Computer Science Competition
Invitational B 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	Baldo
Problem 2	Barb
Problem 3	Constanza
Problem 4	Dilbert
Problem 5	Emily
Problem 6	Fa
Problem 7	Judith
Problem 8	Luann
Problem 9	Lucy
Problem 10	Micaela
Problem 11	Nikita
Problem 12	Peter

1. Baldo

Program Name: Baldo.java Input File: none

You and your classmate Baldo are enrolled in Computer Science 1 at Imaginary High School. Go Dragons! Your teacher, Mr. Snigglefritz, is requiring that every CS 1 student memorize the ASCII value associated with each of the letters of the alphabet both lower and uppercase. Baldo has decided to type up a chart that contains the letters and their ASCII values so that he can study without being near his computer or phone. You on the other hand see a programming opportunity.

Input: None.

Output: A chart of all 26 English alphabet letters and their ASCII values, in natural order. On each row there should be four items listed, with single space separation:

- an uppercase letter
- that letter's ASCII value
- the same letter in lowercase
- the ASCII value for the lowercase letter.

Sample input:
None

Sample output:
```
A 65 a 97
B 66 b 98
C 67 c 99
D 68 d 100
E 69 e 101
F 70 f 102
G 71 g 103
H 72 h 104
I 73 i 105
J 74 j 106
K 75 k 107
L 76 l 108
M 77 m 109
N 78 n 110
O 79 o 111
P 80 p 112
Q 81 q 113
R 82 r 114
S 83 s 115
T 84 t 116
U 85 u 117
V 86 v 118
W 87 w 119
X 88 x 120
Y 89 y 121
Z 90 z 122
```

2. Barb

Program Name: Barb.java Input File: barb.dat

Barb has just learned some basic string manipulation techniques and is fiddling around with them, taking any word that comes to mind and messing with it.

Study the output below and reproduce it, given a single lowercased input word each time.

Input: Several lowercased single words, each on one line, containing only alpha characters.

Output: The original word, plus three more variations as shown in the sample output below.

Sample input:

```
red
yellow
green
```

Sample output:

```
red DER er ED
yellow WOLLEY ley LOW
green NEERG erg EEN
```

3. Constanza

Program Name: Constanza.java **Input File: constanza.dat**

Constanza loves art, especially using the seven colors of the rainbow, indicated by the acronym, ROYGBIV *(red, orange, yellow, green, blue, indigo, violet)*. She has collected numerous pieces of abstract "rainbow" art, and has scanned each image into a file that contains a listing of the rainbow colors of each pixel. For one of her art pieces, the first part of the listing looks like this *(shown here in three columns, but all in one column in the actual listing)*, and continues for many more lines:

```
yellow          green           green
orange          blue            red
green           yellow          violet
indigo          red             yellow
green           blue            indigo
blue            blue            red
orange          indigo          red
```

She wants to analyze the pixel frequency of each color for this art piece, creating a horizontal bar graph that shows this, a display that looks like the one shown below, with the rainbow colors listed in order, each followed by the number of stars indicating the frequency of that color in the art piece.

```
red    ****
orange **
yellow ***
green  ****
blue   ****
indigo ***
violet *
```

Input: A vertical, single column listing of only rainbow colors (red, orange, yellow, green, blue, indigo, violet) in the format as shown above, except with each on one line, and no multiple columns. The file listing is for one and only one art piece. There will be at least one instance of each color.

Output: A single horizontal bar graph display representing the frequency of the colors in the data file. Both the color labels and bars of stars must be left-aligned, and spaced exactly as shown above.

Sample input and output shown in example above.

4. Dilbert

Program Name: Dilbert.java **Input File: dilbert.dat**

Dilbert is the chairman of Firethorne High School Student Council's fundraising committee. The committee's October fund raising activity this year was to sell Halloween candy for customers to pass out on Halloween night to trick or treaters. Dilbert's record keeping for the fundraiser consists of a list of names, dates and dollar amounts turned in each day by various committee members. Dilbert promised a prize to the top seller for the fundraiser, so he needs to know which of his committee members did the best job of selling candy. Since you serve on the technology committee of the Student Council, Dilbert has asked you to write a program that will process his list of names, dates and sales, producing a report that will help him determine the top prize winner.

Input: Several sets of data lines, each containing a first name, last name, date, and dollar amount turned in on that day, with single space separation between each item. All dates will be listed in the format **mm/dd/yyyy**. Dollar amounts range in value from at least one dollar to hundreds of dollars, do not contain a dollar sign ($) and will be listed with two decimal places of precision. Each committee member is listed by first and last name. No two members have the same name.

Output: A list that contains an entry for each committee member where each person's name is on the first line followed on subsequent lines by each date money was turned in by that person, with the amount in dollar format immediately following the date, separated by a single space. The entries for each member are to be sorted by date, earliest to latest. The final line for each committee member should be a line that contains the total sales for that committee member. The list of committee member records should be sorted in **descending** order based on total sales, with the first name listed in the report having the greatest total sales. All dates should be displayed in the **mm/dd/yyyy** format, and all dollar amounts should be preceded by a dollar sign ($), with no space between it and the first digit of the dollar value. Here is a template for each committee member's output:

```
FirstName LastName
10/1/2017 $##.##
10/15/2017 $##.##
10/20/2017 $##.##
10/29/2017 $##.##
Total Sales $##.##
```

Each committee member's record should be followed by a blank line. The last line of the output should be the grand total for all sales during the fundraiser, shown as follows:

```
Grand Total $###.##
```

Sample input:

```
Cory Randolph 10/6/2017 25.25
Idalia Sauceda 10/3/2017 15.00
Alonzo Carrasco 10/6/2017 31.10
Steve Flanary 10/2/2017 20.00
Cory Randolph 10/30/2017 18.50
Steve Flanary 10/15/2017 12.75
Cory Randolph 10/14/2017 35.90
Idalia Sauceda 10/20/2017 26.35
```

Sample output:

```
Cory Randolph
10/6/2017 $25.25
10/14/2017 $35.90
10/30/2017 $18.50
Total Sales $79.65

Idalia Sauceda
10/3/2017 $15.00
10/20/2017 $26.35
Total Sales $41.35

Steve Flanary
10/2/2017 $20.00
10/15/2017 $12.75
Total Sales $32.75

Alonzo Carrasco
10/6/2017 $31.10
Total Sales $31.10

Grand Total $184.85
```

5. Emily

Program Name: Emily.java Input File: emily.dat

In geometry, the class is reviewing area formulas of standard geometric figures, and Emily has decided to write a program to double check her work, especially with trapezoids. She knows that the area of a trapezoid is the **average of the bases times the height**, but wants to make sure she is doing the work correctly.

Input: Several lines of data, each line containing three integers representing the two bases and height of a trapezoid.

Output: The area of the trapezoid, with decimal precision of one place if the value is fractional, otherwise no decimal point showing for whole number results.

Sample input:

```
5 7 3
8 3 12
3 4 5
```

Sample output:

```
18
66
17.5
```

6. Fa

Program Name: Fa.java **Input File: fa.dat**

In a recent thought experiment Fa conducted dealing with balancing lists of values, he imagined a list of random numbers, all non-negative within a limited range (each value less than 100). He stored them in original order in a contiguous data structure, and wanted to find the balancing point, or fulcrum, of the list. He realized that this point could either be in between two values of the list, or at an exact position of one of those values in the list, depending on the sum of the values before and after the fulcrum. He wanted the **"before"** and **"after"** sums to be equal, or as close as possible to being equal, so that the list was balanced at that point.

For example, in a list containing the values, 1 2 3 2 1, the fulcrum clearly is at position 2, right at the value 3, since the sum of the values before and after the fulcrum are equal, a sum of 3 on each side, making a perfectly balanced list.

```
1 2 3 2 1
    ^
```

He decided to indicate his findings for this list showing the **"before"** sum, the position of the fulcrum, and then the **"after"** sum, like this: **`3 ^2 3`**

For another list he tried, 3 6 3 3 5 8, he found the balancing point to be at position 3, with a sum of 12 (3+6+3) before that position, and 13 (5+8) after. He realized this wasn't a perfect balance, but he was content with "almost", since it was the best positioning of a fulcrum to best balance the list.

```
3 6 3 3 5 8
    ^
```

The report for this example was: **`12 ^3 13`**

In a third list of values, 3 3 3 3 4 3, the fulcrum was actually in between positions 2 and 3, with a sum before of 9, and after of 10.

```
3 3 3 3 4 3
    ^
```

Since the fulcrum ended up between two elements, the report format was slightly different, with the two index positions listed before and after the caret symbol, like this: **`9 2^3 10`**

Input: Several data sets, each consisting of two lines. The first line contains an integer N (2 <= N <= 50) indicating the number of elements in the list, followed on the next line by N random non-negative integers, no greater in value than 100.

Output: For each data set, show the final report formatted as shown above, with the "before" sum listed first, the fulcrum next, followed by the "after" sum, with single space separation. ***Note:*** *it is guaranteed there will be a "unique" fulcrum for each data set.*

Sample input:

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

Sample output:

```
3 ^2 3
12 ^3 13
9 2^3 10
```

7. Judith

Program Name: Judith.java **Input File: judith.dat**

Judith is doing research using some very precise floating-point measurement values, some very large, some very small, and her first step is simply to sort them in ascending order. Can you help her do that?

The sample input and output below show an example of what she is dealing with.

Input: A datafile containing several non-negative floating-point values.

Output: An ascending sorted list of the values, each value shown in the **exact format** as was provided in the data file.

Sample input:

```
395.96774839412213864694087795
375.286439548498
8.412190040221
0.382560882678260591251068
0.23943358855017051969126499
95.96038946102548087733835
0.392863116206040
557.91740143772795279
46.8020528027329
0.91986136436504
```

Sample output:

```
0.23943358855017051969126499
0.382560882678260591251068
0.392863116206040
0.91986136436504
8.412190040221
46.8020528027329
95.96038946102548087733835
375.286439548498
395.96774839412213864694087795
557.91740143772795279
```

8. Luann

Program Name: Luann.java **Input File: luann.dat**

This past summer Luann collected data on how many turtles she saw in the pond behind her home. Several days in a row she would go down to the creek and count the turtles in the pond and then record her findings in a data file. Luann didn't count turtles every single day. She would go on consecutive days for a while then skip a few days. She never counted turtles more than nine days in a row. Now that summer is over, Luann would like to chart her findings. She wants a different chart for each set of consecutive days that she counted turtles. Let's help her write a program that will process the data and print out the charts she wants.

Input: A single value C representing the number of charts to be printed. N will be followed by C sets of data each beginning with a number N that represents the number of coordinate pairs to follow. Each coordinate pair will be on a separate line where the first value is x and the second value y. Both x and y will be greater than or equal to 1 and less than or equal to 9. The x values will always begin with one and continue in sequential order with no skips. For example, if there are 7 coordinate pairs, the x values will always be 1 – 7 without any skips.

Output: A chart for each set of data. Each chart should have an x and y axis labeled starting at one (1). The y axis should be formed using pipes (|) and the x axis should be formed using underscores (_). The y axis and its labels should take up two spaces in each row and the x axis and its labels should take up two rows. Each data point (y value) should be marked with a capital X. Any space within the chart that does not have an axis, a label or a data point must be filled with a blank. Each chart should be followed by the line, "=====".

Sample input:

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

Sample output:

```
6|      X
5|    X
4|   X
3|  X  X
2| X
1|X

  _______
  1234567
=====
9|  XXX
8|
7|     X
6|
5|      X
4| X
3|
2|       X
1|X       X

  _________
  123456789
=====
```

9. Lucy

Program Name: Lucy.java Input File: lucy.dat

Lucy is an artist. Her preferred medium is ASCII characters. She creates large images using nothing more than the characters she finds on her computer's keyboard. She stores these images in data files and sends them to a company called ***Print This!*** to be printed on canvas. Lucy then frames the prints and puts them on sale in her booth down on Guadalupe street in Austin. A problem has arisen with sending the files to ***Print This!*** Lucy's artwork has become so big that it is taking too much time and using too much bandwidth to send the files to the printer. Because you are one of Lucy's most treasured friends and she knows that you are a skilled programmer she has asked if you can write a program that will compress her files and make them easier to send to the printer.

A quick check of the internet turns up an algorithm that looks like it will do the job. It is called **run-length encoding**. Run-length encoding is a very simple "loss-less" data compression routine in which runs of characters (characters that are repeated in sequence) are stored as a pair of values, the character itself and the number of times it occurs in succession. This algorithm works best on data that contains many such runs, simple graphic images such as icons, line drawings, and animations. This algorithm is just what we need to compress Lucy's art work without any loss of each image. Run-length encoding for the string `aaaiiOOOOOOqwwwww**********` would result in `a3i2O6q1w5*10`.

Input: The first line of data will contain a value D representing the number of drawings to compress, followed by D drawing data set. Each drawing data set consists of a number N on one line representing the number of lines in the drawing, followed by N lines that make up the drawing. Each line will contain non-digit alpha and symbol characters, including spaces..

Output: For each of the drawings, N lines that consist of the run-length encoding for that line in the original drawing. Each encoded drawing should be ended by this line, "=====".

Sample input:

```
2
13
..................
@................@
.@..............@.
..@............@..
...@@@@@@@@@@@@...
...@..........@...
....@..**..**..@....
....@..**..**..@....
.....@........@.....
....../@...++...@.....
.../&&@..++..@......
../&&&/@@@@@@.......
./&&&/.............
8
.............................
....____.  ..................
....|    |_____ ___  ________...
....|    \__  \\  \/ /\__  \..
/\__|    |/ __ \\   /  / __ \_
\________(____  /\_/  (____  /
.............\/..........\/.
.............................
```

(Lucy – cont)

Sample output:

```
.20
@1.18@1
.1@1.16@1.1
.2@1.14@1.2
.3@14.3
.3@1.12@1.3
.4@1.2*2.2*2.2@1.4
.4@1.2*2.2*2.2@1.4
.5@1.8@1.5
.4/1@1.3+2.3@1.5
.3/1&2@1.2+2.2@1.6
.2/1&3/1@6.7
.1/1&3/1.14
=====
.30
.4_4.1 1.20
.4|1 4|1_4 1_3 2_7.3
.4|1 4\1_2 2\2 2\1/1 1/1\1_2 2\1.2
/1\1_2|1 4|1/1 1_2 1\2 3/1 2/1 1_2 1\1_1
\1_8(1_4 2/1\1_1/1 2(1_4 2/1
.14\1/1.11\1/1.1
.30
=====
```

10. Micaela

Program Name: Micaela.java **Input File: micaela.dat**

Micaela is fascinated with writing encrypted messages and has come up with one she thinks will do the trick. See if you can figure out a program to decipher her message.

For example, she wants to send the word "BLUE" as a part of a message to her friend, Ayden, and decides to use two long nonsense words, both of which contain the word "BLUE", with those four letters in that order, somewhere inside each word, but with many other letters in between. She designs each word so that the word "BLUE" is the longest possible sequence of common letters among both words.

The two code words she sends are **DEBQGLBFUE** and **BMANOPLRSTUEVE**. There are other sequences that are common, but are not as long. For example, **BUE** and **LUE** are both common, but aren't the longest ones. She is also careful to make sure that there are no two "different" common words that are of the longest length, for obvious reasons; it would be quite confusing to the one receiving the coded message!

To mark the end of a thought in the message, she includes two encoded sequences that have nothing in common. In that case, for testing purposes, she just says "NONE" when that happens.

Input: A data file of several pairs of encoded character sequences, in all caps, with each pair on a separate line.

Output: The word that is the longest sequence of characters common to both encoded sequences, or the word "NONE" if there is no matching sequence.

Sample Input:

```
DEBQGLBFUE BMANOPLRSTUEVE
URPWOEGJSDLFJA RARKQENMED
QGWFEDRGQWFD ULIKJOULIKJYUKLIJ
```

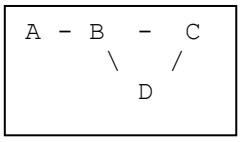
Sample Output:

```
BLUE
RED
NONE
```

11. Nikita

Program Name: Nikita.java Input File: nikita.dat

Nikita's friend Stelios has been researching efficient ways to represent connections within graphs, and has developed a shortest path matrix that shows this. The resulting chart contains the least number of steps it takes to get from one node to any other node. Nikita wants to take Stelios' work and do some analysis about the general nature of any particular graph, especially regarding the accessibility of any node to all other nodes within the graph. For example, he is interested in how many nodes directly connect with each node in the graph, what he considers the "degree" of a node, and generally how "close" each node is to the rest of the graph.



	A	B	C	D
A	0	1	2	2
B	1	0	1	1
C	2	1	0	1
D	2	1	1	0

	1	2
A	1	2
B	3	0
C	2	1
D	2	1

Nikita looks at one of the graphs Stelios has worked with, specifically the four-node graph shown above. The connection matrix in the center shows values that represent how many steps it takes to get to each node from any other node. He observes that the longest path in this situation is 2, and so develops a two-column "length tally" matrix to represent this situation, like the one on the right. This shows that the A node has **one** 1-link connection, and **two** 2-link connections. The B node has **three** 1-link connections, and C and D each have **two** 1-link and **one** 2-link connections.

Just by looking at the graph, Nikita can intuitively judge that the A node seems to be the most isolated, and that B is clearly the most centralized. He thinks about how to mathematically determine this, and comes up with two ways to measure this. First, he wants to measure the "degree" of each node. The formula he comes up with is this:

- Divide the number of 1-link connections for each node by one less than the number of nodes.

Using the length tally matrix shown above, the A node calculation would be 1/3, or 0.33. The B node would be 3/3, or 1.00. The C and D nodes both would be 2/3, or 0.67. Clearly the node with the least degree is A, and the one with the most degree was B, which supports the Nikita's original intuitive observation.

The "closeness" idea is a little more of a challenge, but after pondering a while he decides to use this formula. For each node:

- Find the sum of the number of 1-link connections times 1, 2-link connections times 2, 3-links * 3, etc.
- Divide that sum **into** the value representing one less than the total number of nodes in the graph.

For node A, the "closeness" calculation would be **3 divided by (1*1 + 2*2), or 3/5, or 0.60.**
For node B it would be **3 / (3*1 + 2*0), or 3/3, which is 1.00.**
Nodes C and D both have a "closeness" result of **0.67**.

This "closeness" measure at first seems to show the same result as the "degree", in that the A node appears to be the most remote node, with the least "closeness" value, and that B is the most central node, with the greatest "closeness" value, which leaves Nikita unsatisfied. He decides to look at one more example (see next page).

Nikita - cont

```
A - B - C - D - E - F - G - H
```

```
  A B C D E F G H
A 0 1 2 3 4 5 6 7
B 1 0 1 2 3 4 5 6
C 2 1 0 1 2 3 4 5
D 3 2 1 0 1 2 3 4
E 4 3 2 1 0 1 2 3
F 5 4 3 2 1 0 1 2
G 6 5 4 3 2 1 0 1
H 7 6 5 4 3 2 1 0
```

```
  1 2 3 4 5 6 7
A 1 1 1 1 1 1 1
B 2 1 1 1 1 1 0
C 2 2 1 1 1 0 0
D 2 2 2 1 0 0 0
E 2 2 2 1 0 0 0
F 2 2 1 1 1 0 0
G 2 1 1 1 1 1 0
H 1 1 1 1 1 1 1
```

He tries these measures on one more graph, essentially a single linked chain of nodes, as shown in the box above. The connection matrix is shown in the center, and the length tally matrix on the right. After analyzing that graph, Nikita finally finds a difference in the "degree" and "closeness" measure, and decides to use both to analyze many more graphs.

In this example, there is a six way tie (BCDEFG) for the nodes with the greatest "degree" (0.29) of connection, but only two nodes (D and E) share the greatest "closeness" (0.44) measure. The two outermost nodes, A and H, both measure the least in "degree" (0.14) and "closeness" (0.25).

Input: Several data sets, each consisting of an initial value N, representing the number of nodes in the graph, followed on the next N lines by an NxN matrix of integer values, representing the connection matrix for the graph as described above, with single space separation.

Assumptions: The node labels for each graph will always begin with upper-case A and proceed in alpha order for as many nodes as are represented by the graph.

Output: For each data set, output four values, labeled, formatted and aligned exactly as shown below. List all nodes in alpha order that match each measure. Each output will be followed by a single line of three dashes.

Sample Input:

```
4
0 1 2 2
1 0 1 1
2 1 0 1
2 1 1 0
8
0 1 2 3 4 5 6 7
1 0 1 2 3 4 5 6
2 1 0 1 2 3 4 5
3 2 1 0 1 2 3 4
4 3 2 1 0 1 2 3
5 4 3 2 1 0 1 2
6 5 4 3 2 1 0 1
7 6 5 4 3 2 1 0
6
0 1 1 2 2 2
1 0 2 1 1 1
1 2 0 3 3 1
2 1 3 0 1 2
2 1 3 1 0 2
2 1 1 2 2 0
```

Sample Output:

```
least degree       0.33 A
greatest degree    1.00 B
least closeness    0.60 A
greatest closeness 1.00 B
---
least degree       0.14 AH
greatest degree    0.29 BCDEFG
least closeness    0.25 AH
greatest closeness 0.44 DE
---
least degree       0.40 ACDEF
greatest degree    0.80 B
least closeness    0.50 C
greatest closeness 0.83 B
---
```

12. Peter

Program Name: Peter.java Input File: peter.dat

Peter is all about efficiency in everything he does. He despises wastefulness, and is always looking for ways to utilize his resources in the most frugal way. This includes managing his money when he spends it, specifically in paying for things he buys with the coins he saves up in a jar on his dresser. He is obsessed with using the fewest number of pieces of money to find exact change for the cost of the item.

He also has a hobby of studying obsolete money systems, or even makes up new ones in his mind, just for fun, still finding ways to make exact change in the most efficient way.

In the American system of coins, using pennies, nickels, dimes, quarters, half-dollars and dollar coins, he would pay for an item costing 70 cents with 1 a half-dollar and two dimes, a total of three coins used.

In an imaginary system, with only three coins valued at 1, 3 and 5 units, an item costing 11 units would use two 5 coins and a 1 coin.

He needs you to write a program to help in testing his efficient change-making efforts.

Input: Several sets of data, each set on two lines. The first line contains a single integer C representing the cost of the item to be purchased. The next line contains several integers representing a coin system, always starting with 1, and proceeding upwards in value, with single space separation.

Output: The original value C, followed by a value N representing the least number of coins required to make exact change, followed by an ascending list of each coin, all values with single space separation.

Sample input:

```
11
1 3 5
70
1 5 10 25 50 100
131
1 4 9 14 22 53
```

Sample output:

```
11 3 1 5 5
70 3 10 10 50
131 6 1 1 1 22 53 53
```