Y coordinates (respectively) of a point involved in the test.

```
1
1 0 5 0 6 6
0 2
1 1
2 0
3 -1
4 -2
4 -2.5
```

Sample Output

For each test case, your program must print a single line for each test point involved in the test; print 1 if the point is within the ellipse, or 0 if it is not.

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

Problem 09: Counting Cards

Points: 30

Author: Steve Gerali, Denver, Colorado, United States

Problem Background

You have been hired by Casino Royale in Las Vegas, Nevada to build an application that can determine whether a player or dealer has won a hand of blackjack. Casino Royale will be using this application to support their electronic blackjack operations to determine whether to pay out a player or not based on whether they have a winning hand. The rules of blackjack are provided below along with how to determine whether a player has won, tied, or lost against the dealer.

Problem Description

In the game of blackjack, the player and the dealer will both receive two cards from a shuffled deck of cards. After those cards are dealt to the player and the dealer, the player and dealer may both "hit" as many times as they'd like to receive more cards, or "stay" to keep the cards they have. The objective of the game is to make the sum of your card values as close to 21 as possible without going over 21. If a player or dealer makes 21 exactly, then they have a "blackjack" which cannot be beaten. If either the player or the dealer goes over 21, then they "bust" and lose the round.

The number cards (2 through 10) are worth the number displayed, face cards (i.e. Jack, Queen or King) are worth 10, and an Ace can be worth either 1 or 11 (whichever benefits the player or dealer more). For example, if the player's first two cards are a Jack and an Ace, we would want to count the Ace as 11 since 10 + 11 = 21 and the player would have a blackjack. If the player had already had a hand worth 18, decided to hit and got an Ace, then the player would want to count it as 1, since counting it as 11 would put the player at 29 and they would bust.

The scoring of the game is settled using the following simple rules:

- If the player has a blackjack, they win, unless the dealer also has a blackjack, in which case the game is a tie.
- If the dealer busts and the player doesn't, then the player wins.
- If the player busts, the dealer wins.
- If neither player nor dealer busts, whoever is closest to 21 wins.
- If both the player and the dealer end up with the same score, the game is a tie.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include two lines of text,

representing the player's final hand and the dealer's final hand, respectively. Each line will contain the names of at least two cards, separated by spaces. Card names contain three parts:

- The card rank, either a number from 2 through 10 inclusive or one of the rank names "ACE", "JACK", "QUEEN", or "KING"
- The string "_OF_"
- The name of a suit of cards: "CLUBS", "SPADES", "HEARTS", or "DIAMONDS".

```
3
ACE_OF_DIAMONDS 10_OF_DIAMONDS
QUEEN_OF_HEARTS ACE_OF_HEARTS
5_OF_HEARTS KING_OF_DIAMONDS QUEEN_OF_DIAMONDS
8_OF_CLUBS JACK_OF_HEARTS
7_OF_CLUBS 10_OF_CLUBS
6_OF_HEARTS JACK_OF_HEARTS KING_OF_HEARTS
```

Sample Output

For each test case, your program must print a single line containing the following information, separated by spaces:

- The string "Player Score:"
- The total value of the player's cards
- The string "Dealer Score:"
- The total value of the dealer's cards
- One of the strings "Player Wins!", "Dealer Wins!", or "Tie!", depending on the outcome of the game.

```
Player Score: 21 Dealer Score: 21 Tie!
Player Score: 25 Dealer Score: 18 Dealer Wins!
Player Score: 17 Dealer Score: 26 Player Wins!
```

Problem 10: Are We on Budget?

Points: 35

Author: Sowmya Chandrasekaran, Sunnyvale, California, United States

Problem Background

Earned Value Management (EVM) is a program management technique used by the Department of Defense, and therefore defense companies, to provide situational awareness of program status and to assess cost, schedule, and technical performance of programs. Lockheed Martin's Finance and Business Operations team works closely with technical groups to accurately predict monthly costs and schedules, and conduct analyses each month to showcase to stakeholders how accurate their predictions were. Essentially, this system helps a program's stakeholders understand how money and time are being spent - this is especially important since a single spacecraft can be worth more than a billion dollars!

Problem Description

One of the metrics that Lockheed Martin tracks is "Cost Variance," the difference between the budgeted cost of an item or task and the actual cost incurred. If this variance is equal to 0, that means that the prediction was exactly correct, and the project is on budget. For example, if a program expected to spend $1000.00 but only spent $900.00, it has a cost variance of -$100.00; it was under budget. In order to provide a better overview of the program, cost variances are usually calculated according to individual line items (software development, systems engineering, purchasing parts, etc.) and then those variances are averaged for presentation to stakeholders.

Your team will be given a list of cost estimates and a list of the corresponding actual costs for a program during the previous month. You'll need to write a program that calculates the average cost variance for that month.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include:

- A line containing a single integer, **N**, representing the number of line items in the monthly budget.
- A line containing **N** decimal values, separated by spaces, each representing the budgeted amount for a line item in the program's overall budget.
- A line containing **N** decimal values, separated by spaces, each representing the actual cost for each line item in the program's budget. Each value in this list corresponds with the same-indexed item in the previous list.

```
2
6
123.45 678.90 1234.56 789.01 2345.67 8901.23
321.54 876.09 1432.65 987.10 2543.76 8109.32
6
250.00 349.99 150.45 782.15 650.00 99.99
225.16 299.99 160.14 798.16 650.00 75.00
```

Sample Output

For each test case, your program must calculate the cost variance for each line item, then print out a single line with the average cost variance across all line items. Round your result to two decimal places and include any trailing zeroes.

```
32.94
-12.36
```

Problem 11: Hijacked!

Points: 40

Author: Matt Hussey, Ampthill, Reddings Wood, United Kingdom

Problem Background

Lockheed Martin Space Systems maintains a large number of satellites for various purposes. Recently, they've received reports that one of their communication satellites has been acting oddly. Upon investigation, it seems as though the satellite has been hijacked through a cyberattack! Buried within its real communications are secret messages planted by the attacker! You'll need to decipher these messages to determine how the cyberattack was accomplished and to help authorities identify the culprit.

Problem Description

Streams of data sent by the satellite consist of printable ASCII characters. The hacker's messages are embedded within these streams and can be identified by indicator tokens. The token at the start of each message will consist of three different characters, which are different for each message (e.g. 'rst', 'qzy', or some other combination). The token at the end of a message consists of the same three characters used for the starting token, but in reverse order (using the previous examples, 'tsr' or 'yzq').

Each data stream may contain more than one message, but messages will not overlap each other. To avoid confusion with the indicator tokens, any character within a message that also appears within that message's indicator tokens will be "escaped" by appearing twice. For example, the data stream below contains the hidden message 'hello world'; the starting and ending tokens are highlighted:

```
abcjklhellllo worlldlkjdef
```

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include:

- A line containing a positive integer, N, representing the length of the data stream in characters
- A line containing N printable ASCII characters, representing the data stream. Each data stream contains at least one hidden message.

```
2
26
abcjklhellllo worlldlkjdef
41
tueoafghthhis is code questhgfllmn2023nml
```

Sample Output

For each test case, your program must print each hidden message contained within the data stream, one message per line.

```
hello world
this is code quest
2023
```

Problem 12: Fill in the Blanks

Points: 45

Author: Chuck Nguyen, Rockville, Maryland, United States

Problem Background

Lockheed Martin is a large corporation and produces a huge amount of documents and letters on a daily basis. Sometimes it's easier to use a pre-written template to generate these correspondences, rather than write them all out each time.

Problem Description

You're working with Lockheed Martin's Human Resources department to send letters out to prospective interns. You'll be provided with information about each candidate and the position they're applying for, which must be used to fill out a provided letter template. The template will include several fields which must be replaced with the relevant information to create a complete letter.

By the way, we are hiring! If you're interested in applying for an internship at Lockheed Martin, your coach will be receiving information about how to sign up. We look forward to seeing your resume!

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include:

- A line containing two positive integers separated by spaces; respectively,
  - D, the number of lines of data that will be provided, and
  - T, the number of lines of text in the letter template
- D lines containing the data to be populated into the template. Each line will contain the name of the field (which may contain upper- and lower-case letters and spaces), a colon (:), a space, and the value of the field (which may contain any printable characters).
- T lines containing the template to be filled out. Each line may contain any printable character; fields to be populated will be indicated by the use of square brackets containing the name of the relevant field. Square brackets will not appear in any other context.

```
1
6 8
Candidate Name: John Doe
Business Area: Enterprise Operations
Job Title: Software Engineer
Job Description: engineer software
Start Date: 6/1/2023
Manager Name: Jane Smith
Dear [Candidate Name],
Lockheed Martin [Business Area] is delighted to offer you the position
of [Job Title]!
As part of this position, you will be expected to [Job Description]. We
expect that you will be able to start on [Start Date].
We look forward to seeing you on [Start Date] at the [Business Area]
offices downtown.
Sincerely, [Manager Name]
```

Sample Output

For each test case, your program must print the provided message, with all fields replaced with their given values.

```
Dear John Doe,
Lockheed Martin Enterprise Operations is delighted to offer you the position
of Software Engineer!
As part of this position, you will be expected to engineer software. We
expect that you will be able to start on 6/1/2023.
We look forward to seeing you on 6/1/2023 at the Enterprise Operations
offices downtown.
Sincerely, Jane Smith
```

Problem 13: Who's Your Supervisor?

Points: 50

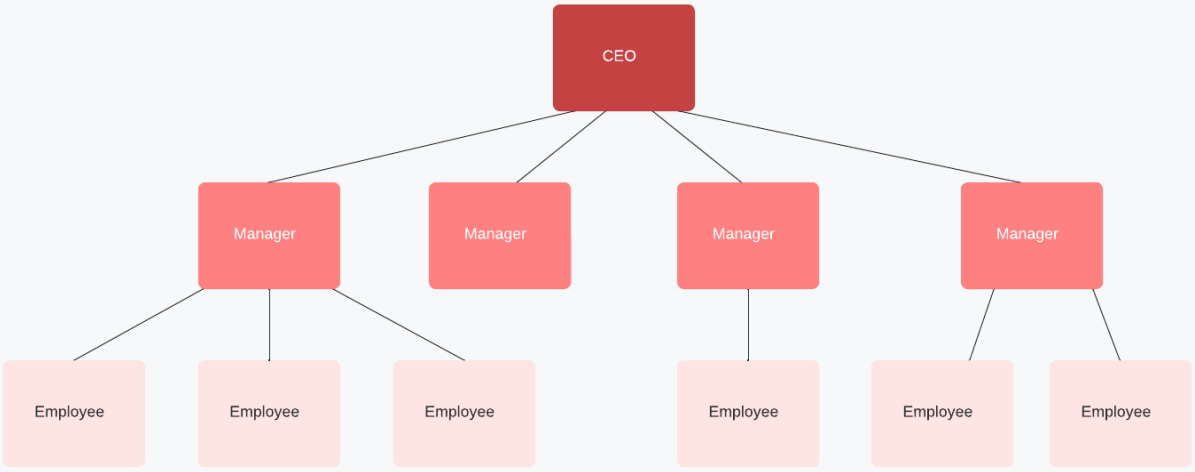
Author: Ryan Veitenheimer, Moorestown, New Jersey, United States

Problem Background

When working for a government contractor, you will find that requesting access to company resources often requires approval from your manager. Some examples of company resources include software installations, access to shared drives, access to applications on the intranet, budget allocation, etc. These types of requests can be automated to send an approval request to the employee's manager without human intervention. But the manager may not be available or may have missed the notification of the approval request. In that case, we would want the system to send the request to the next manager in the chain, the manager of the manager, and so on, until someone approves or denies the request.

Problem Description

Lockheed Martin wants to implement an automated manager notification system. You have been hired to program this system. The human resources department tracks the names of direct reports for each manager in the company, all the way up to the CEO. Each direct report has a single manager, but each manager may have one, zero, or multiple direct reports. The CEO will not have a manager. Some managers are newly hired and have not been assigned direct reports yet. The management structure can be represented as a standard hierarchal structure, like the one below.



Your goal with this program is to create the hierarchal management structure, then print out the entire line of managers for each employee. The data received from human resources will not be in hierarchical order of management. The output must be in alphabetical order by first name for each employee.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include…

- A positive integer, M, representing the total number of managers, including the CEO.
- M sections, each containing information for each manager, including:
  - A single line containing the name of the manager.
  - An integer, N, representing the number of direct reports for that manager.
  - N lines containing the name of each direct report.

```
1
3
Brett Shofiquel
5
Beth Burns
Brennan McCloud
Ibrahim Harvey
Jacob Allard
Shannon Takamoto
Thomas Takamoto
2
Brett Shofiquel
Enida MacConnell
Enida MacConnell
2
Jane Wong
Marjorie Zaremby
```

Sample Output

For each test case, your program must print the name of each employee, followed by the name of each manager in the hierarchy for that employee. The output will be sorted ascending by first name of each employee. Each name in the manager chain will be separated by a forward slash.

```
Beth Burns/Brett Shofiquel/Thomas Takamoto
Brennan McCloud/Brett Shofiquel/Thomas Takamoto
Brett Shofiquel/Thomas Takamoto
Enida MacConnell/Thomas Takamoto
Ibrahim Harvey/Brett Shofiquel/Thomas Takamoto
Jacob Allard/Brett Shofiquel/Thomas Takamoto
Jane Wong/Enida MacConnell/Thomas Takamoto
Marjorie Zaremby/Enida MacConnell/Thomas Takamoto
Shannon Takamoto/Brett Shofiquel/Thomas Takamoto
Thomas Takamoto
```

Problem 14: Time Troubles

Points: 55

Author: Brett Reynolds, Annapolis Junction, Maryland, United States

Problem Background

Time may not actually be money, but it's still very important when it comes to organizing our daily lives. Unfortunately, when working with other people around the world, times tend to be different. What's early in the morning in New Zealand is late at night in France. The world is divided into dozens of different time zones, and being able to work between these time zones is often a problem in computer programming.

Problem Description

Your team is working on a scheduling application that will be used at multiple Lockheed Martin sites around the world. Users of the application will want to see events displayed in their own time zone, but in order to keep everything organized, your application will store times of events in the UTC time zone (Coordinated Universal Time... the acronym is based on French, if you're wondering).

Each time zone in the world is defined using a UTC offset - the number of hours by which that time zone is ahead or behind of UTC. For example, most of the Eastern coast of the United States has a UTC offset of -4 hours for much of the year (written as UTC-4). This means that when it's 11:00 AM in the UTC time zone, it's 7:00 AM in the US Eastern time zone (11 - 4 = 7).

Your program will be given a time entered by the user, and the UTC offset of the user's time zone. You'll need to convert that time from the user's time zone into UTC for storage in your system's database.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include the following values, separated by spaces:

- The date of the event, formatted as MM/DD/YYYY, where "MM" is the two-digit month (January is 01, December is 12), "DD" is the two-digit day of the month, and "YYYY" is the four-digit year
- The time of the event, formatted as HH:MM, using the 24-hour clock and two digits for both numbers (00:00 is 12:00 AM, 23:59 is 11:59 PM)
- A number representing the offset of the user's time zone from UTC, in hours. This number may contain decimal values.

```
3
08/17/2019 12:00 8
08/17/2019 12:00 -4
08/01/2019 04:15 7.5
```

Sample Output

For each test case, your program must output the date and time of the event, converted to the UTC time zone using the given hour offset. The date and time must be formatted as "MM/DD/YYYY HH:MM" as shown in the input.

```
08/17/2019 04:00
08/17/2019 16:00
07/31/2019 20:45
```

Problem 15: Data Forest

Points: 60

Author: Vedant Patel, Stratford, Connecticut, United States

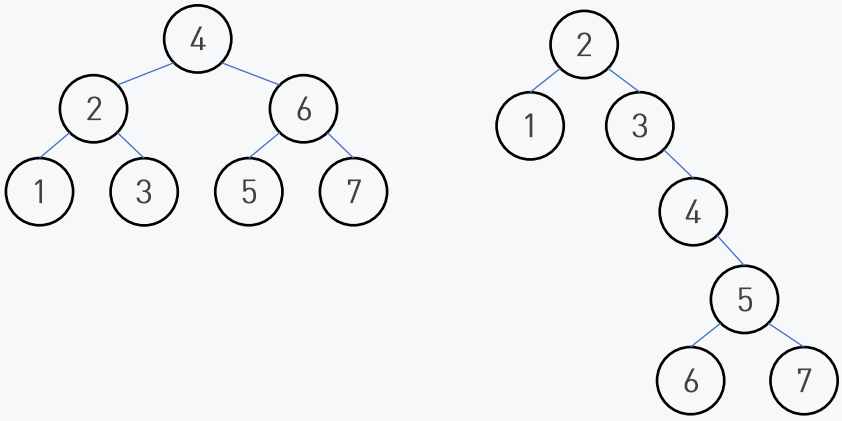
Problem Background

Being able to find data quickly is an important task for any software application. Huge amounts of research have gone into developing different data structures, and the specific data structure used by an application depends on a wide range of factors. For simple data that can be easily sorted, a binary search tree allows that data to be retrieved quickly and efficiently.

A binary search tree is simple to set up. Each node within the tree is associated with a value, and can have up to two child nodes: a left child and a right child. The left child (and its descendants) must have values less than the parent node's value; the right child (and its descendants) have values greater than that of the parent. Searching for a value therefore requires comparing your desired value against the root node of the tree, and moving down the appropriate branch until your target is found.

Problem Description

A binary tree must be built correctly for it to be as efficient as possible. For example, consider the two binary search trees below, which both include the integers from 1 to 7 inclusive:



The tree on the right is poorly built; if searching for 6 or 7, you'll have to do four comparisons before you find them. The tree on the left is much shorter; it will only take two comparisons at most to locate any within the tree. It's what we call "balanced." A balanced binary tree optimizes retrieval time by ensuring that no data is added to a new layer of the tree until the layer before it is filled. Most trees follow a set of rules to ensure that they remain balanced, even as data is added or removed.

You'll need to read in a list of numbers and build a balanced binary search tree from those numbers. For this problem, you'll be guaranteed to get enough data to completely fill the tree; that is, the bottom-most layer of the tree will be completely filled, as shown in the balanced tree diagram above.

Once you've determined how the tree should be laid out, you'll need to print out a visual representation of the tree similar to the above diagram. To ensure consistent indentation, each number within the tree should be printed using three characters, adding leading spaces as necessary. Spaces should be printed at the beginning of each line and between each number, in multiples of three, to ensure that each parent node is positioned directly between both of its children.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include a single line containing a list of unique integers with values between 0 and 999 inclusive, separated by spaces. The number of integers in each line will equal $2^N - 1$ for an undisclosed integer value of N between 1 and 9 inclusive.

```
2
1 2 3 4 5 6 7
106 115 103 111 108 101 104 113 105 110 109 112 102 107 114
```

Sample Output

For each test case, your program must print the structure of the tree, laid out across N lines, indented and spaced as described above. **Do not use tab characters to indent lines or separate numbers, and do not print trailing spaces on any line.**

```
           4
     2           6
  1     3     5     7
                    108
          104                   112
   102          106          110          114
101   103   105   107   109   111   113   115
```

Problem 16: Round It Out

Points: 65

Author: Brett Reynolds, Annapolis Junction, Maryland, United States

Problem Background

Numbers can be extremely precise, but very often we don't really need them to be that precise. For example, when working with money, we only need two decimal places, since no currency uses denominations less than one hundredth of its main unit. If a number is excessively precise, we need to "round" it to a less precise number. However, there are a large number of ways to round numbers, and how you do the rounding makes a very big difference.

You most likely learned the "half up" rounding method in school. In this rounding method, numbers are rounded to the nearest value of the desired precision. For example, if rounding to whole numbers, 0.2 rounds down to 0; 0.8 rounds up to 1. However, 0.5 is the midpoint between 0 and 1; with the "half up" method, the halfway point will round up, to 1. Note that whenever we say "round up" we really mean "round away from 0;" -0.5 rounds to -1, even though this number has a lesser value. Remember to pay attention to more precise values; 0.500000000001 rounds up to 1.

Other rounding methods follow different rules. For this problem, you'll need to appropriately apply those rules to a variety of situations.

Problem Description

Please note that this problem will ask you to round numbers in multiple ways. This problem overrules the general guidelines for rounding numbers provided at the beginning of the Problem Packet.

You will need to write a program that is able to implement the various rounding rules listed below to round provided numbers to a specific number of decimal places.

- Half Up (HU) - Round numbers to the nearest value of the desired precision. 5 rounds away from zero.
- Half Down (HD) - Round numbers to the nearest value of the desired precision. 5 rounds towards zero.
- Up (U) - Round all numbers to the next value of the desired precision, moving away from zero.
- Down (D) - Round all numbers to the next value of the desired precision, moving towards zero.
- Half Even (HE) - Round numbers to the nearest value of the desired precision. 5 rounds to the value that is evenly divisible by 2.
- Half Odd (HO) - Round numbers to the nearest value of the desired precision. 5 rounds to the value that is not evenly divisible by 2.

For example, if we are rounding the following numbers to one decimal place, the results with each rounding method would be:

Original Number	HU	HD	U	D	HE	HO
1.23	1.2	1.2	1.3	1.2	1.2	1.2
1.28	1.3	1.3	1.3	1.2	1.3	1.3
1.33	1.3	1.3	1.4	1.3	1.3	1.3
1.38	1.4	1.4	1.4	1.3	1.4	1.4
1.25	1.3	1.2	1.3	1.2	1.2	1.3
1.35	1.4	1.3	1.4	1.3	1.4	1.3
-1.25	-1.3	-1.2	-1.3	-1.2	-1.2	-1.3
-1.35	-1.4	-1.3	-1.4	-1.3	-1.4	-1.3

Again, the examples above are for rounding to one (1) decimal place. Rounding to zero (0) decimal places means rounding to an integer. Rounding to -1 decimal places means rounding to a multiple of ten.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include a single line with the following information, separated by spaces:

- A decimal number, X
- A one- or two-character code indicating the rounding method to be used:
    - HU = Half Up
    - HD = Half Down
    - U = Up
    - D = Down
    - HE = Half Even
    - HO = Half Odd
- An integer, P, indicating the number of decimal places to which X should be rounded

```
1.23 HU 1
1.23 HO 1
1.23 HD 1
5.0 U -1
5.0 HE -1
-4.205 D 2
```

Sample Output

For each test case, your program must round X to P decimal places using the indicated method and print a single line containing the resulting number. The decimal point should only be printed when needed. Trailing zeroes that occur after the decimal point should not be printed.

```
1.2
1.2
1.2
10
0
-4.2
```

Problem 17: Turing Interpreter

Points: 65

Author: Louis Ronat, Denver, Colorado, United States

Problem Background

Alan Turing is widely credited with creating the first mathematical model of a computer. His model, the Turing Machine, can be thought of as a mechanical machine which holds a strip of paper called a tape. Based on the machine's internal state and the data printed on the tape at the current position, the machine will write new data to the tape, move the tape forward or backwards one step, and update its internal state. The Turing Machine forms the basis of defining whether a machine is "Turing Complete," which is an important determination to be made about programming languages.

Problem Description

Turing Machines take in a set of instructions which describe what to do in each situation the machine might encounter. Each instruction includes five values:

1. $q_{current}$ - The current machine state in which the instruction applies
2. s_{read} - The tape symbol the machine is currently reading
3. s_{write} - The symbol that should be printed on the tape in that position (overwriting s_{read})
4. d - The direction in which to move the tape reader one position after printing (L for left, R for right)
5. q_{new} - The new machine state to set

These values can be combined into a compact form called a 5-tuple:

$$\{q_{current}, s_{read}, s_{write}, d, q_{new}\}$$

A set of 5-tuples forms the complete set of instructions for a Turing Machine, and is known as a Turing Table. A sample Turing Table and its interpretation is shown below:

5-tuple	If the machine is in state…	…and reads the symbol…	…print this symbol in its place…	…then move one space to the…	…and set this as the new machine state
{A, 0, 1, R, A}	A	0	1	right	A
{A, 1, 1, L, B}	A	1	1	left	B
{B, 1, 0, L, B}	B	1	0	left	B
{B, 0, 0, R, A}	B	0	0	right	A

Let's demonstrate how this machine works in practice. The tables below represent a tape being read by the Turing Machine described above. The black cell represents the position of the reader. The machine starts in state A.

0	1	1	**0**	0	0	1

The machine reads in a 0, and so executes the first instruction listed above. It writes a 1, then moves one position to the right, and remains in state A. After this first step, the tape now looks like this:

0	1	1	1	**0**	0	1

The machine is still in state A and is reading another 0. This causes the machine to execute the same instruction again. After the completion of that step, it again finds itself in the same situation, and repeats the first instruction one more time.

0	1	1	1	1	**0**	1
0	1	1	1	1	1	**1**

Now the situation has changed. The machine is still in state A, but now it's reading a 1. It now executes the second instruction: it writes a 1, then moves one position to the left, then sets the state to B.

0	1	1	1	1	**1**	1

With the machine now in state B, it reads a 1, and follows the third instruction: write a 0, move to the left, and remain in state B. This continues until the machine encounters a 0.

0	1	1	1	**1**	0	1
0	1	1	**1**	0	0	1
0	1	**1**	0	0	0	1
0	**1**	0	0	0	0	1
0	0	0	0	0	0	1

With the zero now reached, the machine executes its final instruction: write a 0, move to the right, and set the state to A.

0	**0**	0	0	0	0	1

We now find ourselves in the same situation we started in: in state A, reading a 0. The machine now enters an infinite loop, writing 1's to the tape as it moves to the right, then writing 0's as it moves to the left.

For this problem, you will need to implement your own Turing Machine that follows the instructions provided in a given Turing Table over a certain number of steps. You may operate under the following guarantees:

- The machines you implement will never move beyond the edges of the provided tape.
- Machines will always start in state A.
- Machines will always start in the exact middle of the tape, which will always contain an odd number of symbols.
- The provided 5-tuples will cover every possible scenario the machine may encounter.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include:

- A line containing a single positive integer, T, representing the number of instructions contained in the Turing Table
- T lines, each containing a 5-tuple with values listed in the same order as described above. The 5-tuple will be wrapped with curly braces ({ and }) and each value will be separated by a comma and a space.
    - Machine states ($q_{current}$ and q_{new}) will always be represented by uppercase letters.
    - Symbols (s_{read} and s_{write}) will always be single-digit non-negative integers.
    - The direction d will always be a capital L (for left) or R (for right).
- A line containing symbols separated by spaces, representing the initial tape given to the machine. The tape may contain any odd number of symbols greater than 1.
- A line containing a positive integer, X, indicating the number of steps to simulate for this machine.

```
2
4
{A, 0, 1, R, A}
{A, 1, 1, L, B}
{B, 1, 0, L, B}
{B, 0, 0, R, A}
0 1 1 0 0 0 1
8
8
{A, 0, 1, R, A}
{A, 1, 2, R, A}
{A, 2, 3, R, A}
{A, 3, 0, L, B}
{B, 0, 3, L, B}
{B, 1, 0, L, B}
{B, 2, 1, L, B}
{B, 3, 2, R, A}
3 3 3 3 3 3 3 3 3 0 3 3 3 3 3 3 3 3 3
20
```

Sample Output

For each test case, your program must print the state of the tape after X steps have been completed by the machine. Print the entire tape on a single line, and separate values on the tape with spaces.

```
0 1 0 0 0 0 1
3 3 3 3 3 3 3 2 1 0 0 0 0 3 3 3 3 3 3
```

Problem 18: Modem Mania

Points: 65

Author: Vedant Patel, Stratford, Connecticut, United States

Problem Background

Internet Service Providers (ISPs) are your gateway to the rest of the world. They provide the infrastructure and hardware necessary to allow your ever-growing list of internet connected devices the ability to actually connect to the internet. This internet connectivity is quickly becoming a crucial part of everyday life, a fact that's only been highlighted over the events of the past two years. As a result, many feel that ISPs should be treated as a utility service, like an electric company. If this happens, this may result in some changes in how ISPs operate their businesses.

Problem Description

Lockheed Martin has been contracted by Quickfire Internet Services to develop new hardware for their modems. Quickfire wants to change their billing model to charge users based on the number of devices they're using, rather than on the speed of their internet connection; they're planning to use the new modems to help with this process.

Internet traffic is managed using IP (Internet Protocol) addresses, which you've likely heard of. When you access the internet, your computer connects through your modem, which is assigned an IP address. This address is used by websites to send you data you've requested. However, IP addresses can be shared by multiple devices. Since your modem is actually what's connecting to the internet, it's what receives the IP address visible on the internet; behind the modem, on your local network, could be dozens of devices, all sharing that same connection. This also applies when companies like Lockheed Martin use proxy servers to manage traffic, or countries such as China or Qatar use firewalls to manage and censor internet access to their citizens.

Within these local networks, these devices are assigned internal IP addresses, but they are also identified by a unique MAC (Media Access Control) address. A MAC address is a series of twelve hexadecimal digits assigned by the device's manufacturer. You can think of an IP address as a mailing address used to ship someone a package; the MAC address is then the unique signature a person must give when receiving that package.

Quickfire wants their new modems to track the MAC addresses connected to them and transmit these addresses to their central billing server. Users will then be billed based on the number of unique MAC addresses connected to their modem. Each time a device attempts to access data on the internet, the modem will send a message to Quickfire listing the IP address of the modem and the MAC address of the device. They want your team to develop an algorithm to collate this data for billing purposes.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include:

- A line containing a single integer, X, representing the number of address pairings in the test case
- X lines, each containing an IPv4 address and a MAC address, separated by a space.
    - IPv4 addresses consist of four integers, between 0 and 255 inclusive, separated by periods (.).
    - MAC addresses consist of six pairs of hexadecimal digits, with each pair separated by a colon (:).

```
1
6
143.228.1.43 01:23:45:67:89:0A
192.197.82.189 BC:DE:F0:12:34:56
194.60.0.255 78:90:AB:CE:EF:01
192.197.82.189 23:45:67:89:0A:BC
143.228.1.43 DE:F0:12:34:56:78
143.228.1.43 90:AB:CD:EF:01:23
```

Sample Output

For each test case, your program must print one line for each unique IP address listed in the input, including:

- The IP address
- A space
- The number of unique MAC addresses listed for that IP address

IP addresses should be listed in ascending numeric order; if the first numbers within two IP addresses are equal, break ties with the second numbers, and so on.

```
143.228.1.43 3
192.197.82.189 2
194.60.0.255 1
```

Problem 19: Incoming!

Points: 70

Author: Jonathan Tran, Dallas, Texas, United States

Problem Background

Some applications are interested in finding the longest subarray of contiguous elements that contain the largest sum. For example, if the elements of the array contain the number of rich metal resources in a mining deposit, a mining company would like to focus on the group of contiguous elements with the largest sum of these rich metal resources.

Problem Description

You are located on a top-secret military base. You have received intelligence that an enemy force has launched multiple waves of attack drones, targeting your base. But the enemy does not know that the base has defense technology capable of detecting and destroying the drones. The detection system can not only detect the drones, but also calculate a threat level for each drone. Threat levels are determined based on the offensive and defensive capabilities of each drone. For example, some drones may have machine guns, while others have rocket launchers. The threat level for each drone ranges from 0 to 30, where 0 is a non-threat and 30 is the highest threat. The detection system has determined that each wave of drones have been launched in a contiguous horizontal straight-line formation, meaning that each drone is adjacent to one or two other drones.

The defense system is prepared to engage the drones, but the system is only capable of destroying up to 15 contiguous drones per shot. The program for the defense system currently has no way of determining which 15 drones to shoot first. But the command team has hired you to optimize the program by calculating the largest contiguous subarray within the wave which adds up to the highest sum. That is the largest subarray which has the highest total threat value, up to 15 drones. Once this subarray is found, the defense system will destroy the drones within the subarray, then the detection system will update the array of drones with the previous subarray removed. On the next shot the system will again detect and destroy the largest subarray with the highest threat total, and so on, until all drones within the wave are destroyed. This will be repeated for each incoming wave of drones.

There are two special cases which may arise that the command team has provided instructions for:

- Special Case 1: A group of less than 15 contiguous drones that have a higher threat value than any group of 15 drones.
    - This occurs when the highest threat subarray of drones has zero threat level drones on both the left and right side.
    - For example, if given this array of 16 drones: 0 8 16 11 25 4 17 13 14 17 24 5 0 22 29 0

        - The highest threat level subarray would be: 8 16 11 25 4 17 13 14 17 24 5 0 22 29, because the 0 values at the ends do not add any value to the total threat. But even though they are a non-threat, we still want to shoot them down.
        - This subarray only has 14 drones. We want to shoot down 15. So, which zero should we add to the subarray? Here is how you decide:
            - If there are additional elements remaining on the left side of the subarray, append those values to the shot, until you reach 15 drones.
            - If there are no elements remaining on the left of the subarray, append elements on the right side of the subarray to the shot, until you reach 15 drones.
            - If there are no remaining elements to append on the left and right side, then you can shoot down less than 15 drones. This should only occur on the final shot.
    - This process may increase the total threat value if there are more zeros on the right of the subarray, than on the left, or vice versa.
- Special Case 2: Two or more groups of 15 contiguous drones that have the same threat value.
    - This case should be considered after special case #1 has been accounted for.
    - If there are multiple subarrays of 15 drones that have equal total threat value and have the highest total threat value, we will shoot down the group of drones furthest from the left side of the wave.

Sample Input

The first line of your program's input, received from the standard input channel, will contain an integer representing the number of test cases. Each test case will contain:

- A line containing a positive integer N, representing the size of the wave.
- A line containing N integers separated by spaces, representing the threat levels of each drone in the wave

```
2
28
0 15 1 1 21 1 19 8 11 29 5 24 23 15 29 12 0 7 12 18 16 1 29 12 22 25 17 0
30
22 16 17 24 22 27 18 20 15 28 13 18 24 8 5 11 17 3 28 2 27 26 12 7 5 4 28 10 0 26
```

Sample Output

For each test case there will be a variable number of shots taken. For each shot, there will be eight output lines with descriptive labels; see the sample output provided below for those labels. The eight lines will contain the following data, in order:

- The wave number (X) and shot number (Y), in the format "Wave #X --- Shot #Y"

- The current elements in the wave, in the order originally presented in the input, but omitting any drones that have already been shot down
- The current size of the wave (the number of elements just printed on the previous line).
- The total threat value of the wave.
- The zero-based indices of the first and last elements to be targeted by the current shot.
- The elements that are being targeted, in the order originally presented in the input.
- The number of elements in the targeted range.
- The total threat value of the targeted range.

Include an additional blank line after each shot's report.

Due to the length of the sample output, it is provided on the next page to avoid breaking across pages. Note that some output may break across multiple lines due to its length; please download and consult the provided sample output file to view the output accurately.

```
Wave #1 --- Shot #1
Elements in wave: 0 15 1 1 21 1 19 8 11 29 5 24 23 15 29 12 0 7 12 18 16 1 29 12
22 25 17 0
Current size of wave: 28
Total threat value of wave: 373
Targeted range: Elements 11 to 25
Eliminated elements: 24 23 15 29 12 0 7 12 18 16 1 29 12 22 25
Number of elements destroyed: 15
Total threat value eliminated: 245

Wave #1 --- Shot #2
Elements in wave: 0 15 1 1 21 1 19 8 11 29 5 17 0
Current size of wave: 13
Total threat value of wave: 128
Targeted range: Elements 0 to 12
Eliminated elements: 0 15 1 1 21 1 19 8 11 29 5 17 0
Number of elements destroyed: 13
Total threat value eliminated: 128

Wave #2 --- Shot #1
Elements in wave: 22 16 17 24 22 27 18 20 15 28 13 18 24 8 5 11 17 3 28 2 27 26
12 7 5 4 28 10 0 26
Current size of wave: 30
Total threat value of wave: 483
Targeted range: Elements 0 to 14
Eliminated elements: 22 16 17 24 22 27 18 20 15 28 13 18 24 8 5
Number of elements destroyed: 15
Total threat value eliminated: 277

Wave #2 --- Shot #2
Elements in wave: 11 17 3 28 2 27 26 12 7 5 4 28 10 0 26
Current size of wave: 15
Total threat value of wave: 206
Targeted range: Elements 0 to 14
Eliminated elements: 11 17 3 28 2 27 26 12 7 5 4 28 10 0 26
Number of elements destroyed: 15
Total threat value eliminated: 206
```

Problem 20: Lost in the Mail

Points: 70

Author: Ben Fenton, Ampthill, Reddings Wood, United Kingdom

Problem Background

System Solutions, a division within Lockheed Martin Rotary and Mission Systems, helps to run postal sorting machines that sort around ¼ of the world's postal mail every day. They provide services to companies such as the United States Postal Service (USPS), Royal Mail, and PostNord.

Systems like these use various methods to track and correctly sort mail so it gets to the correct recipient. In the US, a system called intelligent mail barcode (IMb) is used to track all letter-based mail using a barcode which the sorting machines translate to route the post accordingly. This barcode system uses a series of long and short bars to encode information about who is sending some mail, what is being sent, how to send it, and where it's going.

Problem Description

You are a Lockheed Martin Engineer working with USPS on a new mail sorting system using a new version of the IMb system. You have been tasked with writing the program to translate the decoded barcode from the 27-digit payload into the data elements the system needs to route the letter to the correct recipient. The data payload is split into several elements, as described below:

- Digits 1-18: Tracking code, which can be broken down further as follows:
  - Digits 1-3: Service type identifier; for this problem, this will be one of these values:
    - 042 or 261 - Standard Class Mail
    - 040 or 300 - First Class Mail
    - 710 or 712 - Priority Mail
  - Digits 4-9: Mailer ID Code, which uniquely identifies the sender
  - Digits 10-18: Serial Number, which uniquely identifies the mail being sent
- Digits 19-27: Routing code, which can be broken down further as follows:
  - Digits 19-23: A five-digit ZIP code
  - Digits 24-27: A four-digit ZIP+4 extension code

This payload data is concatenated into a single 27-digit number; for example, a company with the mailer ID "123456" might generate the following payload for something they intend send via first-class mail to Lockheed Martin headquarters in Bethesda, Maryland:

040 123456 123456789 20817 1803

This number is then encoded into a 48-character string used to generate the barcode itself, using the following process:

1. Start with a value of 100,001
2. Add the 9-digit routing code to that value
3. Multiply the value by 50
4. For each of the 18 digits in the tracking code, from left to right, multiply the value by 10, then add that digit of the tracking code.
5. Convert the resulting value into a binary number
6. Add leading 0's to bring the binary value to a length of 96 bits
7. For each pair of bits in the binary value, from left to right, identify the resulting barcode character as shown in the table below:

Binary Value	00	01	10	11
Decimal Value	0	1	2	3
Barcode Character				
Description	Tracker	Ascender	Descender	Full Bar
Text Representation	T	A	D	F

This will result in a barcode consisting of 48 bars. This would be printed on envelopes using the barcode characters shown above, but for this problem, we'll represent these bars with the letters shown in the "Text Representation" row. The example payload given above will result in the barcode:

TDTADDAAFDDFAFFFFDDADFTAADATTFTAFTAAATADATFATAAA

Your task is to write a program which reads a series of the new IMb barcodes, decodes them, and compiles a set of statistics regarding the mail identified by those barcodes.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include:

- A line containing a single positive integer, X, representing the number of barcodes to be read in this test case
- X lines, each containing a single 48-character string containing the letters A, D, F, and/or T, representing each of the barcodes to be read

```
1
5
TDTADDAAFDDFAFFFFDDADFTAADATTFTAFTAAATADATFATAAA
DTTADAFFTTFTDFDTFAAADAAFATATAFFTFTFFDDDATTDTDFTA
TFAATTFTAFTAFAAADFDAAFAFFDDADDTTTFTTTDFFFFTAFDAA
AFDDFFDAFATATATTTFTFTTFFTAFFTFFAAAFADATTTAFTTTDA
TDTADDAAFDDFAFFFFFTFTTAAFFFFFADFTDTFFAADFTFDTTDA
```

Sample Output

For each test case, your program must print six lines of output:

- A line containing the text "Packages Sorted: #" where # is the number of barcodes processed
- A line containing the text "Standard Class: #" where # is the number of standard-class packages included in the test case
- A line containing the text "First Class: #" where # is the number of first-class packages included in the test case
- A line containing the text "Priority Mail: #" where # is the number of priority mail packages included in the test case
- A line containing the text "Most frequent mailer ID: #" where # is the six-digit mailer ID that occurred most frequently amongst the processed barcodes. In the event of a tie, list the mailer ID that appeared first in the list of barcodes.
- A line containing the text "Most frequent ZIP code: #" where # is the five-digit ZIP code that occurred most frequently amongst the processed barcodes. In the event of a tie, list the ZIP code that appeared first in the list of barcodes.

```
Packages Sorted: 5
Standard Class: 1
First Class: 2
Priority Mail: 2
Most frequent mailer ID: 123456
Most frequent ZIP code: 20817
```

Problem 21: Maxwell's Demon

Points: 75

Author: Louis Ronat, Denver, Colorado, United States

Problem Background

Maxwell's demon is a thought experiment posed by physicist James Clerk Maxwell which seemingly violates the second law of thermodynamics. According to that law, energy will only naturally flow from hotter (more energetic) areas to colder (less energetic) areas. Maxwell's demon is proposed as an entity who controls a "door" between two chambers containing gases of different temperatures, allowing energy to instead flow in the opposite direction.

Imagine placing a piston between the two chambers; the hotter gas has more energy and will push the piston towards the chamber with less energy. This results in a change in volume that will eventually equalize the temperatures, as the compressed gas will become hotter while the decompressed gas will become cooler. Once this equilibrium is reached, the piston will stop. Now imagine that below the cylinder there is a door which Maxwell's demon controls. As the temperature changes, the demon selectively admits molecules between the chambers to reverse the situation. The gas on the compressed side is now hotter than the gas on the decompressed side and the piston is pushed back the other way. If there is enough gas and the demon is fast enough, a reciprocating engine can be built without any outside driver to the system, granting free work, and a perpetual motion machine for the demon's nefarious purposes.

Problem Description

Your challenge will be to develop a demon that will partition a one-meter-long, one-dimensional container of gas into two chambers. The chamber contains a number of particles; you will be given their masses and initial velocities. You must model the movement of these particles as they collide with each other and the ends of the container. After a certain period of time, your demon will close his door at the halfway point within the container (0.5 meters). This will trap the particles on one side or the other, creating two chambers of (most likely) unequal energy levels. You must then calculate the energy level on each side of the demon's door.

Positions within the container will range from 0.0 meters to 1.0 meters. Note that because we're in one dimension, particles will constantly be colliding with their neighbors and the walls at 0.0 and 1.0 meters. All collisions between particles will be completely elastic; this means that when two particles collide, the combined momentum of those particles prior to the collision will be equal to their combined momentum after the collision. When a particle collides with a wall, it simply reverses direction; its velocity retains the same magnitude, but switches from positive to negative (or vice versa).

You might find these equations helpful as you solve this problem:

Terms and Units	Equations
$x = position\ (meters)$ $t = time\ (seconds)$ $m = mass\ (grams)$ $v = velocity\ \left(\frac{meters}{second}\right)$ $E = energy\ \left(millijoules = \frac{grams \times meters^2}{seconds^2}\right)$ $p = momentum\ \left(\frac{grams \times meters}{seconds}\right)$ $(something)_0 = initial\ something$ $(something)_f = final\ something$	$x_f = vt + x_0$ $E = \frac{1}{2}mv^2$ $p = mv$ $p_{a0} + p_{b0} = p_{af} + p_{bf}$ $v_{af} = \left(\frac{m_a - m_b}{m_a + m_b}\right)v_{a0} + \left(\frac{2m_b}{m_a + m_b}\right)v_{b0}$ $v_{bf} = \left(\frac{m_b - m_a}{m_a + m_b}\right)v_{b0} + \left(\frac{2m_a}{m_a + m_b}\right)v_{a0}$

Again, using these equations, model the interactions between all particles in the container until the stated time has elapsed. At that point, calculate the total energy of all particles on each side of the door located at 0.5 meters. If no particles exist on one side of the door, the energy on that side is 0.

One last important note: Normally we recommend that you do not attempt to round intermediate values when performing calculations, and that you only round the final results. For this problem, we strongly recommend that you round each particle's position and velocity to no fewer than six decimal places following each recalculation. Computers struggle to accurately store decimal numbers, and this will reduce the risk of errors resulting from this difficulty.

Sample Input

The first line of your program's input, received from the standard input channel, will contain a positive integer representing the number of test cases. Each test case will include:

- A line containing the following values, separated by spaces:
  - A positive integer, N, representing the number of particles in the chamber.
  - A positive decimal value, T, representing the time (in seconds) at which the "demon" will separate the chamber into two halves.
- N lines containing information about the particles. Each line contains the following values, separated by spaces:

- A positive decimal number representing the particle's mass in grams
- A decimal number between 0.0 and 1.0 inclusive representing the particle's initial position within the chamber, in meters
- A decimal number representing the particle's initial velocity in meters per second.

```
2
2 4.0
1.8 0.1 0.3
2.6 0.9 -0.1
4 4.0
1.1 0.3 0.4
4.3 0.1 -0.2
0.8 0.4 1.3
2.2 0.7 -0.1
```

Sample Output

For each test case, your program must print a single line, containing the word "LEFT" if the total kinetic energy of all particles on the left side of the door (from 0.0 meters inclusive to 0.5 meters exclusive) is greater than that on the right side of the door (from 0.5 meters exclusive to 1.0 meters inclusive) after T seconds have elapsed, or the word "RIGHT" otherwise. No particles will be located at exactly 0.5 meters when the stated time has elapsed.

```
RIGHT
RIGHT
```

Problem 22: ADFGVX

Points: 85

Author: Brett Reynolds, Annapolis Junction, Maryland, United States

Problem Background

In 1894, an Italian physicist named Guglielmo Marconi discovered a world-changing new technology - the ability to communicate wirelessly through the use of radio waves. Up to that point, all communication required some form of physical connection, either a messenger manually delivering a written message, or a telegraph operator transmitting signals over a wire using Morse Code. This had a tremendous impact on our society as well as many other areas.

Of interest to many countries was the military application of the radio; a unit equipped with a radio could remain in contact with their commanders at all times, and so could receive new orders and relay intelligence reports in real time. However, since anyone with an antenna could receive those same orders and reports - including the enemy - a secure encryption scheme became of paramount importance.

When World War I broke out just 20 years after Marconi's discovery, nations on both sides of the conflict grappled with this problem. At the time, most ciphers used either substitution (replacing letters or groups of letters with different letters/groups) or transposition (switching the order of letters or groups of letters without changing them); however, there were established methods of breaking ciphers that used either approach, and so these ciphers were insecure. Near the end of the conflict, Germany believed they'd found a solution to this problem: the ADFGVX cipher, which employed both substitution and transposition.

Problem Description

The ADFGVX cipher works in two steps: letters are substituted according to a grid, and then are transposed to further obfuscate the message. The key to the cipher consists of the layout of the substitution grid and a keyword which directs how the transposition should be performed. The end result is an encrypted message that uses only the letters A, D, F, G, V, and X. If you're wondering why the cipher uses those specific letters, it's because they're easily distinguishable from each other in Morse Code, reducing the risk that the message would be misunderstood and turned into gibberish.

To begin, a 6-by-6 grid is populated with the digits 0 through 9 and the 26 letters of the English alphabet, arranged in a random order. Each row and column of the grid is associated with one of the letters A, D, F, G, V, or X in sequence, as shown in the table on the next page:

	A	D	F	G	V	X
A	q	c	t	1	o	8
D	w	0	b	d	z	k
F	4	h	p	m	3	j
G	g	s	6	7	e	v
V	l	9	2	f	x	n
X	y	a	u	5	i	r

Next, each letter of the message is located within the grid and replaced with a pair of letters; the labels for that cell's row and column, respectively. For example, using the grid above, 'c' becomes 'AD' - the letter 'c' appears in row A and column D. "code quest 2022" would be encoded as "AD AV DG GV AA XF GV GD AF VF DD VF VF". While this looks like nonsense, it's still easy to break, and so the cipher adds a second step. A new grid is created now, with the top row filled in with the letters of a keyword. The rest of the grid is created by listing the individual letters in the semi-enciphered message from left to right and top to bottom:

M	A	R	T	I	N
A	D	A	V	D	G
G	V	A	A	X	F
G	V	G	D	A	F
V	F	D	D	V	F
V	F				

A	I	M	N	R	T
D	D	A	G	A	V
V	X	G	F	A	A
V	A	G	F	G	D
F	V	V	F	D	D
F		V			

The table on the left shows the populated transposition grid; the table on the right shows how the transposition is performed. Once the message is filled in completely, the columns are reordered to place the letters of the keyword in alphabetical order. The final encrypted message is then read from left to right, top to bottom. As a result, our original message "code quest 2022" is encrypted as:

DDAGAVVXGFAAVAGFGDFVVFDDFV

For this problem, your team will need to decrypt messages that have been encoded using the ADFGVX cipher. To decrypt a message, follow the encryption process in reverse. Good luck!

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include eight lines of text:

- The first six lines will contain six characters each (lowercase letters and/or numbers), representing the layout of the substitution grid used to encode the message.
- The seventh line will contain a single word in uppercase letters, which is the keyword used to perform the transposition step of the cipher. This keyword will not contain any duplicate letters.

- The eighth line will contain a message consisting only of the characters A, D, F, G, V, and/or X, representing the encoded message.

```
3
qct1o8
w0bdzk
4hpm3j
gs67ev
l92fxn
yau5ir
MARTIN
DDAGAVVXGFAAVAGFGDFVVFDDFV
1c62et
iljvm7
d8rgko
suabph
9y354z
fnxq0w
CODE
AFFDVGVDAFXXFFFGGGVFGVDXDGAAXGDAXD
3kcdqg
5iub1f
92me6a
h0l7ry
xto84s
znjwvp
QUEST
GVADDXDDDVAVFVXGFFGADAFFVXGVDVDVVGXDDGAVFGGFXVXG
```

Sample Output

For each test case, your program must print a single line containing the decrypted plaintext message in lowercase letters and numbers. Do not attempt to re-insert spaces or punctuation into the message.

```
codequest2022
cryptographyisfun
thisiscodequests10thyear
```

Problem 23: Time Is in Your Hands

Points: 90

Author: Anonymous Volunteer, Orlando, Florida, United States

Problem Background

With a global company like Lockheed Martin, we have employees scattered around the globe. Of course, we want to use the best people on each team to provide the best skills for our customers. As such, you could have team members in various locations around the world that need to meet virtually.

Lockheed Martin is hiring you to develop a scheduling application. Following agile development methods, you'll work towards smaller goals over the course of several short "sprints," gradually building up to a fully functional application. For your first sprint, your goal is to determine the local time for various members of a team.

Problem Description

For this problem, you'll be focused on an international team of eight members, listed below.

Name / Location	Time Zone	DST Starts	DST Ends
Elizabeth **Smith** Phoenix, AZ, USA	Arizona Time (UTC-7)	Not observed	Not observed
Jane **Sprey** Bethesda, MD, USA	Eastern Standard Time (UTC-5)	02:00, second Sunday of March	02:00, first Sunday of November
Freya **Anderson** Edinburgh, Scotland, UK	Greenwich Mean Time (UTC+0)	01:00, last Sunday of March	02:00, last Sunday of October
Alika **Bolade** Lagos, Nigeria	West Africa Time (UTC+1)	Not observed	Not observed
Ahmed **Hassan** Cairo, Egypt	Eastern European Time (UTC+2)	Not observed	Not observed
Sri **Agarwal** Mumbai, India	Indian Standard Time (UTC+5½)	Not observed	Not observed
Haruki **Sato** Tokyo, Japan	Japan Standard Time (UTC+9)	Not observed	Not observed
Rick **Thomas** Melbourne, VIC, Australia	Australian Eastern Std. Time (UTC+10)	02:00, first Sunday of October	03:00, first Sunday of April

All time zones are defined with an offset in hours from Universal Coordinated Time, or UTC. Converting between time zones is done by combining the offsets of both time zones and adding that

number to the time; for example, when it's midnight (00:00) in Bethesda (UTC-5), it's 10:30 in Mumbai (UTC+5½).

Certain time zones also observe Daylight Savings Time (DST) during the summer months. This adds an additional offset of UTC+1 to that time zone during the specified period. For example, in December, Melbourne has an offset of UTC+11, not UTC+10. This shift comes into effect immediately at the stated time and date; when DST starts in Edinburgh, the time jumps directly from 00:59 to 02:00; when it ends, it reverts from 01:59 back to 01:00 (causing 01:00 to occur twice that day).

For each meeting you'll be asked to schedule, you'll be given a team member's name, when they want to meet (in their local time), and with whom they want to meet. Then, for each of those people, you'll need to report their local time so they know when the meeting is. You'll have to compute the time across those different time zones, and will need to account for DST if it applies for a team member's location and time of the year.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include a single line of text, containing the following information, separated by spaces:

- The last name (surname) of the meeting organizer, as shown in **bold text** in the table above.
- The date of the meeting in the organizer's local time, expressed in YYYY-MM-DD format. All dates will be Saturday, January 1, 2022 or later.
- The time of the meeting in the organizer's local time, expressed in 24-hour HH:MM format.
- A list of one or more additional last names (surnames), indicating the desired meeting attendees. Names will be listed in alphabetical order and will appear as shown in **bold text** in the table above.

```
2
Sprey 2022-03-14 09:00 Agarwal Anderson Bolade Hassan Sato Smith Thomas
Sato 2022-03-29 15:45 Agarwal Anderson Bolade Hassan Thomas
```

Sample Output

For each test case, your program must print output in the following format:

- A line containing the phrase "<Organizer>'s Meeting:", replacing <Organizer> with the last name (surname) of the meeting organizer
- One line for each person attending the meeting, including the meeting organizer, listed in alphabetical order by last name (surname). Each line should include the following information, separated by spaces:
    - The last name (surname) of the attendee
    - The date of the meeting in the attendee's time zone, in YYYY-MM-DD format

- The time of the meeting in the attendee's time zone, in 24-hour HH:MM format

```
Sprey's Meeting:
Agarwal 2022-03-14 18:30
Anderson 2022-03-14 13:00
Bolade 2022-03-14 14:00
Hassan 2022-03-14 15:00
Sato 2022-03-14 22:00
Smith 2022-03-14 06:00
Sprey 2022-03-14 09:00
Thomas 2022-03-15 00:00
Sato's Meeting:
Agarwal 2022-03-29 12:15
Anderson 2022-03-29 07:45
Bolade 2022-03-29 07:45
Hassan 2022-03-29 08:45
Sato 2022-03-29 15:45
Thomas 2022-03-29 17:45
```

Problem 24: Flight Finder

Points: 95

Authors: Cindy Gibson and Danny Lin, Greenville, South Carolina, United States

Problem Background

Heading to the airport always seems to be such a hassle... finding a parking spot, getting your bags checked, going through airport security, and still making it to your flight on time is enough to get anyone grumpy. However, many people's complaints start long before they leave for the airport... when they try to book their flight in the first place! Airline tickets can be very expensive, and it's often hard to find an itinerary that works with your schedule and doesn't risk leaving you stranded due to a missed connection.

Finding a cheap and practical flight isn't just an issue for tourists, however; businesses often have to wrestle with those same problems. Large companies like Lockheed Martin have to develop policies about what sorts of flights their employees are allowed to purchase and work out fare agreements with major airlines in order to keep costs down. A contracting company like Lockheed Martin often includes the cost of any necessary travel in its bids for contracts, so being able to get cheap flights is crucial to keeping costs to the customer at an absolute minimum.

Problem Description

Lockheed Martin is considering a redesign to its business travel tool, and your team has been asked to work on a tool to identify the best flight itineraries for employees. Your tool will need to enforce Lockheed Martin's policies and risk management guidelines by only recommending itineraries meeting the following criteria:

- All flights must be booked with Lockheed Martin's approved carrier (this will be handled for you; you'll only have information for flights provided by this carrier).
- When a connection is required, the layover time (time between arrival of the first flight and departure of the second) must be at least 60 minutes. Overnight layovers (connections involving flights on different days) are not permitted.
- Employees should arrive either the day before or the day of the start of their event, and should depart either the day of or the day after the end of their event.
- When multiple itineraries meet the above criteria, the one with the lowest total cost should be selected. In the event of a tie, the itinerary with the latest outgoing date should be selected. In the event there is still a tie, the itinerary with the earliest return date should be selected.

Your team will be provided with the schedule of flights available from Lockheed Martin's approved carrier, and with details of trips that need to be made during that year. Given this information and using the criteria above, your program will need to identify the best itinerary for each trip.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include:

- A line containing two positive integers separated by spaces:
  - F, the number of flights available by the airline
  - T, the number of trips that need to be taken in the foreseeable future
- F lines detailing the flights available from the airline. Each line contains the following information, separated by spaces:
  - A positive integer, representing the flight number
  - A three-uppercase-letter code representing the departure airport
  - A time, in the 24-hour format HH:MM, representing the departure time
  - A three-uppercase-letter code representing the arrival airport
  - A time, in the 24-hour format HH:MM, representing the arrival time. This time will always be later than the departure time; no overnight flights will be listed.
  - A string of uppercase letters, indicating the days of the week this flight is held. Letters correspond to days of the week as follows, and will always be presented in this order:
    - S = Sunday
    - M = Monday
    - T = Tuesday
    - W = Wednesday
    - H =Thursday
    - F = Friday
    - A = Saturday
  - A decimal number representing the cost of a ticket on that flight
- T lines detailing the trips that need to be taken. Each line contains the following information, separated by spaces:
  - A three-uppercase-letter code representing the employee's home location
  - A three-uppercase-letter code representing the employee's destination
  - A date, in the format DD/MM/YYYY, indicating the start of the event
  - A date, in the format DD/MM/YYYY, indicating the end of the event

```
1
6 2
12 MCO 10:15 DFW 13:00 SMTWHFA 115.10
34 DFW 14:00 MCO 17:45 SMTWHFA 123.50
56 DFW 13:30 BWI 18:00 MWF 140.35
78 BWI 08:05 DFW 12:00 MWF 121.15
90 MCO 09:00 BWI 11:15 SA 290.50
123 BWI 12:30 MCO 15:00 SA 299.99
MCO DFW 26/04/2021 01/05/2021
MCO BWI 26/04/2021 30/04/2021
```

Sample Output

For each test case, your program must print information about the best itinerary for each listed trip. This information must include the following:

- A line containing the phrase "Departure to XXX on DATE", where:
  - XXX is replaced with the three-letter code of the destination airport
  - DATE is replaced with the date of departure in DD/MM/YYYY format.
- One line for each flight in the outgoing leg of the itinerary in chronological order, in the format "Flight NNN from XXX at TIME", where:
  - NNN is replaced with the flight number
  - XXX is replaced with the departure airport for that flight
  - TIME is replaced with the departure time of the flight in 24-hour HH:MM format
- A line containing the phrase "Return to XXX on DATE", where:
  - XXX is replaced with the three-letter code of the origin airport
  - DATE is replaced with the date of return in DD/MM/YYYY format.
- One line for each flight in the returning leg of the itinerary in chronological order, in the format "Flight NNN from XXX at TIME", where:
  - NNN is replaced with the flight number
  - XXX is replaced with the departure airport for that flight
  - TIME is replaced with the departure time of the flight in 24-hour HH:MM format
- A line containing the phrase "Total cost: $", followed by the total cost of all flights in the itinerary. Print two decimal places, including any trailing zeroes.

```
Departure to DFW on 26/04/2021
Flight 12 from MCO at 10:15
Return to MCO on 01/05/2021
Flight 34 from DFW at 14:00
Total cost: $238.60
Departure to BWI on 25/04/2021
Flight 90 from MCO at 09:00
Return to MCO on 30/04/2021
Flight 78 from BWI at 08:05
Flight 34 from DFW at 14:00
Total cost: $535.15
```

Problem 25: A Matter of 0's and 1's

Points: 100

Author: Richard Green, Whiteley, Hampshire, UK

Problem Background

You've probably heard of Sudoku puzzles, and may have even written a program to attempt to solve them. A slightly simpler version uses only 0's and 1's, and so is called Binary Sudoku.

As in Sudoku, the puzzle is made up of a grid of numbers in a square pattern. The challenge is to complete the grid in a way that fulfills a simple set of rules:

- Each box must contain a 0 or a 1.
- The same number may not appear more than twice consecutively within any row or column; for example, you may see 010 or 0110, but 01110 is illegal, as three 1's appear together without a 0 to separate them.
- Each row and column should contain an equal number of 0's and 1's.
- Each row is unique, compared to other rows.
- Each column is unique, compared to other columns.

As with Sudoku, Binary Sudoku boils down to a logic problem; using the rules above to make deductions about empty cells. Your task is to write a program that utilizes a series of deductions to determine how complex a particular puzzle is.

Problem Description

There are advanced methods a computer program can use to solve problems, but the aim here is to determine whether the presented problem can be solved by *people*. Your program should use the methods of deduction outlined below to determine if the puzzle can be solved with those methods alone, and if so, if it can be solved entirely with simple logic or if a degree of deduction is required. **Do not use any other methods of deduction when attempting to solve these puzzles.**

Simple Logic

There are three scenarios which, based on the rules outlined above, allow you to make immediate decisions regarding which number to place in a cell.

- Since a number cannot appear three or more times consecutively, any instance of doubled numbers in a row or column must have the opposite number on either side.

	1	1		→	0	1	1	0
0	0			→	0	0	1	

- Likewise, any time there is a single space between two of the same number, that space must be filled by the opposite number:

0		0	→	0	1	0		
1		1	→	1	0	1		

- Finally, since puzzles are square and each row and column must contain an equal number of 0's and 1's, the number of 1's (and 0's) in each row or column must be equal to half the dimensions of the puzzle. In a puzzle measuring eight cells to a side, a row or column must contain four 0's and four 1's. A row or column that already contains the required amount of one number must fill its remaining cells with the opposite number.

	1		0	1	1	→	0	1	0	0	1	1

Complex Logic

If the three methods above are insufficient to solve a problem, some more advanced methods may be required.

- Each row and column must be unique. If a row (or column) with two incomplete cells is otherwise identical to an already-completed row (or column), the incomplete cells must be filled with the opposites of their counterparts in the complete row (or column).

0	1	1	0	1	0	1	0	→	0	1	1	0	1	0	1	0
0	1			1	0	1	0	→	0	1	0	1	1	0	1	0

- Finally, you may be able to identify cases where a cell cannot legally contain a particular value. If a row or column has all but one of its required number of 0's or 1's, you can put the last remaining number in any open space to see if it is a valid move. In the example below, the row already has three of the four required 0's in place. Placing the final 0 in the rightmost cell is not possible, as filling the remaining cells with 1's would cause three 1's to appear in a row. Therefore, the rightmost cell must be a 1. When resorting to this method, be sure to only use the five rules listed in the Problem Background section to validate the current state of the puzzle. Do not attempt to recursively solve the remainder of the puzzle based on that guess if you can't immediately disqualify the entry.

0	0	1	0	1				→	0	0	1	0	1	1	1	0
0	0	1	0	1				→	0	0	1	0	1			1

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include:

- A line containing a positive even integer, N, representing the length of one side of the puzzle.
- N lines, each containing N characters, representing the puzzle itself. Characters may include the digits 0 or 1, representing pre-filled cells, or a period (.), representing a blank cell.

```
3
4
0.1.
..1.
...0
0...
4
0..1
0...
...0
...1
6
......
..0..1
..11..
....0.
.0...1
0....0
```

Sample Output

For each test case, your program must print the results of your program's effort to solve the given puzzle using the methods outlined above.

If a solution was found, your program must print:

- N lines, each N characters long and containing only 0's and 1's, representing the solved puzzle.
- A line containing one of the two following phrases:
    - "Solved with simple logic" if the puzzle was solved only using methods listed in the "Simple Logic" section above, or
    - "Solved with complex logic" if the puzzle required use of one or both methods listed in the "Complex Logic" section above at least once.

If the provided methods were not sufficient to come to a solution, print a single line with the phrase "Unable to solve with the provided logic".

```
0011
1010
1100
0101
Solved with simple logic
Unable to solve with the provided logic
101001
010011
101100
110100
001011
010110
Solved with complex logic
```

This page has been left blank intentionally.