Plano West Programming Contest

Wolf Invitational 2017



|  |  |  |  |
| --- | --- | --- | --- |
| # | Problem | Short Name | Points |
| 1 | **[Unfinished Business](http://drive.google.com/open?id=10Vb3-zrefbqobZGzRD7je5xWyfV7HpF0WlWh1K5sIlg" \t "_blank)** | business | 2 |
| 2 | **[Allies](http://drive.google.com/open?id=1VVO_H-pzL6E7vNh_37I9n_tHMcivPG1lYyvN7B4hHSw" \t "_blank)** | allies | 2 |
| 3 | **[Traitors](http://drive.google.com/open?id=1J0R-d3VAsycUkzNwp87rm1-BWKijgabF42asIkaLU-U" \t "_blank)** | traitor | 2 |
| 4 | **[Transmitting Messages](http://drive.google.com/open?id=1g_ZSLvfmkLbREfWGHQv01y2woMDRzR0Auam15n065e0" \t "_blank)** | messages | 3 |
| 5 | **[Gas Pump](http://drive.google.com/open?id=1S_rIkfpHtEEqp0MjANsUK5tOZiu45rTgd1FI9OB2uUo" \t "_blank)** | N/A | 3 |
| 6 | **[Super Decimals](http://drive.google.com/open?id=1hWlcxpnxPGqgeF5sYjXYNzbKRBugLn9uGHKinPahswU" \t "_blank)** | decimals | 4 |
| 7 | **[Base Conversion](http://drive.google.com/open?id=19GuXZG_EUKXxJnZH-Vp07WNjOHAAnuCz03Py8CGnzN4" \t "_blank)** | base | 4 |
| 8 | **[String Decoding](http://drive.google.com/open?id=1ueec0Q_ekYbim5eu1P561XbUid8SZT6cDwjvEKqXMKM" \t "_blank)** | decoding | 5 |
| 9 | **[The Name's Bond, Chemical Bond](http://drive.google.com/open?id=1pBw1bdqIMcXfD_dJ120gb1QlfOLv7H5nmjRUXPuS2_I" \t "_blank)** | bond | 5 |
| 10 | **[Position-Based Rotations](http://drive.google.com/open?id=1Xbjf-EtyyhGe1vp4HxiOmnCchqSFf2t8D4CbUUZkcvM" \t "_blank)** | rotations | 6 |
| 11 | **[Summation of Forces](http://drive.google.com/open?id=1Hr3Bdz2C74V0rUmGga7ajF3JrpRJSIW77fJ2fA60ZKw" \t "_blank)** | forces | 7 |
| 12 | **[Pyramid Cipher](http://drive.google.com/open?id=1HfHZpMluwrHVRcR_QMFykcPu2wD5gTN-XfEwY7eRfKw" \t "_blank)** | pyramid | 7 |
| 13 | **[Yazaxawh-wh-what??](http://drive.google.com/open?id=17tqhUPCE7X4uej2DmYLg5yRd2xZ7bxpFrj5sKMe0lvs" \t "_blank)** | yazaxa | 7 |
| 14 | **[Picking Locks](http://drive.google.com/open?id=1qdAYBlj2sXcCCwGCsghanu2FA73K2owHhb7Viq1p0aA" \t "_blank)** | locks | 8 |
| 15 | **[100000000000](http://drive.google.com/open?id=1qGjuMfuYPntnDVxWFHFUw8wjciR-mufFU3D49qRHZpU" \t "_blank)** | 2048 | 10 |
| 16 | **[Roads](http://drive.google.com/open?id=1Z77kqboR5_ZaOe7ONXzaK1XaxICOIPVVPiUhX8qX32Q" \t "_blank)** | roads | 11 |
| 17 | **[Bond’s Escape Route](http://drive.google.com/open?id=1Prh5PoPXn7DfPpu-yahTwMctCKDc6-AiJjInegX3GWM" \t "_blank)** | portals | 13 |
| 18 | **[Cashier Change](http://drive.google.com/open?id=1NqXmJszrhFP1vP-L0w70AURmQZaiE-V70fWrgWR7Ma0" \t "_blank)** | change | 14 |
| 19 | **[Bonding Bridges](http://drive.google.com/open?id=1jibcNS0Fa0JLxdK6vXD8eTYpxNXYxrbU2m80GjukN_8" \t "_blank)** | bridges | 15 |
| 20 | **[Garbage Collector](http://drive.google.com/open?id=1N5V8pr7zbodO-HDPAye7ynpUfz0tkh24_gAEt_jivqE" \t "_blank)** | gc | 16 |
| 21 | **[QR Matrix Reader](http://drive.google.com/open?id=1BUtzzzLulSBZaCkZWziiSI4gcszELrMv7MBtjVmlDQ4" \t "_blank)** | qr | 19 |
| 22 | **[Find X](http://drive.google.com/open?id=1HbDPUmNuap8D4tnCEzSbtkTGcpRphInKyjpClUJ1eNo" \t "_blank)** | findx | 20 |

# Unfinished Business (business.in) [2 Points]

James Bond has recently acquired several cool new gadgets, except he forgot to account for how much money he has left (he uses his special gift card to purchase everything, and he \*\*accidentally\*\* threw away the receipt). Based on a series of item prices, calculate how much money he has left in his deposit, so that he knows how much money he could spend on his lunch after a long day of spying and looking cool.

**Input**

The input starts with an integer, N, the number of items purchased and a decimal number, M, which is the total amount of money, where 0 ≤ |M| < 1000.

The next N lines each consist of any decimal number, P, where 0 ≤ |P| < 1000, representing the price of a purchased item.

**Output**

Output how much money is left over rounded to two decimal places with a dollar sign ($) in front. It is guaranteed to have a positive amount of money left over.

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 4 127.50  50.00  12.00  1.50  42.50 | $21.50 |



# Allies (allies.in) [2 Points]

Bond has a few allies in different countries.

* General Gogol
* Marc-Ange Draco
* Sheriff J.W. Pepper
* Tiger Tanaka
* Jack Wade
* Valentin Dmitrovich Zukovsky

He’s just found a list of people captured by Dr. No and needs to check whether his allies are on the list.

**Input**

The input starts with an integer, *C*, that is the number of people the follow.

Each subsequent line contains a name of a person captured

**Output**

The output should be the names of the allies that have been captured followed by " was captured!"

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 6  General Google  General Gogol  John Wade  Jack Wade  Tiger Zhang  Tiger Tanaka | General Gogol was captured!  Jack Wade was captured!  Tiger Tanaka was captured! |

# Traitors (traitors.in) [2 Points]

Each agent has a secret code. To ensure maximum security, M has made each code have a few special characteristics:

* The code must never be greater than 099 and always greater than 000
* The number should be divisible by 3, 5, or 7

Anyone whose code doesn’t follow those rules is considered a traitor.

**Input**

The input starts with a number, *C*, which determines the number of test cases to follow.

The next C lines will consist of one integer X, representing an agent code in the range of 0 < X < 105.

**Output**

Print out whether or not the agent is a traitor and not actually part of M’s team.

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 4  007  12345  034  035 | AGENT  TRAITOR  TRAITOR  AGENT |

# Transmitting Messages (messages.in) [3 Points]

James is spying on countries around the world, but his telecommunication device is not functioning properly, so the distance that his messages can travel is significantly limited. Given his current position, the position of the station he is communicating to, and the speed of the transmissions, determine whether the message can be successfully transmitted and if yes, how long it would take for it to be transmitted.

**Input**

The first line will consist of a single integer N, which gives you the number of test cases you need to check. For each one of the N test cases, there will be 6 integers: two sets of coordinates giving his current x, y position and x, y coordinate for the destination of his message in kilometers where 0 ≤ x, y ≤ 10,000, D, the maximum distance his messages can travel in kilometers where 0 ≤ D ≤ 15000, and C, the speed of the communication in kilometers per second where 0 < C ≤ 100,000. ***All numbers will be provided as integers.***

**Output**

For each test case, print It will take S seconds for his message to transmit, where S is the number of seconds it will take for the message to be transmitted rounded to the closest integer if it can be transmitted successfully, or His message cannot be transmitted. if James is outside of the transmitting range.

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 2  0 0 300 400 600 50  60 300 7700 2000 5000 1 | It will take 10 seconds for his message to transmit. *(Output shouldn’t wrap)*  His message cannot be transmitted. |



# Gas Pump [3 Points]

Everyone loves drawing ASCII art! Your task today is to draw the iconic James Bond logo, complete with 007 and “gas pump” (definitely not a gun).

**Input**

No input

**Output**

Look at sample output. Copy exactly as shown.

Note:There are no blank lines

|  |
| --- |
| Expected Output |
| \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\*\* =============================  \*\*\*\*\*\*\* \*\*\*\*\*\*\* \*\*\*\*\*\*\* ============  \*\* \*\* \*\* \*\* \* \*\* ( |  \*\* \*\* \*\* \*\* \*\* \_\_\_\_/  \*\*\*\*\*\*\* \*\*\*\*\*\*\* \*\*  \*\*\*\*\*\* \*\*\*\*\*\* \*\* |

# Super Decimals (decimals.in) [4 Points]

Jaws, a villainous henchman and one of Bond’s most familiar foes may have a frightening face - but when it comes to understanding instructions for his master's evil plots, he could use a little help.

When he receives Bond’s coordinates, it’s in a special code - a series of numbers separated by decimal places. To decode it, he has to add up each consecutive decimal number in the code. For example:

Quick, before Bond gets away - can you help him out?

**Input**

The input starts with a number, *C*, which is the number of test cases the follow.

Each test case contains < 104 numbers in the range of 0 < X < 105 separated by a decimal point

**Output**

The output must be rounded to 5 decimal places

**Assumptions**

The super decimal number and the individual numbers will always be positive.

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 2  55.66.77.88  12.34.56.78 | 200.31000  103.68000 |



# Base Conversion (base.in) [4 Points]

M needs to move agents from one military base to another; however, her agents’ codes are listed in different bases! She can find which agent belongs to which military base by what their code is in another numerical base.

When the agent’s number is in base 10, its remainder when divided by 12 gives the base number of the agent. So if an agent’s code number in base 10 is 124, that agent would belong in base 4.

**Input**

The input starts with a number, *C*, which is the number of test cases the follow.

The next C lines have two numbers, A and B, representing the Agent's code number, A, in base B.

0 < A < 231 - 1

1 < B < 11

**Output**

Determine where each agent belongs by converting their Agent code to the base code.

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 4  10 10  20 4  32 5  1000 9 | 10  8  5  9 |

**Explanation**

Input 1 - 10 in base 10 is 10 which when divided by 12 leaves a remainder of 10

Input 2 - 20 in base 4 is 8 which when divided by 12 leaves a remainder of 8

Input 3 - 32 in base 5 is 17 which when divided by 12 leaves a remainder of 5

Input 4 - 1000 in base 9 is 729 which when divided by 12 leaves a remainder of 9

# String Decoding (decoding.in) [5 Points]

On his last mission, Bond found some secret messages that he needs you to decode. Although he can’t decode the messages himself because he missed out on MI6’s computer science course, he found out the way that each message was encoded while spying. The method to decode each message is as followed.

* All letters within the string must be rotated forward by two positions (for example, ‘a’ becomes ‘c’, ‘b’ becomes ‘d’, ‘c’ becomes ‘e’ … ‘y’ becomes ‘a’, and ‘z’ becomes ‘b’.) Case remains the same for each letter shift.
* Any time there are three identical characters in a row, the middle letter must be deleted until there are no more groups of three (for example, if the phrase “aaa” appeared in the encoded string, you would have to convert it to “aa” and “aaaa” would become “aa” because removing one “a” leaves you with a group of three)
* If the last character in the encoded message is a ‘%’ character, the entire string must be reversed without the ‘%’ character (for example, “example%” (after the previous rules have been applied) would become “elpmaxe”)

**Input**

The first line will consist of an integer n, which will give you the number of test cases that you will need to decode. The n following lines will each contain an encoded string.

**Output**

Output each decoded message on its own line.

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 4  Fytc wms qmjtcb rfc npmzjck?  Wcq, G’k emmmmb.  .rqcr y rqsH%  Epclbcj jmtcq Ucyjrfcmu. | Have you solved the problem?  Yes, I’m good.  Just a test.  Grendel loves Wealtheow. |

# The Name’s Bond, Chemical Bond (bond.in) [5 Points]

Q is working on creating a top-secret formula that will enhance the speed and strength of his spies. Currently, he has several potential candidates that he would like to narrow down. He knows that mass must always be conserved in a chemical reaction, so he wants to eliminate the formulas that violate this rule. A reaction is said to be balanced if the sum of the masses of its reactants equal the sum of the masses of its products. Q will provide you with a list of chemical compounds that may be used and their respective masses. Can you help Q eliminate the bad formulas?

**Input**

The input starts with an integer, *N*, that is the number of compounds to follow. The next N lines consist of the name of the compound (with no spaces) followed by an integer, M, representing its mass. Next, there is an integer, C, that is the number of test cases to follow. Each case begins with two integers, R and P, which are the number of reactants and products respectively. The next line contains the reactants followed by an arrow (“->”) then the products. The reactants and products are both formatted the same way: an integer, A, is the number of that compound present followed by the name of the compound.

**Output**

On a separate line for each test case, print Possible candidate if the reaction is balanced or Not possible if the reaction is unbalanced.

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 7  H 1  O2 5  Au 10  C6H12O6 15  H2O 18  CoHN 42  ArI 115  3  2 2  2 O2 5 H -> 1 Au 1 O2  1 1  5 C6H12O6 -> 2 CoHN  3 2  5 Au 1 O2 5 H -> 1 H2O 1 CoHN | Possible candidate  Not possible  Possible candidate |



# Position-based Rotations (rotations.in) [6 Points]

In order to communicate with another secret agent, James has to encrypt a message then send it out so that if the message gets intercepted, no one will know how to decrypt it. James remembers one of the oldest ciphers (algorithm for encrypting and decrypting text) in the book is a rotation or the Caesar’s cipher. In this cipher, one would increment each letter in a text by a certain value called a key value, and Z will cycle around to A, hence the letters are “rotated” around. However, James realizes the issue with this cipher is that one could guess what the key is by using letter frequencies, circumventing the key value. This would prove to be an issue, so James devises a machine that would be initialized to a rotation amount, and this will increment for each letter. For example for a key of value 5, the first letter would be rotated 5 letters forwards, and then the second letter would be rotated 6 letters forwards, and so on. For example, the message AAAAA would be encrypted as FGHIJ. Given a particular plaintext and a key value, help James encrypt each message.

**Input**

The first line consists of an integer N, which determines the number of test cases to follow, each on one line. Each subsequent line will first contain an integer K () and a plaintext to encrypt where K is the key value to encrypt with (separated by a space).

**Output**

The output would consist of N lines where each line is the encrypted text of the corresponding test case.

**Assumptions**

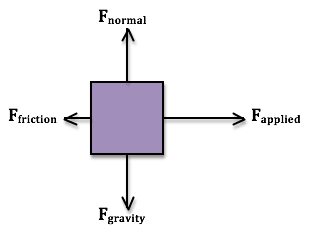
The plaintext contains ONLY UPPERCASE letters and consists of no spaces or punctuations.

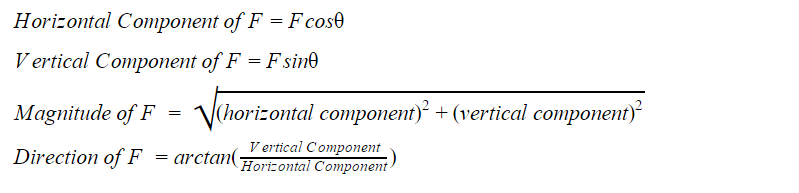
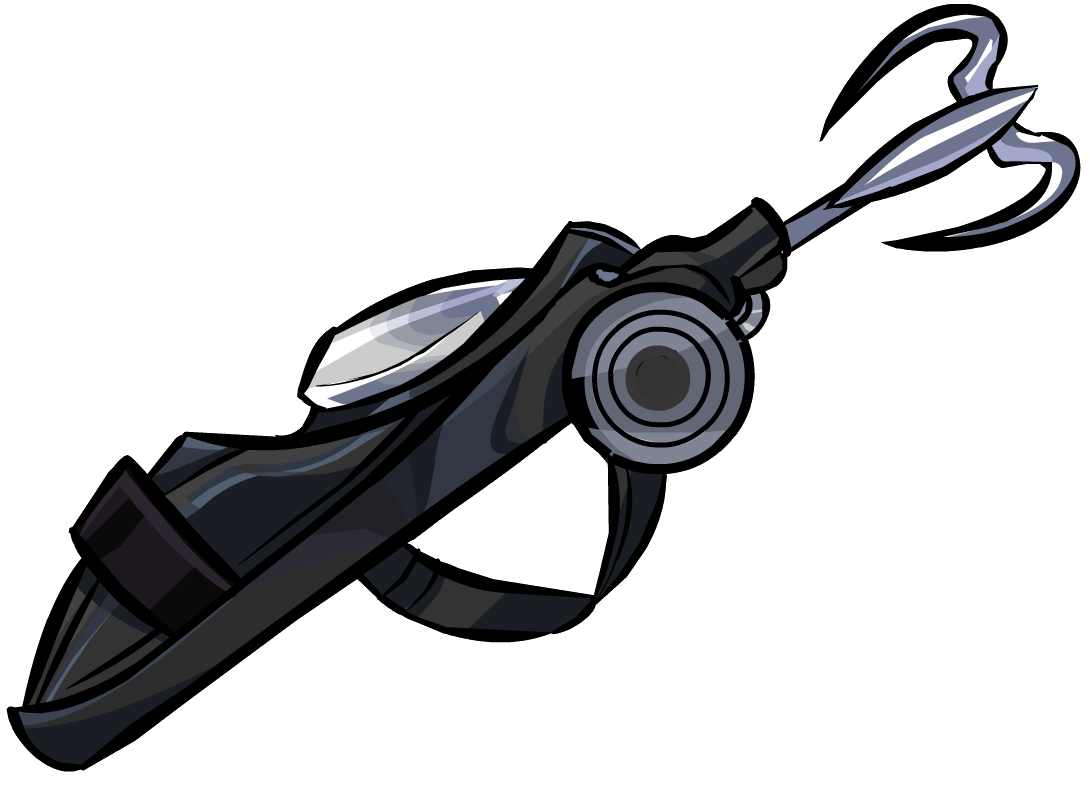
|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 4  0 CIPHER  3 ROTATION  15 ENIGMA  25 CRYPTOSYSTEM | CJRKIW  USYGAQXX  TDZYFU  BRZRWSXEZBNW |

# Summation of Forces (forces.in) [7 Points]

Being an agent takes more than just stunning good looks and a few catchy one-liners - when 007 is on a mission, there’s a lot he has to take into account. For a birthday gift, Q wants to ease the load on his mind a little by adding a special gadget to his grappling hook - something that can calculate the forces so he knows whether or not he’ll make it over that gap or up that wall. Here are the basics of the problem:

Given several forces (tension of grappling hook, gravity, drag) that act upon an object (in this case, Bond himself), determine the magnitude and direction of the net force applied. Remember, forces in opposite directions can cancel each other out. Java measures angles in radians, where

Relevant Equations:



**Input**

The first line consists of an integer, F, which determines the number of forces to follow, each on one line. Each subsequent line contains an integer, M, representing the magnitude of the force and an integer, A, representing the angle in degrees.

**Output**

Output the magnitude and direction of the force on a single line. Round the magnitude to 3 decimal places and direction in degrees with 2 decimal places. It should look like: M @ A degrees (replace M with magnitude and A with angle).

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 3  10 90  5 270  12 0 | 13.000 @ 22.62 degrees |

**Explanation**

First, add up the separate horizontal and vertical components of the net force.

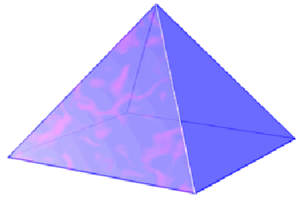
Horizontal:

Vertical:

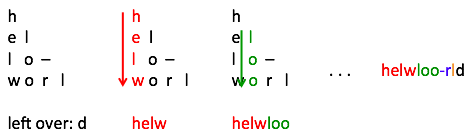
Next, the magnitude can be determined using:

Finally, the direction can be determined using: (converted back to degrees)

# Pyramid Cipher (pyramid.in) [7 Points]

007 needs your help decoding secret messages! The pyramid cipher takes a string of plaintext and encrypts it by creating a pyramid out of its characters. Starting from the beginning of the plaintext, the first character becomes row 1, the next two characters become row 2, the next three become row 3, and so on, until you cannot create another full row. After creating the pyramid, you must traverse it to create your encrypted message. Starting from column 1, traverse straight down the pyramid from each column from left to right. Finally, append any remaining letters that did not fit into the pyramid to the end.

For example, the string “hello-world” (ignore quotes) would be encrypted as follows:



After each step you have:

1. The pyramid to work with
2. Column 1: helw
3. Columns 2-4: loo-rl
4. Add left over letters: d

**Assumptions**

The plaintext strings can contain both capital and lowercase letters, numbers, spaces, and the following characters: . ? ! , : - ' " ( ) { }

**Input**

The first line consists of an integer, N, which determines the number of test cases to follow, each on one line. Each subsequent line contains a sentence to encrypt.

**Output**

Output the encrypted message each on one single line.

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 2  hello-world  Programming is fun! | helwloo-rld  Prgmgorm aiins fun! |

# Yazaxawh-wh-what?? (yazaxa.in) [7 Points]

In order to pass a secret message to a fellow agent, Q devises a new number system called Tercesa (if you reverse the letters…). Q first finds the prime factorization of a number and writes it in this form: 2x \* 3y \* 5z \* ...

For the exponents, he substitutes as such:

za = 0

ya = 1

xa = 2

wa = 3

va = 4

ua = 5

...

Then, Q will reverse the syllables so that there will never be a leading zero. For example, 168 can be written as 23 \* 31 \* 50 \* 71. This can then be written as wayazaya. Finally, it is reversed, so the 168 can be represented as yazayawa. You need to write a program that helps decode the Tercesa numbers and convert them into base-10 integers.

**Assumptions**

There are no prime factorizations that have any exponents greater than 25, and all numbers will be greater than 2 and less than 263 - 1. The Tercesa language will not be longer than 200,000.

**Input**

The first line consists of an integer N, which determines the number of test cases to follow, each on one line. Each subsequent line will contain the Tercesa representation of the number.

**Output**

Output the integer value that the Tercesa number represents.

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 4  yaza  yazayawa  uazawa  yazazazazazazazazazazazaza | 3  168  25000  41 |

# Picking Locks (locks.in) [8 Points]

As a spy you frequently encounter locks that need to be picked. However there are often a bunch of random symbols that are used to decode this lock. Luckily, Q knows the length of the lock so you just need to figure out how many combinations of symbols our lockpick will need to test. Q doesn’t really like big numbers either, so he chooses to mod the total number of combinations by his favorite number 1,000,007.

**Input**

The input starts with an integer, *C*, which is the number of test cases the follow.

Each line contains two integers, N and R, which are the number of symbols to choose from and the length of the passcode lock.

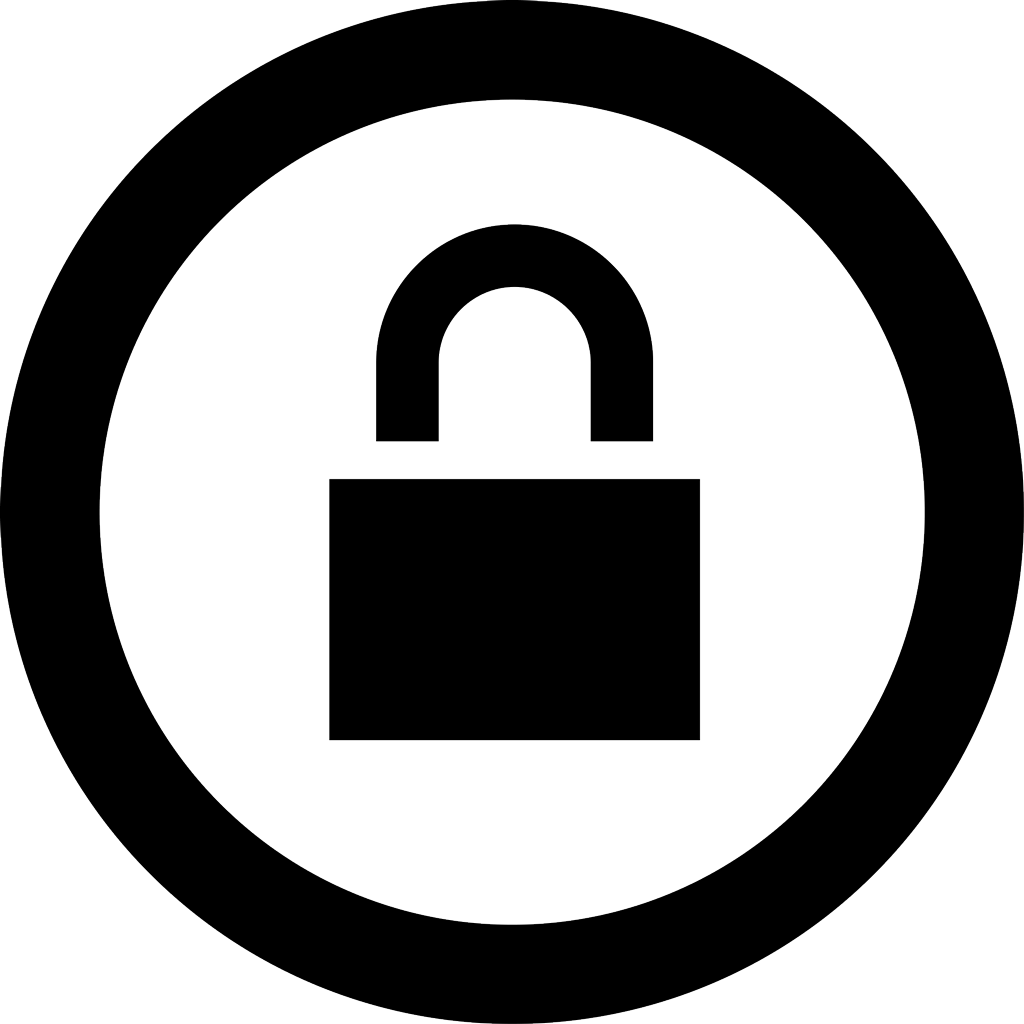
0 < N < 101

0 < R ≤ N

**Output**

The output is the number of possible passwords for the lock.

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 2  10 5  3 2 | 252  3 |



# 100000000000 (2048.in) [10 Points]

Everyone likes to play games every now and then. Bond’s favorite game is “100000000000” (a variant on the popular game 2048). As a refresher to those who played 2048, or to those who never played it, we will briefly explain the game. The game 2048 requires you to slide blocks over to “combine” them.

|  |  |  |  |
| --- | --- | --- | --- |
| 2 |  |  | 2 |
|  |  | 4 | 4 |
| 8 |  |  |  |
|  |  |  |  |

When the right key is pressed all blocks slide over to the right. If two blocks that share a number end up colliding then they combine into one block and double their value.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | 4 |
|  |  |  | 8 |
|  |  |  | 8 |
|  |  |  |  |

The new grid is shown above. When three blocks that share a number end up colliding, only the first two blocks combine and double their value. With four blocks, they will pair up two and two to form two blocks with doubled value.

However in this variation of 2048 all the numbers have been put into scrambled bases! This proves an additional challenge for James Bond. Below shows the order of the digits: 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

For example, for base 16 the digits will be from 0 to F, and then base 37 would utilize digits from 0 to a.

**Input**

The input starts with a number, *C*, which is the number of test cases the follow.

Each test case contains four lines of input with four tokens. Each token contains an integer, *a*, in base *b (1 < b* ≤ *62)* in the format shown

*a b*

Numbers can only be combined if they’re of equal value in base 10

A *-* (dash) signifies an empty square. The sample data contains extra spaces and extra new lines for ease of reading. The actual data would only contain one space between tokens and would not contain empty lines.

**Output**

The output must be the maximum number of “combinations” that can occur within one move.

|  |  |
| --- | --- |
| Sample Input (Extra spaces and new lines provided for ease of reading) | Expected Output |
| 2  2 10 - - 2 10  4 10 - - 4 10  -    - - -  8 10 - - 10 8  11 7 11 3 2 4   -  13 5 4  5 -     10 2  12 6 -    10 16 G 17  8 10 4  7 -     100 2 | 3  3 |

# Roads (roads.in) [11 Points]

Dr. No must build a road to facilitate the spread of his evil minions. For some reason, the area they want to place roads are populated with spiky mountains. To keep the road straight, the construction workers need to level the mountains to be the same height. They cannot simply build a tunnel because all workers have a phobia of tight, dark space. The area is mapped using a table of integers like so:

32101

4321T

553T2

43211

33101

Where each number represents the average height for each cell of land. If the average height between two horizontally/vertically adjacent cells are equal, a road can be constructed in between. There are no such thing as diagonal roads.

Two towns are located on the cell with the letter T. Both of them are in a cell with the same average height.

An operation is defined as removing or adding one unit of height to each cell of mountains. Calculate the minimum operations needed to connect one town to another with flat roads. There will always be two towns.

**Input**

The first line consists of an integer N (1 ≤ N ≤ 1,000), which determines the number of test cases to follow. For each test case, there are three integers H (-25 ≤ N ≤ 25), R (2 ≤ N ≤ 100), and C (2 ≤ N ≤ 100) for the height of the town, number of rows, and columns. The next R lines will be the average height of mountains in C cells.

**Output**

For each test case, output the minimum number of operations needed to connect the two towns together.

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 1  0 6 5  321T1  43210  55302  43211  33101  T2223 | 7 |

In the first sample test case, a road was built in these underlined cells. All underlined cells were leveled to average height of 0:

321**T**1

432**1**0

553**0**2

432**1**1

33**10**1

**T22**23

# Bond’s Escape Route (portals.in) [13 Points]

Even harder than getting into your nemesis’ lair is getting out of it. Luckily, a few of the Bond Girls have decided to help our favorite agent out by installing some portals - but this lair is so twisted that Bond might need even more help.

As is the case with regular mazes, Bond may not go outside the bounds of the maze or through walls. Each portal pair will have a different number from 0-9. In this maze of a lair:

* S = Start
* F = Finish
* 0-9 = portal, where each number corresponds to a different portal pair.
* X = a wall
* . = space where Bond can walk

If Bond lands on a portal (it has a number) he will immediately move to the other portal of that same number. It is always guaranteed to have exactly two portal of same number (if there are any). Going through a portal is only counted as one step (the step Bond used to go into the portal)

**Input**

The first line consists of an integer N, which determines the number of test cases to follow, each on one line. Each test case starts with an integer L, the number of levels to follow, and integers R and C, the number of rows and columns of each maze (it’s the same for each level). The next L mazes consist of R lines with C characters followed by a single line of space to separate mazes.

**Output**

The minimum number of steps it takes to go from start (S) to finish (F). If you cannot exit the maze, output Impossible, Cheater! instead.

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 2  3 5 5  S...X  ..X.X  .1XXX  ...XX  XXX2F  ..0..  X...X  XX.XX  XX..2  XX...  0...X  XXX..  XX.X.  XX...  1..XX  4 3 4  ..S.  XX..  2...  F.XX  X..X  XX.1  0XXX  ....  X1XX  ....  0X2.  .... | 21  16 |

# Cashier Change (change.in) [14 Points]

Paying coins at the cashier is a pain, but Miss Moneypenny only seems to have a pocket of various types of coins (especially when all she wants is just a few dollar bills). Miss. Moneypenny wants to handle the least number of coins as possible (and also receive the least number of coins back from the cashier). So given a value that Moneypenny needs to pay to the cashier, and given the types of coin denominations Moneypenny and the cashier have, find the least number of coins needed in transaction (whether in paying the cashier, or in the change the cashier gives back to Moneypenny). The catch is, depending on where Moneypenny is, the coin denominations happen to change a whole lot (for example, in some parts of the world Moneypenny goes to, there’s a 3 cent coin?)

**Input**

The first line consists of an integer N (0 < N < 25), which determines the number of test cases to follow. For each test case, the first line consists of three space-separated integers, P(0 < P < 500,000), the amount Bond needs to pay, J (0 < J < 1,000), the number of coin denominations James Bond has, and C(0 < C < 1,000), the number of coin denominations the cashier has. The second line consists of J integers listing the coin denominations of James Bond, and the third line consists of C integers listing the coin denominations of the cashier. Each coin denomination, Ci is in the range of (0 < Ci < 1,000). There will always be a one-cent coin.

**Output**

For each test case, print out the minimum number of coins handled.

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 2  43 4 1  1 5 10 25  7  164 5 4  1 7 15 35 10  4 10 3 7 | 3  7 |

For the first test case Bond can give two 25-cent coins and in exchange receive a 7-cent coin totaling to 43 cents in 3 transactions.

For the second test case Bond gives five 35-cent coins and in exchange receives a 7-cent and a 4-cent coin totaling to 164 cents in 7 transactions.

# Bonding Bridges (bridges.in) [15 Points]

After being chased down by his nemesis Goldfinger, James Bond discovers a tall nondescript building standing in front of him. With little options, James decided to enter the building. Inside he sees a network of different bridges leading to different sites. Goldfinger is gaining on him right behind, James Bond decides that one way to escape the building is to walk across every single bridge **exactly once** whilst Goldfinger is chasing right behind him, and then slam the external door on Goldfinger’s face right after he walks across the last bridge. Can he do such a feat? ***This problem is a timed problem. Maximum Execution Time: 5 seconds.***

**Assumptions**

James Bond always starts at site 0, and must come back to site 0.

**Input**

The first line consists of one integer T (0 < T ≤ 100), the number of test cases to follow. For each test case, the first line consists of integer N (1 ≤ 0 ≤ 500), number of sites the bridges could connect to and integer X (0 ≤ B ≤ 1,000), number of bridges to follow. Each of the next X lines will consist of two space-separated integers A, and B in which A and B represent the two sites this bridge will connect and 0 ≤ A, B < N.

**Output**

Print In your face! if James Bond could accomplish this feat, or Oops I'm Lost. if Bond could not accomplish this feat.

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 2  4 7  0 1  0 1  1 2  1 2  2 3  1 3  0 3  4 6  0 1  1 3  3 1  1 2  2 3  3 0 | Oops I'm Lost.  In your face! |

# Garbage Collector (gc.in) [16 Points]

Q is writing a new version of Java called Java 007 to help develop new software for Bond. He’s implementing Java’s garbage collector, which cleans and deallocate memory of unused objects. When an object is recently created, it has a strong reference to whichever function created it. Once that function is exited, that strong reference is removed. If that object is an instance variable, it keeps a strong reference to the parent object, so long as the instance variable still refers to the object. Any objects that can be directly transversed at any time is referred to as a GC Root. This means it is actively in use. Any objects that cannot be traversed via a GC Root will become eligible for garbage collection. In the Sun’s Java 2, there was an introduction of different types of references (Soft, Weak, and Phantom) that can prolong the life of an object beyond eligibility for garbage collection. Soft references will only get garbage collected when the JVM runs out of memory. Weak and Phantom references don’t actually prevent being garbage collected, but it will serve as a notification when they get garbage collected. You will create a simple garbage collector that stops “execution” during each frame and checks whether if any objects are out of reach (yes, JVM’s current garbage collectors are much more complex than this). Each frame contains three different types of instructions:

CREATE X S This creates a new object with a positive valued address of X (0 < X ≤ 100) and a size of S (0 < S ≤ 1,000,000). By default, these newly created objects will have a strong reference with the GC Root. The address X will always be unique.

REF TYPE X Y This creates a bi-directional reference of TYPE, which can be STRONG, SOFT, WEAK, or PHANTOM, between objects X and Y. Y must always be positive. However, X can either be positive or -1. If X is -1, it would mean that object Y is now a GC Root. Any previous references between X and Y is composed on top of this reference.

UNREF X Y This destroys all existing reference between objects X and Y. Again, X can be -1 or positive, but Y must be positive. If X is -1, it means to remove object Y from GC Root.

Once an object cannot be strongly or softly accessible to any GC Roots, that object will immediately be collected and removed. However, whenever the JVM needs more memory (whenever it creates more objects) than possible in the heap, or when it sees fit (when the memory reaches a certain threshold), those objects that are softly accessible will also be garbage collected. For simplicity, you should only garbage collect softly accessible objects when you run out of memory. If the JVM runs out of memory at any time in the execution, it should stop at that frame.

**Input**

The first line consists of one integer N, the number of test cases to follow. The each test case, the first line consists of an integer F (0 < F ≤ 100), which determines the number of frames following this line and integer S (0 < S ≤ 1,000,000), which represents the max heap size (max size of objects that we can hold at any one time).

**Output**

This output will consist of which objects this GC can collect after and before each frame has been executed (with CREATE, this GC would garbage collect before execution if the JVM would run out of memory when allocating this new object). Each frame that has any new objects to be garbage collected, print out the frame number, then a colon space (“: “) then print out a space-separated numerically-ordered list of objects being garbage collected, eg (“5: 3 4 7 8 9 10”). If at anytime, the JVM runs out of memory, you will print the frame number, then a colon space then the “OUTOFMEMORYERROR”

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 1  15 100  CREATE 1 11  CREATE 2 22  CREATE 3 31  CREATE 4 20  REF STRONG 1 4  UNREF -1 4  REF WEAK 1 2  UNREF -1 2  REF STRONG 3 4  UNREF -1 3  CREATE 5 11  REF SOFT 5 4  UNREF -1 5  CREATE 6 50  REF STRONG 6 1 | 8: 2  14: 5  14: OUTOFMEMORYERROR |

**Explanation of sample data:** In this data we first created four objects, and it used up 84 bytes of heap space. Then we created a strong reference between object 1 and 4 and made object 4 no longer GC Root. We created a weak reference between objects 1 and 2 and made object 2 no longer GC Root. This doesn’t prevent object 2 from garbage collection, and hence, 2 is listed at frame 8. Now our heap space has 62 bytes used up. We made a strong reference between 3 and 4 and made object 3 no longer GC Root. Even though object 3 doesn’t have any direct links to any roots, we could traverse through object 4 to object 1. We create object 5 and refer to it softly between objects 5 and 4. Then object 5 is no longer GC root. Right now, it is untraversable via strong references, but it could be traversed with strong and soft references. Finally, we create object 6, which completely blows up our heap, so we first try to free up as much memory as we can (remove all softly linked objects) and hence remove object 5. We still can’t get enough resources and terminate with an OUTOFMEMORYERROR.

# QR Matrix Reader (qr.in) [19 Points]

Oh no! Q’s super-high-tech computer had been hacked! Q was lucky the hackers were just toying with him. The hackers encrypted some of Q’ documents into a binary matrix, resembling like a QR code (not quite though). For your ease, Q has converted each file into a string matrix with 0s (represents the white dots of a QR) and 1s (represents the black dots of a QR). However, there are some extraneous data surrounding the binary matrix. In order to read the matrix correctly, the program needs to read the markers. The markers for the binary matrix are found on the top left corner, the top right corner, and the bottom left corner as shown below.

1  1  1  1  1  1  1

1  0  0  0  0  0  1

1  0  1  1  1  0  1

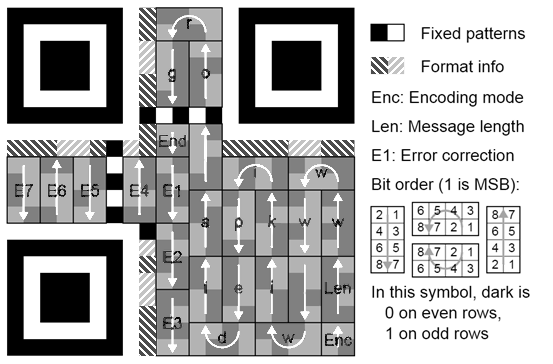
1  0  1  1  1  0  1

1  0  1  1  1  0  1

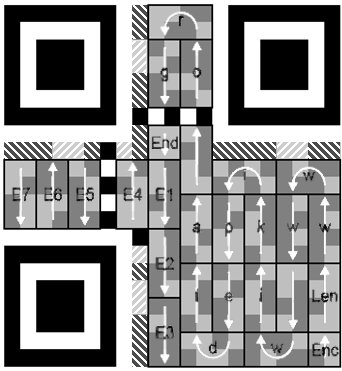
1  0  0  0  0  0  1

1  1  1  1  1  1  1

These squares have a padding of formatting info (shown below). This is a sample of what the QR should look like (courtesy of Wikipedia!)



Notice that the QR matrix appears to be square like (21x21 square, in this case). It will ALWAYS be a square. For simplicity, the size of the QR will ALWAYS be 21x21, formatted as shown (although in reality, the QR matrix can come in different sizes). The red formatting info bits will not be used in this implementation of the program (however, they will still be present in your QR code). Notice that there is a dotted line of black and white dots in the left and top areas. These are also just formatting stuff, and should be ignored, but again, they will be present in your QR code. Each “byte” in the QR code is divided into 4x2 rectangular blocks (except for the end bit and encoding type, which are 2x2 blocks) that form a snake pattern from the bottom right to the right. These rectangular bytes collectively represent one byte (one character) and each dot consists of the bit. Notice how the bit ordering is done based on where you are in the QR code. Furthermore, notice how the bit number 1 is the most significant bit, meaning, when each block is parsed, it is read as binary 12345678. The first byte block on the lower right consists of the length of the string that is being encoded. Thenceforth is the actual characters, followed by an end block. After this message are some error correction blocks. You will not need to worry about these blocks. All QR codes have some sort of mask (basically certain data bits squares get flipped color, black to white, white to black) to avoid having marker tags embedded within the data. For simplicity, we will always apply the same type of mask as shown: row % 2 == 0. For example, all bits on row 0, row 2, row 4, etc. get their bits flipped. Noticed that only the data bits get flipped. After demasked, it looks like this:



**Input**

The first line consists of three integers N, R, and C where N specifies the number of test cases to follow, R specifies the number of lines of QR data to follow, and C is the number of characters for each line of data. Then the subsequent lines will consists of N data sets of R lines that consist of the data encoded in 0s and 1s where 0s represent the white spots and 1s represent the black spots, separated by one whitespace.

**Output**

For each test case, the program will output on a line the text of the QR code.

**Assumptions:** There will only be three markers present in each of the data sets, and they will be positioned as is in a QR code. Furthermore, these markers will not necessarily be placed in the upper left corner of the data set, and also may not extend to the end of the data set.

|  |
| --- |
| Sample Input |
| 1  23 27  101010110111011100111011101  100101101100101110010100001  100111111101110001111111101  010100000101011001000001100  010101110100010001011101010  110101110100011101011101110  110101110101011001011101101  100100000101111001000001010  001111111101010101111111110  110000000001000100000000100  101111001101110011110011101  101001101001010101011011110  110100001101101101010001100  101100010000111010111011101  110011000111101100100001001  110000000001010100011011010  100111111100100111010001101  101100000101111010100010100  101101110100011010101011100  110101110100110110011110110  101101110101100101011011110  100100000101110010111000101  001111111101000110010001011 |
| Expected Output |
| www.wikipedia.org |

# Find x! (findx.in) [20 Points]

Quick! *Bond* lost his girlfriend *X*! We have been searching all over town to hunt down the missing X. No, actually, we need to compute the value of X. So imagine you are back sitting in *Algebra I* trying to figure out how you would solve for X in a series of linear equations. However, the issue is, you are now bombarded with so many of such problems, and you don’t have the time to go through all of them; in addition, some of these problems have some pesky floating decimal values, and you’re too lazy to plug the numbers into the calculator, so you just decide to create a simple equation solver. Since this is Algebra I level problems, you are just dealing with simple single variable problems without complex polynomial stuff (i.e. you won’t see any X2, X3, etc) or rational polynomials (such as 1/x or [x+1]/[x-1]). Luckily, all the equations have been converted into post-fix notation (believe me, you won’t want to read in infix equations).

**Input**

One integer value N describes the number of test cases to follow. Each test case consists of one line of space-separated tokens of either mathematical operations (+, -, /, \*), floating-point numbers, the variable X (note the uppercase) to solve for, or the equals sign in post-fix notation. For simplicity of code there will not be two consecutive numbers/ variables to imply multiplying (i.e. no input would show 3 X, but rather as 3 X \* ).

**Output**

For each test case, the output will consists of X = followed by the value of X, accurate to exactly 3 decimal points. (i.e. should be outputted as X = 5.000 rather than X = 5.0 or X = 5). If X could have multiple values, print out X = ?, and if no values of X can make the equation true, print out Undefined.

**Assumptions**

There will only be ONE equals sign, found at the end of each equation, and each number, variable, and operation, will ALL be separated by EXACTLY one space.

|  |  |
| --- | --- |
| Sample Input | Expected Output |
| 3  5 X \* 1 + 6 =  5 X 3 + \* 2 X \* 3 + =  5 X 3 + \* 2 \* X 3 + = | X = 1.000  X = -4.000  X = -3.000 |