Computer Science Competition
Invitational A 2019
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	Aaditya
Problem 2	Alok
Problem 3	Bojing
Problem 4	Dasha
Problem 5	Dylan
Problem 6	HoJin
Problem 7	Joanna
Problem 8	Lucia
Problem 9	Nina
Problem 10	Raghav
Problem 11	Santino
Problem 12	Vlad

1. Aaditya

Program Name: Aaditya.java Input File: none

Aaditya is excited about participating in Texas UIL Computer Science programming contests but thinks the classic "Hello World" program is totally lame. He challenged his friends to write a program that displays the following messages on the screen.

```
Programming in Java is Awesome!!
Java PROGRAMMERS rock the world!
```

Input: None.

Output: The above messages exactly as shown.

Expected exact output:

```
Programming in Java is Awesome!!
Java PROGRAMMERS rock the world!
```

2. Alok

Program Name: Alok.java Input File: alok.dat

Alok's classmates are collecting donations to purchase a new computer for a friend whose computer was stolen. He needs a program to calculate the total amount of the donations and wants to know the average amount collected per classmate.

Input: A list of several donation amounts, each on one line.

Output: The total amount of all donations and the rounded average donation amount, each on a separate line with a leading dollar sign and two decimal places.

Sample input:

```
39.18
45.87
22.65
19.53
45.23
60.72
```

Sample output:

```
$233.18
$38.86
```

3. Bojing

Program Name: Bojing.java Input File: bojing.dat

Bojing likes to make crossword puzzles. To make it easier to choose words that will fit into his puzzles he needs a list of words that is sorted by length and then alphabetically. For example, he wants all five-letter words grouped together and in alphabetical order, and then all six-letter words alphabetized and so on. He needs a program to sort his word list accordingly and has asked your team to do the job.

Input: Several lines of word lists, each list containing words all in lowercase, single space separation, no punctuation, except for the occasional underscore connecting two words into one.

Output: The entire set of words sorted by length, shortest to longest, and alphabetized within each length group. Each group of words should be on a separate line with single space separation. For example, all five-letter words should be on the same line. Any duplicate words in the input should be deleted from the output. There should be no blank lines in the output.

Sample input:

```
the jaguar is a wild cat species and only
extant member of genus panthera natives to americas present
range extends from southwestern united_states mexico in north_america
across much south paraguay northern argentina though
there are cats now living within_western has
largely_been extirpated since early_century it listed as near
threatened on red list its numbers declining_threats include loss
fragmentation habitat
```

Sample output:

```
a
as in is it of on to
and are cat has its now red the
cats from list loss much near only wild
genus range since south there
across extant jaguar listed living member mexico though
extends habitat include natives numbers present species
americas northern panthera paraguay
argentina
extirpated threatened
largely_been southwestern
early_century fragmentation north_america united_states
within_western
declining_threats
```

4. Dasha

Program Name: Dasha.java Input File: dasha.dat

Dasha is writing a poker game, and needs help shuffling the deck. As a first step, she needs to generate random numbers. Since Dasha likes writing all the code herself, she has decided to implement her own pseudo-random number generator (PRNG).

One of the simplest PRNGs is the linear congruential generator (LCG). An LCG is defined by four values, integers `a, b, m`, and `x_0`. The first "random" value output by the LCG is `x_0`, and subsequent values are given by the formula:

```
x_{i + 1} = (a * x_i + b) % m
```

In other words, to generate the next value of the LCG, multiply the previous value by `a`, add `b`, and take the result modulo `m`.

Dasha picked some values by hand to create her LCG. She then ran the LCG, but found that it wasn't giving all outputs between 0 and m - 1, inclusive. Can you help her figure out how many numbers her LCG doesn't generate?

For example, when `a = 1`, `b = 2`, `m = 4`, and `x_0 = 1`, the LCG outputs an infinite stream of `1, 3, 1, 3, 1, 3,` ... This means that only `2 / 4` (two out of four) values are ever seen.

Input: The first line is an integer T ($0 < T \leq 50$), the number of test cases to follow. Each test case is a single line of four single-space-separated integers, "`a b m x_0`".

$0 \leq a, b < 10^9$
$0 < m < 10^5$
$0 \leq x_0 < m$

Output: For each test case, print the number of values that are possible out of the number of total values, formatted as in the sample. Do **not** perform any simplification of the fraction.

Sample input:
```
4
1 2 4 1
3 6 10 2
1 7 11 4
3 2 17 5
```

Sample output:
```
2 / 4
1 / 10
11 / 11
16 / 17
```

5. Dylan

Program Name: Dylan.java Input File: dylan.dat

Dylan is traveling on a bus to the grand Hilbert Hotel! Everything's bigger in Texas, and this bus is no exception. The bus has a seemingly infinite number of rows, and there are 2 * K seats per row. The seats in a row are numbered 1 through 2 * K, and a center aisle separates seats K and K + 1. The rows are numbered row 1, then row 2, and so on.

When disembarking the bus, people get out one row at a time. The person closest to the center aisle on the left side gets out first, then the person on the right side, and so on. For example, when K = 3, the seats in a row are 1 2 3 4 5 6, with the central aisle between seats 3 and 4. The seats are vacated in this order: 3 4 2 5 1 6.

Row							
1	1	2	3	Center	4	5	6
2	1	2	3	Aisle	**4**	5	6
3	1	2	3		4	5	6
.	1	2	3		4	5	6
.	1	2	3		4	5	6
.	1	2	3		4	5	6

If Dylan is sitting in row 2 seat 4, he wants to know how many people will get off the bus before him. In this situation, the entire first row will exit, followed by the person in seat 3 of row 2, and then Dylan exits after 7 people have gone before him.

As the possible numbers are enormous, they'll need a computer program to do this calculation!

Input: The first line is T (0 < T <= 100), the number of test cases. Each test case has 3 space separated integers, **K** (as defined above), **R** (the number of the row in which Dylan sits), and C (Dylan's seat number in that row).

1 <= K, R <= 10^8, 1 <= C <= 2 * K

Output: For each test case, print a single integer, the number of people who will disembark before Dylan. This value is guaranteed to fit in a 64-bit integer.

Sample input:

```
5
3 2 4
2 1 3
4 4 2
25 10 43
100000000 1 200000000
```

Sample output:

```
7
1
28
485
199999999
```

6. Ho-Jin

Program Name: Hojin.java Input File: hojin.dat

Oh no! Your CS teacher gave you an assignment which extends code you wrote last semester! Unfortunately, you've purged all your old homework (it was a tough semester), but luckily your partner for this project, Ho-Jin, still has his. However, Ho-Jin hates writing comments, and always uses inane variable names. Here's the function he wrote for the last assignment:

```
public long pizza(long cat, long dog) {
    if (cat < dog) {
        return pizza(dog, cat);
    }

    if (dog == 0) {
        return cat;
    } else {
        return pizza(cat - dog, dog);
    }
}
```

This is not only difficult to read, but also very slow. Some of your teacher's data sets take minutes to execute, and neither of you will get a good grade if your code is too slow! Can you speed up this function?

Input: The first line is T ($0 < T \leq 100$), the number of data sets. Each data set has two single space separated integers ($1 \leq$ cat, dog $\leq 10^{18}$). The first data set is Case #1, the second is Case #2, etc.

Output: For each data set, output on a single line the string "Case #", followed by the data set number, a colon symbol, space, and then the output of the "pizza" function when cat and dog are passed in as inputs, formatted as in the samples.

Sample input:

```
5
1 1
6 3
56 21
66 99
2 1000000000000000000
```

Sample output:

```
Case #1: 1
Case #2: 3
Case #3: 7
Case #4: 33
Case #5: 2
```

7. Joanna

Program Name: Joanna.java Input File: joanna.dat

Joanna plans to work a summer job to earn enough money to purchase a new laptop computer and accessories before her senior year in high school. She also hopes to have some money left to spend on summer fun. With multiple job opportunities and numerous options for the computer and accessories she needs a program to evaluate the options. Joanna will not make enough to have income tax deducted from her paycheck but understands that 7.65% will be deducted for social security. Also, sales tax will be applied to the laptop and accessories purchase. She wants to know for each option how much net pay she will get working 10 weeks, how much will be spent on the laptop and accessories including sales tax, and how much will be left for summer spending.

Here is some information to understand job pay:

- Weekly gross pay = pay rate x hours worked per week
- Weekly net pay = gross pay less 7.65% (social security deduction)

Extra attention to accuracy is needed when working with money values!

Input: The first line of the data file contains a count of the number of data sets, with each data set describing a specific option to evaluate. Each of the following lines will contain one complete data set with five numeric values separated by whitespace: hourly pay rate for a job, number of weekly hours as a decimal number, price of the laptop, price of the accessories, and sales tax rate as a percentage. All values are decimal numbers except the initial count which is an integer.

Output: 3 lines of output for each set of input data followed by row of nine stars, "`*********`". Output must use exact labels as shown below with the colon, a single space, and a dollar sign preceding the value. The values are displayed as shown below with the decimal points aligned and commas as necessary. Comments in ( ) are not output.

1. `Net Pay:  $9,999.99` (total 10-week pay less social security deduction)
2. `Purchase: $9,999.99` (total price of laptop and accessories including sales tax)
3. `Fun Cash: $9,999.99` (net pay less purchase)

Sample input:

```
3
8.95 20.5 979.99 135.87 8.25
9.25 25.00 861.19 161.94 8.25
9.65 20.00 986.40 139.68 8.25
```

Sample output:

```
Net Pay:  $1,694.40
Purchase: $1,207.92
Fun Cash: $  486.48
*********
Net Pay:  $2,135.60
Purchase: $1,107.54
Fun Cash: $1,028.06
*********
Net Pay:  $1,782.40
Purchase: $1,218.98
Fun Cash: $  563.42
*********
```

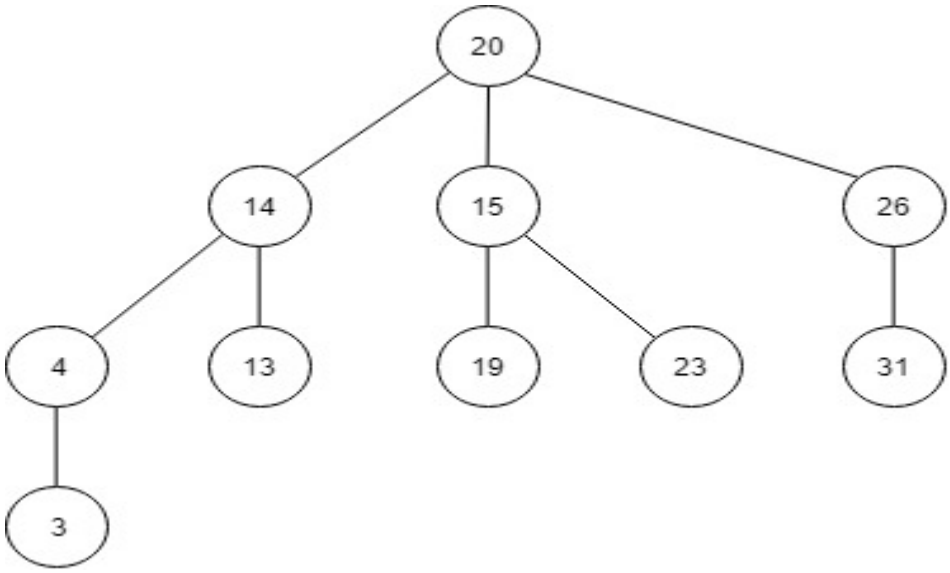
8. Lucia

Program Name: Lucia.java **Input File: lucia.dat**

Lucia just learned about binary trees and thinks it would be cool to make a tri-tree. This is similar to a binary tree, but instead of each node having two possible branches, each node can have up to three branches. Lucia decides to make this tree with numbers, where values over 5 less than the current number go to the left, and values over 5 greater than the current number go to the right. Remaining numbers including duplicates go down the middle.

Here is an example, given the numbers 20, 15, 14, 26, 13, 4, 3, 23, 19, and 31 inserted in order, resulting in the tree shown below.

Your job is to create the tree and then traverse the resulting pre-order and post-order sequence. Pre-order is root, left, middle, right; post-order is left, middle, right, root.



Preorder traversal is: 20 14 4 3 13 15 19 23 26 31
Postorder traversal is: 3 4 13 14 19 23 15 31 26 20

Input: A single list of integers all on one line, with single space separation. **There will be only one data set for this problem.**

Output: Pre-order traversal on one line followed by post-order traversal on the next line, in the exact format as shown below.

Sample Input:
```
20 15 14 26 13 4 3 23 19 31
```

Sample Output:
```
Preorder: 20 14 4 3 13 15 19 23 26 31
Postorder: 3 4 13 14 19 23 15 31 26 20
```

9. Nina

Program Name: Nina.java **Input File: nina.dat**

Nina is the grounds keeper for all the city parks in Jayton, Texas. Recently the Jayton city council voted to create a dog park for the canine citizens of the town, naming it the 5 Star Dog Park. In the spirit of the name the council members would like for the park to be in the shape of a regular pentagon (all sides congruent, all angles congruent). There are a variety of places where the park can be built, all of which would allow for a different size park. The council would like for Nina to determine the area each potential park would cover and the amount of perimeter fencing each would require. Additionally, the council does not want to use up more than one acre of land for the new dog park. Nina has looked on the Internet and found that the area of a regular pentagon can be calculated using this formula:

$$\frac{5s^2}{4\tan\frac{\pi}{5}}$$

where s is the length of a side. The length of a side can be calculated using:

$$2r\sin\frac{\pi}{5}$$

where r is the distance from the center of the pentagon to a vertex (corner). She also found that one acre of land is 43560 square feet. Nina has come to you for help with the calculations. Write a program that will calculate the area and fencing required for each of the proposed locations for the dog park.

Input: A number N representing the number of proposed locations for the park followed by N real numbers each representing the distance in feet from the center of the proposed park to any corner of the pentagon.

Output: For each proposed location print "LOCATION #" followed by the location number followed by one space. For each location that will fit within an acre, print the area in square feet and length of fencing in feet required for the location. Each value should be rounded to the nearest hundredth and separated by one space. For any locations that will not fit within an acre print "LOCATION #" followed by the location number followed by one space and then "WILL NOT FIT".

Sample input:

```
3
100
200
75.5
```

Sample output:

```
LOCATION #1 23776.41 587.79
LOCATION #2 WILL NOT FIT
LOCATION #3 13553.15 443.78
```

10. Raghav

Program Name: Raghav.java Test Input File: raghav.dat

Raghav's math teacher gave an interesting problem with exactly 50 different integers with which to do various calculations.

The difference with this problem versus others is the students need to do the calculations to each number based on where the number is given. For instance, in the sample data below the first value is 72. Since it is first in the list, all calculations done on the number 72 are because it is the first number in the list and have nothing to do with the fact that the number is 72.

Here are the calculations and the order in which to do the calculations:

1) All even locations multiply by 2, odd locations add 7.
2) Multiples of 3 multiply by 5.
3) Multiples of 5, subtract 11.
4) Multiples of 7 square the number.
5) Multiples of 10 divide by 10.
6) Multiples of 11 find the square root.

Note: **Do not round the numbers** but instead provide the lowest integer answer.

For instance:

- Say the 33rd number in the list was 25. Since 33 is odd, a multiple of 3 and a multiple of 11, you would get the following result:

  $\sqrt{((\mathbf{25} + 7) * 5)} = 12$

- For a value 26 in position 45, which is odd, a multiple of 3, and a multiple of 5, this would be the resulting calculation and value in position 45.

  ((**26** +7) * 5) -11 = 154

- For a number in location 50 number, which might be 32, since 50 is even, a multiple of 5 and a multiple of 10, this the changed value in that position would be 5, like this:

  ((32 * 2) -11) / 10 = 5

Input: Exactly fifty integers arranged vertically in the data file, the first to be considered in location 1, an odd numbered location, and the last in position 50, which is even.

Output: Fifty integers, output in vertical format, with the mathematical changes as described above.

Sample Input (First 10 numbers only, arranged vertically in the data file)

```
72    41    25    3     24    3     12    31    11    35
```

Sample Output (First 10 numbers only, to be output in vertical format)

```
79    82    160   6     20    30    361   62    90    5
```

11. Santino

Program Name: Santino.java Input File: santino.dat

Kids these days are getting stranger and stranger. When they're not snap-sta-gramming or smoking Tide Pods, they're making up new playground games. Their latest game is called Hopscotch-plus-plus (abbreviated HPP).

HPP is played on a rectangular grid. Coordinates in the grid are 0-indexed, and are given row, then column, just like indexing matrices in Java. Each cell in the grid has an arrow, which points in one of the 8 cardinal or intercardinal directions (north, northeast, east, southeast, south, southwest, west, northwest). These are given by one or two letter abbreviations (N, NE, E, SE, S, SW, W, NW).

Santino starts in some cell and the students have marked some other cell as the goal. Santino can only jump in the direction given by the arrow in his current cell. Santino can also jump over squares, as long as the destination of his leap is in the direction of the arrow. For example, if the current cell says E, Santino can jump to any square in that row to the right of him in a single leap.

The object of the game is to reach the goal cell from a start cell in as few leaps as possible. Since these wacky kids are making bigger and bigger grids, Santino is worried he won't be able to leave the game and go to lunch. Help him figure out the fewest number of jumps needed.

Input: The first line is T ($T \leq 15$), the number of test cases to follow. The first line has 6 space separated integers. N M RS CS RE CE ($1 \leq N, M \leq 25$, $0 \leq RS, RE < N$, $0 \leq CS, CE < M$). The grid has N rows and M columns. Santino starts in (RS, CS) and can only exit once he reaches (RE, CE).

Output: For each test case, print the minimum number of hops Santino needs to make before he reaches the goal. If there is no way for Santino to reach the goal, print "Lost in the playground" (without quotes). Prefix every line with the case number, formatted like the samples.

Sample input:

```
3
4 4 0 0 3 3
E SW SW W
N S NE SW
NW SE SE N
NE NW E SW
2 3 0 1 1 0
S E S
W N NE
4 3 2 1 2 1
N NE S
SW E NW
W E SE
S E N
```

Sample output:

```
Case #1: 5
Case #2: Lost in the playground
Case #3: 0
```

12. Vlad

Program Name: Vlad.java **Input File: vlad.dat**

Vlad is the chairman of the recruitment committee for the Garden Club at Carlton High School. Go Bulldogs! The committee has decided to place an emphasis on recruiting freshman students to join the club this year. As chairman he has taken the responsibility of obtaining a list of the names of all the freshmen in CHS and dividing that list amongst the committee members so that each freshman is contacted and asked to join. The list of freshmen the registrar provided Vlad has each student listed with their student ID number, last name, first name and middle initial.

He would prefer the list he provides his committee members just be first name, middle initial and last name in alphabetical order by last name. That's where you come in. Write a program that will read the list provided by the registrar and print a list where each student is shown as first name, middle initial and last name in alphabetical order based on last name, then first name and finally by middle initial.

Input: A list of unknown size containing the ID numbers and names of each of the freshman students in Carlton High School each on a separate line. The names will be shown as last name followed by a comma and exactly one space then first name followed by one space and a middle initial followed by a period. Some students might not have a middle initial.

Output: A list of freshman students that is alphabetized by last name then first name and finally by middle initial. The names should be printed first name followed by one space, middle initial followed by one space and then last name. There should be no punctuation within the names and each name should be on a separate line. A name containing no middle initial takes precedence over the same name with an initial.

Sample input:

```
196 Perez, Ava P.
205 Hill, Scarlett M.
369 Gonzalez, Olivia G.
1025 Rodriguez, David D.
1170 Walker, Alexander E.
1401 Davis, Thomas Z.
1492 Clark, Ximena K.
2515 Smith, Daniel B.
2747 Anderson, Jayden F.
2827 Lewis, Jose G.
```

Sample output:

```
Jayden F Anderson
Ximena K Clark
Thomas Z Davis
Olivia G Gonzalez
Scarlett M Hill
Jose G Lewis
Ava P Perez
David D Rodriguez
Daniel B Smith
Alexander E Walker
```