Computer Science Competition
Invitational B 2023
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	Dilmini
Problem 2	Emily
Problem 3	Fiorella
Problem 4	Jacob
Problem 5	Karen
Problem 6	Lautaro
Problem 7	Mario
Problem 8	Petra
Problem 9	Rishita
Problem 10	Shivani
Problem 11	Tushar
Problem 12	Vinay

1. Dilmini

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

Oh No! You broke your friend Dilmini's phone, and he's prone to outbursts when he gets upset! Quick, make him a new phone before he notices.

Input: There is no input for this problem.

Output: Output the ascii image of the phone exactly as shown in the sample output, without the first and last lines of numbers (These lines are only there to help you determine how many underscores are present, the .out file shows exactly what should be output.)

Sample output:

```
12345678901234567890
___________$$$$$$$$$$
__________$__________$$
__________$_$$$$$$$_$$
__________$_$_____$_$$
__________$_$_____$_$$
__________$_$_____$_$$
__________$_$_____$_$$
__________$_$$$$$$$_$$
__________$__________$$
___________$$$$$$$$$$
__________$__________$$
_________$_1__2__3_$$$
________$_4__5__6_$$$
_______$_7__8__9_$$$
_____$_*__0__#_$$$
____$__________$$$
_____$$$$$$$$$$
______$$$$$$$$
12345678901234567890
```

2. Emily

Program Name: Emily.java **Input File: emily.dat**

Emily needs your help writing a program that reads in a simple mathematical expression consisting of an integer operand, a binary operator, and another integer operand. The program should calculate the result of the mathematical operation and display it to the console.

Valid binary operators are:

- + (performs addition)
- - (performs subtraction)
- X, x, and * (performs multiplication)
- / (performs division)

Input: Input will consist of an integer N, the number of test cases. The number of test cases will be in range [1,20]. Each subsequent line will contain the mathematical expression of the form: "operand1 + operand2", "operand1 - operand2", "operand1 X operand2", "operand1 x operand2", "operand1 * operand2", or "operand1 / operand2". operand1 and operand2 are guaranteed to be integers in range of [-255,255]. There will be a guaranteed space between operand1 and the binary operator, as well as a guaranteed space between the binary operator and operand2. Note: the input will never consist of division by 0 and the remainder will be guaranteed to be a nonnegative number.

Output: For each mathematical expression, you are to output the original mathematical expression followed by a space, an equal sign, a space, and the result of the operation. For the division case only, you are to output the result of the integer division, followed by a space, "remainder", space and the remainder of the division that was performed. See the Sample output.

Sample input:

```
7
1 + 2
5 - 1
4 * 3
8 x 6
9 X 4
20 / 6
40 / 10
```

Sample output:

```
1 + 2 = 3
5 - 1 = 4
4 * 3 = 12
8 x 6 = 48
9 X 4 = 36
20 / 6 = 3 remainder 2
40 / 10 = 4 remainder 0
```

3. Fiorella

Program Name: Fiorella.java **Input File: fiorella.dat**

Your friend Fiorella has Algebra 2 homework due next period, and she forgot all about it! You need to write a program to do the homework before the homework is due.

Use the following formula, given w, x, y, z, and G, solve for n:

$$wx^ny + z = G$$

Input: The first line will contain a single integer `b (0 < b < 50)` that indicates the number of data sets that follow. Each data set will consist of 5 integers, `w`, `x`, `y`, `z`, and `G`, `(0 < w, x, y, z, G < 2^31)` all separated by spaces.

Output: Output the value of `n` given by solving the equation for n given all the other values. This value should be formatted as an integer, and printed as shown in the sample output. There will never be a set of numbers for which there is no valid n.

Sample input:

```
4
2 2 2 2 8
2 2 2 2 514
3 2 3 16 52
6 7 8 10 2362
```

Sample output:

```
0
7
2
2
```

4. Jacob

Program Name: Jacob.java **Input File: jacob.dat**

Jacob is fascinated with Math and recently learned that the Math constant π or **pi** is an irrational number that is approximately 3.14159265… and it can be computed to desired precision using the following math series:

$$\pi \approx 3 + \frac{4}{2 \times 3 \times 4} - \frac{4}{4 \times 5 \times 6} + \frac{4}{6 \times 7 \times 8} - \frac{4}{8 \times 9 \times 10} + \cdots$$

Jacob is better with Math than Java programming and has asked for your help computing approximations of π using a variety of number of terms after the initial value of 3. The above expression would be the approximation using 4 terms and produces the first value shown below in the sample output.

Input: First line will contain a positive integer **N**, the number of test cases with $0 \leq$ **N** ≤ 25. The following **N** lines will contain a single positive integer **T**, the desired number of terms following the initial value of 3 to include in the approximation with $0 \leq$ **T** ≤ 1000.

Output: For each test case, display the computed approximation of π with 13 digits after the decimal point.

Sample input:

```
5
4
7
23
101
911
```

Sample output:

```
3.1396825396825
3.1420718170718
3.1416106990405
3.1415928891421
3.1415926539194
```

5. Karen

Program Name: Karen.java **Input File: karen.dat**

Karen has always been fascinated by perfect squares. They have so many interesting characteristics. She realized that since perfect squares alternate between even and odd numbers, any integer will have exactly one integer perfect square that is closest to it. That is, an integer is never halfway between two integer perfect squares.

Your job is to take any integer in the range [1,1000000] and determine the closest integer perfect square to that number.

Input: Input will consist of an integer N, the number of test cases. The number of test cases will be in range [1,20]. Each subsequent line will contain one integer in the range [1,1000000]

Output: Each line of output will consist one number representing the closest integer perfect square to the input.

Sample input:

```
5
90
91
144
500000
1111
```

Sample output:

```
81
100
144
499849
1089
```

6. Lautaro

Program Name: Lautaro.java **Input File: lautaro.dat**

You and your friend Lautaro have secured a job at a telemarketing company. Your job is to go through a list of submitted phone numbers, and determine which ones are valid, valid phone number requirements are defined below:

Valid Phone Number Requirements:

1) The phone number must be made up of 10 digits, in the following example format:
   (123) 456-7890
2) There must be a three-digit area code beginning the phone number, and it must be inside a set of parentheses, with nothing inside the parentheses except for the area code.
3) After the area code, there must be a space before there are any more numbers.
4) After the space, we will have 3 digits, followed by a dash, followed by 4 digits.
5) Any number not following this format exactly is invalid.

Input: The first line will contain a single integer `n (0 < n < 50)` that indicates the number of data sets that follow. Each data set will consist of a string denoting the phone number to be validated, all on one line.

Output: For each data set, if the phone number is valid, output the string `"Valid Phone Number."`. If the phone number is invalid, output the string `"No Calls for You."`.

Sample input:

```
4
(833) 691-2590
(323)-432-3222
(34) 345-2341
(233) 888 7876
```

Sample output:

```
Valid Phone Number.
No Calls for You.
No Calls for You.
No Calls for You.
```

7. Mario

Program Name: Mario.java **Input File: mario.dat**

Mario is trying to create a new way to write words in code so he can leave secret messages to his friends. He has come up with an interesting way to encode a word.

A word will be accompanied by an integer. That integer indicates how many letters at the front and at the back of a word that will be swapped. For example, if the integer is 2 and the word is COMPUTER, the first two letters of COMPUTER "CO" will be placed at the end of the word and the last two letters "ER" will move to the front.

But, to make it just a bit trickier, Mario will reverse the letters in each of the two blocks that are being swapped. Thus, in the example above "OC" moves to the end and "RE" moves to the start.

2 COMPUTER gives us REMPUTOC
3 COMPUTER gives us RETPUMOC

Mario noticed that there would be two special cases.

If the integer is larger than the number of letters in the word, the output is "error".

7 MOUSE gives us error

If the integer does not trigger an error as mentioned above, but is more than half the length of the word, the result would be simply the reversal of the original word.

4 TEXAS gives us SAXET

Write the program to allow Mario to encode his words.

Input: Input will consist of an integer N, the number of test cases. The number of test cases will be in range [1,20]. Each subsequent line will contain one integer in the range [1,100] followed by one space then a string consisting only of upper-case letters. The strings will contain no spaces and no punctuation marks. The string will be of length [1,100].

Output: Each line of output will consist of one string of characters representing the "rearranged" solution. If the inputs create an error code mentioned above, the output will be "error" written in all lower-case letters.

Sample input:

```
5
3 ABCDEFG
4 AB
1 ABCDEFG
5 QWERTY
2 ASDFGH
```

Sample output:

```
GFEDCBA
error
GBCDEFA
YTREWQ
HGDFSA
```

8. Petra

Program Name: Petra.java **Input File: petra.dat**

Petra likes to look at things from all angles. When it comes to numbers, she likes to look at the relationship between a number and its reverse.

The reverse of 135 is 531.

The reverse of 980 is 089 which is actually 89.

The reverse of 5 is 5.

Your job is to find the greatest common factor of an input number and its reverse.

Input: Input will consist of an integer N, the number of test cases. The number of test cases will be in range [1,20]. Each subsequent line will contain one integer in the range [1,1000000] representing the number to be tested.

Output: Each line of output will consist of one integer representing the greatest common factor of the input with its reverse.

Sample input:

```
5
21
44
700
369
12345
```

Sample output:

```
3
44
7
9
3
```

9. Rishita

Program Name: Rishita.java **Input File: rishita.dat**

Rishita's classmate Sunny solved a programming problem from the Invitational A contest about dual-credit courses. She found the resulting list of courses from Sunny's contest problem very useful and has accepted a challenge to expand the list by including more data and a different sort. Rishita is as excited as Sunny about taking UIL programming and developing her skills! The previous data from the Texas Common Course Numbering System (TCCNS) included only course codes and titles but now contains the names of the schools that offer the courses. Courses may be offered by more than one school.

Can you help Rishita implement a multi-level sort?

Input: An unknown number of lines, greater than 1 and less than 100. Each line contains a course title followed by a course code and the name of the school offering the course. Items are separated by commas and there are no other punctuation or special characters except for the dash in the course codes.

Output: A sorted list, first by school as the major sort then by course number as the minor sort, both alphabetical (A…Z). List will be formatted as shown in the sample output with the school listed once and the list of courses indented by 3 spaces containing the course code first followed by the course title, separated by 3 spaces.

Sample input:

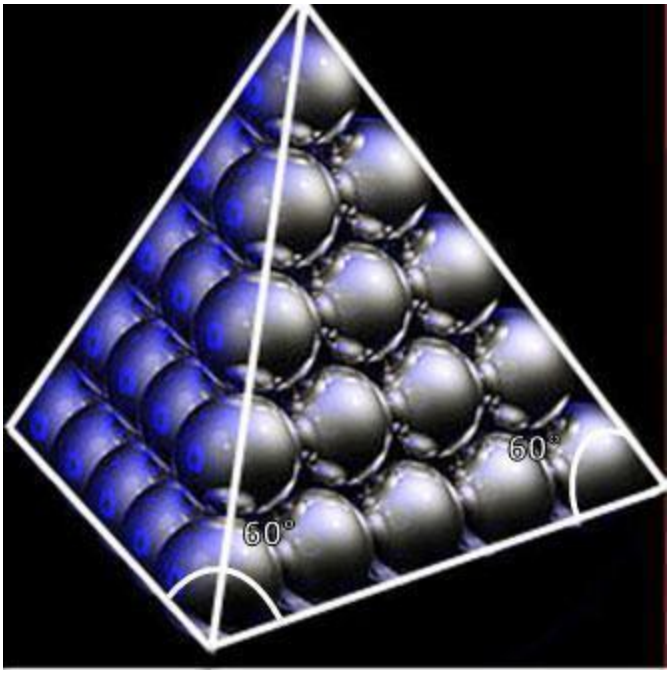
```
UNIVERSITY PHYSICS I LAB,PHYS-2125,TEXAS REPUBLIC COLLEGE
ELEMENTARY STATISTICAL METHODS,MATH-1342,TEXAS REPUBLIC COLLEGE
COMPUTER ORGANIZATION,COSC-2325,BEXAR COMMUNITY COLLEGE
PROGRAMMING FUNDAMENTALS I,COSC-1336,TEXAS REPUBLIC COLLEGE
UNIVERSITY PHYSICS I,PHYS-2325,TEXAS REPUBLIC COLLEGE
INTRODUCTION TO COMPUTER PROGRAMMING,COSC-1315,BEXAR COMMUNITY COLLEGE
C PROGRAMMING,COSC-1320,TEXAS VIRTUAL COLLEGE
CALCULUS I,MATH-2313,TEXAS VIRTUAL COLLEGE
PROGRAMMING FUNDAMENTALS II,COSC-1337,TEXAS REPUBLIC COLLEGE
INTRODUCTION TO COMPUTER PROGRAMMING,COSC-1315,TEXAS REPUBLIC COLLEGE
PROGRAMMING FUNDAMENTALS I,COSC-1336,TEXAS VIRTUAL COLLEGE
PROGRAMMING FUNDAMENTALS III,COSC-2336,TEXAS VIRTUAL COLLEGE
```

Sample output:

```
BEXAR COMMUNITY COLLEGE
   COSC-1315   INTRODUCTION TO COMPUTER PROGRAMMING
   COSC-2325   COMPUTER ORGANIZATION
TEXAS REPUBLIC COLLEGE
   COSC-1315   INTRODUCTION TO COMPUTER PROGRAMMING
   COSC-1336   PROGRAMMING FUNDAMENTALS I
   COSC-1337   PROGRAMMING FUNDAMENTALS II
   MATH-1342   ELEMENTARY STATISTICAL METHODS
   PHYS-2125   UNIVERSITY PHYSICS I LAB
   PHYS-2325   UNIVERSITY PHYSICS I
TEXAS VIRTUAL COLLEGE
   COSC-1320   C PROGRAMMING
   COSC-1336   PROGRAMMING FUNDAMENTALS I
   COSC-2336   PROGRAMMING FUNDAMENTALS III
   MATH-2313   CALCULUS I
```

10. Shivani

Program Name: Shivani.java **Input File: shivani.dat**



You and Shivani are doing a project for your ancient architecture class, and you need to go buy materials. Your project will be a triangular pyramid made of spheres. The top of the pyramid will be one sphere, and the layer below it will be 3 spheres. The nth layer in the pyramid will have n more spheres than the one above it, so 3 + 3 = 6, 6 spheres in the 3rd layer, and so on. You will be given n, the number of layers, and you need to determine how many spheres you will need for the pyramid project.

Input: The first line will contain a single integer `m` that indicates the number of data sets that follow. Each of the following m lines will contain an integer n, the number of layers in your triangular pyramid.

Note: The values for n may be quite large, n may be greater than 2^64 (meaning it is too big for an int or long).

Output: Output the number of spheres needed to form the triangular pyramid with n layers, as shown in the sample output.

Sample input:

```
4
6
12
25
53
```

Sample output:

```
56
364
2925
26235
```

11. Tushar

Program Name: Tushar.java **Input File: tushar.dat**

Tushar really enjoys working with 2-D arrays and is always looking for new ways to work with the array data. Traditional uses typically involve horizontal and vertical processing but Tushar is trying a diagonal pattern and is having some difficulty. Starting at the top left corner, which is indexed as row 0 and column 0, work down and right to compute the average of the elements along a diagonal. Then, work across the top to generate an average for each down and right diagonal. Continue by working down the left side and generate an average, again moving down and right along each diagonal.

Col → ↓ Row	0	1	2	3	4	5
0	1	6	11	15	18	20
1	21	2	7	12	16	19
2	25	22	3	8	13	17
3	28	26	23	4	9	14
4	30	29	27	24	5	10

In the above example with 5 rows and 6 columns, the first shaded diagonal contains 1…5, the second diagonal 6…10, the third shaded diagonal 11…14, the fourth diagonal 15…17, the fifth shaded diagonal 18…19, and finishing across the top with just 20. Working down the left side would be 21…24, then shaded 25…27, then 28…29, and finally just the shaded 30. This produces 10 diagonals with averages shown in sample output below.

Input: First line contains a single integer **N** the number of test cases that follow with **N** ≤ 10. Each test case starts with a line containing 2 integers separated by whitespace: **R**, the number of rows, and **C**, the number of columns, with both 2 ≤ **R**, **C** ≤ 12. That line will be followed by **R** lines of data with each containing **C** integers separated by whitespace containing integers in [-100,100].

Output: For each test case, output 1 row of averages, separated by single spaces. Display the averages with 2 digits after the decimal point.

Sample input:

```
3
5 6
1     6     11    15    18    20
21    2     7     12    16    19
25    22    3     8     13    17
28    26    23    4     9     14
30    29    27    24    5     10
4 4
-13   -9    -5    -1
-14   -10   -6    -2
-15   -11   -7    -3
-16   -12   -8    -4
3 2
1     -6
-2    5
3     -4
```

Sample output:

```
3.00 8.00 12.50 16.00 18.50 20.00 22.50 26.00 28.50 30.00
-8.50 -6.00 -3.50 -1.00 -11.00 -13.50 -16.00
3.00 -6.00 -3.00 3.00
```

12. Vinay

Program Name: Vinay.java **Input File: vinay.dat**

Vinay is obsessed with palindromes. A palindrome is a word or phrase that reads the same backward as forward. For example: racecar is a palindrome since "racecar" spelled in reverse is "racecar". Vinay knows verifying palindromes is a straightforward process, but Vinay wants to up the ante, and try to determine if the letters or numeric characters in any word or phrase can be rearranged such that it forms a palindrome. If the word or phrase can be rearranged to form a palindrome, Vinay wants to know how many distinct palindromes can be formed. For example, looking at the word racecar, the word can be rearranged such that six palindromes are formed:

1. acrerca
2. arcecra
3. carerac
4. craearc
5. racecar
6. rcaeacr

Vinay needs your help writing a program that will not only determine if any given phrase, with all non-alphabetic and all non-numeric characters removed, can be used to form a palindrome, and if so, how many distinct palindrome(s) can be formed. Think you can help?

Input: Input will consist of an integer N, the number of test cases. N will be in range [1,30]. The following N lines will contain a word, or phrase, of which you are to determine how many distinct palindromes can be formed from the word or phrase when all non-alphabetic and all non-numeric characters are removed. Each test case will be guaranteed to be of length [1, 30] characters.

Output: For each input case, you are to output the original word or phrase followed by: "`can not be rearranged to form a palindrome.`" if it is not possible to form a palindrome from the input, or "`can be rearranged to form X distinct palindrome(s).`" where `X` is the calculated number of distinct palindromes that can be formed from the input when all non-alphabetic and non-numeric characters are removed.

Sample input:

```
12
racecar
carrace
a
kayak
12123455436
rotator
wow
noon
saippuakivikauppias
uil
racecars
Mr. Owl ate my metal worm
```

Sample output:

```
racecar can be rearranged to form 6 distinct palindrome(s).
carrace can be rearranged to form 6 distinct palindrome(s).
a can be rearranged to form 1 distinct palindrome(s).
kayak can be rearranged to form 2 distinct palindrome(s).
12123455436 can be rearranged to form 120 distinct palindrome(s).
rotator can be rearranged to form 6 distinct palindrome(s).
wow can be rearranged to form 1 distinct palindrome(s).
noon can be rearranged to form 2 distinct palindrome(s).
saippuakivikauppias can be rearranged to form 45360 distinct palindrome(s).
uil can not be rearranged to form a palindrome.
racecars can not be rearranged to form a palindrome.
Mr. Owl ate my metal worm can be rearranged to form 181440 distinct palindrome(s).
```

UIL Computer Science Competition

Invitational B 2023

JUDGES PACKET - CONFIDENTIAL

I. Instructions

1. The attached printouts of the judge test data are provided for the reference of the contest director and programming judges. Additional copies may be made if needed for this purpose.

2. This packet must remain CONFIDENTIAL. Additional copies may be made and returned to schools when other confidential contest material is returned.

II. Table of Contents

Problem #1
60 Points

1. Dilmini

Program Name: Dilmini.java **Input File: NONE**

Test Input File: NONE

Test Output To Screen:

```
__________$$$$$$$$$
_________$_________$$
_________$_$$$$$$$_$$
_________$_$_____$_$$
_________$_$_____$_$$
_________$_$_____$_$$
_________$_$_____$_$$
_________$_$$$$$$$_$$
_________$_________$$
__________$$$$$$$$$
_________$_________$$
________$_1__2__3_$$$
_______$_4__5__6_$$$
______$_7__8__9_$$$
_____$_*__0__#_$$$
____$_________$$$
_____$$$$$$$$$$
______$$$$$$$$
```

Problem #2
60 Points

2. Emily

Program Name: Emily.java **Input File: emily.dat**

Test Input File:

```
20
1 + 2
5 - 1
4 * 3
8 x 6
9 X 4
20 / 6
40 / 10
2 + 1
1 - 5
3 * 4
6 x 8
4 X 9
6 / 20
10 / 40
-5 + 23
-255 + 0
255 + 6
-4 x -5
-30 / 5
36 / -12
```

Test Output To Screen:

```
1 + 2 = 3
5 - 1 = 4
4 * 3 = 12
8 x 6 = 48
9 X 4 = 36
20 / 6 = 3 remainder 2
40 / 10 = 4 remainder 0
2 + 1 = 3
1 - 5 = -4
3 * 4 = 12
6 x 8 = 48
4 X 9 = 36
6 / 20 = 0 remainder 6
10 / 40 = 0 remainder 10
-5 + 23 = 18
-255 + 0 = -255
255 + 6 = 261
-4 x -5 = 20
-30 / 5 = -6 remainder 0
36 / -12 = -3 remainder 0
```

Problem #3
60 Points

3. Fiorella

Program Name: Fiorella.java **Input File: fiorella.dat**

Test Input File:

```
10
2 2 2 2 8
2 2 2 2 514
3 2 3 16 52
6 7 8 10 2362
2 2 2 2 8192
3 2 3 1 64
1 212 1 1 2
4 2 4 2 258
3 3 3 3 12
3 3 3 3 59052
```

Test Output To Screen:

```
0
7
2
2
10
2
0
4
0
8
```

Problem #4
60 Points

4. Jacob

Program Name: Jacob.java **Input File: jacob.dat**

Test Input File:	Test Output To Screen:
25	3.1396825396825
4	3.1420718170718
7	3.1416106990405
23	3.1415928891421
101	3.1415926539194
911	3.0000000000000
0	3.1666666666667
1	3.1333333333333
2	3.1452380952381
3	3.1415926533405
1000	3.1415926652257
277	3.1415926613640
317	3.1415926566480
433	3.1415926555425
503	3.1415926544835
653	3.1415926516018
500	3.1415926541638
757	3.1415926540116
839	3.1415926538413
997	3.1417360992607
11	3.1415924109720
100	3.1415926228048
200	3.1415926444226
300	3.1415926528640
700	3.1415926532480
900	

Problem #5
60 Points

5. Karen

Program Name: Karen.java **Input File: karen.dat**

Test Input File:

```
10
90
91
144
500000
1111
1
1000000
77777
200
40001
```

Test Output to Screen:

```
81
100
144
499849
1089
1
1000000
77841
196
40000
```

Problem #6
60 Points

6. Lautaro

Program Name: Lautaro.java **Input File: lautaro.dat**

Test Input File:

```
15
(833) 691-2590
(323)-432-3222
(34) 345-2341
(233) 888 7876
888 888-8888
888-888-8888
(333) 333-3333
(432) 33-2345
(8900) 456-4567
(899) 1234-1234
(432) 123-123
(452) 321-12345
(321) 3214-12345
(900) 32-345
Your Teacher is My Teacher.
```

Test Output To Screen:

```
Valid Phone Number.
No Calls for You.
No Calls for You.
No Calls for You.
No Calls for You.
No Calls for You.
Valid Phone Number.
No Calls for You.
No Calls for You.
No Calls for You.
No Calls for You.
No Calls for You.
No Calls for You.
No Calls for You.
No Calls for You.
No Calls for You.
```

Problem #7
60 Points

7. Mario

Program Name: Mario.java **Input File: mario.dat**

Test Input File:

```
10
3 ABCDEFG
4 AB
1 ABCDEFG
5 QWERTY
2 ASDFGH
10 TELEVISION
8 BIFOCAL
2 PRESIDENT
1 CEILING
3 PAJAMAS
```

Test Output to Screen:

```
GFEDCBA
error
GBCDEFA
YTREWQ
HGDFSA
NOISIVELET
error
TNESIDERP
GEILINC
SAMAJAP
```

Problem #8
60 Points

8. Petra

Program Name: Petra.java **Input File: petra.dat**

Test Input File:

```
10
21
44
700
369
12345
45678
301
88
1
999998
```

Test Output to Screen:

```
3
44
7
9
3
6
1
88
1
1
```

Problem #9
60 Points

9. Rishita

Program Name: Rishita.java **Input File: rishita.dat**

Test Input File:

```
UNIVERSITY PHYSICS I LAB,PHYS-2125,TEXAS REPUBLIC COLLEGE
ELEMENTARY STATISTICAL METHODS,MATH-1342,TEXAS REPUBLIC COLLEGE
COMPUTER ORGANIZATION,COSC-2325,BEXAR COMMUNITY COLLEGE
PROGRAMMING FUNDAMENTALS I,COSC-1336,TEXAS REPUBLIC COLLEGE
UNIVERSITY PHYSICS I,PHYS-2325,TEXAS REPUBLIC COLLEGE
INTRODUCTION TO COMPUTER PROGRAMMING,COSC-1315,BEXAR COMMUNITY COLLEGE
C PROGRAMMING,COSC-1320,TEXAS VIRTUAL COLLEGE
CALCULUS I,MATH-2313,TEXAS VIRTUAL COLLEGE
PROGRAMMING FUNDAMENTALS II,COSC-1337,TEXAS REPUBLIC COLLEGE
INTRODUCTION TO COMPUTER PROGRAMMING,COSC-1315,TEXAS REPUBLIC COLLEGE
PROGRAMMING FUNDAMENTALS I,COSC-1336,TEXAS VIRTUAL COLLEGE
PROGRAMMING FUNDAMENTALS III,COSC-2336,TEXAS VIRTUAL COLLEGE
CALCULUS II,MATH-2314,TEXAS REPUBLIC COLLEGE
COLLEGE ALGEBRA,MATH-1314,WATER HOLE COLLEGE
ELEMENTARY PHYSICS,PHYS-1310,VETERAN COLLEGE OF TEXAS
COLLEGE PHYSICS I LAB,PHYS-1101,TEXAS REPUBLIC COLLEGE
PHYSICAL SCIENCE II LAB,PHYS-1117,TEXAS VIRTUAL COLLEGE
CALCULUS III,MATH-2315,TEXAS REPUBLIC COLLEGE
ELEMENTARY PHYSICS I,PHYS-1305,TEXAS REPUBLIC COLLEGE
SOLAR SYSTEM,PHYS-1304,TEXAS REPUBLIC COLLEGE
ELEMENTARY PHYSICS II LAB,PHYS-1107,TEXAS REPUBLIC COLLEGE
PHYSICAL SCIENCE I,PHYS-1315,WATER HOLE COLLEGE
PRE-CALCULUS MATH,MATH-2312,WATER HOLE COLLEGE
MATHEMATICS FOR TEACHERS II,MATH-1351,VETERAN COLLEGE OF TEXAS
UNIVERSITY PHYSICS II,PHYS-2326,BEXAR COMMUNITY COLLEGE
COLLEGE PHYSICS I,PHYS-1301,BEXAR COMMUNITY COLLEGE
ELEMENTARY PHYSICS I LAB,PHYS-1105,TEXAS REPUBLIC COLLEGE
PHYSICAL SCIENCE I LAB,PHYS-1115,TEXAS VIRTUAL COLLEGE
COLLEGE PHYSICS II LAB,PHYS-1102,VETERAN COLLEGE OF TEXAS
CALCULUS II,MATH-2314,BEXAR COMMUNITY COLLEGE
PHYSICAL SCIENCE II,PHYS-1317,TEXAS VIRTUAL COLLEGE
UNIVERSITY PHYSICS II LAB,PHYS-2126,WATER HOLE COLLEGE
LINEAR ALGEBRA,MATH-2318,VETERAN COLLEGE OF TEXAS
MATHEMATICS FOR TEACHERS I,MATH-1350,WATER HOLE COLLEGE
ELEMENTARY PHYSICS II LAB,PHYS-1307,TEXAS VIRTUAL COLLEGE
DISCRETE MATHEMATICS,MATH-2305,BEXAR COMMUNITY COLLEGE
COLLEGE PHYSICS II,PHYS-1302,WATER HOLE COLLEGE
PLANE TRIGONOMETRY,MATH-1316,TEXAS VIRTUAL COLLEGE
SOLAR SYSTEM LAB,PHYS-1104,VETERAN COLLEGE OF TEXAS
MATHEMATICS FOR TEACHERS II,MATH-1351,WATER HOLE COLLEGE
MATHEMATICS FOR TEACHERS I,MATH-1350,VETERAN COLLEGE OF TEXAS
PROGRAMMING FUNDAMENTALS II,COSC-1337,VETERAN COLLEGE OF TEXAS
PROGRAMMING FUNDAMENTALS III,COSC-2336,VETERAN COLLEGE OF TEXAS
INTRODUCTION TO COMPUTING,COSC-1301,TEXAS VIRTUAL COLLEGE
DIFFERENTIAL EQUATIONS,MATH-2320,BEXAR COMMUNITY COLLEGE
```

~ Output continues on next page ~

~ Rishita, continued ~

Test Output To Screen:

```
BEXAR COMMUNITY COLLEGE
   COSC-1315   INTRODUCTION TO COMPUTER PROGRAMMING
   COSC-2325   COMPUTER ORGANIZATION
   MATH-2305   DISCRETE MATHEMATICS
   MATH-2314   CALCULUS II
   MATH-2320   DIFFERENTIAL EQUATIONS
   PHYS-1301   COLLEGE PHYSICS I
   PHYS-2326   UNIVERSITY PHYSICS II
TEXAS REPUBLIC COLLEGE
   COSC-1315   INTRODUCTION TO COMPUTER PROGRAMMING
   COSC-1336   PROGRAMMING FUNDAMENTALS I
   COSC-1337   PROGRAMMING FUNDAMENTALS II
   MATH-1342   ELEMENTARY STATISTICAL METHODS
   MATH-2314   CALCULUS II
   MATH-2315   CALCULUS III
   PHYS-1101   COLLEGE PHYSICS I LAB
   PHYS-1105   ELEMENTARY PHYSICS I LAB
   PHYS-1107   ELEMENTARY PHYSICS II LAB
   PHYS-1304   SOLAR SYSTEM
   PHYS-1305   ELEMENTARY PHYSICS I
   PHYS-2125   UNIVERSITY PHYSICS I LAB
   PHYS-2325   UNIVERSITY PHYSICS I
TEXAS VIRTUAL COLLEGE
   COSC-1301   INTRODUCTION TO COMPUTING
   COSC-1320   C PROGRAMMING
   COSC-1336   PROGRAMMING FUNDAMENTALS I
   COSC-2336   PROGRAMMING FUNDAMENTALS III
   MATH-1316   PLANE TRIGONOMETRY
   MATH-2313   CALCULUS I
   PHYS-1115   PHYSICAL SCIENCE I LAB
   PHYS-1117   PHYSICAL SCIENCE II LAB
   PHYS-1307   ELEMENTARY PHYSICS II LAB
   PHYS-1317   PHYSICAL SCIENCE II
VETERAN COLLEGE OF TEXAS
   COSC-1337   PROGRAMMING FUNDAMENTALS II
   COSC-2336   PROGRAMMING FUNDAMENTALS III
   MATH-1350   MATHEMATICS FOR TEACHERS I
   MATH-1351   MATHEMATICS FOR TEACHERS II
   MATH-2318   LINEAR ALGEBRA
   PHYS-1102   COLLEGE PHYSICS II LAB
   PHYS-1104   SOLAR SYSTEM LAB
   PHYS-1310   ELEMENTARY PHYSICS
WATER HOLE COLLEGE
   MATH-1314   COLLEGE ALGEBRA
   MATH-1350   MATHEMATICS FOR TEACHERS I
   MATH-1351   MATHEMATICS FOR TEACHERS II
   MATH-2312   PRE-CALCULUS MATH
   PHYS-1302   COLLEGE PHYSICS II
   PHYS-1315   PHYSICAL SCIENCE I
   PHYS-2126   UNIVERSITY PHYSICS II LAB
```

Problem #10
60 Points

10. Shivani

Program Name: Shivani.java **Input File: shivani.dat**

Test Input File: (*indented lines are continuations of previous line*)

```
15
6
12
25
53
123456789
234567890
345678901
1234567890
1
2
3
12345670987654123456789876543345678765
123454321234
234567654334567876545678987653576886567865643546787654357987867564387986756435675867867564367586798675643
0
```

Test Output To Screen: (*indented lines are continuations of previous line*)

```
56
364
2925
26235
313612736252315226397035
215106948284414107056018
6884420214044052050454651
313612729393604748070560180
1
4
10
313612125648698381628978249890078539444825647697249986989585021840047472475488546518835042618878109765064006208120
313593922690148606789819349221940
215106297193670771803814128384475041290554774651543484144705038436998341167933939307340866201413423279684182378111659236516025054647480701505322683497330697968818205640361578456408363933756514398435086087952706795503990018873029954628904111106589828458262983777704310137571524067705444646297856080176946029880239090
0
```

Problem #11
60 Points

11. Tushar

Program Name: Tushar.java **Input File: tushar.dat**

Test Input File: (*rows of data indented and right-aligned here for readability, actual data single tab delimited*)

```
7
5 6
1     6     11    15    18    20
21    2     7     12    16    19
25    22    3     8     13    17
28    26    23    4     9     14
30    29    27    24    5     10
4 4
-13   -9    -5    -1
-14   -10   -6    -2
-15   -11   -7    -3
-16   -12   -8    -4
3 2
1     -6
-2    5
3     -4
2 2
-33   4
-83   2
12 12
-85   -2    100   -23   39    -96   -98   35    -3    -18   -85   -95
-79   22    46    -80   5     31    20    -71   64    32    -35   -38
-3    87    65    -90   -12   30    -50   -74   91    14    -38   100
24    33    -13   -2    7     53    30    -26   44    55    70    90
-27   69    -4    20    -76   -48   0     36    -24   -54   -78   33
9     91    51    21    21    -51   29    -69   3     84    -37   -18
41    94    89    80    -92   -2    -88   17    -12   72    23    -83
-19   -39   -54   -16   71    -95   25    29    5     -40   51    41
95    -66   40    79    -100  -15   76    -56   -34   -83   37    36
-67   -36   -58   -70   95    36    -12   29    -88   -95   -16   -59
61    -18   6     18    21    58    -67   -49   -35   53    -36   -33
85    59    40    47    -55   -22   31    19    96    83    62    -27
3 12
-51   -49   94    21    -3    96    -22   31    37    60    -82   -28
-52   86    -26   -2    81    -10   -29   43    -99   -10   -92   26
46    23    77    -42   59    59    22    47    92    88    -69   7
```

~ Input & Output continues on next page ~

~ Tushar, continued ~

```
12 3
-3    1     55
-28   26    -23
4     -92   89
-56   16    -43
61    -17   -24
-49   37    82
77    -40   -60
63    -91   -58
-78   -73   88
-17   -69   -58
-22   -59   76
-2    41    -77
```

Test Output To Screen: (*indented lines are continuation of previous line*)

```
3.00 8.00 12.50 16.00 18.50 20.00 22.50 26.00 28.50 30.00
-8.50 -6.00 -3.50 -1.00 -11.00 -13.50 -16.00
3.00 -6.00 -3.00 3.00
-15.50 4.00 -83.00
-31.50 -15.27 -8.20 26.67 14.75 -40.00 -19.83 43.20 20.25 15.67 -61.50 -95.00
   2.73 1.30 35.00 3.13 44.57 -4.83 -36.00 28.00 -15.00 60.00 85.00
37.33 -39.00 50.33 53.67 3.00 38.00 37.67 6.67 -14.00 -8.33 -28.00 -28.00 -14.50
   46.00
37.33 -11.00 55.00 -54.33 -1.33 3.00 12.67 -49.00 24.67 -22.67 -23.67 -51.00
   9.50 -2.00
```

Problem #12
60 Points

12. Vinay

Program Name: Vinay.java **Input File: vinay.dat**

Test Input File:

```
29
deified
Do geese see God?
Was it a car or a cat I saw?
Rats live on no evil star
Live on time, emit no evil
Step on no pets
Don't nod.
Evil olive.
Amore, Roma.
Yo, banana boy!
Dammit, I'm mad!
Borrow or rob?
I did, did I?
Draw, O coward!
Wonton? Not now!
Never odd or even.
Step on no pets.
Live not on evil.
Rise to vote, sir!
Stella won no wallets.
Won't lovers revolt now?
Delia saw I was ailed.
Too bad I hid a boot.
Red rum, sir, is murder.
Nate bit a Tibetan.
Ah. Satan sees Natasha.
Nat bit a Tibetan.
Step on no pe.
112233445566778899aabbccdde
```

~ Vinay Output on next page ~

~ Vinay Output ~

Test Output To Screen: (lines that are indented are continuation of previous line)

```
deified can be rearranged to form 6 distinct palindrome(s).
Do geese see God? can be rearranged to form 360 distinct palindrome(s).
Was it a car or a cat I saw? can be rearranged to form 60480 distinct
   palindrome(s).
Rats live on no evil star can be rearranged to form 3628800 distinct
   palindrome(s).
Live on time, emit no evil can be rearranged to form 907200 distinct
   palindrome(s).
Step on no pets can be rearranged to form 720 distinct palindrome(s).
Don't nod. can be rearranged to form 6 distinct palindrome(s).
Evil olive. can be rearranged to form 24 distinct palindrome(s).
Amore, Roma. can be rearranged to form 24 distinct palindrome(s).
Yo, banana boy! can be rearranged to form 120 distinct palindrome(s).
Dammit, I'm mad! can be rearranged to form 60 distinct palindrome(s).
Borrow or rob? can be rearranged to form 30 distinct palindrome(s).
I did, did I? can be rearranged to form 6 distinct palindrome(s).
Draw, O coward! can be rearranged to form 120 distinct palindrome(s).
Wonton? Not now! can be rearranged to form 180 distinct palindrome(s).
Never odd or even. can be rearranged to form 2520 distinct palindrome(s).
Step on no pets. can be rearranged to form 720 distinct palindrome(s).
Live not on evil. can be rearranged to form 720 distinct palindrome(s).
Rise to vote, sir! can be rearranged to form 720 distinct palindrome(s).
Stella won no wallets. can be rearranged to form 181440 distinct palindrome(s).
Won't lovers revolt now? can be rearranged to form 181440 distinct
   palindrome(s).
Delia saw I was ailed. can be rearranged to form 20160 distinct palindrome(s).
Too bad I hid a boot. can be rearranged to form 2520 distinct palindrome(s).
Red rum, sir, is murder. can be rearranged to form 20160 distinct palindrome(s).
Nate bit a Tibetan. can be rearranged to form 2520 distinct palindrome(s).
Ah. Satan sees Natasha. can be rearranged to form 30240 distinct palindrome(s).
Nat bit a Tibetan. can not be rearranged to form a palindrome.
Step on no pe. can not be rearranged to form a palindrome.
112233445566778899aabbccdde can be rearranged to form 6227020800 distinct
   palindrome(s).
```