Computer Science Competition Invitational A 2025
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	Alberto
Problem 2	Anisha
Problem 3	Danielle
Problem 4	Dominik
Problem 5	Filip
Problem 6	Helen
Problem 7	Jason
Problem 8	Juliana
Problem 9	Ksenyia
Problem 10	Martyn
Problem 11	Prachi
Problem 12	Valery

1. Alberto

Program Name: Alberto.java **Input File: None**

Alberto is very excited to have started 2025 off with a bang! So far, he has kept his New Year's resolution, and so to celebrate he has asked you to print out 2025 in big digits. Note: each line has exactly 26 characters.

Input: None

Output: 2025 as a 6-line x 26-character message.

Sample input:
```
None
```

Sample output:
```
 ___    ___  ___   _____
|__ \  / _ \|__ \ | ____|
   ) || | | |  ) || |__
  / / | | | | / / |___ \
 / /_ | |_| |/ /_  ___) |
|____| \___/|____||____/
```

2. Anisha

Program Name: Anisha.java **Input File: anisha.dat**

You just got a new cat, and you've decided to name her Anisha. Anisha needs a lot of water throughout the day, so you need to write a program to determine if she has enough water in her bowl, or if she needs more. If Anisha has less than 15 water in her bowl, she needs more.

Input: The input will begin with an integer, `n` (`0 < n <= 1000`), denoting the number of test cases to follow. Each test case will consist of two space separated integers, `c` and `m`, denoting the current amount of water in the bowl, and the max amount of water in the bowl. The maximum will never be less than 15.

Output: For each test case, if Anisha has enough water, output the string "`Way to go, H2O.`". Otherwise output the string "`On my way to Dehydration Station.`".

Sample input:
```
5
19 36
10 36
25 25
30 35
3 16
```

Sample output:
```
Way to go, H2O.
On my way to Dehydration Station.
Way to go, H2O.
Way to go, H2O.
On my way to Dehydration Station.
```

3. Danielle

Program Name: Danielle.java **Input File: danielle.dat**

Danielle recently learned about different numerical bases, namely binary, octal, decimal, and hexadecimal. Danielle is already comfortable with converting between bases but wanted to make an extension of this to reflect her exposure to music counting patterns. From her high school choir experience, she knows that, in a typical 4/4 pattern, musicians count beats within a measure as follows:

1, 2, 3, 4; 2, 2, 3 4; 3, 2, 3, 4; 4, 2, 3, 4; 5, 2, 3, 4; 6, 2, 3, 4; …

However, Danielle is curious as to what will happen once the first number exceeds 9 (and therefore has more than one digit). Unfortunately for Danielle, she has never been able to figure this out in her choir class. Therefore, she has taken it upon herself to figure out exactly what this should look like.

She has devised the rule that, after a number in a given significant position reaches the value 9, attempting to add one more to that will instead reset that number back to its original value, and increment the number in the next significant position. Note that, unlike a typical mathematical base/counting system, the least significant digit is the left-most digit, while the most significant digit is the right-most digit. Moreover, not all positions reset back to the same number when they go past the maximum value (in this case, 9). Depending on how significant a digit is, it will reset to a different number. The least significant digit will reset back to 1, the second least-significant digit will reset back to 2, the third least-significant digit will reset back to 3, and the most-significant digit will reset back to 4. Help Danielle develop a program that converts from her musical notation back to the measure she is currently on.

Input: The first line of input will consist of a single integer n $(1 \leq n \leq 500)$ denoting the number of test cases to follow. The next n test cases will consist of a single line that denotes the musical notation of which beat of which measure Danielle has currently counted to. This line will be in the format of "d_1, d_2, d_3, d_4" where d_1, d_2, d_3, and d_4 are all integers and $d_1 \in [1,9]$, $d_2 \in [2,9]$, $d_3 \in [3,9]$, and $d_4 \in [4,9]$.

Output: For each of Danielle's n requests, print out the which measure she has counted to.

Sample input:
```
11
1,2,3,4
2,2,3,4
3,2,3,4
4,2,3,4
9,2,3,4
1,3,3,4
2,3,3,4
9,9,3,4
1,2,4,4
9,9,9,4
1,2,3,5
```

Sample output:
```
1
2
3
4
9
10
11
72
73
504
505
```

4. Dominik

Program Name: Dominik.java **Input File: dominik.dat**

Your professor Dr. Dominik has assigned you some homework involving the differenced in time between given historical events. You will be given a "study sheet" of historical events and the dates they occurred. Then, for your homework assignments you will be given pairs of events, and you need to determine the number of days between them.

Input: The input will begin with two space-separated integers, `n` (`0 < n <= 1000`) and `m` (`0 < m <= 1000`), denoting the number of dates and test cases to follow, respectively. Each of the following n lines will contain a date in the format (`yyyy/mm/dd`), followed by a space, followed by a name of a historical event. The following m lines will contain the names of two events, separated by a colon. Note that the names of historical events may contain spaces.

Output: For each pair of events, output the number of days between them, as an integer, with commas where they should appear.

Sample input:
```
4 3
1776/07/04 Declaration of Independence Signed
1564/04/23 William Shakespeare Born
1804/05/18 Napoleon Gains Power in France
1959/01/03 Alaska Becomes 49th State
Declaration of Independence Signed:Napoleon Gains Power in France
Declaration of Independence Signed:William Shakespeare Born
William Shakespeare Born:Alaska Becomes 49th State
```

Sample output:
```
10,179
77,504
144,160
```

5. Filip

Program Name: Filip.java **Input File: filip.dat**

Professor Filip is your new teacher from Venezuela. His latest assignment is to create a program to print out a rectangle of a given character of a given size.

Input: The input will begin with an integer, `n` (`0 < n <= 1000`), denoting the number of test cases to follow. Each test case will consist of two integers `r` (`0 < n <= 100`) and `c` (`0 < n <= 100`) denoting the number of rows and columns in the rectangle to be printed, a character `k` denoting the character of which the rectangle is to be made up of, and a Boolean `b` denoting whether or not the rectangle should be filled in. If not filled in, only the outside edge should be made up of `k`, and the rest should be empty spaces. If filled in, the entire rectangle should be `k`.

Output: For each test case, output a rectangle with `r` rows and `c` columns made up of character `k`. If `b` is true, the rectangle should be filled in.

Sample input:
```
2
5 6 * true
4 7 # false
```

Sample output:
```
******
******
******
******
******
#######
#     #
#     #
#######
```

6. Helen

Program Name: Helen.java **Input File: helen.dat**

Helen has taken over your Algebra class and is making all of you do all of her homework for the whole year. You need to write a program to solve equations for Helen so that she doesn't make you clap erasers.

Input: The input will begin with an integer, `n` (`0 < n <= 1000`), denoting the number of test cases to follow. Each test case will consist of a string of characters denoting an equation, consisting of any of the following characters:

- `term` – A term will always be either a `constant` or a `variable`.
- `constant` – A constant will be an integer value.
- `variable` – A variable will be a single lowercase alphabetic character, which may or may not have a coefficient (a number before the variable denoting the number to multiply by the variable). The coefficient will always be an integer (positive or negative) if present.
- `symbol` – A symbol will be one of the following 4 symbols (`*`, `/`, `+`, `-`).

The equation will always be in the following format with single spaces separating the items:

```
term symbol term = term
```

It is guaranteed that only one of the terms will be a variable (the other two will be constants).

Output: For each equation, output the variable that was solved for, followed by a space, an equal sign, another space, and the corresponding value (rounded to 3 decimal places).

Sample input:
```
3
2x + 7 = 15
13 * 26 = 3x
x - 2 = 9
```

Sample output:
```
x = 4.000
x = 112.667
x = 11.000
```

7. Jason

Program Name: Jason.java Input File: jason.dat

Jason has just decided to move to Rainchester, a booming city with promising tech startups eagerly hiring. One thing about Rainchester that Jason finds interesting, is how much it rains throughout the year. Jason would like to write a program that creates a nice histogram based on the monthly average rainfall in Rainchester. However, he just started a new job and doesn't have the time. He decided to hire you to do the job for him.

Input: The input will be a single line with exactly 365 space-separated values **R** (0.0 <= R <= 20.0) representing the rainfall for each day of the year in sequence from Jan 1st to Dec 31st. The months of the year have the following days 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 (no leap years).

Output: The output should be a histogram, with the month abbreviation below the x-axis and a bar of asterisks above showing the average rainfall for that month rounded to the nearest inch. Each row of five *'s should represent an inch of rain.

Sample input: All values are on a single line, but have been wrapped below (see input file for entire data)

```
12.90 11.65 3.29 2.02 1.56 19.32 9.68 12.22 13.68 2.10 18.01 17.48 8.20 16.95 9.22
3.57 9.38 4.42 13.38 18.83 18.97 14.88 17.57 7.80 19.25 14.42 3.06 7.14 7.32 17.68
14.11 18.50 16.35 7.52 1.73 3.49 11.15 14.60 12.26 7.22 11.76 4.09 2.90 8.24 1.92
11.05 3.94 15.45 16.97 1.41 11.36 5.30 19.61 11.10 1.94 16.40 1.50 12.40 18.26 12.60
13.40 13.65 11.01 10.83 7.60 16.06 12.07 5.21 12.20 9.74 3.33 5.23 9.07 9.87 3.69 2.31
14.69 13.70 19.01 6.70 6.09
```

(***continues, see iput file to view all data...***)

Sample output:

```
                                                            *****
*****                         ***** *****       *****       *****
***** ***** *****       ***** ***** ***** ***** ***** ***** ***** *****
***** ***** *****       ***** ***** ***** ***** ***** ***** ***** *****
***** ***** ***** ***** ***** ***** ***** ***** ***** ***** ***** *****
***** ***** ***** ***** ***** ***** ***** ***** ***** ***** ***** *****
***** ***** ***** ***** ***** ***** ***** ***** ***** ***** ***** *****
***** ***** ***** ***** ***** ***** ***** ***** ***** ***** ***** *****
***** ***** ***** ***** ***** ***** ***** ***** ***** ***** ***** *****
***** ***** ***** ***** ***** ***** ***** ***** ***** ***** ***** *****
***** ***** ***** ***** ***** ***** ***** ***** ***** ***** ***** *****
***** ***** ***** ***** ***** ***** ***** ***** ***** ***** ***** *****
 Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
```

8. Juliana

Program Name: Juliana.java **Input File: juliana.dat**

J.U.L.I.A.N.A. is a new “AI” software you have been tasked to write at your new internship. Your job is to use LinkedIn networks to find qualified candidates for jobs at your company. You will be given an employee name and a job listing, and you need to see if anyone in that employee’s network is qualified for the job.

Input: The input will begin with two integers, `n` (`0 < n <= 1000`) and `e` (`0 < n <= 1000`), denoting the number of test cases to follow, and the number of people listed in the LinkedIn input you are given. The following `e` lines will each contain a listing of a person on LinkedIn in the following format [without parentheses and brackets]: `FirstName LastName: (years of experience) (major) (field) [0 or more Full Names denoting everyone they are connected to]`. Following these `e` lines, the n test cases will follow. Each test case will consist of two lines. The first will contain a first and last name, denoting the person whose network we will be testing for qualified candidates. The next line will contain job requirements in the following format: `(years of experience) (major/field)`. In order for a candidate to be qualified, they need at least the required years of experience, and they must work in the given field or have the correct college major.

Output: For each test case, output an alphabetized list of the full names of all qualified candidates who are connected to the given employee, with multiple names separated by a comma and a space. If there are no qualified candidates in the given employee’s network, print an empty line.

Sample input:
```
3 5
Sam Franklin: 6 CS CS John Adams Tom Jennings
John Adams: 3 Music Accounting
Tom Jennings: 2 CS Accounting Sam Franklin
George Paul: 2 Accounting CS Tom Jennings
James Lebron: 4 Music CS John Adams
Sam Franklin
2 CS
James Lebron
4 Accounting
George Paul
2 Accounting
```

Sample output:
```
Tom Jennings

John Adams, Tom Jennings
```

9. Ksenyia

Program Name: Ksenyia.java **Input File: ksenyia.dat**

You have created a website with your friend Ksenyia. Your job is to build a program the validity of given passwords and usernames. There is a list of rules for both as follows:

Username Rules:

- Every username must begin with a capital letter.
- Usernames can only be made up of letters and digits.
- A username has maximum length of 20 and minimum length of 8.
- Any username that has been used before cannot be used after it is set the first time.

Password Rules:

- Passwords must contain at least one digit, capital letter, lowercase letter, and special character (`!@#$%^&*?+`).
- A password has maximum length of 30 and minimum length of 10.
- Any password that has been used before cannot be used after it is set the first time.
- A password cannot have 3 or more duplicate characters in a row.
- A password cannot contain any spaces.

Usernames/passwords will only be used if both are valid, so a valid username with an invalid password has not been used.

Input: The input will begin with an integer, `n` (`0 < n <= 1000`), denoting the number of test cases to follow. Each test case will consist of two lines, with the first containing the username, and the second containing the password for a given user.

Output: For each test case, if both username and password are valid output the string "`Valid`", if neither is valid output the string "`Both Invalid`". If only the username is valid output the string "`Password Invalid`", and if only the password if valid output the string "`Username Invalid`".

Sample input:
```
3
ComedicTiming17
Rocket@1776ABC
BenHornSheep
qw3RTYuiooo!
JabbaTheDominoHuttIsBig
ForsakenMY$45
```

Sample output:
```
Valid
Password Invalid
Username Invalid
```

10. Martyn

Program Name: Martyn.java **Input File: martyn.dat**

Martyn's two favorite things that he learned in his Computer Science class are Palindromes and Numeric Bases… why not combine the two? Martyn is interested in knowing, among a list of bases, which bases, if any, a given unsigned integer is palindromic in. Help Martyn in writing a program to list out those such bases.

Input: The first line of input will consist of a single integer n ($1 \leq n \leq 2.5 \cdot 10^5$) denoting the number of test cases. The next n test cases will consist of a single unsigned integer m_i (for all i, $1 \leq i \leq n$: $0 \leq m_i \leq 2^{31} - 1$), denoting the i^{th} number that Martyn is interested in checking.

Output: For each of Martyn's n requests, print out a comma-and-space-separated list of the bases which a given request m_i is palindromic in, or the string "`None.`" if there are none. Note that a valid solution should consider all integer bases starting from base 2 and up until base 64. For the sake of simplicity (traditionally, bases after 36 break this rule), you may assume that a given base b uses the first b symbols from the following alphabet:

```
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/
```

When converting to a given base b, you should express the number to its minimal width. That is, if a number m_i can be represented with a minimum of k characters, then disregard whether the number may be palindromic for any equivalent padded version of that string with more than k characters. Moreover, for bases like base 64 which traditionally prepend '='s to pad a number out to a certain width, disregard the need for such padding.

Sample input:
```
5
33
180149434
31
187544
0
```

Sample output: (***indented lines are continuation of previous line***)
```
2, 10, 32, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
      50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64
16
2, 5, 30, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
      49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64
33
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
      23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
      41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58,
      59, 60, 61, 62, 63, 64
```

11. Prachi

Program Name: Prachi.java **Input File: prachi.dat**

You and Prachi are doing an experiment for your physics class. You will be given a rectangular plate that you will pour water onto, and you need to determine what the plate will look like once you pour the water. The plate will have a series of etchings that can hold water. You will be given a picture of the plate and you'll need to color in the sections that will be filled with water if you pour a continuous stream of water into a certain spot on the plate.

Input: The input will begin with an integer, `n` (`0 < n <= 1000`), denoting the number of test cases to follow. Each test case will begin with a line containing four integers, `r`, `c`, `sr`, and `sc`, denoting the number of rows and columns in the picture of the given plate, and the position on the plate you will pour water into. The picture will consist of two kinds of characters, `#` denotes an unaltered section of the plate (water cannot flow onto this), and **.** denotes an etching in the plate, where water can flow. Note: The starting point is guaranteed to be a **.** character.

Output: Output the picture of the plate, with all the spots that contain water after the experiment filled in with a `$` character. Water cannot flow diagonally, and cannot flow through any space except for a **.**

Sample input:
```
3
5 4 1 1
####
#..#
####
#..#
####
6 6 1 4
###.##
#....#
###...
#..##.
#.##..
..#...
3 8 1 5
#..#.#..
..#...##
.#...#..
```

Sample output:
```
####
#$$#
####
#..#
####
###$##
#$$$$#
###$$$
#..##$
#.##$$
..#$$$
#..#$#..
..#$$$##
.#$$$#..
```

12. Valery

Program Name: Valery.java **Input File: valery.dat**

Valery just started her new job at the post office. During her onboarding, they explained something she found rather interesting about how they make deliveries between other post offices. Each post office makes some number of deliveries each day to other post offices. The delivery driver at each office must wait for all packages to arrive from the expecting offices before making their delivery runs. Since Valery knows you're so good at programming, she has asked you to write a program telling her what order the post offices will make their deliveries.

Input: The first line of input will contain a number N (1 <= N <= 100) denoting the number of post offices. Each following line will contain two values, S and M (1 <= M <= 100) being the name of the post office and the number of deliveries that driver must make respectively. The following line will contain M space separated values denoting the names of post offices to make deliveries to.

Note: There will always be at least one post office which does not need to wait for any other deliveries.

Output: Output the correct ordering of deliveries, arrow separated as shown in sample output, such that each post office driver waits for all deliveries before leaving. If there are multiple, output the lexicographically smallest.

Sample input:
```
6
A 3
B C D
B 0
C 2
D F
D 1
B
E 1
D
F 1
E
```

Sample output:
```
A->C->F->E->D->B
```