January 2016 Packet 1

**Computer Science Competition**

Hands-On Programming 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 | Days Gone By |
| Problem 2 | Fruit Salad |
| Problem 3 | King |
| Problem 4 | Lawn Mower |
| Problem 5 | Name, Rank and Serial Number |
| Problem 6 | Overtime |
| Problem 7 | Palinum |
| Problem 8 | ParseCon |
| Problem 9 | PP |
| Problem 10 | Stuck Robot |
| Problem 11 | That’s a Wrap! |
| Problem 12 | Tri, Tri Again |

**1. DaysGoneBy**

**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 a pair of three integer value sets, each representing a date in the same year, with the first set guaranteed to be earlier by at least one day than the second set, calculate and output the number of days gone by.

**Input**: Several pairs of three integer value sets (M, D, Y), representing the month, day, and year. for each date. Each set is on one line, and the Y value for each set will be any year 1700<=Y<=2020, but will be the same value for both dates.

**Output**: A sentence that reports the two dates and the number of days gone between the two dates.

**Sample input:**

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 2015-01-01 to 2015-01-10.

There are 364 days gone by from 1714-01-01 to 1714-12-31.

There are 29 days gone by from 2000-02-04 to 2000-03-04.

**2. Fruit Salad**

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

Sally is throwing a party, and wants to make a fruit salad. Cost is of no concern to her, but the quality of the fruit at the store is, so she buys all of the fruits that she deems to be of good quality, whether or not she puts them in the salad. She also 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 possible list of fruits, and the number of fruits to be used in the salad, show all the possible fruit combinations, each combination with the fruits listed in alpha order, and the entire list of combinations also alphabetized.

**Input**: Several data sets, each on one line, with an integer N representing the number of desired fruits in the salad, followed by a list of fruits. Each data set will be on one line, and each item in the data set will be separated by one space. There will be no duplicate fruits listed, and each fruit will be a single string, but the list of fruits may not necessarily be in alpha order.

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

**Assumptions**: For each data set, 2<=N<= number of fruits purchased.

**Example Input:**

3 strawberry orange banana kiwi

10 pineapple orange tomato banana kiwi avocado strawberry blueberry mango cherry

2 banana kiwi strawberry blueberry cherry pineapple orange

Example Output:

banana kiwi orange

banana kiwi strawberry

banana orange strawberry

kiwi orange strawberry

avocado banana blueberry cherry kiwi mango orange pineapple strawberry tomato

banana blueberry

banana cherry

banana kiwi

banana orange

banana pineapple

banana strawberry

blueberry cherry

blueberry kiwi

blueberry orange

blueberry pineapple

blueberry strawberry

cherry kiwi

cherry orange

cherry pineapple

cherry strawberry

kiwi orange

kiwi pineapple

kiwi strawberry

orange pineapple

orange strawberry

pineapple strawberry

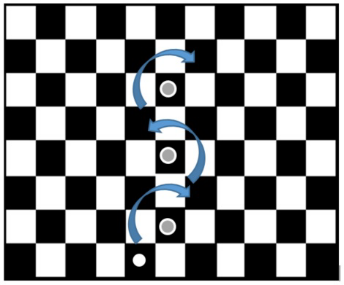
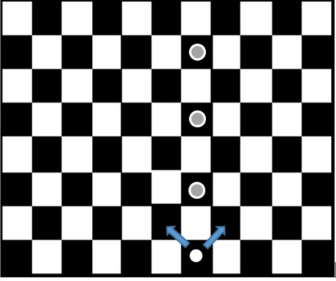
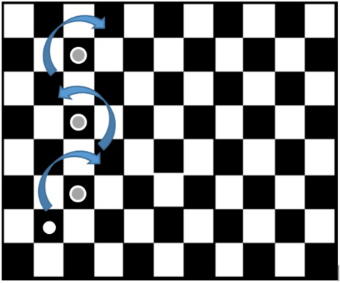
**3. King**

**Program Name: King.java Input File: king.dat**

A checker board is an 8X8 grid of red and black squares (in this example, white and black), but only the black squares are used. In the first board shown below, a white checker piece is at position (1,5) with opposing grey pieces at positions (2,6), (4,6), and (6,6), respectively. The white piece is able to make three jumps, but falls one row short of making it to the top row where it could become King.

In the second board, the white piece at (2,2) can also make three jumps, and lands on the top row where it is "Kinged".

On the third board, only one move is possible, either to the left or right diagonally.



Your job is to determine how many moves/jumps (if any) can be made from a given situation, and whether or not the final move results in King status for the white piece.

**Input**: The data file will contain several lines of data, each line containing several integers, the first two of which indicate the location of the white piece, then an integer N, followed by N pairs of integers indicating the locations of the grey pieces. It is guaranteed that only one white piece will be on the board, and that all pieces are on black squares.

Output: For each data set, determine how many moves in one turn the white piece can make, and whether or not the turn results in King status for the piece, formatted as shown below, with a comma, space, and the word KING following the integer for any turn that results in King status.

Sample input:

1 5 3 2 6 4 6 6 6

2 2 3 3 3 5 3 7 3

1 7 3 3 7 5 7 7 7

1 7 2 2 6 4 4

Sample output:

3

3, KING

1

2

**4. Lawn Mower**

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

Joe's Lawn Service has been hired to mow a rectangular lawn, which has thick hedgerows through which the lawnmower cannot pass, but through which the workers can pass, barely.

When starting the mower at a particular place in the lawn,your job is to show how much of the lawn can be mowed  at one time WITHOUT requiring the workers to physically pick up the mower over their heads to squeeze through hedges.

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

**Input:** The data file will contain an initial value N, indicating how many data sets will follow. Each data set consists of two integers R and C, indicating the size of the lawn in rows and columns, followed by and 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. Assume the mower has been complete removed from the lawn by the workers. One blank line will follow each lawn output.

**Sample input:**

2

5 5

::::X

:::::

:::::

&&&&&

:::::

12 20

::&&:&&::&&&:::&&:::

:&::::::::::::&&&&&&

&:::::::::::::::::::

:&:::::::::&&:::::&&

::&::::::&&&&&::::&&

:::&&&&&&&&&&:::::::

::::&&:::&&&::::::::

::::&&&:&&&:::::&&::

:::::::&:::::::&&:::

:::::&&&&&&::::&&&::

:::&&&&:::&&&&::::::

::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, although it seems lately the rules may have changed a bit.

Nevertheless, you are to take information from a list given to you and output what each person on the list is supposed to say in the event they are captured. The list contains names of well known real or fictional military persons in history, literature, TV, and movies.

**Input:** Several data sets, each on one line, each containing a first name, last name, rank, and serial number.

**Output:** For each data set, output the sentence exactly as shown.

**Sample input:**

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 work-a-day world at a job that pays $10 an hour (not bad considering minimum wage in Texas is currently $7.25 as of February 2015). 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 $1.50. Furthermore, if your weekly total exceeds 40 hours, $2.50 is paid for each additional hour. Finally, a 125% bonus is paid for whatever you earn on a Saturday, and a 50% bonus for 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 $1.50) for Thursday, for a total gross pay of $403.00.

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:** Several sets of seven integer values, all on one line, separated by single spaces, representing the hour 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, rounded to the nearest penny, each output on a separate line, with no blank lines in between.

**Sample Input:**

0 8 8 8 10 6 0

4 0 0 0 0 6 0

8 7 6 7 8 7 6

**Sample Output:**

$403.00

$120.00

$627.50

**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:** Several integers in a data file, each on one line.

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

**Sample input:**

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 , 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**: Several standard form equations for a circle or an ellipse, each on one 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:**

x^2+y^2+4x-6y-3=0

x^2+4y^2-6x-16y-11=0

x^2+y^2-25=0

**Sample Output:**

1 0 1 4 -6 -3

1 0 4 -6 -16 -11

1 0 1 0 0 -25

**9. PP**

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

An interesting but probably useless number exercise might be counting how many times any particular numeric digit occurs in a series of values, such as prime numbers from 1 to 100, or even powers of primes 1000 to 9999. How about products of prime powers from 10,000 to 99,999? Let's do it.

Suppose A and B are two prime numbers, N and M are non-negative integer exponents, and D is a digit that may or may not appear in a value produced by the expression  in the range 10,000 to 99,999, inclusive. Your job is to take two given prime numbers and a digit value D and count how may times D occurs in the value produced by the expression indicated above, using the two given values of A and B.

For example, if A was 5 and B was 13, one value that falls within the specified range is 5^1\*13^ 3= 10985, which contains the digits 1, 0, 9, 8 and 5. Another is 5^5\*13^1=40625.

**Input:** Several data sets, each on one line, containing two prime numbers and a digit (0-9).

**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:**

5 13 6

5 13 0

11 37 2

7 17 1

**Sample output:**

3

2

1

8

**10. Stuck Robot**

**Program Name: StuckRobot.java Input File: stuckrobot.dat**

You and Tony want to build an AI robot to write essays for you. Tony has an idea about how this robot should work, but you disagree with the idea. The notion is to feed the robot a chunk of text, and give it a starting letter, and the robot will be able to reproduce english-like text to write essays for you.

Given the starting letter, the robot should look through the text to find all two letter sequences that start with the starting letter. From these sequences choose the one whose 2nd letter is the most frequent.. Once this most frequent letter is found, the robot then looks through the text to find all two letter sequences that start with this letter and then finds the most frequent letter that follows that letter, so on and so on, until a character limit is reached.

For example, if the text is:

**The down brown fox jumped over the lazy dog**

the character limit is 5, and the starting letter is e, then the robot will produce:

e dow

because e is the first letter, and the most frequent character to follow the e is ‘ ‘, and the most frequent character to follow the space is d, and the most frequent character to follow d is o and the most frequent character to follow o is w. The process stops at this point since the resulting string has reached a length of 5, the character limit provided.

In your skepticism, you believe this robot design will result in an AI that is jibberish, that the essay it produces will be nonsense, and you would get an F if you turned it in. However, Tony is being stubborn, so to prove him wrong you are going to code it up and show him that it isn’t going to work well.

**Input**: The sample text will all be on the first line of the data file. The lines following the first line will be an integer **c**, representing the character limit, and a character **s**, representing the first character for the robot.

**Output**: The string of **c** characters starting with character **s** that the robot will produce under Tony’s AI.

**Assumptions**: There will never be a tie for the most frequent letter. The given character will exist at least once in the paragraph. The new line and the empty string are **not** considered characters however the space and all punctuation are.

**Example Input:**

abcdefghijklmnopqr stuvwyzfffghijijiklm

10 a

12 j

5 p

**Example Output:**

abcdefffff

jijijijijiji

pqrst

**11. That's A Wrap!**

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

Encoding can be simple or complex, depending on the process used. This is one is fairly simple. For any uppercase letter from A-E, multiply its numerical value by 2 and produce the resulting letter. For example, B is second in the alphabet, and would be encoded to a D, the fourth letter in the alphabet. For the letters F-J, divide the numerical value by 3 and multiply the integer remainder by 5. For K-O, divide by 4 and multiply the integer part of the quotient by 8. For P-T (16-20), add the two digits together and multiply by 10. P would result in 7 X 10, or 70. For the letters U-Z, find the largest integer factor of its numerical value less than the value itself and multiply that by 12.

Since some calculations will exceed 26, the numerical value for Z, you must "wrap" around the alphabet enough times until the final value settles within the range 1-16. For example, the value 70 from the letter P in the above example encodes to R.

**Input**: Several characters, each on a separate line.

**Output**: A mapping for each letter that shows the original letter paired to its encoded letter, as shown below.

**Sample input:**

B

P

G

**Sample output:**

B==>D

P==>R

G==>E

**12. Tri, Tri Again**

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

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

The challenge, of course, is whether you can do it algorithmically, using only one output statement producing only one "\*", within an appropriate control structure, instead of just using hard-coded brute force output statements. One way is much more elegant and impressive, and the other is just, well, not.

Think about it and give it your best shot!

**Output:**

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*