Computer Science Competition
State 2018
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	Ankur
Problem 2	Catalina
Problem 3	Dennis
Problem 4	Dinesh
Problem 5	Emma
Problem 6	Johnny
Problem 7	Konstantinos
Problem 8	Leonardo
Problem 9	Milo
Problem 10	Oscar
Problem 11	Richa
Problem 12	Romero

1. Ankur

Program Name: Ankur.java Input File: none

Ankur has just learned about transcendental numbers and wants to output these variations of the two most well-known mathematical constants in various formats, as shown below. Can you help him write a program to do that? *Here's a hint that might be helpful:* ***e, f,*** *and* ***g****.*

Input: None

Output: These six values formatted exactly as shown.

Expected output:

```
3141.59
3141.592654
3.141593e+03
271.828
271.828183
2.718282e+02
```

2. Catalina

Program Name: Catalina.java **Input File: catalina.dat**

Catalina loves acronyms and has discovered that several everyday words commonly used are acronyms. For example, in the military, **AWOL** means **Absent WithOut Leave**, and **SNAFU** is used in a slightly sarcastic and cynical way to say things aren't really going great, as usual, standing for **Situation Normal All Fouled Up**. She decides to write a program that will take the phrase and reduce it to its acronym, taking only the significant letters, which she uppercases in the actual input sentence, and then parses through the sentence and forms the acronym from those letters.

Input: Several phrases, each on one line, with key letters uppercased.

Output: The resulting one-word acronym made up of the uppercased letters in the phrase, eliminating spaces and all other letters and symbols.

Sample input:

```
Absent WithOut Leave
Self-Contained Underwater Breathing Apparatus
Completely Automated Public Turing test to tell Computers and Humans Apart
Situation Normal, All Fouled Up
Junior Reserve Officer Training Corps
Missing In Action
```

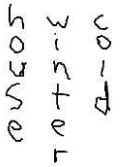
Sample output:

```
AWOL
SCUBA
CAPTCHA
SNAFU
JROTC
MIA
```

3. Dennis

Program Name: Dennis.java **Input File: dennis.dat**

Last winter on a very cold day Dennis was looking at the icicles hanging from the eve of his house and he began to wonder what words would look like if they were icicles. He got a piece of scratch paper and tried it out. He rewrote the words "**house"**, "**winter"** and "**cold"** as if they were icicles hanging from his house. It came out like this:



As you can see, Dennis' penmanship is quite poor. So, to make his word icicles look better, Dennis decides he needs a program to read his words and print the icicles.

Input: A file that contains several sets of words to be printed as if they were icicles. The first line will contain a value S that represents the number of sets of words followed by S sets of words. For each set there will be a line that contains a number N representing the number of words in the set followed by N words. There are no blank lines between sets of words.

Output: Each set of words printed side by side in columns spelled from the top down. There should be as many rows as there are characters in the longest word and no more. Any space in a column not taken up by a letter in a word must be a space character. Each set should be followed by a row of asterisks (*) exactly as wide as the number of columns used to display the set. There should not be any blank lines between the sets.

Sample input:

```
2
3
house
winter
cold
5
cat
turtle
aardvark
dog
fish
```

Sample output:

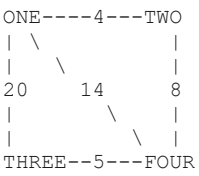
```
hwc
oio
unl
std
ee
 r
***
ctadf
auaoi
trrgs
 td h
 lv
 ea
  r
  k
*****
```

4. Dinesh

Program Name: Dinesh.java **Input File: dinesh.dat**

In the recent advanced algorithms unit, Dinesh has learned about the concept of shortest path, where there is an undirected graph of nodes with each edge a certain distance between a pair of nodes, and where the task is to find the shortest distance required to travel between the two nodes.

For example, here is a picture of a graph, and the data that represents it.



The first line of data represents the four nodes of the graph, followed by a value N, then followed by N edge connections between two nodes, stated as the two names followed by an integer representing the distance between that pair of nodes. Following the edge designations is another value M, followed by M pairs of nodes. Between each pair we want to know the shortest path.

The first one is easy since they are adjacent. The distance between ONE and TWO is 4. The next one is a bit more complex since there are three different ways to travel between ONE and THREE. The direct route has a distance of 20, but the route through node FOUR is 19, and even better the route that goes through TWO and FOUR has a distance of 17, clearly the shortest path, even though it seems to be the long way around.

Please help Dinesh as he struggles to code the solution to this problem.

------sample data set-----

```
ONE TWO THREE FOUR
5
ONE TWO 4
ONE THREE 20
THREE FOUR 5
ONE FOUR 14
TWO FOUR 8
5
ONE TWO
ONE THREE
ONE FOUR
THREE FOUR
THREE TWO
```

Input: First will be listed an integer G, indicating G graph data sets to follow. Each graph data set will consist of a row of node names, all on one line, all uppercased, with single space separation. On the next line will be an integer N, followed on the next N lines by a pair of nodes and an integer indicating the distance between that pair of nodes. Finally, an integer M will be followed by M pairs of nodes, between which the shortest path is to be determined.

Output: The pair of nodes in question followed by the shortest path between that pair of nodes, in the exact format shown in the sample output below.

(Continued on next page)

4. Dinesh (continued)

Sample input:

```
2
ONE TWO THREE FOUR
5
ONE TWO 4
ONE THREE 20
THREE FOUR 5
ONE FOUR 14
TWO FOUR 8
5
ONE TWO
ONE THREE
ONE FOUR
THREE FOUR
THREE TWO
ALPHA BETA GAMMA DELTA EPSILON
6
ALPHA BETA 1
ALPHA GAMMA 7
ALPHA EPSILON 3
BETA EPSILON 6
GAMMA EPSILON 2
EPSILON DELTA 3
3
ALPHA DELTA
BETA GAMMA
EPSILON BETA
```

Sample output:

```
ONE to TWO:4
ONE to THREE:17
ONE to FOUR:12
THREE to FOUR:5
THREE to TWO:13
ALPHA to DELTA:6
BETA to GAMMA:6
EPSILON to BETA:4
```

5. Emma

Program Name: Emma.java **Input File: emma.dat**

Emma likes box-like patterns that can be produced with different computer algorithms and has designed one she thinks is pretty interesting. She needs your help though because her skills aren't quite ready to tackle the pattern she created. She decided to base it on a single integer value, like 3, and then create a box three times that size, a 9x9 box, which contained an X pattern of stars in the middle and four 3x3 boxes of stars in each corner. For lack of a better term, she decides to call it the **X-Box star pattern**. It looks like this:

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

For an input value 4, the pattern would be similar, but slightly different, like this:

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

Input: Several integers N, such that 0<N<=10, all on one line with single space separation.

Output: An X-Box pattern as shown above, characterized by a grid three times the size of N, in other words, 3Nx3N, with an NxN box of stars in each corner, connected by an X pattern in the middle. Each complete output will be followed by the line "**==========**".

Sample input:

```
3 4 5
```

(continued on next page)

5. Emma (continued)

Sample output:

```
***    ***
***    ***
***    ***
    * *
     *
    * *
***    ***
***    ***
***    ***
==========
****     ****
****     ****
****     ****
****     ****
     *   *
      **
      **
     *   *
****     ****
****     ****
****     ****
****     ****
==========
*****      *****
*****      *****
*****      *****
*****      *****
*****      *****
      *    *
       * *
        *
       * *
      *    *
*****      *****
*****      *****
*****      *****
*****      *****
*****      *****
==========
```

6. Johnny

Program Name: Johnny.java **Input File: johnny.dat**

A huge Star Trek fan for as long as he can remember, Johnny is fascinated with the Universal Translator used to help with communications between various species in the series. He decides to make up a simple lexicon of words that might be used by English speakers and Vulcans, using a different symbol to represent each word. For example, the word EARTH would be the symbol ", and VULCAN or VULCANS would be #. I or WE would be represented by the ! symbol, IS or AM or ARE would all be represented by =.

Here is a subset of a much larger collection of words that might be used to symbolize these different ideas.

```
! I/WE
# VULCAN/VULCANS
+ BIG
- SMALL
" EARTH
\ AND
= IS/AM/ARE
] HAS/HAVE
: MOON/MOONS
```

Using these words, a phrase like **I AM VULCAN** would be simply translated as **!=#**.
VULCAN IS SMALL AND EARTH IS BIG would be symbolized as **#=-\"=+**.

He needs your help translating simple English phrases into the symbols they represent in a simple "word for word" manner.

Input: Several one-character non alphanumeric symbols, each followed by one or more words that symbol represents. Any multiple words for each symbol are separated by the / character. Following the list of symbol/word combinations are a listing of sentences, in all uppercase, to be translated into symbols. **Note:** the judges data may have more symbol/word pairs than shown in the sample data.

Output: The symbolic representation of each sentence.

Sample input:

```
! I/WE
# VULCAN/VULCANS
+ BIG
- SMALL
" EARTH
\ AND
= IS/AM/ARE
] HAS/HAVE
: MOON/MOONS
I AM VULCAN
VULCAN IS SMALL AND EARTH IS BIG
VULCAN AND EARTH HAVE MOONS
```

Sample output:

```
!=#
#=-\"=+
#\"]:
```

7. Konstantinos

Program Name: Konstantinos.java **Input File: konstantinos.dat**

In geometry class Konstantinos has been learning about circles being tangent (or not) in various situations. He knows that two circles are externally tangent when they look like this, intersecting at exactly one point, with no other points in common:

For two circles to be externally tangent, mathematically he knows that the sum of the radii for the two circles is equal to the distance between the two centers. He thinks he can figure out the math for other situations shown below but needs your help to write a program to do that. He remembers the distance formula from algebra class, which is:
$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

NON-INTERSECTING INTERSECTING NESTED INTERNALLY TANGENT

Input: Several data sets, each on one line consisting of six integer values x1, y1, r1, x2, y2, r2, representing the (x,y) circle center coordinates and corresponding radii for two circles. All values will have single space separation.

Output: For each data set, output the way the two circles intersect (or not), based on the five situations depicted by the diagrams shown above.

Sample input:

```
0 0 2 3 0 2
0 0 2 0 3 1
0 0 4 0 2 2
```

Sample output:

```
INTERSECTING
EXTERNALLY TANGENT
INTERNALLY TANGENT
```

8. Leonardo

Program Name: Leonardo.java **Input File: leonardo.dat**

According to the website Leonardo has found at URL (https://thoughtcatalog.com/nico-lang/2013/08/55-celebrities-whose-real-names-will-surprise-you/), the real names of several famous personalities are very interesting indeed. He decides to play around with these names, converting both stage and real names to initials, but with a twist. For the given name, he decided to make up initials from the last letter of each name part.

For example, **`Marilyn Monroe = Norma Jean Mortensen`** would convert to **`MM = ANN`**. Help him out by writing a program to do this simple but fun task.

Input: Several names of famous people, each on one line, first their personality name, followed by an equal sign, and then their real original given name. Either name will have one or more parts, but no extra designation like Jr. or II or anything else like that, just names.

Output: The initials for each name pair, first letters for the personality name, last letters for the real name, in the format demonstrated above and shown below.

Sample input:

```
Marilyn Monroe = Norma Jean Mortensen
Ben Kingsley = Krishna Pandit Bhanji
Katy Perry = Katy Hudson
Abigail Van Buren = Pauline Ester Friedman
Elvira = Cassandra Peterson
```

Sample output:

```
MM = ANN
BK = ATI
KP = YN
AVB = ERN
E = AN
```

9. Milo

Program Name: Milo.java **Input File: milo.dat**

Since neither the Cowboys or the Texans were in the Super Bowl this past February, Milo quickly became bored as he watched the game. Milo noticed that the logo for the game was Super Bowl LII. Roman numerals? Who uses those? Milo's grasp of the Roman numeral system is a little rusty, so he looked them up on Wikipedia. Here is some of what he found:

> The original pattern for Roman numerals used the symbols I, V, and X (1, 5, and 10) as simple tally marks. Each marker for 1 (I) added a unit value up to 5 (V), and was then added to (V) to make the numbers from 6 to 9: I, II, III, IIII, V, VI, VII, VIII, VIIII, X.
>
> The numerals for 4 (IIII) and 9 (VIIII) proved problematic and are generally replaced with IV (one less than 5) and IX (one less than 10). This feature of Roman numerals is called subtractive notation.
>
> The numbers from 1 to 10 are expressed in Roman numerals as follows:
>
> I, II, III, IV, V, VI, VII, VIII, IX, X.
>
> The system being basically decimal, tens and hundreds follow the same pattern:
>
> Thus 10 to 100 (counting in tens, with X taking the place of I, L taking the place of V and C taking the place of X):
>
> X, XX, XXX, XL, L, LX, LXX, LXXX, XC, C.
>
> Note that 40 (XL) and 90 (XC) follow the same subtractive pattern as 4 and 9.
>
> Similarly, 100 to 1000 (counting in hundreds):
>
> C, CC, CCC, CD, D, DC, DCC, DCCC, CM, M.
>
> Many numbers include hundreds, units and tens. The Roman numeral system being basically decimal, each "place" is added separately, in descending sequence from left to right, as with "arabic" numbers. For example, the number 39 is XXXIX, (three tens and a ten less one), 246 is CCXLVI (two hundreds, a fifty less ten, a five and a one). As each place has its own notation there is no need for place keeping zeros, so "missing places" can be simply omitted: thus 207, for instance, is written CCVII (two hundreds, a five and two ones) and 1066 becomes MLXVI (a thousand, a fifty and a ten, a five and a one)
>
> *Roman numerals. (2018, January 21). In Wikipedia, The Free Encyclopedia. Retrieved 21:06, February 20, 2018, from https://en.wikipedia.org/w/index.php?title=Roman_numerals&oldid=821541836*

In addition to the information from Wikipedia, Milo finds that to represent a number greater than 3999 requires using an over bar to indicate that a number must be multiplied by 1000. $\bar{V} = 5000$.

Milo needed to kill a little time until the half time show so he decided to whip up a little program to convert Roman numbers to Arabic numbers.

Input: A file that contains an unknown number of Roman numbers each listed on a separate line. All of the numbers will be greater than or equal to I and less than or equal to MMMCMXCIX. All of the Roman numbers in the file will be valid numbers as described above.

Output: The Arabic equivalent of each Roman numeral, each printed on a separate line.

Sample input:	**Sample output:**
IV	4
V	5
VII	7
XIX	19
XX	20
CV	105
MDCLXXVIII	1678

10. Oscar

Program Name: Oscar.java **Input File: oscar.dat**

Oscar works down at the sign company. His job is to make 24-inch wide signs with the phrases and words that are presented to him. Each sign has a 2-inch margin where nothing is printed except a border line. Each sign can have up to three lines of text, each letter or character 2 inches wide. The makeup of each sign must follow these rules:

1. A word cannot be divided.
2. Each line of words should be centered on the line.
3. If the spaces before and after a word or words on a line are unequal, place the larger number of spaces after the text.
4. If the sign has just one word, it should be placed on the middle line.
5. If the sign has two words, they should be placed in the first two lines.
6. If there are three or more words, all three lines must be used even if all the words will fit on fewer lines.
7. After at least one word has been placed on each line, lines should be filled with as many words as possible, starting with the top line.
8. If all the words will not fit on three lines the sign cannot be made.
9. If any of the words to be placed on the sign are too long to fit on a line, the sign cannot be made.
10. None of the text being considered will contain a period but may contain other symbols.

For the words, "Don't Mess With Texas", the first two words, "Don't Mess" go on the first line, followed by "With" and "Texas" on the next two lines. It would be possible to have the words, "Mess With" on the second line, but this violates Rule 7 above.

Input: A file that contains an unknown number of lines each of which contains the text that should be placed on each sign.

Output: Each sign printed following the rules described above. Blank spaces between and around the text should be indicated by printing a period. The two-inch margin around the edge of the sign will be shown by "*" symbols. If the number of spaces on either side of the line of text is uneven, the larger number of spaces should be placed after the text. If a sign cannot be made, print **`SIGN CANNOT BE MADE`**. Each of the signs should be separated by one blank line.

Sample input:

```
Don't Mess With Texas
Get Your Crackerjacks Here
UIL Sign Company
Turn Right
```

Sample output:

```
************
*Don't.Mess*
*...With...*
*..Texas...*
************

SIGN CANNOT BE MADE

************
*...UIL....*
*...Sign...*
*.Company..*
************

************
*...Turn...*
*..Right...*
*..........*
************
```

11. Richa

Program Name: Richa.java Input File: richa.dat

The concept of GPA (grade point value) is new to Richa, a high school freshman this year, and she needs some help figuring out hers. The catalog of courses to be considered contains ten courses numbered from 140 to 149, and are designated either Honors or not, with three different possible credit lengths, a full year (4 credits), semester only (2 credits), or a quarter course (1 credit). To practice the calculation process on a simple level, she only wants to consider two courses at a time.

Grade point values are awarded as follows: A=4.5, B=3.5, C=2.5, D=1, and F=0. A '+' or a '-' can be added to each grade (except for F). This either adds or subtracts a quarter point from the value. Honors courses with a grade of C- or better earn an additional half point.

For example, a grade of A- for an Honors course would earn a grade point value of 4.75, subtracting a quarter point from 4.5, and then adding a half point. A D+ for a regular course would earn 1.25 points.

The total number of points earned for a course is the product of the grade point value earned and the number of credits earned. For example, the total points for an A- in a full year Honors course would be calculated as 4.75 times 4, which is a total of 19 grade points. A D+ in a semester course would earn a total of 1.25 times 2, or 2.5. The total GPA for these two courses would be the sum of the two grade point values, divided by the total number of credits, which is the calculation (19 + 2.5) / (4 + 2) = 21.5 / 6 = 3.583.

A C+ for course 144 and a B- for course 142 would result in a GPA of 3.417.
An A for 141 and a B for 148 earns a GPA of 4.0.

Input: An initial integer value N representing N course designations to follow, each consisting of a course number and a two character code, made up of one of the characters 'Y' of 'N' designating Honors or not, and then one of the three characters 'Y' for a full year course, 'S' for semester, and 'Q' for quarter. Following these N course data sets, several four-part data sets follow, each consisting of a letter grade A, B, C, D or F, possibly followed by a + or -, the course number, another letter grade, and then the second course. All data elements on a line are separated by single spaces.

Output: The resulting GPA of each pair of courses, formatted and rounded to three places of precision.

Sample input:

```
10
140 NS
141 NY
142 YY
143 YS
144 NS
145 YQ
146 NQ
147 YY
148 NY
149 YY
A- 149 D+ 143
C+ 144 B- 142
A 141 B 148
```

Sample output:

```
3.583
3.417
4.000
```

12. Romero

Program Name: Romero.java **Input File: romero.dat**

Mrs. Romero is the principal at Firethorne High School. Go Hotspurs! One of the many and varied duties she must perform is to sell and take tickets at the home football game on every other Friday night. Oddly enough, FHS changes the price for tickets for each game. For example, they charged $5.00 for adults and $3.00 for students and children not yet in school to attend the home opener. For the next game they charged $6.00 for adults but only $2.50 for students and children. Students always pay less than adults. At the end of each game Mrs. Romero knows the total attendance and the gross receipts for that game. She does not keep track of how many adults attended or how many students and children attended. At the end of the season the Firethorne ISD board of trustees decided that they would like to see the effect changing ticket prices had on attendance of each group of ticket buyers. They have requested a report showing the count of how many adults, students and children attended each game.

Luckily Mrs. Romero is a former algebra teacher and knows how to use a system of equations to calculate the attendance of each group. Here is how she did it.

At the first game total attendance was 3250. Gross receipts totaled $14,250. This provides enough information to come up with these two equations if she lets x be adults and y be students and children:

x+y=3250
5x+3y=14250

Now she multiplies the first equation by 5 to get:

5x+5y=16250
5x+3y=14250

Next she adds the opposite of the second equation to the first and gets:

2y=2000

Solves for y to get:

y=1000

Now she substitutes 1000 for y in the first equation:

x+1000=3250

Solves that for x and finds that:

x=2250

So, for the first game she can report that 2250 adults and 1000 students and children attended the game.

Mrs. Romero has a whole season's worth of data stored in a file and doesn't want to process all that data by hand. That is where you come in. Write a program that will process all her data and print a report of the date, total attendance, gross receipts, ticket prices, adult attendance and student/child attendance for each of the home football games that Firethorne High played this past season.

Input:

An initial value G representing the number of home games played this past season followed by G lines of data for each game. Each line of game data will contain a date shown as **mm/dd/yyyy** (all games were played in September, October, November or December), total attendance (stadium capacity is 4000), gross receipts for that game, ticket price for adults and ticket price for students and children. Neither ticket price was ever more than $9.50 and are never the same. All the data items on each line will have single space separation.

Output:

A report containing the data compiled for each home game in the exact format, column size and left alignment described here and shown in the examples. The report should begin with column headings for each data item: Date, Attendance, Gross, ATP, STP, Adults, Stu/Child. Each row should display the date as a long date (Month Day, Year), the gross receipts, adult ticket price and student ticket price using a dollar sign, comma separator if necessary, and two decimal places ($##,###.##) and display total, adult and students/children attendance as whole numbers. All the data should be left aligned within its column and there should be at least one space between each column. A final **pipe " | "** character is to end each line of the report, as shown.

12. Romero (continued)

No data or calculated values will exceed the column size requirements, and there will always be a minimum of one space separating the data elements in each column. For example, the Date column will contain dates no longer than 19 spaces, Attendance always starts in column 20, Gross in column 31 with values <$100K, etc.

Sample input:

```
2
9/8/2017 3250 14250 5.00 3.00
9/22/2017 2980 15027.50 6.00 2.50
```

Sample output:

```
Date               Attendance Gross      ATP   STP   Adults Stu/Child|
September 8, 2017  3250       $14,250.00 $5.00 $3.00 2250   1000     |
September 22, 2017 2980       $15,027.50 $6.00 $2.50 2165   815      |
```