Computer Science Competition Invitational B 2021 Programming Problem Set

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.
2. All problems have a value of 60 points.
3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.
4. Your program should not print extraneous output. Follow the form exactly as given in the problem.
5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Names of Problems

Number	Name
Problem 1	Regina
Problem 2	Arthur
Problem 3	Denis
Problem 4	Eugene
Problem 5	Hannah
Problem 6	Isamu
Problem 7	Klaudia
Problem 8	Manasa
Problem 9	Melanie
Problem 10	Paaus
Problem 11	Sharon
Problem 12	Tiffany

1. Regina

Program Name: Regina.java **Input File: none**

Regina would like to print an ASCII representation of the French flag.

The flag of France is a tricolor flag featuring three vertical bands colored blue (hoist side), white, and red. Write a program that will print a 15 character tall by 60 character wide representation of the French flag using capital ‘B’ for blue, capital ‘W’ for white and capital ‘R’ for red. Show the flag where blue is on the left and red is on the right.

Input: none

Output: An ASCII representation of the flag of France.

Sample output:
```
BBBBBBBBBBBBBBBBBBBBWWWWWWWWWWWWWWWWWWWWRRRRRRRRRRRRRRRRRRRR
BBBBBBBBBBBBBBBBBBBBWWWWWWWWWWWWWWWWWWWWRRRRRRRRRRRRRRRRRRRR
BBBBBBBBBBBBBBBBBBBBWWWWWWWWWWWWWWWWWWWWRRRRRRRRRRRRRRRRRRRR
BBBBBBBBBBBBBBBBBBBBWWWWWWWWWWWWWWWWWWWWRRRRRRRRRRRRRRRRRRRR
BBBBBBBBBBBBBBBBBBBBWWWWWWWWWWWWWWWWWWWWRRRRRRRRRRRRRRRRRRRR
BBBBBBBBBBBBBBBBBBBBWWWWWWWWWWWWWWWWWWWWRRRRRRRRRRRRRRRRRRRR
BBBBBBBBBBBBBBBBBBBBWWWWWWWWWWWWWWWWWWWWRRRRRRRRRRRRRRRRRRRR
BBBBBBBBBBBBBBBBBBBBWWWWWWWWWWWWWWWWWWWWRRRRRRRRRRRRRRRRRRRR
BBBBBBBBBBBBBBBBBBBBWWWWWWWWWWWWWWWWWWWWRRRRRRRRRRRRRRRRRRRR
BBBBBBBBBBBBBBBBBBBBWWWWWWWWWWWWWWWWWWWWRRRRRRRRRRRRRRRRRRRR
BBBBBBBBBBBBBBBBBBBBWWWWWWWWWWWWWWWWWWWWRRRRRRRRRRRRRRRRRRRR
BBBBBBBBBBBBBBBBBBBBWWWWWWWWWWWWWWWWWWWWRRRRRRRRRRRRRRRRRRRR
BBBBBBBBBBBBBBBBBBBBWWWWWWWWWWWWWWWWWWWWRRRRRRRRRRRRRRRRRRRR
BBBBBBBBBBBBBBBBBBBBWWWWWWWWWWWWWWWWWWWWRRRRRRRRRRRRRRRRRRRR
BBBBBBBBBBBBBBBBBBBBWWWWWWWWWWWWWWWWWWWWRRRRRRRRRRRRRRRRRRRR
```

2. Arthur

Program Name: Arthur.java **Input File: arthur.dat**

A Pythagorean triple consists of three positive integers a, b, and c, such that $a^2 + b^2 = c^2$ given that $a + b \leq c$. For example, $a = 3$, $b = 4$, and $c = 5$ is an example of a Pythagorean triple. This is true because $3^2 + 4^2 = 5^2 = 25$. A triangle whose sides form a Pythagorean triple is called a Pythagorean triangle, and is guaranteed to be a right triangle. Arthur needs your help in determining if three, unsorted integers do in fact form a Pythagorean triple.

Input: Input begins with an integer N (1 <= N <= 10), the number of different test cases. Each of the following N lines will contain three, unsorted integers guaranteed to be greater than 1 and less than 100.

Sample Output: For each test case, you are to output the three integers in sorted order, formatted as shown in the sample output, and whether the list is a Pythagorean triple or not followed by a period.

Sample Input:

```
6
3 4 5
5 13 12
3 2 1
6 10 8
31 480 481
85 76 36
```

Sample Output:

```
3, 4, 5 is a Pythagorean triple.
5, 12, 13 is a Pythagorean triple.
1, 2, 3 is not a Pythagorean triple.
6, 8, 10 is a Pythagorean triple.
31, 480, 481 is a Pythagorean triple.
36, 76, 85 is not a Pythagorean triple.
```

3. Denis

Program Name: Denis.java **Input File: denis.dat**

Denis is helping his little sister with her math homework which covers reducing fractions. However, she usually needs help when Denis is at UIL Computer Science practice. Denis decides to write a program that she can use while he's away that will check whether she has correctly reduced a given fraction or not.

Denis needs your help writing a program, that given a fraction, will reduce it. To reduce a fraction, all you have to do is find the greatest common divisor (GCD) of both the numerator (the number above the line in a fraction) and the denominator (the number below the line in a fraction), then divide both the numerator and denominator by the GCD.

Input: Input begins with an integer C (1 <= C <= 20), the number of different test cases. Each of the following C lines will contain a fraction in the form of N/D, where N is the numerator and D is the denominator. N will be guaranteed to be an integer in range $[-500,500]$, D will be guaranteed to be in range $[-500,0) \cup (0,500]$, and there is no restriction that N be less than, equal to, or greater than D. N/D may be given in reduced form initially, so no reduction may be needed.

Exact Output: For each fraction, you are to output: N/D reduced is X/Y, where X is the reduced numerator and Y is the reduced denominator. Negative fractions should be manipulated so that the minus sign goes with the numerator. Zero is represented by zero (0) as the numerator and one (1) as the denominator.

Sample Input:
```
10
2/4
1/2
3/4
18/24
0/5
6/8
-4/-8
4/-8
-4/8
8/4
```

Sample Output:
```
2/4 reduced is 1/2
1/2 reduced is 1/2
3/4 reduced is 3/4
18/24 reduced is 3/4
0/5 reduced is 0/1
6/8 reduced is 3/4
-4/-8 reduced is 1/2
4/-8 reduced is -1/2
-4/8 reduced is -1/2
8/4 reduced is 2/1
```

4. Eugene

Program Name: Eugene.java Input File: eugene.dat

Eugene is in charge of creating usernames for new members of the Firethorne High School's White Hat Hackers Club. The usernames along with a password to the club's web site must be generated for each student. The usernames consist of the lower case version of the member's first and last initial followed by a four digit number. The number is simply the ASCII value of each of the uppercase versions of the initials. For example, Tim Jones' username would be tj8474. 84 is the ASCII value of T and 74 is the ASCII value of J.

Of course, one or more members might have the same initials. When this turns out to be the case, Eugene is going to prevent the duplicate by adding one to the ASCII value of the first initial until there is no longer a duplicate username.

Input: The first line of the input file will contain a single whole number N that is the number of names in the file. There will be no more than 50 names in the file. N will be followed by N club member names each on a separate line where the first and last names are separated by a single space. There may be members with the same first name or the same last name, but no members will have the exact same first and last name. There will not be enough duplicates to require a three digit value for the first initial.

Output: N lines where each line contains the member's first and last name and their username. The first name, last name and the username should each be separated by a single space.

Sample input:
```
6
Sung Habel
Rachal Vandyke
Jewell Krouse
Sally Hess
Claretta Mattinson
Rob Fillmore
```

Sample output:
```
Sung Habel sh8372
Rachal Vandyke rv8286
Jewell Krouse jk7475
Sally Hess sh8472
Claretta Mattinson cm6777
Rob Fillmore rf8270
```

5. Hannah

Program Name: Hannah.java **Input File: hannah.dat**

Hannah knows there's most likely an arithmetic question on the written portion of the UIL Computer Science test, that most likely is not going to be in decimal (base 10). Hannah has the idea to write a program that will perform simple arithmetic, either addition, subtraction, multiplication or division on two numbers in a given base. This way, Hannah can make up her own sample questions, and check to see if she computed the correct answer. Can you help Hannah with this?

Input: Input begins with an integer N (1 <= N <= 10), the number of different test cases. Each of the following N test cases will consist of two lines. The first line will be an integer B, the base of the numbers to be given as well as the base for the arithmetic. The base B will be in range $[2,16]$. The second line will contain two operands, op1 and op2, separated by an arithmetic operator, s. The input will be formatted as seen in the sample input. The operands will be guaranteed to be valid integers for the given base B and will be in range $[-B^8, B^8]$. The operators will either be: +, -, /, or *.

Exact Output: For each test case you are to output: op1 s op2 = resultant, where op1 is operand 1, s is the sign, op2 is operand 2, and resultant is the arithmetic result of the arithmetic performed in base B. The resultant should be of type integer and all mathematical operations are of type integer. If a letter is needed to represent the resultant, use only capital letters.

Sample Input:
```
7
10
10 + 15
2
1010 - 1111
4
31 * 1123
16
2E / F
7
123456 + 654321
3
2212 / 12
2
11111111 - 11111111
```

Sample Output:
```
10 + 15 = 25
1010 - 1111 = -101
31 * 1123 = 102133
2E / F = 3
123456 + 654321 = 1111110
2212 / 12 = 120
11111111 - 11111111 = 0
```

6. Isamu

Program Name: Isamu.java **Input File: isamu.dat**

Isamu just got the latest edition of *Pouch Monsters* and cannot wait to go adventuring! In *Pouch Monsters*, you build a team of the title creatures and visit all the cities in a region (the specific region changes between games). Each city in a region has a dojo, and beating the game requires defeating every dojo.

To travel between cities, Isamu uses the routes in the game. Each route connects two cities, and can be travelled in either direction. Isamu can use each route multiple times, and can also visit the same city multiple times. The only restriction is that he must visit each city at least once. Given the layout of cities and routes in the game, what is the minimum amount of distance Isamu needs to travel to beat the game?

Input:
The first line of input is a positive integer T (T <= 20), the number of test cases. The first line of each test case has two integers C (1 <= C <= 8) and R (C - 1 <= R <= 28), the number of cities and the number of routes. The next line has C space separated names, the name of each city. Isamu begins his adventure in the first city listed. The next R lines each have two cities and a distance. The distance between any two cities is at most 100 units. Routes can be taken in either direction, and there is at most one direct route between any pair of cities. It is always possible to travel from one city to another using the given routes.

Each city name is a string of at most 15 lowercase English letters.

Output:
For each test case, output the minimum distance Isamu must travel to visit every city, formatted as shown in sample ouput.

Sample Input and Output on next page…

Sample Input:
```
3
3 2
alpha beta gamma
alpha gamma 7
gamma beta 4
4 4
austin elpaso houston dallas
austin houston 3
dallas houston 5
dallas austin 4
austin elpaso 9
5 8
green brown blue yellow saffron
green brown 52
blue yellow 26
saffron brown 19
yellow saffron 12
green blue 21
saffron green 36
yellow brown 56
blue brown 66
```

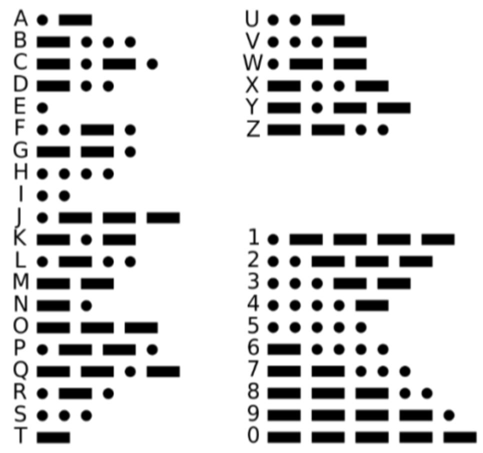
Sample Output:
```
Case #1: 11
Case #2: 21
Case #3: 78
```

Sample Explanation:
In the first sample test case, Isamu can travel from `alpha` to `gamma` to `beta`, using the only routes available for a total cost of 11.
In the second sample test case, one possible optimal route is `austin`, `houston`, `dallas`, `austin`, `elpaso.`

7. Klaudia

Program Name: Klaudia.java **Input File: klaudia.dat**

Klaudia has been sent a message coded using Morse code. The message is quite long so Klaudia needs a program to convert the Morse code to text. Morse code is based on a series of dots and dashes typically transmitted using sound or light. However, Klaudia's message is stored as a series of text dots and dashes. Here is a key for each of the letters and for the digits. Your job is to write a program that will read the Morse code in the file and then print the corresponding text message.



By Rhey T. Snodgrass &amp; Victor F. Camp, 1922 - Image:Intcode.png and Image:International Morse Code.PNG, Public Domain, https://commons.wikimedia.org/w/index.php?curid=3902977

Input: The first line of the input file will contain a single whole number N. The first line will be followed by N lines of Morse code. Each letter in the code will correspond to the table shown above. There will be a single space between each letter and a forward slash / will signify a new word. There will not be any punctuation in the message.

Output: N lines where each line contains the words in a single line of Morse code. Each word should be separated by a single space. All of the output should be uppercase letters or numbers.

Sample input:

```
3
.... . -.-- / -.- .-.. .- ..- -.. .. .- / -.. --- / -.-- --- ..-
.-- .- -. - / - --- / --. --- / - --- / - .... . / --.. --- ---
- --- / ... . . / .-.. .. --- -. ... / .- -. -.. / - .. --. . .-. ...
```

Sample output:

```
HEY KLAUDIA DO YOU
WANT TO GO TO THE ZOO
TO SEE LIONS AND TIGERS
```

8. Manasa

Program Name: Manasa.java **Input File: manasa.dat**

Manasa the superhuman likes swimming long distances in rivers, and relishes the challenge of swimming against a river's current. However, even superhumans struggle while straining against the current for long periods of time, so she takes breaks while swimming to regain her strength.

Formally, Manasa can swim at a rate of V meters per second for T seconds. After T seconds, she has to recharge for S seconds, during which time the river carries her back at a rate of W meters per second.

Given that Manasa's source and destination are D meters apart, how many times does she need to start swimming in order to reach the other side?

Input:
The first line of input is a positive integer N (N <= 50), the number of test cases. Each test case is a single line with 5 integers V T W S D in order. Each of these is a positive integer at most 1,000,000,000.

Output:
For each test case, output the number of times Manasa must start swimming in order to reach her destination. If it is impossible for Manasa to reach her destination, output "Impossible". Format your output with the case number as in the samples.

Sample Input:
```
3
2 4 1 1 10
1 1 1 1 2
31415 92653 58979 32384 626433827
```

Sample Output:
```
Case #1: 2
Case #2: Impossible
Case #3: 1
```

Sample Explanation:
In the first sample, Manasa can swim 2 meters per second for 4 seconds, then has to take a break. After she swims 8 meters, she takes a 1 second break, during which the river's current takes her backwards by 1 meter, leaving her 7 meters ahead of where she began. She reaches her destination 10 meters from where she began during her second period of swimming.

In the second sample, Manasa swims a meter forwards, but then the current takes her back where she began. She will never reach two meters out from where she began.

In the third sample, Manasa shows she is truly superhuman, and makes it to the shore during her first swim.

9. Melanie

Program Name: Melanie.java **Input File: melanie.dat**

An anagram is a reordering of the letters in a word of phrase. For example, you can rearrange the letters of `stressed` to get the word `desserts`. You can even make `detesssr` and `tsesrsed`, which are both anagrams of `stressed` even if they are not legitimate English words. Melanie would like to write a program, that given a word, not necessarily a legitimate English word, that will output the number of possible unique anagrams that can be formed from the given word.

Input: Input begins with an integer N (1 <= N <= 10), the number of different test cases. Each of the following N test cases will contain a word (not necessarily a legitimate English word) made up of letters only. The letters may be upper or lower case, and upper and lower cases letters are to be treated as distinct letters. The word will have a length of at least 1 and no more than 20.

Exact Output: For each test case you are to output the total possible unique anagrams that can be made from the given word. Each anagram must use every letter, i.e., letters cannot be omitted to make a shorter word.

Sample Input:
```
7
abc
att
aabbcc
ordeals
abcdABCDabcd
desserts
terraced
```

Sample Output:
```
6
3
90
5040
29937600
3360
10080
```

10. Paaus

Program Name: Paaus.java **Input File: paaus.dat**

There are four types of integer literals in Java. To differentiate between the different kinds of literals, each group has its own prefix.

The kinds of integer literals are:

- decimal literals, which allow programmers to write numbers in base 10, with no prefix
- hexadecimal literals, which allow programmers to write numbers in base 16, with prefix "0x"
- octal literals, which allow programmers to write numbers in base 8, with prefix "0"
- and binary literals which allow programmers to write numbers in base 2. with prefix "0b"

For example, 0xb2, 178, 0262, and 0b10110010 are all different ways of writing the same number.

Pauus has a positive number he wants to type into his Java program, but one of the keys on his keyboard is broken. Help him find a way to input his desired number using the fewest number of characters.

Input:
The first line is a positive integer T (T <= 60), the number of test cases. Each test case has N, the number Pauus wants to type, and K, the broken key on his keyboard. N is a non-negative integer written in base 10 less than or equal to 10^9, and K is either a single digit or a lowercase english letter in the range a—f.

Output:
For each test case, output the shortest integer literal that is equal to N and does not include the digit K. Use lowercase English letters when outputting hexadecimal digits. If there are multiple literals of the same length, output the one in the lowest base. If there are no way to write the value, output "Impossible". Format your answer with the case number as in the samples.

Sample Input:
```
3
15 5
33 2
17 1
```

Sample Output:
```
Case #1: 017
Case #2: 33
Case #3: Impossible
```

Sample Explanation:
In the first sample, Paaus cannot write "15" as a decimal literal because it has a "5" in it. Instead he chooses to write "017", the octal literal representing the same value.
In the third sample, it is impossible to write the value 17 as a Java literal without the "1" digit.

11. Sharon

Program Name: Sharon.java **Input File: sharon.dat**

Sharon would like to have a program that will simulate a game she is calling Drop Out. For each round of the game there are a different number of players seated in chairs on a stage in a circle numbered from 1 to the number of players. There is a trap door under each chair. At the beginning of the game Sharon rolls one or two dice to determine the "cycle" for the game. Sharon counts from one to the cycle number then pulls a lever that drops the player she lands on through the trap door. (Don't worry! There is a giant soft foam rubber pad under the stage.) After that player takes the plunge, Sharon counts to the cycle number again and drops the next player through the stage. When counting around the circle, Sharon only counts the players left in the game. When only one player is still on the stage, they are declared the winner and then … Sharon goes ahead and drops them through the stage as well. Sharon is kind of evil that way!
In a game with 5 people named Tran, Alia Eun Marylee and America seated in order in chairs 1 through 5 Sharon rolls a 3 on her die. This means that Eun goes out first, Tran is next, then America, next to last is Alia and the only one left is Marylee and she is declared the winner.

Input: The first line of the input file will contain a single whole number G that represents the number of Drop Games to be played. The first line will be followed by G lines where each line is a game. Each line will begin with a number C representing the cycle number for this game. C will be followed by P the number of players in this game. Finally, P will be followed by P names each separated by a space.

Output: The name of the winner of each game.

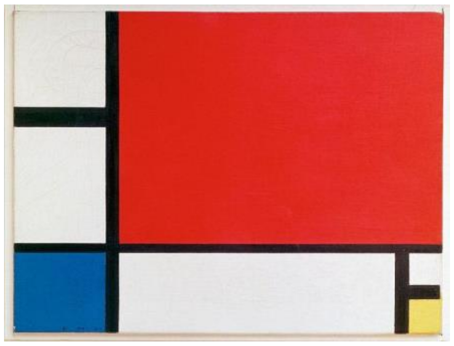
Sample input:
```
3
3 5 Tran Alia Eun Marylee America
4 3 Shantel Sydney Irving
2 8 William Deangelo Rolando Lieselotte Toby Yetta Dolly Roxy
```

Sample output:
```
The winner is Marylee
The winner is Sydney
The winner is William
```

12. Tiffany

Program Name: Tiffany.java **Input File: tiffany.dat**

Tiffany is a fan of the Dutch art movement *De Stijl*. She especially likes the works of artist Piet Mondrian.



Piet Mondrian, Composition with Red, Blue, and Yellow, 1930.

Many of Mondrian's best known works are made of straight lines and solid colors like the one above. They typically use few colors. Inspired by this art, Tiffany set out to make her own paintings. Here is one of Tiffany's paintings:

1	1	2	2
1	1	2	2
3	3	3	3

Tiffany's canvas is made of R rows, each with C cells. Each cell in her painting has a color, represented by a positive integer. A *square* is defined using two cells: an upper left cell (r1, c1) and a lower right cell (r2, c2), where r2 - r1 = c2 - c1. A square is *monochromatic* if every cell (r, c) where r1 ≤ r ≤ r2 and c1 ≤ c ≤ c2 is the same color. For example, using 1-based indexing for the rows and columns, (1, 1) to (2, 2) is a monochromatic square (all cells have color 1), but (2, 2) to (3, 3) is not (there is a cell of color 1, a cell of color 2, and 2 cells of color 3). Furthermore (3, 1) to (3, 4) is not a square, because there are a different number of rows and columns.

Given the layout of Tiffany's painting, find the number of monochromatic squares in the painting.

Input: The first line of input has an integer T ($1 \leq T \leq 20$), the number of test cases. Each test case begins with two positive integers R and C, the number of rows and columns in Tiffany's painting. The next R lines each have C space separated integers, the color of each cell of her painting.

The sum of R * C over all test cases will not exceed 6,000,000. Each color in the painting is a positive integer between 1 and 1,000,000,000, both bounds inclusive.

Output: For each test case, output the number of monochromatic squares in the grid, formatted with the case number as in the samples.

Sample Input:
```
2
3 4
1 1 2 2
1 1 2 2
3 3 3 3
4 3
8 7 8
5 3 1
2 4 6
6 2 1
```

Sample Output:
```
Case #1: 14
Case #2: 12
```

Sample Explanation: In the first sample, there are 5 squares of color 1, 5 squares of color 2, and 4 squares of color 3, for a total of 14 monochromatic squares.
In the second sample, all monochromatic squares are 1x1 squares.