October 29, 2016

Taylor High School

Hands-On Contest

Problems

Number	Title
1	Frog
2	Creepy Crawlies
3	Elements of Madness
4	Boo's Deathday Bash
5	Loopanthrope
6	Ghoulgreens
7	Hallow Map
8	Spooky Names
9	Currency Conversion
10	Haunted Zoo Enclosures
11	Discount
12	Janitor

1. Frog

Program Name: Frog.java **Input File: N/A**

Info: You had a pet frog named Pepe who was unfortunately used in a witch's stew. So, you have decided to commemorate his passing by creating an ASCII replica of him.

Input: None

Output: An exact replication of the ASCII frog shown below in the sample output. **Note that the eyes are zeroes.**

Sample Input:
None

Sample Output:

```
**********.--._.--.**********
**********( 0     0 )**********
**********/    . .   \**********
*********.`._______.`.*********
********/(           )\********
******_/  \   \   /  /  \_******
***.~   `  \  \ /  /  `   ~.***
**{    -.   \  V  /   .-    }**
.__\-------^/__|__\^-------/__.
```

2. Creepy Crawlies

Program Name: Crawlies.java **Input File: crawlies.in**

Info: Amenhotep the IV died in 2200 B.C. Unfortunately, his subjects forgot to bury him with any friends, which means that he has a **LOT** of free time. Recently he has been doing some spring cleaning and decided to sort his priceless collection of scarabs by color. He knows that a simple RGB sort results in a nasty color distribution, so he opts instead for the more aesthetically pleasing alternate, sorting by HSV. He knows that if he has the RGB of each color, he can calculate the HSV using the following formula:

$$max = \max(R, G, B)$$
$$min = \min(R, G, B)$$
$$\Delta = max - min$$
$$H = \begin{cases} 60 * \left(\frac{G-B}{\Delta}\right) mod 6, & max = R \\ 60 * \left(\frac{B-R}{\Delta}\right) + 2, & max = G \\ 60 * \left(\frac{R-G}{\Delta}\right) + 4, & max = B \end{cases}$$
$$S = \begin{cases} 0, & max = 0 \\ \frac{\Delta}{max}, & max \neq 0 \end{cases}$$
$$V = \{max\}$$

Then, to sort, he just compares the Hue, then the Saturation, and finally the Value. He wishes to arrange his scarabs increasingly.

Input: The first line will contain a number *n*, which is the number of test cases. Each test case is composed of a line containing a single number *k,* followed by *k* lines each containing three integers from 0-255 separated by a space. You will get no gray values for this problem.

Output: For each input, output the RGB values of the colors in the format R G B sorted by their HSV values in increasing order. The end of each output should be a blank line.

Sample Input:

```
2
5
122 33 44
49 255 77
188 36 115
255 0 0
200 68 117
9
134 120 161
67 193 144
```

```
221 80 240
128 72 122
166 105 144
118 49 168
86 186 78
56 56 76
183 105 59
```

Sample Output:

```
188 36 115
200 68 117
122 33 44
255 0 0
49 255 77

128 72 122
166 105 144
183 105 59
86 186 78
67 193 144
56 56 76
134 120 161
118 49 168
221 80 240
```

3. Element of Madness

Program Name: Elements.java **Input File: elements.in**

Info: Rick the mad scientist wants to study the chemical composition of words. However, words aren't made up of atoms, so he has to put them into his Multiverse String Theory Defibrillator to convert it into its elemental equivalent. The MSTD takes in a word, and results in the same word written as a sequence of element symbols.

Input: The first few lines will contain every single element you will need, each separated by a space. The next line will contain a number *n,* where *n* is the number of Strings you need to turn into elements. Every single line of input can be perfectly represented as a collection of elements with the same length as the original and same characters, but different capitalization.

Output: Output each line as a properly capitalized representation made up of elements. If a word has multiple representations, print all of them on the same line separated by a space and ordered increasingly by their ascii value.

Sample Input:

```
H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar K Ca Sc Ti V Cr Mn Fe Co
Ni Cu Zn Ga Ge As Se Br Kr Rb Sr Y Zr Nb Mo Tc Ru Rh Pd Ag Cd In Sn Sb
Te I Xe Am
5
Neon
Park
Nap
Harambe
Nascar
```

Sample Output:

```
NeON
PArK
NaP
HArAmBe
NAsCAr NaSCAr NaScAr
```

4. Boo's Deathday Bash

Program Name: Deathday.java **Input File: deathday.in**

Info: Boo the big fat friendly ghost is hosting his very special 419th death-day bash. He's been dead for 419 years! He wants to know who else shares his deathday so he uses the following formula which he found on SkullOverflow that gives the percent of spirits expected to share the same deathday in a graveyard full of *n* spirits

$$1 - \frac{(365)!}{(365-n)!\,(365)^n}$$

Input: The first line contains a number *n*, the number of test cases. The following *n* lines contain a single number *k*, where $0 \leq k \leq 10{,}000$ and *k* represents the number of spirits in the graveyard.

Output: Output the chance that at least 2 animals have the same birthday to two decimal places following the format.

Sample Input:

```
10
1
4
5
7
19
22
23
25
50
77
```

Sample Output:

```
0.00%
1.64%
2.71%
5.62%
37.91%
47.57%
50.73%
56.87%
97.04%
99.98%
```

5. Loopanthrope

Program Name: Loopanthrope.java **Input File: loopanthrope.in**

Info: Dave the werewolf loves to code. However, the fact that he turns into a wolf with a savage mentality means that his computers don't have a long lifespan. To save some cash, Dave has bought a low-budget, reinforced laptop with Spooktel HD Graphics. Thankfully, Dave heard of a technique called loop unrolling which could help optimize his code. Loop unrolling can consist of taking a for-loop with a fixed length, and turning it into a series of commands without any logic checks. For example, the following loop:

```
for(int i=0;i<3;i++){
      add(i);
}
```

Can be rewritten as:

```
add(0);
add(1);
add(2);
```

While the second option takes up more space, it has a better performance.

Input: The first line contains a number *n*, the number of test cases. Each test case is composed of a single number *k*, which is the number of lines in the test case. The first line will contain the first line of a for-loop in a predetermined syntax. You can assume that the variable being initialized is always an integer, the condition will only include (>,<,>=,<=), and the incrementation will always be by one. You can also assume that any time the variable is being used in an operation, it will be surrounded by a single space on each side.

Output: Output a comment saying "Unrolled", followed by the unrolled loop and a blank line at the end.

Sample Input:

```
3
3
for(int k=0;k<3;k++){
    list.get(k).move(k);
}
4
for(int j=40;j<50;j++){
    move();
    attack(j);
}
5
for(int zed=0;zed<5;zed++){
    move( zed -2);
    rotate( zed *3);//More input on next page
```

```
    clean();
}
```

Sample Output:

```
//Unrolled
list.get(0).move(0);
list.get(1).move(1);
list.get(2).move(2);

//Unrolled
move();
attack(40);
move();
attack(41);
move();
attack(42);
move();
attack(43);
move();
attack(44);
move();
attack(45);
move();
attack(46);
move();
attack(47);
move();
attack(48);
move();
attack(49);

//Unrolled
move(0-2);
rotate(0*3);
clean();
move(1-2);
rotate(1*3);
clean();
move(2-2);
rotate(2*3);
clean();
move(3-2);
rotate(3*3);
clean();
move(4-2);
rotate(4*3);
clean();
```

6. Ghoulgreens

Program Name: Ghoulgreens.java **Input File: ghoulgreens.in**

Info: Mr. See finds himself stuck at his convenience store job late at night on Halloween. The demand is through the roof, so he wants to be as efficient as possible in the coins he gives out. He wishes to find the minimum number of coins he would need to make change. If he had coins of 1, 3, 5, and 6 units, the customer paid 50, and the price was 39, the change would be 11. He would need a 1 (5) coin and 1 (6) coin to make 11, so the minimum number is 2.

Input: The first line is the number of test cases. Each test case is two lines. The first line contains the denominations of the coins you have available, and the second line contains two integers: what the customer paid, and what the item was worth. The customer will always pay more than what the item was worth.

Output: Output the minimum number of coins you need to make change with the coins you have available.

Sample Input:

```
4
1 3 5 6
50 39
1 10 20
54 50
5 10 25
75 45
1 10 2
114 99
```

Sample Output:

```
2
4
2
4
```

7. Hallow Map

Program Name: HallowMap.java **Input File: hallowmap.in**

Info: Tommy is about to go trick or treating! But like any mom who loves their child to excess, his mom created a map of the neighborhood for him to guide around the neighborhood. Along with the map comes with markings that represent which houses he can and can't go to for candy. However, when Tommy looks at the map, he finds it a little odd because it doesn't seem to match the neighborhood around him. Help him write a program to decipher if the map is just turned a multiple of 90 degrees or if the map is completely wrong.

Input: The first line of the input file contains the integer, *n*, giving the number of data sets following. Each dataset contains an integer, *d*, on the first line followed by multiple lines of groups of 1s and 0s. The first *d* lines are the correct *d* by *d* map of the neighborhood with a 1 and 0 representing a house, 1 meaning Tommy can go to that house for candy and 0 meaning he can't go to that house for candy. The second *d* lines contain the map that his mom drew that might be inaccurate.

Output: If the map mom drew is turned a multiple of 90 degrees in reference to the correct map, the map is correct and thus forth you print out "Happy Trick or Treating!" If it doesn't match the correct map no matter where you turn it, print out "Ask Mom for another Map."

Sample Input:

```
3
4
1001
0110
1001
0110
0101
1010
1010
0101
5
10001
00110
00100
11100
00100
00100
11100
00100
11111
01010
5
11001
00111
11110
00101
```

```
10001
11011
01100
01110
10100
10101
```

Sample Output:

```
Happy Trick or Treating!
Ask Mom for another Map
Happy Trick or Treating!
```

8. Spooky Names

Program Name: SpookyNames.java **Input File: spookynames.in**

Info: Tommy finally got the correct map from his mom and is about to go trick or treating, but his paranoid mother suddenly instructs Tommy to avoid certain families. Tommy's mom gave him a list of the last names of families in their neighborhoods. Your job is to print out her response to each of the families based on the instructions outlined in the output so that Tommy can finally show off his super cool Bob the Builder costume.

Input: The first line of the input file contains an integer that says how many lines are to follow. Each line contains the last name of the family in the neighborhood.

Output: If the last name starts with J, K, S, or X, they are considered "bad" letters to Jake's mom so you should print out her response with "Stay away from the " + last name of the family + " family." If the last name starts with B, H, L, or R, they are considered "okay" letters to Jake's mom so you should print out her response with "Don't ask The" + family last name + "family for too much candy." And for all other letters print out "Tell The" + family last name + "I said hi!" Make sure the responses have periods at the end when the last names are "bad" or "okay" letters.

Sample Input:

```
4
Cunningham
Elliot
Rodham
Stanley
```

Sample Output:

```
Tell The Cunningham family I said hi!
Tell The Elliot family I said hi!
Don't ask the Rodham family for too much candy.
Stay away from the Stanley family.
```

9. Currency Conversion

Program Name: Conversion.java **Input File: conversion.in**

Info: Styx, the keeper of the river between Earth and the Underworld, requires a certain currency in order to transport you to the Underworld. However, you realize that you accidentally died with the wrong kind of currency in your pockets and you really don't feel like swimming. Fortunately, you spot a currency exchange nearby.

Input: The first given line is a sequence of currencies each followed by the USD to that currency's ratio. The following line is a number, *n*, with *n* lines following it each with a currency value, the currency, and the target currency, all separated by a space.

Output: The current currency's value and the currency and the equivalent conversion and currency. The target conversion must be to two decimal places.

Sample Input:

```
usd 1.00 euro 0.89 bp 0.76 yen 102.42 ad 1.34 cd 1.32
3
3.00 usd euro
5.00 euro cd
4.25 bp yen
```

Sample Output:

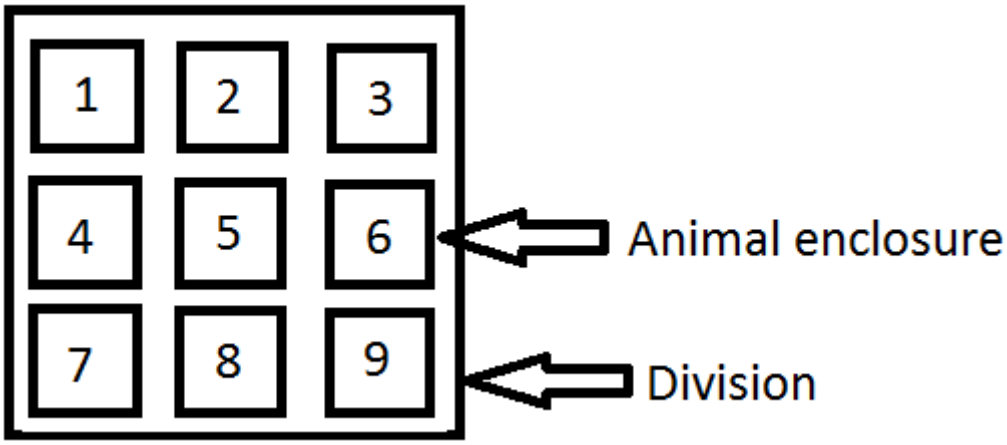
```
3.00 usd = 2.67 euro
5.00 euro = 7.42 cd
4.25  p = 572.74 yen
```

10. Haunted Zoo Enclosures

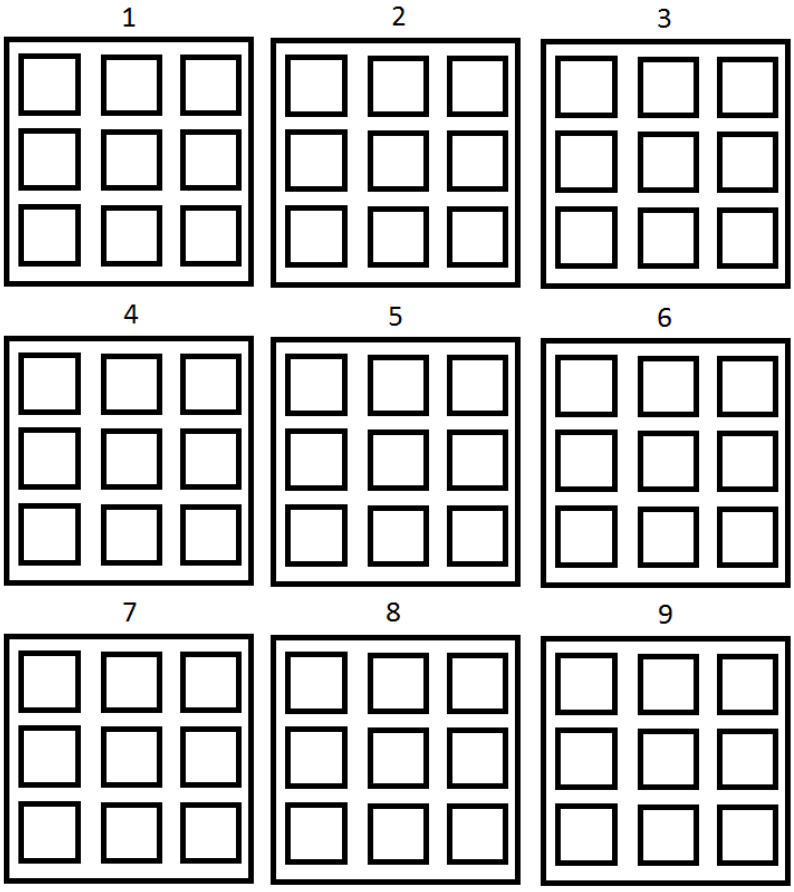
Program Name: Enclosures.java **Input File: enclosures.in**

Info: There is a haunted zoo filled with the unfriendly spirits of dead animals, one being Haramboo. In this zoo there are 9 divisions each with 9 animal enclosures. Each animal enclosure is haunted by a certain animal spirit.

Below is an example division with its enclosures and the order in which the enclosures will be given in the input data.



The full layout and the order in which the divisions will be given in the input data.

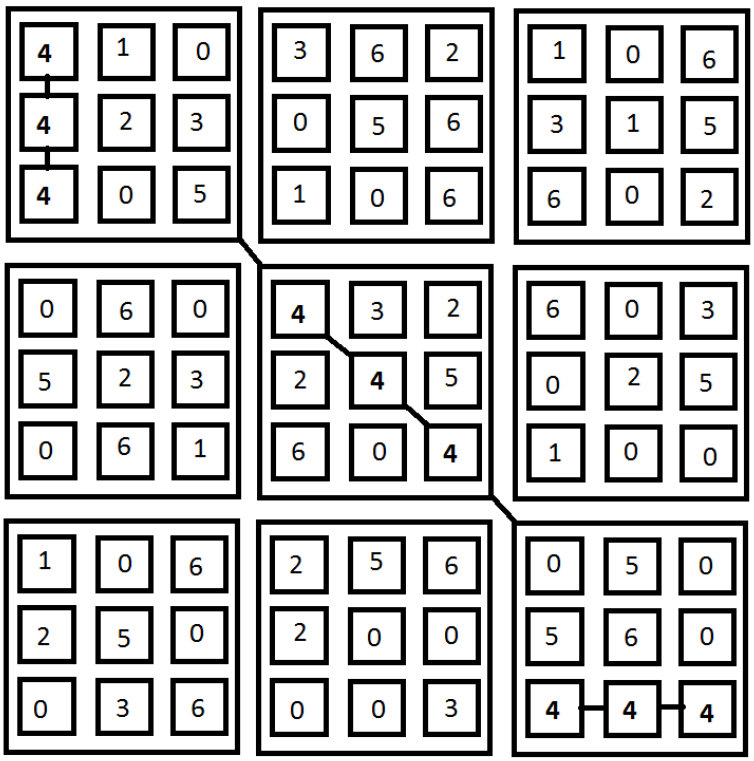


A zoo administrator needs to find out if there is an existing line of divisions that share the same type of dominant animal enclosure, and if there is, he must report the dominant type of enclosure of the line as well as its alignment (horizontal, vertical, or diagonal). He has asked you to analyze the schematics of the zoo's divisions and determine just that for him. A division has a dominant type of enclosure if there is an existing line of enclosures that have the same type, similar to a winning line of tokens in a game of tic-tac-toe. It is safe to assume that a division will never have more than 1 dominant type of enclosure; however, it should be noted that there may not be a dominant type of enclosure at all. A line of divisions is simply a consecutive series of divisions that all have the same type of dominant enclosure.

Type of Enclosure	Representative Number
Empty	0
Bird	1
Tiger	2
Bear	3
Gorilla	4
Reptile	5
Aquarium	6

Note that the type of enclosure "empty" is only used as a placeholder and should never be reported as a dominant type of enclosure.

A diagonal line of divisions with a dominant enclosure type of gorillas



Input: The first line will contain a number, *n*, denoting the number of following test cases. Each test case will be a series of 9 sequences (divisions) with each sequence containing 9 numbers (animal enclosures) ranging from 0 to 6 inclusive.

Output: For each test case, the dominant type of enclosure in a line of divisions and the alignment of the line separated by a single space. Should there not be an existing line of divisions with a defined dominant enclosure, simply print "None" without the quotes.

Sample Input:

```
6
000000000 000000000 000000000 000000000 000000000 000000000 000000000
000000000 000000000 //A single line
014234554 026605010 144403524 412456400 555005053 220634031 204140430
412304511 013326426
026065632 165560260 615265656 042320445 256513314 425400260 254244442
664033216 513250613
222001000 105254105 136015630 365420033 550222001 420351201 432066032
612241434 063201222
102111266 413424410 046445121 025111056 463635430 051056006 210111430
036346266 221242200
344256454 061632305 432235332 535624306 325233544 141014241 345551166
125214462 452644152
```

Sample Output:

```
None
Gorilla Vertical
Aquarium Horizontal
Tiger Diagonal
Bird Vertical
None
```

11. Discount

Program Name: Discount.java **Input File: discount.in**

Info: Mike is buying candy for Halloween. The store has discounts on all the types of candy there are. Mike needs you to write a program to determine which type is cheaper to buy including discounts and then give the amount he saves on choosing one over the other. If they come out the same, just buy the first type.

Input: The first line of input will consist of an integer, n, indicating the number of data sets and/or lines to follow. Each data set consists of Brand A name (String), original price of Type A (double), Type A's discount in percent form, a comma, and the same things for Type B.

Output: For each input data set, you will print out "Buy ", the name of the brand calculated to cost less, a comma, and "Saves: $ " plus the amount saved from choosing that brand. The cost should be to two decimal places. If the brand cost with discounts on both Type A and B are equal, just print "Buy (Type A), Paid the Same."

Sample Input:

```
6
Hershey's $2.46 20%, Snickers $3.32 45%
Skittles $1.99 20%, M&Ms $2.05 10%
AirHeads $12.50 84%, SourPatchKids $6.25 32%
DumDums $4.57 43%, HariboGummyBears $5.01 50%
ButterFingers $5.19 70%, Twizzlers $5.10 75%
JollyRanchers $20.00 70%, Good&Plenty $10.00 40%
```

Sample Output:

```
Buy Snickers, Saves: $0.14
Buy Skittles, Saves: $0.25
Buy AirHeads, Saves: $2.25
Buy HariboGummyBears, Saves: $0.10
Buy Twizzlers, Saves: $0.28
Buy JollyRanchers, Paid the Same
```

12. Janitor

Program Name: Janitor.java **Input File: janitor.in**

Info: A ghost haunts a local high school and wants to spook the janitor who works there early in the morning. Help the ghost determine where the janitor is based on his weekly schedule so that he doesn't have to waste energy moving through walls.

Monday Schedule

Time	Class
1:00 AM – 1:45 AM	Algebra II
1:46 AM – 2:31 AM	English IV
2:32 AM – 3:17 AM	Physical Education
3:18 AM – 4:03 AM	Calculus BC
4:04 AM – 4:49 AM	Computer Science I
4:50 AM – 5:35 AM	Chemistry

Tuesday Schedule

Time	Class
1:00 AM – 1:45 AM	Precalculus
1:46 AM – 2:31 AM	Geometry
2:32 AM – 3:17 AM	Algebra I
3:18 AM – 4:03 AM	Calculus AB
4:04 AM – 4:49 AM	Spanish I
4:50 AM – 5:35 AM	Spanish II

Thursday Schedule

Time	Class
1:00 AM – 1:45 AM	English III
1:46 AM – 2:31 AM	Government
2:32 AM – 3:17 AM	Economics
3:18 AM – 4:03 AM	Speech
4:04 AM – 4:49 AM	Health
4:50 AM – 5:35 AM	Computer Science II

Input: The first line is a number, *n*, which signifies that there are *n* test cases following it. Each test case is composed of a character(s) which represents the day (M, T, W, Th, F), a time, and the meridiem (AM or PM), all separated by a space.

Output: The class the janitor is currently cleaning. Should he not be in the school at the given time, simply print "Not Present".

Sample Input:

```
5
M 2:35 AM
T 4:30 AM
Th 4:13 AM
F 5:27 PM
Th 3:39 PM
```

Sample Output:

```
Physical Education
Spanish I
Health
Not Present
Not Present
```