Computer Science Competition
District 2 2016
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	Aiguo
Problem 2	Ba Tu
Problem 3	Chin
Problem 4	Daisuke
Problem 5	Ekaterina
Problem 6	Felipe
Problem 7	Gary
Problem 8	Heather
Problem 9	Isabel
Problem 10	Jakub
Problem 11	Kartik
Problem 12	Lauren

1. Aiguo

Program Name: Aiguo.java Input File: aiguo.dat

Aiguo is an avid reader and really loves grammar. He enjoys reading proofs of works, and gets to make changes to these works before they get published! However, he has noticed that it is incredibly time consuming to really highlight the changes he makes in these electronic versions. What Aiguo really wants is the ability to just make the changes he sees fit with his expertise and have a program analyze the changes he made to report to the publisher.

The changes that Aiguo makes can be represented by deletions or additions to the original text. He gets an initial work, then will either add or remove words and characters to improve and revise the works. He would like you to write a program that can identify which characters he removed and added by looking at the initial and final version of his work.

Input: The first integer will represent the number of data sets to follow. The first line of each data set will be the initial work, before Aiguo's changes. The second line of each data set will be the final work, after Aiguo's changes.

Output: Each data set should have 3 lines of output. The first, labeled "added: " will be the characters that Aiguo added to the initial text. The second, labeled "removed: " will be the characters that Aiguo removed from the initial text. The last, labeled "unchanged: " will be the characters that Aiguo did not change. Each of these should be listed in the order that the changes were made. If there exists an exchange where it was ambiguous if a correction was an additional of one word or the removal of another, choose to state the correction that comes first in the text.

Assumptions: Each work will be on one line.

Sample Input:

```
3
the brown fox curiously jumped over the sad dog
the quick brown fox jumped over the lazy dog
Ceci n'est pas ma femme.
Ceci est ma femme.
I love chocolate!
He loves chocolates!
```

Sample Output:

```
removed: curiously sd
added: quick lzy
unchanged: the brown fox jumped over the a dog
removed: n'pas
added:
unchanged: Ceci est ma femme.
removed: I
added: Hess
unchanged:  love chocolate!
```

2. Ba Tu

Program Name: BaTu.java Input File: batu.dat

Ba Tu has just learned to play Sudoku, but still needs help figuring out what values any particular square could be. The rules are that every row, column, and every 3X3 sub-box within the large grid have unique values from 1 to 9.

Your job is to help Ba Tu get started with any particular empty box, showing him what digit, or digits, could possibly work for that empty box.

In the puzzle shown here, the empty blank at position 1,1 at the top left corner of the puzzle cannot be a 5, 1, or 4 since that row already contains those numbers, and it cannot be 2, 3, or 9 since that column contains those, and it cannot be a 7 since there is a 7 in the sub-box for that blank. That means that 6 or 8 would work in that blank.

_ 5 _ 1 _ 7 _ _ _	_ 1 _ _ _ _ 9 _ 5	_ 4 _ 6 _ 2 _ _ _
2 _ 8 _ 4 _ 9 _ 1	_ 3 _ _ 7 _ _ 8 _	5 _ 1 _ 2 _ 4 _ 6
_ _ _ 3 _ 4 _ 2 _	4 _ 1 _ _ _ _ 6 _	_ _ _ 7 _ 9 _ 1 _

For the blank at row 4, column 2, between the 2 and 8, the only values that could work are 6 and 7, since that row already has the digits 1, 2, 3, 5 and 8, the column also has 4, and that sub-box also contains the 9.

Input: Nine strings of nine digits, each on a separate line, representing the beginning values of a 9X9 Sudoku puzzle. A zero digit represents an empty blank. After the initial nine strings will be several pairs of integers representing the column and row of an empty blank, each pair on a separate line.

Output: All the possible digits that could be placed in the empty blank indicated by each pair of digits. The digits are to be listed in ascending order with no spaces between.

Sample input:

```
050010040
107000602
000905000
208030501
040070020
901080406
000401000
304000709
020060010
1 1
4 2
2 5
3 5
```

Sample output:

```
68
67
4
24
```

3. Chin

Program Name: Chin.java Input File: chin.dat

In calculus, there exists an operation called a derivative. This operation will tell you the slope of the curve at a given point. The rules for finding the derivative for a polynomial are very well defined, and rather formulaic. Chin is getting behind on his calculus homework, and he would like you to write an evaluator, that given an equation and the point at which to evaluate it, determines the slope at that point. This means, you will have to take the derivative of the polynomial, as defined by the rules below, and evaluate the new equation produced by taking the derivative, at the point given.

The rules for determining the derivative are as follows:

1. if there is a term without an x in it, remove it
2. if there is a multiplication, e.g., ***a * x***, keep only the coefficient, e.g., ***a***. (Note, x / 7 is equivalent to (1 / 7.0) * x)
3. if there is an exponent, e.g., ***a * x^b***, multiple the x by the exponent, and reduce the value of the exponent by 1, e.g., ***a * b * x ^ (b-1)***.

Following these rules in the equation: **5 * x^4 + 3 * x + 2** gives us:

By rule 1, the "**+ 2**" at the end is removed. By rule 2, "**3 * x**" becomes "**3**". By rule 3, the "**5 * x^4**" becomes "**5 * 4 * x^3**".

All of Chin's equations will be exponents (^), multiplication (*), divisions (/), subtractions (-), and additions (+). All coefficients and exponents will be integers. There will be no parenthesis or brackets and typical order of operations do apply.

Input: The first integer will represent the number of data sets to follow. The first line of the dataset will be the polynomial Chin wants you to take the derivative of. The second line of each data set will be one floating point number that will be the point (value of x) at which to evaluate the expression derived.

Output: For each output, display the value of the derivative of the given polynomial evaluated at the given point, rounded to two decimal places.

Assumptions: All multiplications, additions, subtractions, and divisions will be separated by a space. All values will be positive.

Sample Input:

```
3
5 * x^2 + x / 7 + 9
1372.3
7 * x^3 + 7 * x + 15
100.1
1 / 7 + 17 + 9 * x + 15 * x^3
0.0
```

Sample Output:

```
13723.14
210427.21
9.00
```

4. Daisuke

Program Name: Daisuke.java **Input File: daisuke.dat**

Daisuke loves to create symmetrical character patterns, like the one shown below. Your job is to write a program that outputs this exact example of his work.

Input: NONE

Expected output:

```
\"""/
%\''/%
%.\/.%
%./\.%
%/''\%
/"""\
```

5. Ekaterina

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

Ekaterina has just read the classic Jules Verne story **Around The World In Eighty Days**, where the main character Phileas Fogg claimed he could do the title task, making a huge bet with some of his friends, and then proceeding to head east, on his journey around the world. Phileas was a fanatic about time, and was always checking the time and measuring how far he had come, and what time it was in London, his demarcation point, and ultimate destination.

He would also try and predict what time it was in future destinations. For example, if he was in London, he would be in the zero time zone, otherwise known as Greenwich Mean Time, based on the location of the Royal Observatory in Greenwich, a suburb of London. If he wanted to know the current time in France, all he would do is add an hour, since France was in the next time zone to the east.

Currently all of the UK, Portugal, and parts of West Africa, are in the GMT zone. As you head East into continental Europe and beyond, you pass into other time zones, each one roughly 15 degrees to the East, with 360 degrees representing a full circumnavigation of the globe.

Each new time zone eastward has a time difference of one hour more than the previous one. For example, France, Germany and Italy all are at 1 PM if Big Ben in London strikes noon. Beyond that into Eastern European countries, the next zone shows 2 PM, and so on, around the world.

Heading west, the same thing happens, except in reverse. Each time zone 15 degrees further west is another hour earlier. For example, New York City is 5 time zones to the west, 5 hours earlier, so when it is 7AM in New York, it is already noon in London.

A number line to show this would have zero in the middle, indicating Greenwich time, 15E starting the time zone containing Western Europe, 30E starting the Eastern European zone, and so on. Halfway around, at 180 degrees, is the International Date Line.

```
                                NY              GMT
IDL 165W 150W 135W....90W 75W 60W 45W 30W 15W 0 15E 30E 45E...135E 150E 165E IDL.
```

Phileas would have appreciated a computer program to help him keep track of all of this, like the one Ekaterina wants to write to solve this problem, but needs your help.

Input: Several data sets, each on one line, consisting of four items. The first indicates the starting location, with a degree value between 0 and 180, inclusive, followed by the letter 'W' or 'E', the second a three letter abbreviation for the current day of the week, the third the two digit time of the current location, followed by "AM" of "PM", and the fourth the remote time zone, using the same format as the first data item.

Output: The day and time at the remote location, in the format HHXM, with HH meaning the two-digit time and the X either the letter 'P' or 'A'.

Sample input:

```
20E WED 01PM 0E
65W FRI 07AM 5E
4E SAT 06AM 23E
157W MON 07AM 110W
170E SAT 06PM 140W
```

Sample output:

```
WED 12PM
FRI 12PM
SAT 07AM
MON 10AM
FRI 09PM
```

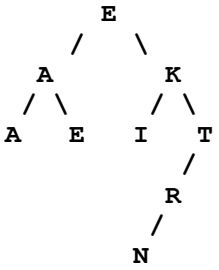
6. Felipe

Program Name: Felipe.java Input File: felipe.dat

Felipe is fascinated with binary search trees, especially with the four different traversals he just learned in class: inorder, preorder, postorder, and reverse order.

According to the rules he learned in class about building a character tree using a word, he practices doing this with his friend's names, and especially enjoys using his beautiful girlfriend's equally beautiful name, EKATERINA, with whom he is head-over-heels in love!

He starts with the first letter, E, as the root, and then sees that the next letter, K, is alphabetically after E, so he inserts it as a right node to the E, and then the A as a left node to the E. When he gets to the T, he goes right at the E, and then right again at the K, and proceeds on through the rest of the letters of her name, resulting in the binary search tree shown below. He decides to allow duplicate letters and slides those to the left as they reach a matching node.



He then proceeds to traverse in order, which means start at the top of the tree, going as deep to the left along each branch, and only outputting a node when reaching the end of a branch (a leaf node) or outputting a node after its left branch has been completely processed.

The inorder traversal (alpha order) of this tree would be: **AAEEIKNRT**
The preorder traversal (output a node BEFORE traversing either the left or right subtrees) would be: **EAAEKITRN**
Postorder traversal (output after both branches are completely traversed): **AEAINRTKE**
The reverse order traversal is just the inorder traversal backwards, which Felipe thinks about for a bit, and then realizes what to do, resulting in: **TRNKIEEAA**

Input: Several uppercased names containing no other symbols , each on one line, and each followed by a single character indicating the traversal to be performed: **E**(preorder), **O** (postorder), **I** (inorder), **R** (reverse order).

Output: The indicated traversal for each name, as shown in the examples.

Sample Input:
```
EKATERINA I
EKATERINA E
EKATERINA O
EKATERINA R
```

Sample Output:
```
AAEEIKNRT
EAAEKITRN
AEAINRTKE
TRNKIEEAA
```

7. Gary

Program Name: Gary.java Input File: gary.dat

Gary has just encountered boolean postfix expressions in class, and is a bit confused. He understands the easy ones, like, "true and false" becomes "true false and", and knows how to evaluate them according the following rules of thumb:

- "and": evaluates to "false" only when both operands are "true"
- "or": evaluates "false" only when both operands are "false"
- "xor": evaluates to "true" only when the operand values are opposite
- "not": reverses the value from "true" to "false", or vice versa

It's evaluating the more complex ones in postfix form that give him a bit of trouble.

For example, "true or false and true" is confusing to him. "*What is the postfix version?*", he wonders, "*and then what is the answer?*" Well, his teacher helps him out and just gives him the postfix version, for now, which is "true false true and or", and then shows him how to process it.

His teacher says, "*Take the first two boolean values and see if there is an operator immediately after them. If there is, do that operation, and replace the result in place of that expression. If not, ignore the first operand for now, and look at the next pair to see if there is an operator following, and evaluate it and replace it with an answer.*"

"*In the example, "true false true and or", there is no operator for the first two operands, so you ignore the first operand and look at the next two, which have an "and" after them. Well, "false and true" evaluates to "false", so you replace the "false true and" with just "false", which now gives you the expression, "true false or", which is "true", and is the final value of the expression.*

"*If you have a "not" in the expression, like this one, "true false or not", the result here would first be, "true not", since "true false or" becomes "true", and then "false", since the "not" operator only needs one operand, and "not true" means "false".*"

Input: Several lines of data, each line with a postfix boolean expression made up of the words, "true", "false", "not", "and", "or", and "xor", with no parentheses, all separated by single spaces.

Output: The final **true** or **false** value of the postfix boolean expression.

Sample input:

```
true true or
true false and
true false true and or
true false or not
```

Sample output:

```
true
false
true
false
```

8. Heather

Program Name: Heather.java **Input File: heather.dat**

Heather has just learned that it is possible to have any base you want, other than base ten, two, eight, and sixteen. She figures since there are 10 counting digits and 26 letters of the alphabet, you could easily represent a value up to base 36 if you wanted to.

She decided to make up an exercise where two values were represented, each value in a different base, anywhere from base 2 to base 36, where three of the four value/base items were known, and the fourth unknown. She then proceeded to try and find the missing item.

For example, if she knew that 25 base 10 was equal to *"something"* in base 8, she would do some math and determine that 31 was the missing value, and she would write the original four items in the order 25 10 X 8, with X indicating the missing value.

On the other hand, she could write the expression 25 X 31 8, and try to find the correct base that has 25 as the value, and is equal to 31 base 8. The answer, of course, is 10.

The expression 25 10 31 X, would result in 8 as the missing value.

She then tried something really strange, using X 23 FACE 16, wondering if she could figure this one, which she did by converting FACE, base 16 to base 10, and then to base 23, with a final answer of 568D.

Given four items N1 B1 N2 B2 in an expression, with any of the four items replaces with an X, write a program to find the missing value of X in each expression.

Input: Several sets of data, each consisting of four items representing two pairs of value/base pairs, with one of the four items indicated by the letter X, representing a missing value for one of the values, or one of the bases.

Output: The missing value for each data set, with any alphabetic characters shown in uppercase.

Sample input:

```
25 10 X 8
25 X 31 8
25 10 31 X
X 23 FACE 16
A5 X BC 13
```

Sample output:

```
31
10
8
568D
15
```

9. Isabel

Program Name: Isabel.java Input File: isabel.dat

In computer science class, Isabel has been studying binary numbers, and has made up a game to practice her binary skills. She understands the binary place value system, where the first (rightmost) column is worth 1, the second worth 2, and then 4, 8, and so on. She has decided to label these columns A, B, C, etc., with A being the label for the rightmost ones place, B the 2s place, C the 4s place, etc. It helps her to see the whole picture when she arranges the numbers in column format, like this:

```
Base Two    Base ten
C B A
0 0 0 =     0
0 0 1 =     1
0 1 0 =     2
0 1 1 =     3
1 0 0 =     4
1 0 1 =     5
1 1 0 =     6
1 1 1 =     7
```

She sees that for this base ten range of values from 1 to 7, the number 6 has 1s in columns B and C. The numbers that all have 1s in the A column are 1, 3, 5, and 7. The numbers that have 1s in the B column are 2, 3, 6, and 7. In the C column, 1s correspond to the values 4, 5, 6 and 7.

Her game goes like this. She picks a random number between 1 and 500 representing the range of values within which her mystery number will be. She then picks a random letter that would represent one of the columns in the binary form of the maximum number in her range, and then one or more other letters, also column labels within the range of the maximum number's binary form. She then tries to figure out the resulting number, where there would be a 1 in each of the binary columns represented by the letters she picked. After she gets the number, she then counts how many other values have a 1 in the column of the first letter she picked.

For example, if she picked 7 as the maximum value in her mystery number range, and then picked first the letter B, as well as another letter C, the binary form of the mystery number would be 110, since there is a 1 in both the C and B columns but not the A column. This number converted to base ten is the value 6, her mystery number. The other numbers between 1 and 7 that also have a 1 in the B column (the first letter she picked) of their binary forms are 2, 3, and 7, making a total of 4 values that have 1s in the B column within that range.

In another example, with 15 as the maximum range value, and only the letter A picked, the resulting binary form of the mystery number only has a 1 in the A column, making 0001, which is equal to the decimal value 1. Other numbers from 1 to 15 that also have a 1 in the A column are the values 3, 5, 7, 9, 11, 13, and 15, for a total of 8 values.

Input: Several data sets, each on one line with single space separation, consisting of an integer M for the maximum range value ($1<M<=500$), an integer N for the number of letters to follow, and then N capital letters. The letters given will correspond to the binary place values, with A always indicating the ones place, B the 2s place, C the 4s place, D, E, F, and so on.

Output: Two integer values, the first of which is the base ten mystery number represented by 1s in the input letter column labels, as described above, and then a value representing the number of integers within the given range, whose binary form contains a 1 in the column matching the first letter listed in the data set.

Assumptions: All letters in each data set will be unique, and the number and nature of the letters given will not exceed the number of columns required for the binary form of the maximum value indicated.

Sample input:

```
7 2 B C
15 1 A
10 2 B A
130 4 F D E G
```

Sample output:

```
6 4
1 8
3 5
120 64
```

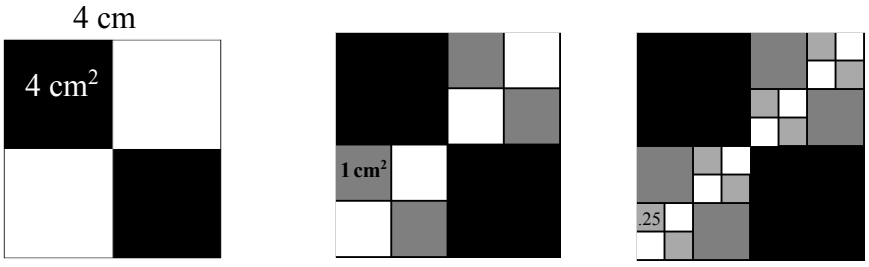
10. Jakub

Program Name: Jakub.java Input File: jakub.dat

Jakub is quite the graphic artist and has been experimenting with drawing square-based patterns, dividing a square into four parts, coloring two diagonal squares, and then dividing the other two smaller squares into four parts, coloring two, dividing the other two, and so on until the divided square reaches a size smaller than 1 centimeter, where they become too small to divide anymore.

His analytical mind wonders what the area is of the colored squares, and sets out to do the math, but has some trouble figuring this out. Please help Jakub calculate how much are of his repeatedly divided square is colored in.

For example, he might start with a 4 cm square, like the one shown below, divide it into four 2X2 squares, each with an area of 4 cm^2, two of which he colors in for a total of 8 cm^2. The two remaining squares are then divided into four 1 cm squares, adding 4 cm^2 to the colored area total, for a total of 12 cm^2, and the remaining 1 cm squares are each divided into squares measuring 1/2 by 1/2, adding a total area of 2 cm^2 more (8 times 0.25 cm^2), for a total of 14 cm^2 of colored-in area.



In another example using a 5 cm square, the total colored-in area would be 21.875 (6.25 + 6.25 + 1.5625 + ….).

Input: Several integers N (1<=N<=100), each on one line, each representing the side length of a square.

Output: The total area for each NXN square colored in using the process described above, rounded to three decimal places of precision.

Sample input:

```
4
5
24
16
```

Sample output:

```
14.000
21.875
558.000
248.000
```

11. Kartik

Program Name: Kartik.java Input File: kartik.dat

Kartik loves to mess around with his friend's names, as well as his own, and has come up with several processes to do this. For example, he may take the leftmost character of his name, delete it, and put a dash on the other end. He calls this a left shift 1, or LS-1, for short. He would notate this as: **LS-1 KARTIK**, and would write the result as:

KARTIK ==> ARTIK-

Similarly, he would do a Right Shift, followed by a number, and call it RS-N, with N being some integer value, no longer than the name he is dealing with. He would make sure all of the numbers he chooses for any of his processes don't go beyond the number of letters in the name he is messing with.

He also does a "circulate", which works like the "shift" operation, except that he inserts each removed letter on the other end instead of putting a dash. He can circulate the letters, either to the right or to the left, with an N value indicating how many times to circulate, but again no greater than the length of the name he is messing with. His notation for this is either RC-N, or LC-N. For example, if he did RC-2 AIGUO, the result would be:

AIGUO ==> UOAIG

where the rightmost two letters of the name are circulated back to the front.

Another circulate process he uses is Mid Circle, and he uses four items to control this one: S, indicating a starting position in the name, 1 being the first letter of the name, L being a number that indicates how many letters of the name to be circulated, starting from S, X indicating how many times to circle the letters, and finally the character R or L to indicate the direction of the circulate. For example, MC-243R EKATERINA would take "KATE", which is 4 letters starting at position 2, and circulate those letters 3 times to the right, which would make "ATEK", and thus the full effect would be:

EKATERINA ==> EATEKRINA

The last process he likes to use is a reverse, but not of all of the letters, but just a few somewhere in the name. For example, if he applied a REV-42 to his friend's name, DAISUKE, he would start with the fourth letter, an S, taking 2 letters from there, "SU", reversing them within the name, resulting in:

DAISUKE ==> DAIUSKE

What he really likes to do is combinations. For example, RS-2 REV-24 HEATHER, would first produce HHTAEER with the reverse, and then --HHTAE. Sometimes he goes crazy and puts several combinations (no more than 6) with a name, always doing the one closest to the name first, and then working backwards until all have been completed.

Input: Several lines of data, each on one line, each consisting of a process, or series of processes, ending with a name, all in caps. Single spaces separate each item in the data line.

Output: The original name, followed by **" ==> "**, and then the resulting name after all processes have been completed.

Sample input:

```
LS-1 KARTIK
RC-2 AIGUO
MC-243R EKATERINA
REV-42 DAISUKE
RS-2 REV-24 HEATHER
```

Sample output:

```
KARTIK ==> ARTIK-
AIGUO ==> UOAIG
EKATERINA ==> EATEKRINA
DAISUKE ==> DAIUSKE
HEATHER ==> --HHTAE
```

12. Lauren

Program Name: Lauren.java Input File: lauren.dat

Lauren loves to travel. She just got her brand new passport, and she is excited to travel the world. She is overwhelmed by the number of cities she wants to visit. To be the most efficient, she wants to be sure to visit every city on her list, but she wants to make sure to do it covering the least amount of distance as possible. Oh, and she doesn't really care about getting back home after reaching the last city…at least for a while.

Input: The first integer will represent the number of data sets to follow. The next line will have an integer representing the number of cities she wants to visit on this trip, followed by a list of cities, each on one line and will contain no spaces. The first city in this list represents the city that Lauren is currently in. The next line will be an integer representing the number of available flights. The following lines will be two cities, separated by a space, and an integer representing the number of miles that flight covers.

Output: For each data set, print the names of the cities in the order that she should visit them. She must start in the city that she is currently in. Each flight is represented by a "=>" sign between the cities.

Assumptions: There will always be a flight to the city Lauren wants to visit. If there is a flight from city A to city B there is assumed to be the same flight from city B to city A. If there is no flight listed between two cities it is assumed she cannot fly between them. There will be at least two cities to travel between. There will always exist a route between any two cities, although it may take several hops. The distance between any two adjacent cities will be greater than 1 mile. The best solution will never require Lauren to backtrack.

Sample Input:

```
3
3
SanFrancisco
NewYork
LosAngeles
3
SanFrancisco LosAngeles 383
SanFrancisco NewYork 2905
LosAngeles NewYork 2789
5
Austin
Istanbul
Calcutta
NewYork
London
10
Austin Istanbul 6500
Austin Calcutta 8773
Austin NewYork 1513
Austin London 4921
Istanbul Calcutta 3652
Istanbul NewYork 5020
Istanbul London 1556
Calcutta NewYork 7929
Calcutta London 4853
NewYork London 3465
4
Reykjavik
Aarhus
Geneva
Budapest
5
Reykjavik Aarhus 1515
Reykjavik Geneva 2379
Reykjavik Budapest 2443
Aarhus Budapest 933
Geneva Budapest 787
```

Sample Output:

```
SanFrancisco => LosAngeles => NewYork
Austin => NewYork => London => Istanbul => Calcutta
Reykjavik => Aarhus => Budapest => Geneva
```