Computer Science
Competition 2017 Invitational B
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	Akio
Problem 2	Danna
Problem 3	Evelyn
Problem 4	Fengge
Problem 5	Isidora
Problem 6	Juan
Problem 7	Kostas
Problem 8	Nastya
Problem 9	Olga
Problem 10	Pedro
Problem 11	Roy
Problem 12	Samantha

1. Akio

Program Name: Akio.java Input File: akio.dat

Akio loses things frequently. At the end of the night of partying out on the town, Akio realizes that he is missing a lot of his things. Luckily, people have called him to let him know where all of his things are. But, since it is late and all of the places are closing, he needs the shortest path from the door of each place to his lost item in the room so he can get in and out quickly.

All the rooms are represented with walls/tables as **"#"** indicating places where Akio cannot walk. All places where Akio can walk are indicated with a **"."** The item in the room that Akio is seeking is indicated with an **"o"**. Akio always enters the room where the **"."** is on the edge of the room, indicating the door. In each room he has only left one item, but some of the rooms can have multiple doors, so you must always choose the closest door for the path for Akio.

For example, in the room shown below, the shortest path to get to the item is from the first door at the top of the diagram. The solution on the right shows this path with the bolded numbers marking the steps.

```
####.##.##
###.....##
###...####
###.######
#○..######
###.######
###......#
###.######
###.....##
#######.##
```

```
####1##.##
###32...##
###4..####
###5######
#876######
###.######
###......#
###.######
###.....##
#######.##
```

To help Akio, you don't need to show him the steps, but just tell him the length of the shortest path.

Input: The first integer will be the number of data sets to follow. Each data set is made up of a 10x10 grid of characters representing one of the rooms where Akio has left one item, as described above. Each 10x10 grid will be followed by a **"-"** for separation.

Output: For each room map, output the length of the shortest path from a door to his lost item.

Sample Input:

```
3
####.##.##
###.....##
###...####
###.######
#○..######
###.######
###......#
###.######
###.....##
#######.##
-
##########
###....○##
###...####
###.######
#...#....#
###.######
###.#....#
###.######
###.....##
#######.##
-
#######.##
#.....###
#.########
#.........#
########.#
#.........#
#.########
#.########
#......○##
##########
-
```

Sample Output:

```
8
17
35
```

2. Danna

Program Name: Danna.java **Input File: danna.dat**

At a recent visit to her doctor, Danna was informed of her BMI (Body Mass Index), a calculation based on her height and weight (BMI = kg/m^2), and was told she was classified as marginally overweight (height 5'10", weight 175 lbs), which surprised her a bit. She did not consider herself overweight, and decided to do more research. She discovered that BMI was a weight-to-height ratio, calculated by dividing one's weight in kilograms by the square of one's height in meters and is used as an indicator of obesity and underweight. She also discovered that the scale was different for women than men.

According to an online source she found during her research (http://halls.md/bmi-chart-women/), *BMI is used as a screening tool to identify possible weight problems for adults. However, body mass index is not a diagnostic tool. To determine if excess weight is a health risk, a healthcare provider would need to perform further assessments.* Danna also found that *calculating BMI is one of the best methods for population assessment of overweight and obesity*, but was not really the most appropriate method for individual diagnosis, which relieved her somewhat.

She decided to do some research at her school using the parameters she discovered for young women around her age, which stipulated that a *BMI under 18.5 is considered very underweight and possibly malnourished, 18.5-24.9 is a healthy weight range, 25.0 to 29.9 is considered overweight and over 30 is considered obese*. She asked female students she knew in her classes in school for their anonymous participation, which many did by providing their height and weight on notecards, but without providing their name.

She then processed the data, classifying each person's BMI individually, and then doing an overall report, showing how many fell into each category, based on the parameters listed above, and solely based on height in inches and weight in pounds. She used the conversion values of 1 inch = 0.0254 meters and 1 pound = 0.453592 kilograms to convert the notecard data to metric units. For example, her own data was 70 inches and 175 pounds, which converted to 1.778 meters and 79.3786 kilograms, which then produced a BMI of 25.11 ($79.3786/1.778^2$), putting her on the borderline just into the "overweight" category. Another person's data reported in the "underweight" category with a height and weight of 68 inches and 105 pounds, for a BMI of 15.97.

She did this same calculation for all of the data set collected, reporting a list that contained the original data, the BMI index for each data set, and the corresponding classification. Finally, she took all of the results and calculated the percentage of how many of the students fell into each of the four categories.

Input: Several pairs of integers H and W, representing height in inches and weight in pounds, each pair on line.

Output: The original data for each set, H and W, the BMI calculation, expressed to a precision of two decimal places, and the corresponding classification, each output value separated by exactly one space. After the last data set report line, the total number of students surveyed is reported, followed by four lines, each showing the percentage for each classification, precise to one decimal place, in the same order as shown in the sample output.

Sample input:

```
70 175
68 105
63 110
60 130
65 130
68 115
66 120
61 170
72 125
64 135
70 140
```

Sample output:

```
70 175 25.11 overweight
68 105 15.97 underweight
63 110 19.49 normal
60 130 25.39 overweight
65 130 21.63 normal
68 115 17.49 underweight
66 120 19.37 normal
61 170 32.12 obese
72 125 16.95 underweight
64 135 23.17 normal
70 140 20.09 normal
11 total students surveyed
27.27% underweight
45.45% normal
18.18% overweight
9.09% obese
```

3. Evelyn

Program Name: Evelyn.java **Input File: evelyn.dat**

Evelyn has invented a new base 10 system. This system behaves very similarly to the existing base 10 system we use today, but some of the symbols look a little different.

```
& add & is &
& add ' is '
& add ( is (


' add ' is (
' add ( is )
' add ) is *
' add * is +
' add + is ,
' add , is -
' add - is .
' add . is /


' add / is '&


/ add / is '.
* add + is ''
```

The term **add** is defined as the classical form of addition and **is** indicates equality. Using this new base representation, given a depth n, the Fibonnaci sequence is defined as:

f(0) is '
f(1) is '
f(n) is f(n-1) add f(n-2)

Evelyn has asked you to display the nth encrypted Fibonnaci value, preceded by all other values in the sequence, separated by single spaces.

Input: One integer, n, the depth of the Fibonnaci value to be displayed.

Output: The Fibonnaci value of depth n in Evelyn's base representation, preceded by all previous values in the sequence.

Sample Input:

```
1
2
10
```

Sample Output:

```
' '
' ' (
' ' ( ) + . ') (' )* ++ ./
```

4. Fengge

Program Name: Fengge.java **Input File: fengge.dat**

In computer science class, Fengge has been learning about creating output box patterns. He decides to write a program that creates a box pattern that is hollow. For example, a 5 X 6 box made of stars (**"*"**) would look like this:

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

A 9X7 box made of the letter **"a"** would look like this:

```
aaaaaaa
aaaaaaa
aa   aa
aa   aa
aa   aa
aa   aa
aa   aa
aaaaaaa
aaaaaaa
```

Input: Several data sets, each set on one line, consisting of two integers R and C, each greater than 4 and less than 20 in value, and a single character A, with single space separation.

Output: A hollow box with R rows and C columns, with a double edge border filled with the input character A. One blank line follows the box.

Sample input:

```
5 6 *
9 7 a
```

Sample output:

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

aaaaaaa
aaaaaaa
aa   aa
aa   aa
aa   aa
aa   aa
aa   aa
aaaaaaa
aaaaaaa
```

5. Isidora

Program Name: Isidora.java **Input File: isidora.dat**

Isidora has just learned about bit string flicking, specifically the **left shift, right shift, left circle,** and **right circle** operations. She needs help writing a program that will correctly output the resulting binary string for one of these four operations.

She wants to use RS for Right Shift, LS for Left Shift, RC for Right Circle, and LC for Left Circle as the first part of the command. This will be followed by a dash "-", a value, then a space, and then the base ten integer value to be shifted or circulated.

For example, the command RS-4 45 will take the value 45, convert it to the binary string 101101, and then perform a Right Shift 4, which means the rightmost 4 bits are eliminated, leaving 10 as the result.

The command LS-2 13 will convert the value 13 to its binary equivalent of 1101, and perform a Left Shift 2 operation, which essentially adds two zeroes to the back of the binary string, resulting in 110100.

The circle commands result in a same size binary string as the original, with a Right Circle taking the rightmost bits and circling them to the other side, and likewise for the Left Circle, taking the leftmost digits and circling them to the other side.

For example, the command RC-3 81 will take the binary value for 81, or 1010001 in binary, take the three right most digits, 001, and circle them to the front of the string resulting in 0011010.

Input: Several commands as described above, each on one line. It is guaranteed that the command will work within the length of the string, with all results greater than or equal to zero.

Output: The resulting bit string for the command.

Sample input:

```
RS-4 45
LS-2 13
RC-3 81
```

Sample output:

```
10
110100
0011010
```

6. Juan

Program Name: Juan.java Input File: None

Juan has just joined the UIL team at his school and knows a little bit about programming, but not too much. He decides to submit to the judges a simple output programming that introduces his team, hoping they will give him some points for it. He only includes the first names of his team, and the initials of his high school, in order to maintain some anonymity.

Input: None

Output: The first name of each of your team's members, either three or four members, followed by the initials of your school, each on one line.

Sample output:

```
Tom
Janie
Kendra
Joe
AHS
```

7. Kostas

Program Name: Kostas.java **Input File: kostas.dat**

Kostas has just discovered the world of Leet Speak and found some references to it on the internet, including this one from www.urbandictionary.com:

> Originating in the early 1980's, leet speak was first used by hackers as a way to prevent their websites/newsgroups from being found by simple keyword searches. Leet speak grew and became popular in online games such as Doom in the early 1990's as a way of suggesting that you were a hacker (h4x0r), and therefore to be feared. Leet, or 1337, is a short form of "elite," commonly used by video gamers to suggest that they are skilled.

And this one from en.wikipedia.org:

> Leet (or "1337"), also known as eleet or leetspeak, is an alternative alphabet for many languages that is used primarily on the Internet. It uses some characters to replace others in ways that play on the similarity of their glyphs via reflection or other resemblance. For example, leet spellings of the word *leet* include *1337* and *l33t*; *eleet* may be spelled *31337* or *3l33t*. The term leet is derived from the word *elite*. The leet alphabet is a specialized form of symbolic writing. Leet may also be considered a substitution cipher, although many dialects or linguistic varieties exist in different online communities. The term leet is also used as an adjective to describe formidable prowess or accomplishment, especially in the fields of online gaming and in its original usage—computer hacking. ()

He decides to create one very similar using his own rules for the simple substitute cypher, based on the following table, including the digits 4, 8, 3, 6, and 1 for the letters A, B, E, G, and L, and the pipe symbol **"|"** included in the substitutions for I, K, and P:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
4	8	[	)	3	|=	6	#	|	]	|<	1	^^	^	0	|o	9	2	5	7	(_)	\/	vv	><	/	%

He also decides on the following additional rules of priority, to give it a computer science flavor.

1. Numbers and spaces are not replaced.
2. Replace ER at the end of a word with XOR
3. Replace OR at the end of a word with ZOR
4. Replace AND, ANNED, and ANT at the end of a word with &
5. Replace ED at the end of a word with *D
6. Replace all remaining letters using the chart above.

Input: A list of words or phrases, each on one line.

Output: The "leet speak" version ("**1337 5|o34|<**") of the word, according to the rules stipulated above.

Sample input:

```
UIL
R2D2
COMPUTER SCIENCE
STATE CHAMPION 2017
```

Sample output:

```
(_)|1
22)2
[0^^|o(_)7XOR 5[|3^[3
57473 [#4^^|o|0^ 2017
```

8. Nastya

Program Name: Nastya.java **Input File: nastya.dat**

After seeing the simple box pattern Fengge did in class, Nastya, not to be outdone, decides to do a one-up on him and create something a bit more elaborate, according to these rules.

Her pattern will be rectangular, simulating a wall of a house, with three values input, A, B, and C. The value A represents the thickness of the wall, B the number of columns between the left and right walls,and C the height of the wall. The walls will be indicated by the letters I and X. For walls with a thickness of 2 or less, Is will be used, otherwise Xs will be used for interior columns between the two Is. Cross letters begin inside each wall, starting at the top of the left wall, with Rs that descend diagonally until they hit the opposite wall, or the floor. Likewise, Ls will start at the top of the right side wall and descend diagonally until they hit the opposite wall of the floor. Anytime an R and L occupy the same cell, and S will be used instead. If an R or L reaches a wall, or the bottom row, then that becomes the last row for the cross letters.

For the input values A = 2, B = 4, and C = 5, the wall pattern is as follows:

```
IIR  LII
II IL II
IIL  RLL
II    II
```

The data set 3 10 8 results in:

```
IXIR        LIXI
IXI R      L IXI
IXI  R    L  IXI
IXI   R  L   IXI
IXI    RL    IXI
IXI   L  R   IXI
IXI  L    R  IXI
```

The input values 1 5 6 produce:

```
IR   LI
I R L I
I  S  I
I L R I
IL   RI
I     I
```

Input: Several data sets, each set on one line, as described above.

Output: A wall cross-section pattern, also as described above. Each pattern will be followed by a blank line.

Sample input:

```
2 4 5
3 10 8
1 5 6
```

Sample output: (see above)

9. Olga

Program Name: Olga.java Input File: olga.dat

To make input and output practice a bit more interesting, Olga has decided to read in the names of her friends and output them with special characters around each name, depending on the characteristics of the name.

For any name that has an odd number of characters, she will but the name in single quotes, putting even number length names in double quotes.

In addition to that she will put single slashes outside of names that begin with letters in the first half of the alphabet, and double slashes outside of names that start with letters in the last half of the alphabet.

Finally, she will put single dashes around names with less than 6 characters, and the equals sign around longer names.

For example, her name will appear in the output as **-\\"Olga"//-**, double quotes for an even number of characters, double slashes since her name starts with 'O', which is in the last half of the alphabet, and single dashes since her name is less than 6 characters long.

The name of her friend Isidora will appear as **=\'Isidora'/=**, single quotes for odd length, single slashes for first half of the alphabet, and equals signs for a longer name.

Input: A list of names, each on one line. There will be no special characters or spaces inside of any name.

Output: Each name surrounded by special characters, as described above.

Sample input:

```
Olga
Isidora
Fengge
```

Sample output:

```
-\\"Olga"//-
=\'Isidora'/=
=\"Fengge"/=
```

10. Pedro

Program Name: Pedro.java Input File: pedro.dat

Pedro really likes cats, and is at the shelter to adopt some. All the cats cost the same amount, and he is at the shelter, so he knows how many cats he wants and how many the shelter has. However, he wants your help in showing him which combinations of cats he can adopt.

Given the number of cats **i** Pedro can adopt, and the list of cats that are at the shelter, show Pedro all the possible cat combinations utilizing **i** cats that he can adopt (in alphabetical order).

Pedro will never try to adopt more cats than the shelter currently has.

Input: An integer representing the number of cats Pedro wants to adopt, followed by series of strings representing the cats' names separated by spaces. All data sets will be separated by a new line.

Output: All the possible cats, in alphabetical order, represented by a list of cats' names separated by space. Each output set should be followed by a blank line. All the cats in the given list will be unique.

Example Input:

```
3 orange-kitty blue-kitty white-kitty black-kitty
10 scarecrow clarissa harley simba tiger maya mojo fluffy reena lucy
2 paintjob ethel-cat spot reinma venmo carno
```

Example Output:

```
black-kitty blue-kitty orange-kitty
black-kitty blue-kitty white-kitty
black-kitty orange-kitty white-kitty
blue-kitty orange-kitty white-kitty

clarissa fluffy harley lucy maya mojo reena scarecrow simba tiger

carno ethel-cat
carno paintjob
carno reinma
carno spot
carno venmo
ethel-cat paintjob
ethel-cat reinma
ethel-cat spot
ethel-cat venmo
paintjob reinma
paintjob spot
paintjob venmo
reinma spot
reinma venmo
spot venmo
```

11. Roy

Program Name: Roy.java **Input File: roy.dat**

Roy is playing a new card game. This game is a card game where the goal is to get as many duplicates in your hand as possible. There are 104, two normal decks of cards in this game labelled A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2. This game starts by dealing each player 5 cards. Then each player is dealt a card and the player has to choose a card to remove from their hand such that they always have only 5 cards in their hand at a time. Roy has decided to apply a simple algorithm to his card keeping choices. When Roy gets a card he removes the card in his hand that has the least number of occurrences. If there are two different valued cards that have the same number of occurrences then the one of lower value is eliminated. Value for the cards is 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < J < Q < K < A.

Example: For input A A Q K 10 A K K, the hand starts with 5 cards A A Q K 10, the next card that Roy receives is A. Q, K, and 10 all have equal number of occurrences so 10 is removed. The next card is received K. Q has the least number of occurrences and is removed. The next card that is received is K. Since A and K have the same number of occurrences, K is removed because it is less in value. The final hand is K K A A A.

Input: A string of characters representing the cards that Roy has received. The first 5 characters are the cards that Roy is dealt, all the following cards are received applying his algorithm.

Assumptions: The input string will have at least 5 characters.

Output: The final hand, arranged in sequential order from lowest to highest rank.

Example Input:

```
A A Q K 10 A K K
K Q J A A A 10 J
K K Q K J J 10 A 2 2
K K K K Q Q Q Q A
```

Example Output:

```
K K A A A
Q K A A A
J J K K K
K K K K A
```

12. Samantha

Program Name: Samantha.java **Input File: samantha.dat**

Samantha's boss has given her a bunch of data to process, but she is not clear on what it represents, nor is she told. However, the criteria for processing the data is given with some formulas and expected outputs, so she happily starts to code the solution.

The format of each data set is a series of integers, each on one line. The first integer is an index value X to be used in a formula, followed by an integer N, followed by N more integers representing the data, sometimes in ascending order, but not always.

The three requested outputs are:

- A mean of the list of values
- The median value of the list
- A value (whose meaning is unclear) based on a complex formula

All three values are to be output as truncated integers.

The formula she is to use for calculating the third output is based on three variables and a few constants.
The three variables are:

- The index value X at the front of each data set.
- The sum of all values below the mean, called the low sum, or LS, for short.
- The sum of all values equal to or above the mean, called the high sum, or HS, for short.

The secret formula for the third output is: (.06 + X * .07) * LS) +(4 * X * .0789 * HS)

Input: Several integer values, all on one line with single space separation, including X and N followed by N values.

Output: Three integer values as described above.

Sample input:

```
4 5 10 75 1000 25000 75000
5 6 200 300 1000 50000 200000 750000
7 7 1 5 10 400000 500000 750000 1000000
```

Sample output:

```
20217 1000 126608
166916 25500 1520214
378573 400000 5854388
```