A+ Computer Science February 2012

Computer Science Competition
Hands-On Programming 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.

6. 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.

Number	Name
Problem 1	Picture
Problem 2	Boxes
Problem 3	Backwards
Problem 4	Rocks
Problem 5	Triangles
Problem 6	Hoop
Problem 7	DimeSweeper
Problem 8	Alliteration
Problem 9	Stock it to me
Problem 10	SooperSizeMe
Problem 11	Combo
Problem 12	Sum Digits

Good luck!

1. Picture

Program Name: Picture.java Input File: none

General Statement : Print out the picture as shown below.

Input: none

Output: Print out the picture as shown below.

Assumptions – Helpful Hints : none

Example Input File
none

Example Output to screen:
```
####################
#UILCOMPUTERSCIENCE#
####################
#UILCOMPUTERSCIENCE#
####################
####################
#UILCOMPUTERSCIENCE#
####################
#UILCOMPUTERSCIENCE#
####################
```

2. Boxes

Program Name: Boxes.java Input File: boxes.dat

General Statement : Print out each box as shown below.

Input: The first line in the data file will indicate the number of data sets to follow. Each data set will contain the size of the box to be printed.

Output: Print out each box of the appropriate size as shown below.

Assumptions – Helpful Hints : none

Example Input File

```
3
3
5
4
```

Example Output to screen:

```
$$$
$ $
$$$

$$$$$
$   $
$   $
$   $
$$$$$

$$$$
$  $
$  $
$$$$
```

3. Backwards

Program Name: Back.java Input File: back.dat

General Statement : Read in a string and determine if that string reads the same forwards as backwards.

Input: The first line in the data file will indicate the number of data sets to follow. Each data set will contain a single string with no spaces.

Output: Print out `SAME` if the word reads the same forwards as backwards or `NOT SAME` if the word does not read the same forwards as backwards.

Assumptions – Helpful Hints : none

Example Input File

```
3
mom
dog
pumpkin
```

Example Output to screen:

```
SAME
NOT SAME
NOT SAME
```

4. Comp Sci Rocks!!

Program Name: Rocks.java Input File: rocks.dat

General Statement : You think Comp Sci Rocks!! and you just like saying that over and over again. Read in a number from the data file and print out Comp Sci Rocks!! that number of times.

Input: The data file will contain an unknown number of lines.

Output: Print out the number of lines in the data file.

Assumptions – Helpful Hints : none

Example Input File

```
3
2
5
1
```

Example Output to screen:

```
Comp Sci Rocks!!
Comp Sci Rocks!!

Comp Sci Rocks!!
Comp Sci Rocks!!
Comp Sci Rocks!!
Comp Sci Rocks!!
Comp Sci Rocks!!

Comp Sci Rocks!!
```

5. Triangles

Program Name: Triangles.java Input File: triangles.dat

General Statement : Read in a letter and a number. The number indicates how big the letter triangle should be. The number indicating the size of the triangle will have a range from 0 to 250. num>=0 and num<=250

Input: The first number indicates the number of data sets to follow. Each data set will contain one letter and one number. All letter input will be uppercase.

Output: Print out the appropriate sized letter triangle as shown below. The letter provided is the starting letter and that letter is printed once and then the sequence continues.

Assumptions – Helpful Hints : The letters must wrap around from Z to A. If you start with Z and have to print 5 levels, you must wrap around and start with A after the Z level is complete. The triangle lettering starts at the bottom and goes up.

Example Input File

```
3
5 A
3 Z
4 C
```

Example Output to screen:

```
EEEEE
 DDDD
  CCC
   BB
    A

BBB
 AA
  Z

FFFF
 EEE
  DD
   C
```

6. HoOp

Program Name: Hoop.java Input File: none

General Statement : Print out the word `Hoop` as shown below.

Input: none

Output: Print out the word `Hoop` as shown below.

Assumptions – Helpful Hints : none

Example Input File
none

Example Output to screen:

```
#   # #### #### ####
#   # #  # #  # #  #
#### #  # #  # ####
#   # #  # #  # #
#   # #### #### #
```

7. DimeSweeper

Program Name: DimeSweeper.java Input File: dimesweeper.dat

There is an old teacher joke – how do you knock out two teachers at the same time—drop a quarter between them! You create a video game where a teacher tries to guess where the coins are in the squares of the tiles in the hallway of your school and locate them. The board is a matrix which contains the characters "01234D" indicating either the number of coins adjoining that cell or "D" indicating that a coin is on that cell. There are several options for mouse input in this GUI program:

1. You can right-click on the cell you suspect contains a coin to mark it with a flag.
2. You can left-click on the cell that contains the coin and the game ends—the teacher is knocked unconscious!
2. If you left-click and the digits "1234" are present, you may continue.
3. If you left-click and the digit "0" is present, if will uncover all the adjoining cells that are also "0". Adjoining cells are defined as cells that are touching the current cell horizontally, vertically, or diagonally.

This sounds vaguely familiar somehow….

Your job is to write the code for these 4 options.

1. If there is a right click on a "D", then change that cell to "F".
2. If there is a left click on a "D", then output "GAME OVER" and print out the matrix. (It would be the last move in the game)
3. If there is a left click on any of these digits "1234", then do not change that cell.
4. If there is a left click on a "0" digit, then change all adjoining cells horizontally, vertically or diagonally to a space " ".

If there is a left-click on a "D", that will be the last move in the data set. However, the other options may have more valid moves after, so print the matrix after the remaining moves (at the end of the data set).

Input: The first line will indicate the size of the game board (R, C). The next R lines contain the game board, consisting of the characters "01234D." The next line will indicate the number of data sets N (using the same initial board). The first line in each data set contains the number of moves, each on one line. Each move consists of the letter L or R followed by the row (0,1…R-1) and column (0,1,…C-1).

Output: Output "GAME OVER" and the matrix, or just the modified matrix at the end of each game. Leave a blank line between separate games.

(Continued on next page…)

(Problem 7 contin.)

Example Input file

```
15 20
00000000000000000000
00111000001110000000
001D1000001D21000000
111111110012D1000000
D10001D1000111000000
11000111000000123210
000000000000002DDD10
000000000000002D4210
00000000000000111000
11111111111111111111
D11D11D11D11D11D11D1
11111111111111111111
00122100000000000000
002DD100000000000000
002D3100000000000000
3
10
L 1 2
L 1 3
L 1 4
R 2 3
L 0 0
L 2 10
L 1 10
L 2 12
R 2 11
R 3 12
2
L 2 2
L 2 3
5
L 0 0
L 13 0
L 12 6
R 10 0
R 10 3
```

(Continued on next page…)

(Problem 7 contin.)

Output to screen:

```
  111     111
  1F1     1F21
11111111  12F1
D1   1D1   111
11   111      12321
              2DDD1
              2D421
              111
1111111111111111111
D11D11D11D11D11D11D1
1111111111111111111
0012210000000000000
002DD100000000000000
002D3100000000000000

GAME OVER
0000000000000000000
0011100000111000000
001D1000001D21000000
111111110012D1000000
D10001D1000111000000
1100011100000123210
0000000000000002DDD10
000000000000002D4210
0000000000000111000
1111111111111111111
D11D11D11D11D11D11D1
1111111111111111111
0012210000000000000
002DD100000000000000
002D3100000000000000


  111     111
  1D1     1D21
11111111  12D1
D1   1D1   111
11   111      12321
              2DDD1
              2D421
              111
1111111111111111111
F11F11D11D11D11D11D1
1111111111111111111
  1221
  2DD1
  2D31
```

8. A Lot of Alliteration

Program Name: Alliteration.java Input File: alliteration.dat

Alliteration is a term that describes sequential words that begin with the same letter (used in tongue-twisters, poetry and works of literature). You will take a sentence or phrase and determine if it has a lot of alliteration. If there are two words in a row, that could be an accident. If there are three words in a row, that is "some alliteration." If there are 4 or more, that is "a lot of alliteration." Determine the amount of alliteration in each sentence/phrase and find the most absolutely awesome alliterative addition to this completely clever collection!

Input: The first line will indicate the number of data sets. Each data set will consist of a sentence or phrase, which may contain capital letters, with words separated by one space.

Output: For each line, show the following options:

- "none" for no alliteration at all
- "accident" for two words in a row
- "some alliteration" for 3 words in a row
- "a lot of alliteration" for 4 or more words in a row

The last line of output should show the maximum number of all data sets in the form:
"Line __ has __ words in a row." There will be no two data sets that tie for maximum number.

Example Input file

```
8
An apple an afternoon always affects aardvarks awfully.
We don't need no education.
This sentence has no alliteration at all.
The quick brown fox jumped over the lazy dog.
A balloon came down every Friday going over the rainbow.
Ten tiny tots came really running roughly round the corner.
Sally Sue sells seashells by the seashore.
Shel Silverstein sunburns selectively sometimes in the summer.
```

Output to screen:

```
a lot of alliteration
accident
some alliteration
none
none
a lot of alliteration
a lot of alliteration
a lot of alliteration
Line 1 has 8 words in a row.
```

9. Stock It To Me

Program Name: Stock.java Input File: stock.dat

Many stock traders have tried to use data trends to predict the stock market to try to "make it big." You are employed by another such trading firm. Your job is to track a stock price for a month (20 trading days) and decide whether to buy, sell, or hold. You want to buy low at the start of a trend, and sell high at the end of a trend. Here are the determining factors:

- If the price goes up for 10 straight days, then sell that stock, predicting it will fall.
- If the price goes down for 10 straight days, then buy that stock, predicting it will rise.
- If the price goes up and down, but has a general increase of 20% from the initial price at 10 days ago, then sell that stock for a profit.
- If the price goes up and down, but has a general decrease of 20% from the initial price, then buy that stock hoping for a future increase.
- Otherwise, hold the stock until a trend develops.

Here is an example:
`1.00 2.00 2.50 3.00 3.50 4.00 4.50 5.00 5.50 6.00 6.50 7.00 6.50 6.00 ...`
For the previous trend, you should sell at 6.50 due to 10 straight days of increase.

Another example:
`6.00 5.00 5.50 6.00 7.50 4.00 4.50 5.00 6.50 7.00 8.50 7.00 6.50 6.00 ...`
For the previous trend, there was fluctuation up and down. On the 11th day, there was a 42% increase from the 1st day (from 6.00 to 8.50), so you should sell at 8.50.

To simplify this program assume:
- There can only one trend per month.
- The trend may not start immediately, but will start by the 11th number (the last 10 could be a trend).

Input: The first line will indicate the number of lines in the data file. Each subsequent line contains 20 stock prices.

Output: Show one of the following possible outcomes:
- `buy at __`
- `sell at __`
- `hold`

(Continued on next page…)

(Problem 9 continued)

Example Input file (Note: in the actual file all 20 numbers are on a single line)

```
5
1.00 2.00 2.50 3.00 3.50 4.00 4.50 5.00 5.50 6.00 6.50 7.00 6.50 6.00 7.00 6.00 7.00 8.00
8.00 8.00
6.00 5.00 5.50 6.00 7.50 4.00 4.50 5.00 6.50 7.00 8.50 7.00 6.50 6.00 6.00
6.00 7.00 6.00 5.99 5.98
8.00 7.99 7.98 7.97 7.96 7.95 7.94 7.93 8.00 7.92 7.91 7.90 7.89 7.88 7.87
8.00 7.86 7.85 7.84 7.83
9.87 12.55 11.94 11.25 20.12 19.87 15.45 13.98 12.50 11.99 10.54 10.44 10.32
10.01 9.23 9.21 10.21 11.00 12.00 13.00
10.00 9.50 10.00 9.50 10.00 9.50 10.00 9.50 9.25 9.15 9.00 8.75 8.50 10.00 8.50 8.25 8.22
8.05 7.99 7.01
```

Output to screen:

```
sell at 6.50
sell at 8.50
hold
buy at 9.23
buy at 7.01
```

10. SOOPSer Size

Program Name: SOOPSerSize.java Input File: soopsersize.dat

Someone in the McDonald's home office has a "great idea." They decide to give a deal where they double the price of an item and give a larger amount of food. However, they aren't too hot in math, and they decide to square the amount of food. OOPS! You will write the software to update the McDonald's corporate database. Given a price and food amount, double the price and square the amount of food! You also decide to cash in your 401k before McDonald's stock price declines when the profits disappear.

Input: The first line consists of the number of data sets in the file. Each line consists of a dollar amount followed by an integer (the ounces, number of items….) representing the portion size.

Output: Print out the price doubled and the amount squared. Round the price to the nearest hundredth (cents) with a preceding dollar sign ($).

Example Input file

```
3
0.99 12
1.00 6
4.99 2
```

Output to screen:

```
$1.98 144
$2.00 36
$9.98 4
```

11. Lazy Combo

Program Name: Combo.java Input File: combo.dat

Your best friend keeps a combination lock on his bicycle at school. It has 4 rollers with the numbers 0-9, so there are potentially 10,000 different combinations. It also has a magnetic mechanism that allows you to change the combination. However, your friend is lazy and only changes 1 of the digits when he locks it! Not very smart! You want to show him the error of his ways, so you decide to write a computer program that will determine his combination by analyzing 4 days of how he leaves his bike lock (by only changing one digit). Hopefully your friend will see this and start being smart about either randomizing his combo when locking his bike, or periodically changing his combo.

For example, let's say he leaves his lock in the following configurations for 4 days:

1234	1357	1101	8878
1235	3257	1011	2878
1236	3397	0111	9877
1237	3355	2111	9870
123X, where X could be 0,1,2,3,8,9	Only combo is 3357	Only combo is 1111	Only combo is 9878
1230, 1231, 1232, 1233, 1238, 1239 are possible combinations			

If the same digit appears 2, 3 or 4 times, then it is the actual digit in the combination. If that digit changes for all 4 possibilities (appears only once), then those digits are not possible digits in the combination.

Input: The first line will indicate the number of data sets. Each data set will consist of a single line with 4 combinations of 4 digits each.

Output: Print out the possible combinations on a line for each data set, in numerical order. There will be at least one correct combination (and no more than 6) per data set.

(Continued on next page…)

(Problem 11 contin.)

Example Input file

```
4
1234 1235 1236 1237
1357 3257 3397 3355
1101 1011 0111 2111
8878 2878 9877 9870
```

Output to screen:

```
1230 1231 1232 1233 1238 1239
3357
1111
9878
```

12. Sum Digits

Program Name: Digits.java Input File: digits.dat

When young students in elementary school learn math, they begin with their math facts. Digit summing is another early math fact process they must learn. Write a program that will read in any number and sum up all of its base 10 digits. You might find % quite useful for this problem.

Input: The first line consists of the number of data sets in the file. Each line consists of a single integer.

Output: Print out the sum of all of the digits in each number.

Example Input file

```
5
100
88
123456
999
10101010101010101010
```

Output to screen:

```
1
16
21
27
10
```