PSCSTA Programming Contest
Spring 2016 Packet

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 14.

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	Days Gone By
Problem 2	Fruit Salad
Problem 3	N'bonacci
Problem 4	Lawn Mower
Problem 5	Name, Rank, Serial Number
Problem 6	Overtime
Problem 7	PaliNum
Problem 8	ParseCon
Problem 9	PP
Problem 10	Expanded Form
Problem 11	That's a Wrap!
Problem 12	Tri, Tri Again
Problem 13	Magic Squares
Problem 14	Four is Cosmic

1. Days Gone By

Program Name: Days.java Input File: days.dat

From January 1st to January 10th, 9 days have gone by. In a non-leap year, there are 364 days from January 1st to December 31st. From the first of the month of May to the end of that same month, 30 days have passed. From February 4th to March 4th in a leap year, 29 days have gone by.

Given two dates in the same year in which the first date is guaranteed to be earlier by at least one day than the second, calculate and output the number of days gone by.

Input: The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set is a pair of lines, each line containing three integers (M D Y), representing the month, day, and year. *The Y value for each set will be any year 1700<=Y<=2020, but will be the same value for both dates. A convenient rule of thumb is that leap years are years divisible by 4 but not by 100. The exception to this rule is that years divisible by 400 are also leap years.*

Output: A sentence like as below that reports the two dates and the number of days gone by between the two dates. Note: Please use "are" and "days" for 1 day; DO NOT correct the grammar to "is" and "day".

Sample input:

```
3
1 1 2015
1 10 2015
1 1 1714
12 31 1714
2 4 2000
3 4 2000
```

Sample output:

```
There are 9 days gone by from 1-1-2015 to 1-10-2015.
There are 364 days gone by from 1-1-1714 to 12-31-1714.
There are 29 days gone by from 2-4-2000 to 3-4-2000.
```

2. Fruit Salad

Program Name: Fruits.java Input File: fruits.dat

Sally is throwing a party and wants to make a fruit salad. She wants to try different combinations in the salad, perhaps not using all of the fruit she buys. She needs your help in showing her what possible fruit salads she can make.

Given a list of fruits and the number of fruits to be used in the salad, show all the possible combinations using exactly three fruits.

Input: The first line will contain a single integer `n` that indicates the number of data sets that follow, with each on one line. Each data set consists of a list of at least three fruits. All values in the data set are separated by a single space. There will be no duplicate fruits listed, each fruit will be a single word, and the list of fruits will be in alphabetical order.

Output: All of the possible combinations of fruit salad, with the fruits listed in alphabetical order for each combination, and all combinations listed in alphabetical order. The fruits in each combination should be separated by a single space. Each combination should be on one line. And each complete set of combinations for a single input should be followed by a blank line.

Example Input:

```
2
banana kiwi orange strawberry
banana blueberry cherry kiwi strawberry
```

Example Output:

```
banana kiwi orange
banana kiwi strawberry
banana orange strawberry
kiwi orange strawberry

banana blueberry cherry
banana blueberry kiwi
banana blueberry strawberry
banana cherry kiwi
banana cherry strawberry
banana kiwi strawberry
blueberry cherry kiwi
blueberry cherry strawberry
blueberry kiwi strawberry
cherry kiwi strawberry
```

3. N'bonacci

Program Name: Nbonacci.java **Input File: nbonacci.dat**

Most people have heard of the Fibonacci sequence. It is a sequence given two base terms, where each term after term zero and one are the sum of the previous two. The fibonacci sequence is as follows: 1, 1, 2, 3, 5, 8, 13… and so on. There is such thing as a tribonacci sequence as well, where every term depends on the previous three: 1, 1, 1, 3, 5,9,17,… and so on. Help simulate an n'bonacci sequence, where each term is the sum of the last n terms.

Input: The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will start with two integers x and y, with x denoting the number of previous terms every term depends on, and y representing the position of the n'bonacci term to print (where the first term is in position 1, etc.). The next line will consist of x integers, representing the first x numbers of the sequence.

Output: Output the y^{th} term of the given n'bonacci sequence

Example Input:

```
3
2 6
1 1
3 8
1 2 3
5 8
1 1 1 1 1
```

Example Output:

```
8
68
17
```

4. Lawn Mower

Program Name: Lawn.java Input File: lawn.dat

Joe's Lawn Service has been hired to mow a rectangular lawn with thick hedgerows through which the lawnmower cannot pass. When starting the mower at a particular place in the lawn, your job is to show how much of the lawn can be mowed without crossing a hedge.

The grown grass to be mowed is indicated by a `":"` and the hedgerows by a `"&"`. The mower can only go along the rows and columns of the lawn, and cannot squeeze through any hedgerow bushes, even diagonally.

Input: The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set consists of two integers R and C, indicating the size of the lawn in rows and columns, followed by an RxC grid of characters indicating the grass and hedgerows, with a single `'X'` character indicating the starting location of the lawn mower.

Output: Show the lawn after the mower has cut as much grass as possible from its starting location. The `"."` character is used to indicate cut grass. The mower should not appear in the output. One blank line will follow each lawn output.

Sample input:

```
2
5 5
::::X
:::::
:::::
&&&&&
:::::
10 18
::&&:&&::&&&:::&&:::
&::::::::::::::::::
:&::::::::::&&:::::&&
::&::::::&&&&&::::&&
:::&&&&&&&&&&:::::::
::::&&&:&&&:::::&&::
::::::::&::::::::&&:::
:::::&&&&&&::::&&&::
:::&&&&:::&&&&::::::
::X:::::::::::&&&&&&
```

Sample output:

```
.....
.....
.....
&&&&&
:::::

::&&:&&::&&&:::&&:::
&::::::::::::::::::
.&::::::::::&&:::::&&
..&::::::&&&&&::::&&
...&&&&&&&&&&:::::::
....&&&:&&&:::::&&::
.......&::::::::&&:::
.....&&&&&&::::&&&::
...&&&&...&&&&::::::
..............&&&&&&
```

5. Name, Rank, Serial Number

Program Name: NRS.java **Input File: nrs.dat**

Some common ranks in the military are Private, Corporal, Sergeant, Lieutenant, Captain, Major, Colonel and General, with numerous variations in between. When captured by the enemy, according to established rules, military personnel are only required to give their name, rank, and serial number.

You are to take information from a list given to you and output what each person in the input is supposed to say in the event they are captured. The input contains names of well known real or fictional military persons in history, literature, TV, and movies. Each first name, last name and rank are single words without spaces and you may assume capitalization is correct.

Input: The first line will contain a single integer `n` that indicates the number of data sets that follow. Each remaining line contains one data set consisting of a first name, last name, rank, and a 5-digit serial number.

Output: For each data set, output the sentence, exactly as shown (on two lines).

Sample input:

```
4
Jessica Lynch Private 51679
Mel Brooks Corporal 19064
Bob Ross Sergeant 94358
George Bush Lieutenant 42913
```

Sample output:

```
My name is Jessica Lynch, Private,
serial number five one six seven nine!
My name is Mel Brooks, Corporal,
serial number one nine zero six four!
My name is Bob Ross, Sergeant,
serial number nine four three five eight!
My name is George Bush, Lieutenant,
serial number four two nine one three!
```

6. Overtime

Program Name: Overtime.java **Input File: overtime.dat**

You have just joined the working world at a job that pays $10 an hour. However, you do get some additional bonuses. The normal workday is 8 hours in length, but any hour beyond 8 for that day is paid an additional $2. Furthermore, if your weekly total exceeds 40 hours, $4 is paid for each additional hour (the $2 and $4 are additive and may both be earned for the same hour). Finally, a 100% bonus is paid for whatever you earn on a Saturday, and a 50% bonus for whatever you earn on Sunday.

For example, if you work 8 hours each on Monday, Tuesday and Wednesday, 10 hours on Thursday, and 6 hours on Friday, you have worked exactly 40 hours (40 X $10 = $400), but get paid two hours overtime (2 X $2) for Thursday, for a total gross pay of $404.

The next week you only worked 4 hours on Sunday and 6 hours on Friday, so your pay is $40 for Sunday and $60 for Friday, but since Sunday gets a 50% bonus, the pay for that day increases to $60, for a total gross pay of $120 for the week.

Input: The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set consists of seven integer values, all on one line, separated by single spaces, representing the hours worked for the seven days of the week, Sunday through Saturday.

Output: For each data set, output the total gross pay for that week, in dollar format, each output on a separate line with no blank lines in between.

Sample Input:

```
3
0 8 8 8 10 6 0
4 0 0 0 0 6 0
8 7 6 7 8 7 6
```

Sample Output:

```
$404
$120
$650
```

7. PaliNum

Program Name: PaliNum.java **Input File: pal.dat**

A PaliNum (palindrome number) is like the number 4884, or 56265, where the digits are in the same order forwards and backwards. There is a theory that all numbers will eventually become palinums under a special process, such as the one described below..

Given a number, reverse its digits and add the resulting number to the original one. If the result is not a palinum, repeat the process. For example, 87 eventually becomes the palindrome number 4884, going through the process described. 87 + 78 = 165, 165 + 561 = 726, 726 + 627 = 1353, and 1353 + 3531 = 4884.

Your job is to find the PaliNum for each given integer, or output the non-palinum result after five steps of the process.

Input: The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set consists of a single integer on its own line.

Output: The resulting PaliNum for each integer, or the non-Palinum after five steps in the process.

Sample input:

```
3
87
196
1689
```

Sample output: (Note the second output is NOT a PaliNum, but is the result after five steps)

```
4884
52514
56265
```

8. ParseCon

Program Name: ParseCon.java **Input File: parsecon.dat**

The general form of all quadratic equations involving two variables that result in a conic section is $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$, with A, B, and C not all equal to zero. For circles, A and C are both positive, and A is always equal to C, but for ellipses A is not equal to C, with both positive, and B is always 0.

Given an equation for a circle or an ellipse, determine and output in order the six values of A through F for the general equation.

Input: The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set consists of a standard form equation for a circle or an ellipse on its own line, containing no spaces. Any term with a zero coefficient is not shown.

Output: The six integer values of A through F, in the order and format as shown below.

Sample Input:

```
3
x^2+y^2+4x-6y-3=0
x^2+4y^2-6x-16y-11=0
x^2-xy+y^2-25=0
```

Sample Output:

```
1 0 1 4 -6 -3
1 0 4 -6 -16 -11
1 -1 1 0 0 -25
```

9. PP

Program Name: PP.java **Input File: pp.dat**

Given primes A and B, and a digit D, count the number of times D occurs in the values produced by the expression $A^N B^M$ for any values of N and M. N and M are any (all) non-negative integers that result in the expression $A^N B^M$ being in the range 10,000 to 99,999, inclusive.

For example, if A = 5 and B = 13, then all values for the expression in the range 10,000 to 99,999 are:

10985, 15625, 21125, 28561, 40625, 54925, 78125

If the given D was 4, the correct output would be 2.

If the given D was 1, the correct output would be 6.

Input: The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set consists of two prime numbers and a digit (0-9) on a single line separated by a single space.

Output: For each data set, output how many times D occurs in all the values produced by the expression mentioned above using the input values for A and B, in the indicated range.

Sample input:

```
4
5 13 6
5 13 0
11 37 2
7 17 1
```

Sample output:

```
3
2
1
8
```

10. Expanded Form

Program Name: Expanded.java Input File:expanded.dat

A common elementary exercise is to write numbers in expanded form. Expanded form involves writing a number as a sum of multiples of powers of ten, with the bigger numbers on top and the smaller ones on bottom. Write a program to simulate this process.

Input: The first line will contain a single positive integer `n` that indicates the number of data sets that follow. Each data set will consist of a single integer to be converted to expanded form.

Output: Print the number in expanded form, one power of ten multiple per line. make sure all numbers are properly aligned. and after all of the numbers to be added, create a line of a single plus followed by as many dash characters as needed, and afterwards print out the final sum.

Example input:

```
2
123456
10010
```

Example output:

```
100000
 20000
  3000
   400
    50
     6
+-----
123456

10000
   10
+----
10010
```

11. That's A Wrap!

Program Name: ThatsAWrap.java **Input File: wrap.dat**

Implement this simple encryption algorithm. The algorithm applies the following rules:

1. All lowercase letters map to the lowercase letter 8 letters above the original letter. You should wrap around the beginning of the alphabet if necessary. For example:

   a ⇒ i, d ⇒ l, z ⇒ h

2. All uppercase letters map to the uppercase letter 4 letters below the original letter. You should wrap around to the end of the alphabet if necessary. For example:

   F ⇒ B, Y ⇒ U, B ⇒ X

3. The following punctuation maps to the article of punctuation 2 positions above, using the following ordering:

   , (comma) . (period) ? (question mark) ! (exclamation point)

   You should wrap around to the beginning of the list if necessary. For example:

   , ⇒ ? ! ⇒ .

All other characters remain unchanged between the input and the output.

Input: The first line will contain a single positive integer `n` that indicates the number of data sets that follow. Each data set will be a sentence on a single line. You should encrypt all lower and uppercase letters and the punctuation listed in (3) above - for all other characters leave them unchanged in the output.

Output: An encrypted version of each input sentence, with each output on its own line.

Sample input:

```
3
Hello, Bob!
y'all come back no, ya hear?
See ya Later AlliGator.
```

Sample output:

```
Dmttw? Xwj.
g'itt kwum jiks vw? gi pmiz,
Omm gi Hibmz WttqCibwz!
```

12. Tri, Tri Again

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

Write the code to output the triangle below, exactly as you see it, in any way you can. No data file input is required.

Output: (40 lines)

```
*
**
***
****
*****
******
*******
********
*********
**********
***********
************
*************
**************
***************
****************
*****************
******************
*******************
********************
*********************
**********************
***********************
************************
*************************
**************************
***************************
****************************
*****************************
******************************
*******************************
********************************
*********************************
**********************************
***********************************
************************************
*************************************
**************************************
***************************************
****************************************
```

13. Magic Squares

Program Name: Magic.java **Input File: magic.dat**

A magic square is a square of numbers such that every row, column, and diagonal adds up to the same value. In addition, for a square with side length = x, every number from 1 to x^2 must appear exactly once in the square. Write a program that determines whether a given square of numbers is a magic square. Note: there are many ways to fail being a magic square, not all are included in the examples, so be careful!

Input: The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will start with a single integer `x` denoting the side length of each potential magic square. The next `x` lines will each contain `x` numbers, representing one potential magic square.

Output: Output either `"magic square"`, or `"not a magic square"` for each input, with each output on its own line.

Example input:

```
3
2
1 1
1 1
3
4 4 7
3 5 7
8 6 1
4
 1  8 12 13
14 11  7  2
15 10  6  3
 4  5  9  16
```

Example output:

```
not a magic square
not a magic square
magic square
```

14. Four is Cosmic

Program Name: Cosmic.java Input File: cosmic.dat

The ultimate math algorithm: Four is Cosmic. This algorithm begins by transforming a number into a sentence. Then a rule is applied to the sentence to create a new sentence. Then the same rule is applied to the new sentence, and so on.

An example sequence following the Four is Cosmic algorithm:

100 => one hundred is ten, ten is three, three is five, five is four, and four is cosmic

See the pattern? The next term in the sequence is the number of letters in the current term. Given an initial term, trace the pattern until it terminates at "four is cosmic". Do not consider spaces part of a number's character count. There should also be no "and" or punctuation in the spellings of any numbers.

Input: The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set is comprised of a single integer `x` denoting the starting number of each pattern.

5<=x<1000

Output: Output one step of each pattern on each line, ultimately printing out "four is cosmic". Separate each complete set of steps by a single blank line.

Example input:

```
3
100
999
207
```

Example output:

```
one hundred is ten
ten is three
three is five
five is four
four is cosmic

nine hundred ninety nine is twenty one
twenty one is nine
nine is four
four is cosmic

two hundred seven is fifteen
fifteen is seven
seven is five
five is four
four is cosmic
```