Computer Science Competition
2016 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	Adam
Problem 2	Bartek
Problem 3	Camila
Problem 4	Daiki
Problem 5	Eun-suh
Problem 6	Francisco
Problem 7	Grace
Problem 8	Huang
Problem 9	Irina
Problem 10	Jorge
Problem 11	Kalyani
Problem 12	Lipun

1. Adam

Program Name: Adam.java Input File: adam.dat

Adam has recently begun exploring the new Java Lambda expressions, now available in Java 8!

Below is a program he wrote that uses the **BiFunction** lambda expression provided in the **java.util.function** package. It reads a series of pairs of values and outputs **true** or **false**, indicating if the first value is greater than the second value.

You may use his program, if you wish, or if you're not using Java 8, he has provided an alternate solution. If you want to solve it your own way, go for it. Have fun!

Input: Several pairs of integers, each pair on one line, separated by a single space.

Output: The word "true" or "false", indicating whether the first integer of each pair is strictly greater than the second integer.

Sample input:

```
1 2
2 1
14 -14
100 200
```

Sample output:

```
false
true
true
false
```

```
-------------------------------------
//UIL Invitational B, 2016, Adam – Solution
import java.util.*;
import java.io.*;
import java.util.function.*;
import static java.lang.System.*;
public class Adam
{
      public static void main(String...args) throws IOException
      {
            Scanner f = new Scanner(new File("adam.dat"));
            BiFunction<Integer,Integer,Boolean> match
                     = (Integer a, Integer b)->a.compareTo(b)>0;
            while(f.hasNext()){
                  Integer x = f.nextInt();
                  Integer y = f.nextInt();
                  out.println(match.apply(x,y));
                  //out.println(x.compareTo(y)>0);//non-lambda solution
            }
      }
}
```

2. Bartek

Program Name: Bartek.java **Input File: bartek.dat**

Bartek loves to write programs that generate ASCII patterns, but sometimes finds them quite challenging. Here's a good one he tried recently! Given an integer N, he prints an octagon whose sides are of length N.

How you solve this problem - brute force or dynamic/elegant - is up to you.

Input: Several integers N (1<N<=10), all on one line, separated by single spaces.

Output: For each integer N, print a size N octagon, made up of stars, as shown below, each output aligned on the left edge, and no blank lines between outputs.

Sample input:

```
2 3 4
```

Sample output:

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

3. Camila

Program Name: Camila.java Input File: camila.dat

Camila is fascinated with last names, and has been doing some research about how they are represented. She has discovered that some last names, even though spelled differently, sound alike, like SMITH and SMYTH, OWEN and OWENS, or PHILLIP and PHILIP. Furthermore, she has discovered a technique used by large databases, which categorizes last names using a coding process. The code for each last name is made up of a letter and three digits. The letter is the first one of the name, and the digits represent the remaining letters, according to the following rules:

- In the rest of the name, disregard all instances of the letters A, E, I, O, U, H, W, and Y
- The digit 1 represents the letters B, F, P and V
- The digit 2 represents the letters C, G, J, K, Q, S, X, and Z
- The digit 3 represents the letters D and T
- The digit 4 represents the letter L
- The digit 5 represents the letters M and N
- The digit 6 represents the letter R

The initial letter, plus the first three digits representing the rest of the letters in the name, make up the code for the name, with zero as a filler for any remaining digit locations.

For example, **SMITH** has a code of **S530**: S as the starting letter, 5 for M, I is ignored, T is 3, and H is ignored, with a zero added at the end for the 3rd digit. **SMYTH** works out the same way: S, 5 for M, Y is ignored, T is 3, and zero at the end. **CAMILA** would be **C540**, **ADAM** would be **A350**, and **BARTEK** would be **B632**. **WASHINGTON** would be **W252**, with W, 2 for S, 5 for N, and 2 for G, with the remaining letters ignored.

Any adjacent letters with the same digit code will be treated as one letter, like **ALLEN** (the first L counts, the second L is ignored, code is **A450**) or **PFISTER** (P counts, F is ignored, code is **P236**) or **JACKSON** (C counts, K is ignored, code is **J250**).

Input: Several last names, all in uppercase, each on one line. There will be no spaces or symbols in any name.

Output: The 4-digit code for each name, as described and demonstrated above.

Sample input:
```
SMITH
SMYTH
CAMILA
WASHINGTON
ALLEN
JILLIANNE
PFISTER
JACKSON
```

Sample output:
```
S530
S530
C540
W252
A450
J450
P236
J250
```

4. Daiki

Program Name: Daiki.java Input File: daiki.dat

This is one is kinda weird, but is real, believe it or not!

Daiki has a collection of weird Japanese toys, each of which has several characteristics. He is trying to figure a way to display his collection in some logical order, and has decided on the following process, based on these characteristics. The characteristics of his weird toys are: dominant color, weight, and overall "weirdness".

He decides to make the primary difference to be their weight (heaviest first), then color (in alpha order), then overall weirdness on a scale of 1 to 10, 10 being the weirdest, the weirdest listed first. If there is a tie after all three characteristics are considered, the name of the toy is used to break the tie, again in alpha order.

Of course, the weirdness factor is totally subjective, but it is Daiki's opinion that matters, regardless of what anyone else thinks.

Some of his toys are:

Poop and Pee Plushies, color brown, 120 grams, weird factor of 2
H-Bouya USB Toy, blue, 25, 1
Face Bank, red, 200, 4
Virus Plush, yellow, 60, 3
Road Kill Cat, white, 120, 5

If you think this is all made up, just check it out on the web **AFTER THIS CONTEST IS OVER** at:
http://www.tofugu.com/2013/09/19/ten-japanese-toys-you-might-want-to-reconsider-buying-for-your-children/

Input: Several weird toys, each on one line, with toy name, color, weight in grams, and weirdness factor, each part separated by commas, as shown below.

Output: All of the toys in sorted order, according to the criteria listed above.

Sample input:
```
Poop and Pee Plushies,brown,120,2
H-Bouya USB Toy,blue,25,1
Face Bank,red,200,4
Virus Plush,yellow,60,3
Road Kill Cat,white,120,5
```

Sample output:
```
Face Bank               (heaviest)
Poop and Pee Flushies
Road Kill Cat           (tied in weight with Poop and Pee Plushies, but loses tie on color)
Virus Plush
H-Bouya USB Toy
```

5. Eun-suh

Program Name: EunSuh.java **Input File: eunsuh.dat**

Eun-suh has learned a fun new game with digit sequences, and is trying to write a program to simulate this game. Your job is to help her in this effort!

Here are the rules of the game. Given a sequence of digits, modify the sequence by:

- Delete all zeroes, if any, and all digits to the left of the zeroes.
- Find the largest remaining digit and reduce it by 1 if it is odd, or by 2 if it is even.
  - If there are two or more instances of the largest remaining digit, use the rightmost one as the largest.
- Repeat these steps until the sequence of digits is completely deleted.

For example, for the starting sequence 3580254, the first step reduces it to 254, removing the zero and all digits to the left of the zero. Next comes 244, where the largest digit 5 is reduced by 1 to 4, then 242, 222, 220, and then an empty sequence, for a total of 6 moves.

The sequence 830751 first becomes 751, then 651, 451, 441, 421, 221, 201, 1, 0, and finally empty, taking a total of 10 moves.

Input: Several sequences of digits, each on one line.

Output: The number of moves it takes to completely remove all the digits from the sequence.

Sample input:
```
3580254
830751
550604
```

Sample output:
```
6
10
4
```

6. Francisco

Program Name: Francisco.java **Input File: francisco.dat**

Francisco loves to play the game, **Tile Bonanza**, which uses a square board with black and white tiles. The goal of this game is to rearrange the pieces into an optimal configuration with a limited number of moves. The rules for a move are that any one piece can be moved to any empty space among its adjacent locations (up, down, left, right, or any of the four diagonal locations). A piece can only be moved to an empty spot. The optimal configuration encourages grouping of pieces of like color and separating pieces of different color. To calculate the scoring, for every neighboring piece of like color in an adjacent square, 10 points is awarded. For each adjacent piece of opposite color, 5 points is deducted. Given this scoring system, with the number of moves indicated, find the optimal configuration and report the score, and output the board as well if only one optimal configuration is possible.

```
B 0 0
0 W 0
W 0 B
```

For example, in the board to the left, before any moves are made, the score is 0. Francisco can see this by looking at each piece on the board individually. Going from left to right, top to bottom, the first Black piece has a White piece in one of its adjacent squares, bringing the total to -5. The White piece in the center has a White and two Blacks next door, which totals 0, keeping the score at -5. The next White piece in the bottom left has an adjacent White piece, which adds 10, bringing the total to 5. Finally, the Black piece in the bottom right has an adjacent White piece, subtracting 5, bringing the total score to 0.

Given exactly one allowed move, the optimal configurations to maximize the score for this board results in the following two optimal scoring boards, both with a point total of 10. The middle White piece can either move down one space, or left one space, both moves resulting in a score of 10.

```
B 0 0
0 0 0
W W B

B 0 0
W 0 0
W 0 B
```

Given a board where the digit 0 represents a space, the digit 1 represents Black and the digit 2 represents White, and the number of allowed moves to find an optimal score, print that score and the number of boards that result in that score. If there is only one configuration that yields the optimal score, display that configuration.

Input: The first integer will represent the number of data sets to follow. The first integer M (0<M<=K) of each data set represents the number of allowed moves. The next integer, K (2<K<=10), will be the size of the board. The next K lines contain a square grid made up of the digits 0, 1, and 2 (single spaces separating each digit, no blank lines between rows) representing the initial configuration of the board. There will always be the same number of black and white pieces, and at least one piece of each color. There will always be at least one empty square.

Output: The maximum score is prefixed by the sentence: "**MAXIMUM SCORE: <score>** ", followed by "**THERE ARE <num> OPTIMAL BOARDS.**" where **<score>** represents the optimal score, and **<num>** represents the number of different configurations that yield that maximum score. If there is only one such board, print "**THERE IS 1 OPTIMAL BOARD.**", and then also **print the board immediately below**. Print a blank line after each output set.

Sample Input:

```
3
1
3
1 0 0
0 2 0
2 0 1
1
3
0 0 1
0 2 1
2 0 0
3
5
0 0 1 0 2
0 2 0 2 0
1 0 1 0 1
0 2 0 2 0
0 0 1 0 0
```

Sample Output:

```
MAXIMUM SCORE: 10
THERE ARE 2 OPTIMAL BOARDS.

MAXIMUM SCORE: 40
THERE IS 1 OPTIMAL BOARD.
0 0 1
2 0 1
2 0 0

MAXIMUM SCORE: 70
THERE ARE 8 OPTIMAL BOARDS.
```

7. Grace

Program Name: Grace.java Input File: grace.dat

Grace is just learning to program, and is exploring how to do some simple math. The program she is solving inputs an integer, and outputs the square of it if the number is divisible by 3, the square root of it if the number has a remainder of 1 after being divided by 3, or the cube root of it if the remainder is 2, again after dividing by 3. She has decided to format all of the values to one rounded place of precision.

Your challenge is to write a program to help solve Grace's problem.

Input: Several integers N (0<N<=10000) arranged vertically in a data file.

Output: A value for each integer, as indicated above (square, square root, or cube root), as shown below.

Sample input:
```
9
10
11
```

Sample output:
```
81.0
3.2
2.2
```

8. Huang

Program Name: Huang.java Input File: huang.dat

Huang is in a science class studying how tree rings show the age of a tree. Fascinated with this notion, he decides to experiment with an exercise that will celebrate this cool aspect of trees by writing a spiral program using the characters in various tree names that are common to the world. Although he is aware that tree rings are NOT in a spiral configuration, he doesn't really care…it's close enough.

For example, bamboo is prevalent through many countries in the tropical regions, and is used quite often in constructing houses and other elaborate structures. He decides to use the letters in the word BAMBOO to make a 5X5 spiral that looks like this:

```
BAMBO
BOOBO
MO*AB
AOBMA
BOOBM
```

Notice how the letters go across the top, then down the right side, across the bottom, then up the left side, spiraling towards the middle until they stop when they reach the center. The last instance of the word BAMBOO may be partial and not fit exactly, but Huang is OK with that. It's just fun to see the patterns. He also decides to put a `'*'` in the very center of the grid to mark the center of the spiral pattern.

Input: Several names of trees, all in uppercase, no spaces or symbols, each followed by an odd positive integer N (2<N<20) indicating the size of the spiral to be created.

Output: Create and output an NXN spiral grid of letters using the name of the tree, with a '*' in the very center. Output a blank line after each grid.

Sample input:
```
BAMBOO 5
BLOODWOOD 7
CEDAR 9
```

Sample output:
```
BAMBO
BOOBO
MO*AB
AOBMA
BOOBM

BLOODWO
WOODBLO
DODWOOD
OOO*OOB
OLLBDDL
LBDOOWO
BDOOWDO

CEDARCEDA
EDARCEDAR
CCEDARCRC
RREDARECE
AACR*CDED
DDRADEADA
EEADECRAR
CCRADECRC
RADECRADE
```

9. Irina

Program Name: Irina.java Input File: irina.dat

Irina is in the middle of a bank robbery. She has very little time because the authorities are on the way. Around her are sacks of money containing various coin denominations. Each bag contains only one denomination, and is clearly labeled with that denomination. She already knows how much each coin weighs, and can do amazingly quick mental mathematics, but is still not as quick as a computer program. Also, as much as she has been working out, building her strength for this job, she can still only carry up to 45 kg of weight. Write a program to help her decide which bags she should take to maximize the profits of her bank heist!

The individual weights in grams of the different denominations of US coins are:

- Penny 2.5
- Nickel 5.0
- Dime 2.25
- Quarter 5.6
- Dollar coin 8.1

Note: 1 kg = 1000 g

Input: The first integer N will indicate the number of data sets to follow. Each of the following data sets will begin with an integer, I, representing the number of bags Irina has to choose from. The next I lines will be an integer, W, representing the weight of the bag in kg and a single word of {pennies, nickels, dimes, quarters, dollars} representing the denomination of coin in the bag.

Output: The string "**`GRAB THE <w> KG BAG OF <denomination>`**" for each of the bags that Irina should take that overall doesn't exceed her carrying capacity but maximizes her profits. All output should be sorted in decreasing order by weight of the bag, and then by decreasing denomination (dollars, quarters, dimes, nickels, pennies). Print an empty line between data sets.

Assumptions: All bags will be uniquely identifiable by their weight and coin denomination contained. There will be at least one bag for Irina to rob. All solutions will be unique.

Sample Input:

```
3
3
40 DOLLARS
15 DIMES
4 DOLLARS
10
15 DIMES
4 DOLLARS
300 PENNIES
7 DOLLARS
5 QUARTERS
45 QUARTERS
40 NICKELS
40 PENNIES
35 NICKELS
20 QUARTERS
5
20 QUARTERS
26 DIMES
11 NICKELS
11 DIMES
12 NICKELS
```

Sample Output:

```
GRAB THE 40 KG BAG OF DOLLARS
GRAB THE 4 KG BAG OF DOLLARS

GRAB THE 20 KG BAG OF QUARTERS
GRAB THE 7 KG BAG OF DOLLARS
GRAB THE 5 KG BAG OF QUARTERS
GRAB THE 4 KG BAG OF DOLLARS

GRAB THE 26 KG BAG OF DIMES
GRAB THE 11 KG BAG OF DIMES
```

10. Jorge

Program Name: Jorge.java   Input File: jorge.dat

Jorge has just learned to play poker, but has some trouble remembering the different hand rankings. He has decided to write a program to help, but needs your assistance in writing it.

He has decided to represent each card using a unique number from 1 to 52, with the diamonds numbered from 1-13, hearts from 14-26, spades 27-39, and clubs 40-52. Card #1 is the Ace of Diamonds, #2 the Two of Diamonds, #13 the King of Diamonds, #14 the Ace of Hearts, and so on. Card #52 is the King of Clubs.

His program will input the five numbers of his hand, and will tell him what his best hand is, according to the rules of poker, the rankings of which are, in order of best to worst:

- FOUR OF A KIND - all four of the same kind, like four 2s or four Kings.
- FULL HOUSE - three of one kind and two of another, like three 8s and two Aces.
- FLUSH - five cards, all of the same suit, like five spades or five diamonds, in no particular order.
- STRAIGHT - five cards, of at least two different suits, all in a row, like Ace, 2, 3, 4, 5, or 10, Jack, Queen, King, Ace, or any sequence somewhere in the middle of the suit. Wraparound sequences, like Queen, King, Ace, 2, 3, do not count as a straight.
- THREE OF A KIND - three of one kind, like three 7s or three Jacks.
- TWO PAIRS - 2 different pairs of the same kind, like two 5s and two 9s.
- PAIR - one pair of the same kind, like two 4s or two Kings.
- NONE - a terrible hand indeed...you should fold, immediately, unless you are a good bluffer!

Note: Neither a Straight Flush or Royal Flush will be considered for this problem.

Input: Several poker hands of five cards each, represented by five integers, all on one line, separated by single spaces, according to the description above.

Output: The best poker hand for each set of five cards, as shown in the description above and examples below.

Sample input:
```
18 44 7 21 23
18 44 31 22 38
18 44 31 34 21
18 44 31 5 9
```

Sample output:
```
PAIR
THREE OF A KIND
FULL HOUSE
FOUR OF A KIND
```

11. Kalyani

Program Name: Kalyani.java **Input File: kalyani.dat**

Kalyani loves numbers, and has been experimenting with an exercise with ranges of values. She uses an overall range of real number values from 0.0 to 9.9, using only the one-place precision values, i.e., 0.0, 0.1, 0.2..4.5, 4.6,...9.7, 9.8, 9.9. Each of those values is then mapped to another decimal value of any precision, for instance 2.365. She'll start with all of the values 0.0 through 9.9 mapping to 2.365, but then reassigns various ranges to different values. For instance, she could go from 1.3 up to but not including 4.5, where a new value is mapped to each number in that range, say 8.340. The data set that would indicate this range adjustment would be **`1.3 4.5 8.340`**.

A summary of those ranges would look like this:

```
0.0 1.2 2.365
1.3 4.4 8.340
4.5 9.9 2.365
```

Another range adjustment, using the values **`3.6 7.5 4.23`** (which means remap the values from 3.6 to 7.4 to the value 4.23), would yield a new summary:

```
0.0 1.2 2.365
1.3 3.5 8.340
3.6 7.4 4.23
7.5 9.9 2.365
```

A third adjustment, using the data line **`3.5 5.0 6.87`**, would be summarized as:

```
0.0 1.2 2.365
1.3 3.4 8.340
3.5 4.9 6.87
5.0 7.4 4.23
7.5 9.9 2.365
```

Input: An initial value N, followed by N sets of data. Each data set starts with a decimal value representing the initial value for the entire range, followed by an integer Q representing the number of adjustments to follow . Q sets of three values follow, all on one line, the first two indicating the range to be remapped, and the third the new value mapped to that range. All input values will be non-negative.

Output: The final summary of the value mappings, as shown in the example above and sample output below.

Sample input:

```
2
2.365
3
1.3 4.5 8.340
3.6 7.5 4.23
3.5 5.0 6.87
8.0
5
0.0 3.4 6.0
0.0 2.5 4.0
5.5 6.5 2.0
7.8 10.0 1.0
7.2 7.4 5.0
```

Sample output:

```
0.0 1.2 2.365
1.3 3.4 8.340
3.5 4.9 6.87
5.0 7.4 4.23
7.5 9.9 2.365

0.0 2.4 4.0
2.5 3.3 6.0
3.4 5.4 8.0
5.5 6.4 2.0
6.5 7.1 8.0
7.2 7.3 5.0
7.4 7.7 8.0
7.8 9.9 1.0
```

12. Lipun

Program Name: Lipun.java Input File: lipun.dat

Lipun has just learned some theory regarding binary search trees, specifically about depth, leaf nodes, internal and external nodes and path length. He needs some help from you writing a program that will build a binary search tree from some text data, and then report all of these new concepts. He wants to take any word, like INVITATIONAL, build a binary search tree in alpha order, allowing duplicate letters, which will be inserted to the left when the duplicate is encountered, and then calculate and report the following:

- Depth - distance from root to farthest leaf node
- Number of Leaf Nodes - number of nodes with no children
- Number of External Nodes - number of potential nodes (right or left child null pointers)
- Internal Path Length - sum of all distances from each internal node to the root
- External Path Length - sum of all distances from external nodes to the root

For example, the tree for the word INVITATIONAL is:

```
        I
      /   \
    I       N
   /       / \
  A       N   V
 / \     /   /
A   I   L   T
           /
          T
         /
        O
```

The expected output is: `5 4 13 29 53`

- The depth of this tree is **5**, since the letter "O" is 5 levels away from the root.
- There are **4** leaf nodes (nodes with no kids) - `A, I, L` and `O`.
- There are **13** external nodes, e.g., right or left child null pointers where a new node could be inserted. The letters I on level 1, N and V on level 2, and both Ts have one available null pointer for a new node, as well as all four leaf nodes each having two available null pointers, for a total of 13 external nodes.
- The internal path length is the sum of the depths of all internal nodes, the nodes actually in the tree. The nodes I and N are at level 1, and therefore are each 1 level away from the root, for an internal path length so far of 2. A, N and V are at level 2, for a total of 6, A, I, L and T are level 3, totaling 12, and then T and O add 4 and 5 to the sum, for a grand internal path length total of 2+6+12+4+5 = **29**.
- The external path length is calculated from the 13 external nodes, 1 at level 2 (right child pointer from the I), 2 each from level 3 (both right child pointers for N and V) for a total EPL of 6 so far, 7 each at level 4 (both pointers from A, I, and L, plus the right child pointer for T) for a level 4 EPL total of 28, plus a level 5 right child pointer from T, and two level 6 pointers from O. The total external path length is: 2+6+28+5+12 = **53**.

Input: Several words made up of only uppercase characters, each on one line, no word greater in length than 25.

Output: Five integers for each binary search tree resulting from the word, indicating tree depth, number of leaf nodes, number of external nodes, internal path length, and external path length.

Sample input:

```
INVITATIONAL
UIL
DISTRICT
```

Sample output:

```
5 4 13 29 53
2 1 4 3 9
4 4 9 16 32
```