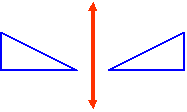
**Java program:** Prob09.java

**Input File:** Prob09.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

Reflections – you deal with them on a daily basis when you look in the mirror or drive a car. In your math classes, you may have dealt with functions that get reflected. Today, we will be dealing with pictures that will be reflected (sort of).

You will be given a picture built out of ASCII characters, and you will be asked to build the reflection of the picture you are given in one of three ways:

* Around the x axis: for this type of reflection, you will flip the picture up and down. Be careful not to add extra spaces at the ends of lines during this type of reflection.
* Around the y axis: for this type of reflection, you will flip the picture left and right. Since the starting picture will be left justified, your reflected picture should be right justified.
* Around the line y=x: for this type of reflection, the x values and y values are switched. This type of reflection is how you find the inverse of a function in mathematics. For our purposes, the origin will be the top left of the picture.

**Program Input**

The first line of the file Prob09.in.txt will contain a positive integer T denoting the number of test cases that follow. Each test case will have the following input:

* The first line of each test case will be a positive integer N denoting how many lines tall the picture is.
* The next N lines of each test case will contain the picture. The lines might vary in length, so you will need to take this into account when performing the reflection.
* The last line of each test case will contain one of these three strings:
  + X – if you should reflect the picture up and down
  + Y – if you should reflect the picture left and right
  + INVERSE – if you should switch the x and y values

**Example Input:**

3

19

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

!\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/!\

!! !! \

!! !! \

!! !! !

!! !! !

!! !! !

!! !! !

!! !! !

!! !! /

!!\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_!! /

!/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\!/

\_\_\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_/!\_

!\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_!/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/oooooooooooooooo /!

/ooooooooooooooooooooooo/ /

/ooooooooooooooooooooooo/ /

/C=\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_/

X

6

\_\_\_\_\_ \_ \_\_\_\_ \_ \_\_\_\_\_ \_ \_

/ \_\_\_\_| | | / \_\_ \ | | | \_\_ \ | | | |

| | \_\_\_ \_\_| | \_\_\_ | | | |\_ \_ \_\_\_ \_\_\_| |\_ | |\_\_) |\_\_\_ \_\_\_| | \_\_\_\_\_| |

| | / \_ \ / \_` |/ \_ \ | | | | | | |/ \_ \/ \_\_| \_\_| | \_ // \_ \ / \_\_| |/ / \_\_| |

| |\_\_\_| (\_) | (\_| | \_\_/ | |\_\_| | |\_| | \_\_/\\_\_ \ |\_ | | \ \ (\_) | (\_\_| <\\_\_ \\_|

\\_\_\_\_\_\\_\_\_/ \\_\_,\_|\\_\_\_| \\_\_\_\\_\\\_\_,\_|\\_\_\_||\_\_\_/\\_\_| |\_| \\_\\_\_\_/ \\_\_\_|\_|\\_\\_\_\_(\_)

Y

8

\_>-. \_\_\_\_ .-<\_

>-.\.-'\_\_\_\_`-./ ,<

.','"\\_\_/"`.`<\_,-,

\_/ / \\_/ \\_/ \! (D\

\ \\_/ \\_\_/ \\_/!\_ (D/

`.`../\_\_\,.'.< `-`

\_>-'/`-.\_\_\_\_.-'\ `<\_

>-' `-<

INVERSE

**Program Output**

Your program should output the appropriate picture after the specified reflection.

**Example Output:**

/C=\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_/

/ooooooooooooooooooooooo/ /

/ooooooooooooooooooooooo/ /

/oooooooooooooooo /!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

!\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_!/

\_\_\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_/!\_

!/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\!/

!!\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_!! /

!! !! /

!! !! !

!! !! !

!! !! !

!! !! !

!! !! !

!! !! \

!! !! \

!\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/!\

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_ \_ \_\_\_\_\_ \_ \_\_\_\_ \_ \_\_\_\_\_

| | | | \ \_\_ | | | \ \_\_ / | | |\_\_\_\_ /

| |\_\_\_\_\_ | |\_\_\_ \_\_\_| )\_\_| | \_| |\_\_\_ \_\_\_ \_ \_| | | | \_\_\_ | |\_\_ \_\_\_ | |

| |\_\_ / /| |\_\_ / \ \_ // \_ | |\_\_ |\_\_ /\ \_ /| | | | | | | \ \_ /| `\_ / \ \_ / | |

|\_\ \_\_\< |\_\_( | )\_( \ \ | | \_| \ \_\_\/\_\_ | |\_| | |\_\_| | /\_\_ | |\_( | )\_( |\_\_\_| |

)\_(\_\_\_\\_\|\_|\_\_\_\ /\_\_\_\\_\ |\_| |\_\_\/\_\_\_||\_\_\_\|\_,\_\_\\\_\\_\_\_\ |\_\_\_\|\_,\_\_\ /\_\_\_\\_\_\_\_\_\

\_ \_

>> \_ >>

-- /\ --

... `''

\'/\./

., \_``

-'\/.-

'"\_ ..

\_\_\/\/\_

\_\_\_ \_\_\_

\_\_\_ \_\_\_

\_\_/\/\\_

`"\_ ,.

-`/\.-

.. \_''

/`\/.\

. <!!< `

-,\_ \_ `-

<<, `<<

\_ -((-\_

,DD`

\/

**Java program:** Prob10.java

**Input File:** Prob10.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

Tennis scoring is similar to other games, but it has been cleverly encoded to sound strange to non-tennis fans. Today, you will break this code and write a program to keep track of the score of a tennis game.

**Game, Set, Match**

When two players play each other in tennis, they are playing a tennis match. The overall objective is to win the match. A tennis match is made up of sets, and a set is made up of games. We will not concern ourselves with the match or set scoring today – only a single game.

A tennis game is made up of points. A player wins a tennis game by being the first player to win four points, but you must win by two. There are no tie-breakers in games.

**What’s love got to do with it?**

So far, tennis sounds easy, right? The clever encoding of tennis scores comes at the point level. The following table describes the different names for tennis points:

|  |  |
| --- | --- |
| Number of Points Won | Name of Score |
| 0 | love |
| 1 | 15 |
| 2 | 30 |
| 3 | 40 |

Table 1: Tennis point names

Tennis scores are usually separated by a dash with the server’s score first, and in our games player 1 will always be serving. Some example scores are 15-30 and 40-love.

When the score is tied, tennis does things a little differently. Tie scores of 15-15 and 30-30 are called “15-all” and “30-all” respectively. After this point, when the score is tied at 40-40 or beyond, the score is referred to as “deuce”.

Once a deuce situation is encountered, the score is called out according to which player has the “advantage”, meaning the player that needs to win the next point to win the game. So, after a deuce the only two possible scores would be “Advantage Player 1” or “Advantage Player 2”. Depending on who wins the point after that, either the game is over or the score is tied at deuce again.

**Program Input**

The file Prob10.in.txt will contain an unknown number of lines. Each line will either contain a 1 or a 2, signifying which player won the point. Hint: if you’re using a BufferedReader to get your input (like the Hello World problem does), the readLine method will return null if the end of a file is encountered. So, a statement like while ((inLine = br.readLine()) != null) { might be useful to you.

**Example Input:**

1

1

2

1

1

1

2

1

2

1

2

1

2

2

2

**Program Output**

Your program should print out the score of each game as it progresses. At the beginning of each game, you should print the text “Game start”, and when a game is won you should print “Game Player x”, where x is the number of the player that won. Your program should play as many games as it can until the input runs out.

**Example Output:**

Game start

15-love

30-love

30-15

40-15

Game Player 1

Game start

15-love

15-all

30-15

30-all

40-30

deuce

Advantage Player 1

deuce

Advantage Player 2

Game Player 2

**Java program:** Prob11.java

**Input File:** Prob11.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

Soundex is a phonetic hashing algorithm that groups together names that sound similar yet have minor differences in spelling. It can be useful for genealogical studies by identifying variations for a given surname. The hashing part of the algorithm uses the following character groups:

* Group 1: b, f, p, v
* Group 2: c, g, j, k, q, s, x, z
* Group 3: d, t
* Group 4: l
* Group 5: m, n
* Group 6: r
* Wild: h, w
* Vowels: a, e, i, o, u, y

An American Soundex code for a name consists of a letter followed by a three digit number. The code can be determined by the following steps:

1. Find the first letter in the name that belongs to one of the numbered groups. Starting from that letter and working left to right, if two or more letters from the same numbered group are adjacent to one another, remove all but the first letter. Note that h and w are “wild” meaning that they will match letters from any group 1-6.
2. Retain the first letter of the name and remove all vowels and wild letters.
3. Retain the first letter of the name and replace all other letters with their group number.
4. If there are less than three numbers, add zeroes until there are three. If there are more than three numbers, just keep the first three. Make sure the letter at the beginning is capitalized.

Examples:

Ashcroft -->Asroft -->Asrft --> A2613 --> A261

Pfister -->Pister -->Pstr --> P236

Williams -->Wiliams -->Wlms --> W452

**Program Input**

The first line of the file Prob11.in.txt will contain a positive integer T denoting the number of test cases that follow. Each test case will have the following input:

* The first line of each test case will contain a positive integer N denoting the number of names that follow.
* The next N lines will contain one name per line.

**Example Input:**

2

3

williams

ashcroft

pfister

10

Gary

Clare

Jane

Gore

Geier

June

Claire

George

John

Jenny

**Program Output**

Your program should print out the list of Soundex codes generated by the list of names (ordered alphabetically) and the number of times that code was generated separated by a space for each test case. The first line of each test case’s output should be the word OUTPUT.

**Example Output:**

OUTPUT

A261 1

P236 1

W452 1

OUTPUT

C460 2

G600 3

G620 1

J500 4