**1. Accounting**

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

In accounting currency formatting, very large values are separated by commas, and negative values are expressed inside parentheses. Below are some examples:

$250.34

$(500.19)

$2,343,555.55

$(59,216.99)

As each transaction is accounted for, the balance is updated by adding the credits and subtracting the debits, so the running balance for the numbers above would be as shown below, with negative balances shown in parentheses, and all values $1,000 or more separated out using commas:

$250.34

$(249.85)

$2,343,305.70

$2,284,088.71

Your job is to write a program that shows the credits, debits, and running balance, formatted as shown below, complete with the top and bottom "\*\*\*\*." lines and the column headings, aligned exactly as shown.

**Input -** Several dollar amounts in accounting currency format as shown above, each on one line. It is guaranteed that all transaction values will be less than 1 billion dollars.

**Output -** A formatted running balance as shown below, complete with the header and footer lines.

**Sample Input**

$250.34

$(500.19)

$2,343,555.55

$(59,216.99)

**Example Output to Screen**

\*\*\*\*.\*\*\*\*.\*\*\*\*.\*\*\*\*.\*\*\*\*.\*\*\*\*.\*\*\*\*.\*\*\*\*.

Transaction : Balance

$250.34 : $250.34

$(500.19) : $(249.85)

$2,343,555.55 : $2,343,305.70

$(59,216.99) : $2,284,088.71

\*\*\*\*.\*\*\*\*.\*\*\*\*.\*\*\*\*.\*\*\*\*.\*\*\*\*.\*\*\*\*.\*\*\*\*.

**2. Awe Pairs**

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

This problem deals with pairs of characters, putting them into groups. Initially, each letter is in its own group. The A is in the A group, the B in the B group, and so on. For each pair of letters you are given to "pair-up", or put in the same group, you first determine which group the letter is currently in, if not its own. If they are both in the same group, you do nothing. If they are in different groups, move all of the letters in the group of the second letter into the group of the first letter, adding them to the back of the list. Then delete the group of the second letter.

For example, in the following string of letters, ABCDEABF, take two at a time and pair them up. For the first two pairs, B is added to the A group, then D is added to the C group. For the EA pair, A and B are added to the E group. For the BF pair, F is added to the E group, of which B is a member. The resulting groups are: CD and EABF. Your job is to output the largest group (or groups if there is a tie), after all the pairs have been processed. The order in which letters are added to the group is the order they are to be output.

**Input -** Several strings of pairs of characters, all uppercase letters.

**Output -** For each input string, output the group (or groups) that contain the most letters, in the output format shown below. One space needs to separate any multiple groups.

**Sample Input**

ABCDEABF

ABCDEFGHDE

UILCONTEST

ALLTHEWAYTOSTATE

**Example Output to Screen**

EABF

CDEF

STE

YWALTHE

**3. Cursed**

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

Nigel has invented a time machine, however he has noticed that he has been randomly having bad days. A variety of horrible things can happen to him on these days, which he calls his ***“cursed”*** days. After several of these days, Nigel noticed these days only happen when the date in MMDDYYYY form is a palindrome. Since he is a young time traveler, he needs your help in recognizing which of these days he should not travel on.

**Input -** A date with 3 parts, the month, the day followed by a comma, and the year.

**Output -** For each date print it in MMDDYYYY format, and the message “DON’T TRAVEL” if the date is a palindrome or “OK TO TRAVEL” if it is not a palindrome.

***Assumptions:*** The input will be a real date in AD. Note that the MMDDYYYY form has 2 characters for month, 2 for day, and 4 for year, if the provided date does not contain enough characters for these fields, they should be padded with leading ‘0’s. (For example the year 109 becomes 0109 to fill 4 characters in year).

**Sample Input**

June 1, 1998

January 9, 109

December 31, 1321

**Example Output to Screen**

06011998: OK TO TRAVEL

01090109: OK TO TRAVEL

12311321: DON'T TRAVEL

**4. Data Count**

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

For this problem, read in each word. Words will be in all caps or all lowercase. If the words are all caps, then print them out in reverse. If the words are all lowercase, print them out in uppercase. There may be numbers and symbols in the words, but no spaces. There will always be at least 1 letter [a-zA-Z] in each word.

**Input –** There will be an unknown number of words.

**Output –** Print out each word per the specifics described above.

**Sample Input**

FUN

go!

HOP1

racecar22

A1B2C3

**Example Output to Screen**

NUF

GO!

1POH

RACECAR22

3C2B1A

**5. GPA**

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

Lake Singkarak High School offers several courses, indicated by a course number (three digit number between 100 and 999), honors designation (YES or NO), and length of the course (year, quarter, or semester).

Here is a sample list of courses, in ascending order by course number:

107 YES YEAR

118 NO YEAR

202 YES YEAR

216 NO QUARTER

335 YES SEMESTER

494 NO SEMESTER

731 NO YEAR

915 YES QUARTER

Points are given for a letter grade in each course, with an A worth 4.5 points, B 3.5 points, C 2.5, D 1, and F worth zero points. A "plus" or "minus" causes an additional 0.25 points to be either added to or subtracted from the grade. Honors courses earn an additional 0.5 points, but only if the grade is C minus or better. Finally, some courses meet for the entire year (4 credits), others for a semester (2 credits), and others for a quarter (1 credit).

A student's GPA is determined by adding the points earned in each course after first multiplying each one by the number of credits in the course, and then dividing by the total number of credits earned. For example, the GPA of a student with a B- in course 202 and a C+ in course 494 would be calculated as follows:

The B- is worth 3.75 points (3.5 - 0.25 + 0.5) and the C+ is worth 2.75 points (2.5 + 0.25). Multiplying each by the credits for each course gives a GPA of (3.75\*4+2.75\*2)/(4+2) = 3.4167.

**Input -** An initial integer N, followed by N courses, each on one line as shown above, followed by another integer S, followed by S student lines. Each student line consists of a student's first name, followed by several grade/course pairs indicating which courses were taken and what grade was earned for that course.

**Output -** The name of each student followed by his (or her) GPA, with the name of the student with the highest GPA printed at the end.

**Sample Input**

8

107 YES YEAR

118 NO YEAR

202 YES YEAR

216 NO QUARTER

335 YES SEMESTER

494 NO SEMESTER

731 NO YEAR

915 YES QUARTER

4

ADAM C+ 494 B- 202

DON B 118 C+ 107 A 202 B+ 494 D- 915

CHUCK A 731 B- 494 F 118 C 107

BRETT C 494 A- 202 B 118

**Example Output to Screen**

ADAM 3.417

DON 3.683

CHUCK 2.607

BRETT 3.800

BRETT

**6. Grid Search**

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

Grid searching has applications in many ways, and this is a fundamental example of this technique. Consider the following grid of numbers:

1 2 3 4 5 4

2 5 4 3 2 1

**3** 5 **3 4 5 1**

**4** 4 3 4 5 2

**5** 3 4 3 4 3

**1** 4 5 2 3 4

Your job is to find a 4-digit number in the given grid, by looking in rows from left to right and from right to left, and in columns from top to bottom and from bottom to top. For example, the number **3451** appears in the grid twice: once at row 3, column 1, going down, and then again at row 3, column 3, going right. Numbers cannot "wrap around", so you will not find **5412** in the grid.

**Input -** An initial integer N, followed by an NXN grid of numbers with digits 1-5, and then another integer M followed by M 4-digit numbers, also containing only the digits 1-5.

**Output -** Output each number, followed by all the places it is found in the grid. You need to print the row and column of the first digit of the number where it is found, followed by the word "right", "down", "left", or "up" indicating the direction it goes, in the exact format and order as shown in the example below. It is guaranteed that all numbers will appear in the grid at least once.

**Sample Input**

6

123454

254321

353451

443452

534343

145234

5

5435

3454

4112

1543

3451

**Example Output to Screen**

5435 3,5,left

3454 1,3,right

4112 1,6,down

1543 3,6,left 6,1,up

3451 3,1,down 3,3,right

**7. HistoNum**

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

Histograms are just bar graphs, and this problem is a fairly simple example of one. Given a series of integers, count the number of occurrences of each digit from 0-9 and output a horizontal bar graph for each digit. If no digit occurs in the number series, do not output that bar.

**Input -** Several input integers, each on one line.

**Output -** Output the corresponding histogram, as shown below. Note carefully that in this sample data, a 4 never occurs, therefore the bar for 4 is not shown in the histogram.

**Sample Input**

35987

176253

20293805

2387

3981

**Example Output to Screen**

0|\*\*

1|\*\*

2|\*\*\*\*

3|\*\*\*\*\*

5|\*\*\*

6|\*

7|\*\*\*

8|\*\*\*\*

9|\*\*\*

**8. Hollow Floyd**

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

Floyd's triangle is a classic triangular number pattern in mathematics, but for this exercise we'll just use stars to make a physical version of the pattern. Your job is to replicate that pattern, with one catch...the triangle must be hollow.

**Input -** Several input integers, each on one line.

**Output -** Output the corresponding hollow Floyd triangle using stars, as shown below.

**Sample Input**

4

6

10

**Example Output to Screen**

\*

\*\*

\* \*

\*\*\*\*

\*

\*\*

\* \*

\* \*

\* \*

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

\*

\*\*

\* \*

\* \*

\* \*

\* \*

\* \*

\* \*

\* \*

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

**9. Labyrinth**

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

You have been captured by the Borg, and they have imprisoned you in their labyrinth, where they hunt you until they can find you. You need to escape, but the labyrinth shifts dimensions every second, shifting between 3 different dimensions. The first dimension is our current dimension, the second is the opposite, all walls are open spaces and all open spaces are walls, and the third dimension contains no walls. The Borg have sensors in the labyrinth, but you can fool them as long as you keep moving, you must always take a step or the Borg will find you, you also cannot retrace steps or you will be found. There is a space where you will be able to contact the Enterprise in the Labyrinth, and they will able to beam you out of the Labyrinth. Every step takes one second, and the dimensions shift every second, going from 1 to 2 to 3 and back to 1 again, and continuing. You start in the first dimension, your first step puts you in the opposite dimension.

**Input**

The first line will contain a single integer n that indicates the number of data sets that follow. Each data set begins with two integers, r and c, denoting the number of rows and columns in the matrix to follow. The next r lines contain the matrix itself made up of '#' - walls, '.' - open spaces, 'S' - your starting position, and 'E' - the point where you can contact the Enterprise.

**Output**

If it is possible to make it out of the labyrinth, output "ESCAPED IN S STEPS", where s is the number of steps required to escape the labyrinth. If it is impossible to escape, output "WELCOME TO THE BORG".

**Sample Input**3  
5 5  
##S##  
##.##  
#...#  
...#.  
E..##  
4 4  
S...  
####  
....  
###E  
1 4  
S..E

**Example Output to Screen**

ESCAPED IN 6 STEPS

ESCAPED IN 6 STEPS

WELCOME TO THE BORG

**10. Shuffle**

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

Given a sentence, output all unique words in alphabetical order.

**Input -** Several sentences, each on one line, all words in uppercase letters, with no other characters of any kind included.

**Output -** All unique words from each sentence in alphabetical order.

**Sample Input**

PRINT ONLY THE UNIQUE WORDS IN ALPHABETICAL ORDER

JINGLE BELLS JINGLE BELLS JINGLE ALL THE WAY

MARES EAT OATS AND DOES EAT OATS AND LITTLE LAMBS EAT IVY

SHE SELLS SEASHELLS ON THE SEASHORE

ONE LITTLE TWO LITTLE THREE LITTLE INDIANS

**Example Output to Screen**

ALPHABETICAL IN ONLY ORDER PRINT THE UNIQUE WORDS

ALL BELLS JINGLE THE WAY

AND DOES EAT IVY LAMBS LITTLE MARES OATS

ON SEASHELLS SEASHORE SELLS SHE THE

INDIANS LITTLE ONE THREE TWO

**11. Sine Up**

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

The Law of Sines for Trigonometry can be used in several different ways. It goes like this:

*Given the diameter D of the circumcircle of a triangle ABC, with sides a, b, and c opposite their corresponding angles, the ratios* ***a/(sin A), b/(sin B), and c/(sin C)*** *are all equal to the length of the diameter D.*

In this problem you are given the diameter D of the circumcircle of the triangle ABC, and the degree measures of the two angles A and B. You are to find the third angle, and the lengths of the three sides, and output all of the information as shown below.

*Note: A circumcircle of a triangle is a circle that contains all three vertices of the triangle.*

**Input -** Several lines of data, each line consisting of three integer values for the diameter of a circumcircle and two angle degree measures of the corresponding triangle.

**Output -** All of the information for the situation, formatted exactly as shown below.

**Sample Input**

100 40 60

50 45 45

70 90 60

**Example Output to Screen**

Circumcircle diameter = 100

Angles are 40, 60 and 80

Corresponding sides are 64, 87 and 98

Circumcircle diameter = 50

Angles are 45, 45 and 90

Corresponding sides are 35, 35 and 50

Circumcircle diameter = 70

Angles are 90, 60 and 30

Corresponding sides are 70, 61 and 35

**12. Supercomputer**

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

Gaelia is doing work on a supercomputer. The cells of the supercomputer are arranged in a 4x4 cube. Each cell of this cube is numbered from 0 - 63 depending on its location in the cube.



Where cell 0 is the front most, top most, left most cell, and cell 63 is the back most, bottom most, right most cell. The numbers are written ascending from left to right, top to bottom, front to back.

To avoid overheating, this cube can undergo 3 rotations labeled A, B, and C. Each of these are 180 degree rotations in 3D space. The A rotation is a 180 degree clockwise rotation about the axis running **front** to **back** through the cube. The B rotation is a 180 degree clockwise rotation about the axis running **top**to **bottom** through the cube. The C rotation is a 180 degree forward rotation about the axis running **left** to **right** through the cube. The numbering system referring to the cells in the cube are positional, meaning that after the cube has been rotated, the new cell that is in the front most, top most, left most square is now cell 0.

Unfortunately, some of the cells inside the cubes have been crashing due to overheating. When this happens, a failure in that cell is reported, but since the system undergoes so many rotations, it can be difficult to ascertain which cells are actually having failures. For example, in this line of data, **0 0 0 0 21 21 21 42 42 63 A**, the numbers **0 0 0 0** mean four failures in cell 0, followed by three failures in cell 21, 2 failures in cell 42, and 1 failure in 63. This gives a final front to back failure grid summation of:

4 0 0 0

0 3 0 0

0 0 2 0

0 0 0 1

However, since an A rotation (spin right on the front to back axis) is applied at the end, the final summation grid also rotates, and is now:

0 0 0 4

0 0 3 0

0 2 0 0

1 0 0 0

Given a list of cells with failures and the rotations the system undergoes, display the final summation grid of the cube from front to back. The final summation grid of the cube is the sum of all of the cell failures from front to back.

**Input** - Each data set will be on one line, and will consist of a list of integers between 0 - 63 and the characters A, B, or C delimited by a space. Each character indicates that rotation and each integer a failure in the corresponding cell.

**Output -** Display the final summation grid of the cube after all of failure reports and rotations in each data set.

**Sample Input**

0 0 0 0 21 21 21 42 42 63 A

21 42 21 B 22 41

0 A 0 B 0 C

**Example Output to Screen**

0 0 0 4

0 0 3 0

0 2 0 0

1 0 0 0

0 0 0 0

0 1 1 0

0 2 1 0

0 0 0 0

1 0 0 1

0 0 0 0

0 0 0 0

0 0 0 1