Frisco FirstBytes Programming Contest

October 14, 2017

NOVICE PROBLEM SET

The final Novice winner is decided by the total number of Novice *and* Advanced problems solved.

Novice teams will receive credit for solving Advanced problems only if they solved ALL the Novice problems.

DO NOT OPEN THIS PACKET UNTIL TOLD TO DO SO

Problem I. **Taking the Middle Offer**

Have you ever noticed that on most shopping sites and stores you are always presented with 3 prices? Small, Medium, and Large drinks at Quick-Trip. Short, Tall, Grande, Grande, Vente and Trenta (but notice the multiple of three?) at Starbucks. Good, better, and best oil change options at Goodyear are just a some examples.

Well there is a reason for this; retail strategists (yes, they exist) have done studies and know that the majority of people choose the middle product! So guess what? The middle product has the highest profit margin.

Your job is going to be to find and print out that "middle choice" product, based on its price, and print it out from a list of those products for the new "know how marketers are trying to trick you" web site that you hope to sell basic, enhanced, and total access plans for.

Input: The first line of the input is an integer (n) that represents the number of product names and prices (3 of each) that follow. Each line contains 3 product choice labels (a, b, c) followed by the corresponding 3 (i, j, k) prices for each. Each product choice label is followed by a comma and space. Each price item is separated by a space.

Name of Data File: middle.txt

Output: Your program should produce n lines of output (one for each line of data in the file). Each line will show the product label and corresponding price.

Assumptions: Value for n will not exceed 1000.
The length of a, b, c will not exceed 50 characters.
The values for i, j, k will be between 0.00 and 1000.00
The values a, b, c or i, j, k on each line are NOT exclusive and may be duplicates.

Sample Input:
```
4
Small Drink, Medium Drink, Large Drink, .79 1.29 1.79
Short Mocha, Tall Mocha, Venti Mocha, 2.99 3.99 4.99
Good Oil, Better Oil, Best Oil, 24.95 29.95 34.95
Semi-Ok, Ok, Wow!,  79.95 79.95 92.95
```

Output:
```
Medium Drink 1.29
Tall Mocha 3.99
Better Oil 29.95
Ok 79.95
```

Problem J. **Finding Your Way, Old Style**

You've probably never had to find your way without the GPS on your phone, but it can be done! Luckily you kept that old Boy Scout compass that shows you the cardinal directions - N, NE, E, SE, S, SW, W, NW. Using just that and a map, you will get pointed in the right direction and get to your destination yet!

Input: The first line of the input is an integer (n) that represents the number of destinations you are trying to reach. Each line contains a pair of cardinal directions a, b separated by a single space. In the order listed above, each pair of consecutive cardinal directions is separated by 45 degrees, and the pair NW and N is also 45 degrees.

Name of Data File: direction.txt

Output: Your program should produce n lines of output (one for each line of data in the file). Each line should contain a single integer representing the smaller degree difference between the two cardinal directions a and b. If a and b are in opposite directions, output 180; if a and b are the same direction, output 0. Note that the order of input does not affect the output. (that is, an input of a b should give the same output as b a).

Assumptions: Value for n will not exceed 1000.
The cardinal directions a and b may be the same
The output value will be between 0 and 180, inclusive

Sample Input:
```
3
NW S
NE NE
E W
```

Output:
```
135
0
180
```

Problem K. **Simple Text Cypher**

So the school rules don't let you use your phone in class but darn it, you haven't checked in with your friends in the last 5 minutes. But hey, you're in Computer Science with a computer! Easy peasy to fire up SnipChit and fire off a missive, BUT you know the school is monitoring everything you do ;-((. AHAH! A simple text cypher is the way to go. You'll send two snippets and remove the overlapping portions between the pairs of words to generate a single word. They'll never figure this one out.

Input: The first line of the input is an integer (n) that represents the number of snippets you are sending that will make up your message. A snippet consists of two words each on a separate line. The first line of the snippet will contain string a. The second line of the snippet will contain string b. Both string a and b will contain lowercase characters between a and z, inclusive

Name of Data File: snipchit.txt

Output: Your program should produce n lines of output (one for each snippet in the file). Each line should contain a single string as output representing the joining of non-overlapping characters of the two strings a and b. The overlap is determined by the largest number of characters (m) such that the last m characters of string a exactly match the first m characters of string b

Assumptions: Value for n will not exceed 1000.
The length of string a and b as well as the output string will not exceed 100

Sample Input:
```
4
meat
atet
doggy
gy
ado
dot
joe
ys
```

Output:
```
meet
dog
at
joeys
```

Problem L. **Big and Little Integers**

So numbers are made from other numbers but rather than using the + to add numbers, you're going to use + as the concatenation operator to create both the smallest number and largest number from a list of numbers.

Input: The first line of the input is an integer (n) that represents the number of lines of integers you will "add" together. Each line will contain from 1 to n integers separated by a space. No integer will start with a 0, so 0124 is not a valid integer.

Name of Data File: intlist.txt

Output: Your program should produce n lines of output (one for each line of data in the file). Each line should contain two integers, each containing ALL the integers for that line. The first integer will be the SMALLEST integer you can create by concatenating all the integers and the second will be the LARGEST integer you can create the same way.

Assumptions: Value for n will not exceed 1000.
Each integer will be greater than 0 and less than 9999

Sample Input:
```
4
5 56 50
79 82 34 83 69
420 34 19 71 341
17 32 91 7 46
```

Output:
```
50556 56550
3469798283 8382796934
193413442071 714203434119
173246791 917463217
```

Problem M. **Miles to go before you sleep**

Well here you are again. Miles from home and you need to be there by midnight to make curfew and get in bed. Otherwise you'll be grounded yet again.

The good news—you know how far you are from home (17.74 miles) and that you need to be in the door at midnight (12:00:00). Speed is the key, but you already have two speeding tickets; one more and not only will you be grounded, no more car for the year! The question is, how much longer can you stay out, and how fast do you need to go (but still under the speed limit) to make it in time if you stay just a bit longer?

Input: The first line of the input is an integer (n) that represents the number of starting times you will leave to go home. Each line will contain 3 integers representing hours (h), minutes (m), and seconds (s).

Name of Data File: miles.txt

Output: Your program should produce n lines of output (one for each line of data in the file). Each line will show the time you want to leave and how fast you need to go in miles per hour and kilometers per hour rounded to the nearest whole number.

Assumptions: Value for n will not exceed 1000.
The value for h will be between 1 and 11 assumed to be 1 pm to 11 pm.
The values for m and s between 0 and 59.
1 mile = 1.60934 kilometers

Sample Input:
```
4
5 56 23
7 23 18
10 15 18
11 43 0
```

Output:
```
If you leave at 5 56 23 pm, you will need to go 3 mph or 5 km/h
If you leave at 7 23 18 pm, you will need to go 4 mph or 6 km/h
If you leave at 10 15 18 pm, you will need to go 10 mph or 16 km/h
If you leave at 11 43 0 pm, you will need to go 63 mph or 101 km/h
```

Problem N. **Super Hero Fighter!**

Time to fight! You are going to recreate a version of the text displayed in the classic 1987 version of the game that started it all. Except instead of Ken and Ryu you will use various superheroes and their unique moves.

Input: The first line of the input is an integer N that represents the number matches that you will narrate. Each match is represented by multiple lines. The first line of the match specifies the hero that strikes first, X, the second line the other hero, Y. The third line will contain a integer A representing the number of attacks player X makes. The next A lines will contain the attack player X makes each round. Following, there will be a single integer B listing the attacks known by player Y. The next B lines will contain the attacks player Y makes each round.

Name of Data File: heroFight.txt

Output: For each battle, first output “Match #b Begin!” where b is the number of the current match, starting at 1. Follow this by outputting the attack made by each player in order, alternating between the two antagonists starting with player X and output “X uses a!” where a is the corresponding attack from the file. Then output “Y uses b!” where b is corresponding attack for player Y.

When a player has run out of attacks, the opponent who has more attacks strikes one last blow. Output “p defeats q using h!” where p is the winning player, q is the losing player, and h is the last attack that won the match. Output a single blank line after each match.

Assumptions: Neither player will have more than 9 attacks per match.
Either player X or player Y will have fewer attacks than the other.

Sample Input:

```
2
Batman
James Bond
5
Cloaking Device
Remote-Controlled Batarang
Bat Beacon
Collapsible Bat Sword
Bat-Shark Repellent
4
Pen Gun
Grappling Suspenders
Radioactive Lint
Shark Gun
Curly
Moe
2
Nuk-Nuk-Nuk
Ear Bite
3
Nose Twist
Eye Poke
Head Slap
```

Output:

```
Match #1 Begin!
Batman uses Cloaking Device!
James Bond uses Pen Gun!
Batman uses Remote-Controlled Batarang!
James Bond uses Grappling Suspenders!
Batman uses Bat Beacon!
James Bond uses Radioactive Lint!
Batman uses Collapsible Bat Sword!
James Bond uses Shark Gun!
Batman uses Bat-Shark Repellent!
Batman defeats James Bond using Bat-Shark Repellent!

Match #2 Begin!
Curly uses Nuk-Nuk-Nuk!
Moe uses Nose Twist!
Curly uses Ear Bite!
Moe uses Eye Poke!
Moe uses Head Slap!
Moe defeats Curly using Head Slap!
```

Problem O. **Fishy Factory**

Congratulations! After a long 4 years you graduated from FFU (Flat Fish University) and got a job on the fish nugget line at the local factory. The amazing Flotsam Fish Nuggets (FFN) are made from the finest fish, fish parts and seaweed (don't ask) caught by local fisherman. Your job is to pick those parts out of the catch made that day that is coming down the factory line in front of you. An example of the line:

~~O<><O~#

The only items that appear on your line is seaweed (~), fish (<>< or ><>), rocks (O), and an empty net (#). Only whole fish appear and won't overlap.

You can calculate how many FFN can be made by how many useful parts there are in the ASCII representation above. You can see that 6 items are useful on this line - a whole fish can be made into 3 nuggets by itself and you also have 3 pieces of seaweed.

Input: The first line of the input is an integer (n) that represents the number of lines you will process that day. Each line will contain characters representing the items on the line.

Name of Data File: fishLine.txt

Output: For each line, output "Line # i will produce x Flotsam Fish Nuggets"

Assumptions: Value for n will not exceed 200
Each line will contain only the characters <, >, ~, O, #

Sample Input:
```
3
~~O<><O~#
~O~O~~~~~~~OOO~~O~O~
<><><>###
```

Output:
```
Line #1 will produce 6 Flotsam Fish Nuggets
Line #2 will produce 13 Flotsam Fish Nuggets
Line #3 will produce 6 Flotsam Fish Nuggets
```

Problem P. **Not All Who Wander are Lost**

Farmer Al is planting a plot of land. He has hired a city lad, Roy, who has a strange way of planting. Roy doesn't plant row by row, but prefers to wander. Al will pay Roy a bonus only if he visits each square once.

To track Roy, a grid of squares is laid out to represent the plot of land in rows and columns. Roy can move forward, backward, left or right, but not diagonally and never leaves the plot until he is finished.

The first time he visits a square it is numbered, starting from 1. He will visit every square at least once. If he visits a square more than once, he does not number that square again, but will continue to number the squares when he reaches an unvisited square.

You must determine if Al will pay Roy a bonus.

14	13	12	11
15	16	9	10
2	1	8	7
3	4	5	6

Bonus

3	2	12	11
4	1	8	9
5	6	7	10

No Bonus

Input: The first line of the input is an integer (n) that represents the number of test cases. On the first line of the test case, there are two integers representing rows (r) and columns (c). The following r lines contain c integers, which represent the order in which he first visited that square.

Name of Data File: wander.txt

Output: For each line, state either "Bonus" or "No Bonus" for each test case.

Sample Input:
```
2
4 4
14 13 12 11
15 16 9 10
2 1 8 7
3 4 5 6
3 4
3 2 12 11
4 1 8 9
5 6 7 10
```

Output:
```
Bonus
No Bonus
```

Problem Q. **Finding a Way Out**

You are one of the Time Bandits who just stole the Map of the Universe from the Supreme Being. The Map of the Universe shows all the portals the Supreme Being created for his convenience. But the map marks his other creations like the Black Planet of Doom, the Rock of Ages, the Golden Orb of Glory, and many other places. And you're only interested in the portals so you can use them to find the ***The Most Fabulous Object in the World.***

Your job, write a program that will identify all the portals on the map so the adventure can begin!

Map Section A

*	*	*	-	-
*	-	*	-	-
*	*	*	-	*
-	-	-	-	-
-	-	*	*	-

Map Section B

-	-	*	-	-
*	*	-	*	-
-	-	*	-	*
-	-	*	*	-
-	-	*	*	-

An asterisk (*) indicates an object or part of the ring around the portal. A portal will have a connected series of asterisks with a blank square in the middle. A blank space is indicated by a dash (-). The portal must have a dash in the center and must, at minimum, be surrounded by asterisks on the top, bottom, and sides; diagonals are not necessary.

Map section A has 1 portal and map section B has 2 portals. The shaded squares above represent the centers of portals.

Input: The first line of input will be an integer n, indicating the number of map sections. The first line of the map section data will have 2 integers $0 < r < 1024$ and $0 < c < 1024$, separated by a space indicating the number rows and columns in that section. The next r lines will contain c characters which are either asterisks or dashes.

Name of Data File: map.txt

Output: For each map section output "x Portals Found" where x is the number of portals found. If none are found, output "No Portals Found".

Sample Input:

```
3
5 5
- - * - -
- * - * -
- * * - -
- - - - -
- - * * *
5 5
- - * - -
* * - * -
- - * - *
- - * * -
- - * * -
5 5
- * - - *
* * * - -
- * * - -
- - - * -
* - - * -
```

Output:

```
1 Portals Found
2 Portals Found
No Portals Found
```