PSCSTA Programming Competition
December 13th, 2014

ADVANCED DIVISION

- Complete the problems in any order.
- All problems have a value of 60 points.
- Your program should not print extraneous output. Follow the format exactly as given in the problem.
- There is a 5-point penalty for each incorrect submission for problems that are eventually judged correct.
- Time will be used to break point ties.

Problems	**Points**	**Notes**
Gorf		
Word Find		
Pokey		
Pac-Dude		
Last One Standing		
Clock-Paper-Scissors		
Sammy Says		
Game Reviews		
Nomopoly		
Bopscotch		
Chaseball		
Invaders		
Chest		
Card Sort		
Scavenger Hunt		

Good luck!

All problems in this packet were created by Frances Davies.

Gorf

Input file: gorf.dat

In Gorf, players hit a small, white ball with a Gorf club. The white ball flies through the air towards a hole in the ground. The Gorf player's goal is to hit the ball into the hole. You are to use the quadratic formula (shown below) to predict how far away from the player the ball will land.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The starting location will be given by using the equation above with - before the square root. The location the ball will land is given by using the equation above with + before the square root. To find the distance, you must subtract the starting location from the landing location.

Input:

The first line will contain the number of lines to follow. Each line will contain a, b, and c from the quadratic equation. You do not need to check the validity of the input. They will be separated by single spaces.

Output:

For each set of a, b, and c, output the ball's distance traveled rounded to the nearest tenth. Each output should be on its own line.

Example Input:

```
3
-1 4 0
-1 10 -11
-0.08 5 -50
```

Example Output:

```
4.0
7.5
37.5
```

Word Find

Input file: wordfind.dat

A Word Find is a puzzle that is made up of a 10x10 square of letters. Hidden in rows and columns of the puzzle are words. In this simplified version of the game, the words will only appear horizontally and spelled left to right, or vertically and spelled in a downwards direction. Write a program that finds a certain set of words in the grid.

Input:

The first line will contain the total number of words to be found in the grid. The following lines will each contain one of those words. Finally, the last 10 lines will each contain 10 letters separated by single spaces (these represent the 10x10 grid).

Output:

For each word found in the grid, output the x and y coordinates (separated by a single space) of the first letter of each word in the square. The order of the coordinates must match the order of words presented in the input. (Hint: you will have to print a coordinate more than once if multiple words start from the same letter.) Each output should be on its own line.

Example Input:

```
3
array
dawg
graph
a r r a y a s d f g
s p o i u y t r e w
t e f v r g b t h n
r t r f g b v o o k
i r t y d f g h d p
n e r e c f t g a i
g g r a p h f t w d
j d i c g f o k g n
a s d f g h j k l p
e r t y u i o p w q
```

Example Output:

```
0 0
8 4
1 6
```

Pokey

Input file: pokey.dat

In the game of Pokey, players draw 5 cards and place bets based on their hands. Write a program that prints the best hand you can make. Here are the types of hands that exist, in order from worst hand to best hand:

Zilch - No cards are the same and all five cards are not in sequence, meaning there is no Straight.
Pair - Two cards are the same.
Triple - Three or four cards are the same.
Straight - All five cards are in sequence. For example: 8 9 10 J Q.
Full House - Three cards are the same, and the other two cards are the same

Cards do not loop, meaning K A 2 3 4 is not a straight. A is high.

Input:
The first line will contain the number of lines of input to follow. The following lines each contain a hand of 5 cards. Each card will be separated by a single space. A card is represented by a number 2 to 10, jack, queen, king, or ace (represented by J, Q, K, or A, respectively).

Output:
Output the best type of hand the five cards make up. For example, in every triple there will also be a pair. However, "triple" should be printed because it is a better hand. Each hand should be on its own line.

Example Input:
```
4
8 2 6 6 7
7 8 9 10 J
4 4 10 4 10
2 3 5 9 Q
```

Example Output:
```
PAIR
STRAIGHT
FULL HOUSE
ZILCH
```

Pac-Dude

Input file: pacdude.dat

You are controlling Pac-Dude, a character eats dots while avoiding ghosts. Write a program to control Pac-Dude while he eats his way through a maze. Pac-Dude's goal is to reach the fruit without running into any ghosts. In this version of Pac-Dude, ghosts can't move, and Pac-Dude never retraces his steps.

Input:
The first line will contain the number of mazes to be processed. Each maze will start with a line "X Y" (where X and Y are separated by a single space) indicating that there are X rows and Y columns in the maze. X and Y will never exceed 10. The maze will follow. The grid will use "#" for walls, "." for dots, "X" for the fruit, "G" for the ghosts, and "O" for Pac-Dude.

Output:
If Pac-Dude can reach the fruit, output "NOM-NOM" followed by a maze with Pac-Man at his destination and empty spaces where dots were eaten. If Pac-Dude cannot reach the fruit without running into a ghost or getting stuck behind a wall, output "UH-OH!". Each output should be on its own line.

Example Input (split over two pages):

```
3
6 6
######
#O...#
####.#
#X.G.#
#G...#
######
8 8
########
#O.....#
####.###
#..#.#X#
##...#.#
#..G.#.#
#.##...#
########
6 6
```

```
######
#O...#
######
#G...#
#.G.X#
######
```

Example Output:

```
NOM-NOM
######
#    #
#### #
#O G #
#G   #
######

NOM-NOM
########
#    ..#
#### ###
#..# #O#
##.. # #
#..G # #
#.##   #
########

UH-OH!
```

Last One Standing

Input file: lastonestanding.dat

The game of Last One Standing is similar to dodgeball, except there are no teams and only one ball. Players try to gain possession of the ball and use it to tag other players. If you are tagged, you must sit in “jail” until the person who tagged you is tagged by someone else. Because of this, in order to win, you must tag every other player at least once. The game is over when all players but one are in jail.

Your job is to determine if a game is valid, meaning: no one in jail tagged anyone, no one in jail was tagged, and no one tagged themselves. Your program must also declare the winner if the game is over.

Input:
The first line will contain the number of games to be processed. Each game will begin with a number indicating how many tags occurred during that game, followed by the tags. A tag will be presented as “Name1 tagged Name2”.

Output:
For each game, if the game is valid output “VALID GAME”. On the next line, output either “Winner: ” followed by the name of the winner of the game, or “Unfinished” if there is no winner. If the game is not valid, output only “INVALID GAME”. There should be a newline between sets of output.

Example Input (split over two pages):

```
3
11
Justin tagged Jake
Justin tagged Mary
Sam tagged Alice
Diana tagged Sam
Diana tagged Justin
Mary tagged Jake
Alice tagged Mary
Alice tagged Jake
Alice tagged Diana
Alice tagged Sam
Alice tagged Justin
3
Justin tagged Jake
```

```
Jake tagged Alice
Justin tagged Sue
4
Jake tagged Justin
Jake tagged Alice
Daryl tagged Jake
Daryl tagged Alice
```

Example Output:

```
VALID GAME
Winner: Alice

INVALID GAME

VALID GAME
Unfinished
```

Clock-Paper-Scissors

Input file: clockpaperscissors.dat

In a match of Clock-Paper-Scissors, two players each pick either clock, scissors, or paper, and then compare their choices. The game is played two-out-of-three, which means a player must win at least two out of three matches to win the whole game. Write a program to determine the winner of a game.

Input:
The first line will contain a number indicating how many games are being played. Each game will be represented by three numbers. A 1 indicates that Player 1 wins that match, while a 0 indicates that Player 2 wins that match. Numbers will be separated by a single space.

Output:
For each game, output "Player 1" if Player 1 wins or "Player 2" if Player 2 wins. Each output should be on its own line.

Example Input:
```
3
1 1 0
0 1 0
1 1 1
```

Example Output:
```
Player 1
Player 2
Player 1
```

Sammy Says

Input file: sammysays.dat

In the game Sammy Says, Sammy is telling you what to do. However, you should only do what he says if his sentence starts with “Sammy says ”.

Input:
The first line will be a number indicating how many commands Sammy will give you. Each line after will contain a command.

Output:
If a command starts with “Sammy says” output the command that follows. Each output should be on its own line.

Example Input:
```
5
Sammy says walk
Sammy says jump
Take three steps
Sammy commands you to do a cartwheel
Sammy says crawl forward
```

Example Output:
```
walk
jump
crawl forward
```

Game Reviews

Input file: gamereviews.dat

Before buying a board game or video game, you should always check its reviews online. In this problem, you will process a list of game reviews and output the average star rating that each game received.

Input:
The first line will be a number indicating how many reviews need to be processed. Each review will contain the name of the reviewer, the name of the game, and the rating, in that order. These will be separated by commas and single spaces. The name of the reviewer will always be one word.

Output:
For each game, output (in alphabetical order based on game name) "______ gets __ stars" where the first blank is replaced by the name of the game, and the second blank is replaced by the average number of stars the game received, rounded to the nearest 10th.

Example Input:
```
10
Alex, Super Luigi Planet, 4
Bob, Nomopoly, 2
Carly, Pac-Dude, 5
Doris, Settlers of Catan, 5
Bob, Super Luigi Planet, 3
Doris, Nomopoly, 5
Ernie, Pac-Dude, 1
Ernie, Nomopoly, 3
Alex, Pac-Dude, 5
Bob, Pac-Dude, 4
```

Example Output:
```
Monopoly gets 3.3 stars
Pac-Dude gets 3.7 stars
Settlers of Catan gets 5.0 stars
Super Luigi Planet gets 3.5 stars
```

Nomopoly

Input file: nomopoly.dat

In Nomopoly, you acquire properties. If another player lands on a property you own, they have to pay you money based on how many houses you have built on that property. For each house on the property, the amount of money a player must pay if they land on that property increases by 125%. Write a program to figure out how much a player would owe you if they landed on your property. Because there are no coins in Nomopoly, after calculating the rent, your program should round *down* to the nearest dollar.

Input:

The first line will be a number indicating how many properties you will examine. Then, each property will take up three lines. The first of these three lines will be the name of the property; the second line will be the original rent of the property, when no houses were built, in the format "$xx.xx"; and the third line will be the number of houses on the property now.

Output:

Output "If someone lands on ______, they will owe me $______." where the first blank is the name of the property and the second blank is the rent, rounded *down* to the nearest dollar. Each output should be on its own line.

Example Input:

```
2
Atlantis Ave
$30.00
2
Engelberg Way
$58.00
1
```

Example Output:

```
If someone lands on Atlantis Ave, they will owe me $151.
If someone lands on Engelberg Way, they will owe me $130.
```

Bopscotch

Input file: bopscotch.dat

In Bopscotch, players hop across a chalk pattern, then return to their starting point. Your job is to write a program that draws the pattern.

Input:
The first line will contain the number of patterns to draw. The following lines will each contain a number indicating the number of steps in that pattern. The number of steps will never be less than 2 or greater than 100.

Output:
Output the Bopscotch pattern. Numbers should go from the bottom up and from left to right. Interior lines (lines that are not the top or the bottom) should alternate between containing one number and containing two numbers. In some cases, when there are not enough numbers left at the upper interior line to make a pair, the upper two interior lines will each contain one number. Patterns should be separated by a newline.

Example Input:
```
5
2
5
9
10
11
```

Example Output (see next page):

Example Output:

```
/ 2 \
\ 1 /

/ 5 \
[3][4]
 [2]
\ 1 /

/ 9 \
 [8]
[6][7]
 [5]
[3][4]
 [2]
\ 1 /

/ 10 \
 [9]
 [8]
[6][7]
 [5]
[3][4]
 [2]
\ 1 /

/ 11 \
[9][10]
 [8]
[6][7]
 [5]
[3][4]
 [2]
\ 1 /
```

Chaseball

Input file: chaseball.dat

In Chaseball, players hit a ball with a bat and then run around a diamond-shaped field. Your job is to write a program that draws the field.

Input:
The first line will contain the number of fields to draw. The following lines will each contain a number indicating the number of slashes that make up one side of that field. Each number will never be less than 1 or greater than 50. Fields should be separated by a newline.

Output:
The field.

Example Input:

```
2
2
5
```

Example Output:

```
 /\
/  \
\  /
 \/

    /\
   /  \
  /    \
 /      \
/        \
\        /
 \      /
  \    /
   \  /
    \/
```

Invaders

Input file: invaders.dat

In Invaders, you control an alien race whose mission is to take over earth. At the beginning of the game, the aliens are confined to one place and spread outwards (in all directions) at a rate of 1000 miles per day. Each day, for every new city the aliens reach, their spread rate doubles. This new rate comes into effect at the beginning of the following day. For example, if the aliens reach two cities in a single day, their spread rate will have doubled twice at the beginning of the next day. The spread rate will never be greater than 250000. You will record the process of the aliens' takeover.

Input:
The first line will contain the number of cities that exist in the game. Each city will be represented by a name, followed by a single space, followed by a colon, followed by another single space, followed by that city's distance from the aliens' starting location.

Output:
Each day, you will print the day number, the current spread rate of the aliens, and a message for each city they were able to reach in the order that they were reached. Each message will read "____ has fallen!" where the blank is the name of the city reached. City names will not include spaces or colons. The current spread rate and city messages should each be preceded three spaces, so they appear indented. When the aliens have taken over every city, print "The aliens have taken over!" This message should also be preceded by three spaces.

Example Input:

```
5
NYC : 500
Auburn : 3000
Boise : 6770
Seattle : 3100
Sydney : 30000
```

Example Output (see next page):

Example Output:

```
Day 1
   Rate: 1000 Miles Per Day
   NYC has fallen!
Day 2
   Rate: 2000 Miles Per Day
   Auburn has fallen!
Day 3
   Rate: 4000 Miles Per Day
   Seattle has fallen!
   Boise has fallen!
Day 4
   Rate: 16000 Miles Per Day
Day 5
   Rate: 16000 Miles Per Day
   Sydney has fallen!
   The aliens have taken over!
```

Chest

Input file: chest.dat

In Chest, differently shaped pieces with different abilities move on an 8 by 8 board. The Knight piece always moves in an L shape, either two spaces horizontally and one space vertically, or two spaces vertically and one horizontally.

Input:
The first line will contain the number of board locations to follow. Each board location will be represented by a capital letter (A - H, where A is left and H is right) for the column and a number (1 - 8, where 1 is bottom and 8 is top) for the row.

For reference, in the example below, K is at location F6.

```
  A B C D E F G H
8 - - - - - - - -
7 - - - - - - - -
6 - - - - - K - -
5 - - - - - - - -
4 - - - - - - - -
3 - - - - - - - -
2 - - - - - - - -
1 - - - - - - - -
```

Output:
For each location, output a board printout in which the Knight is represented by the letter K, the locations it could reach are represented by question marks, and other locations are represented by hyphens. There should be a single space between each symbol in within rows. There should also be a newline between each board.

Example Input:

```
3
E4
G6
A1
```

Example Output (see next page):

Example Output:

```
- - - - - - - -
- - - - - - - -
- - - ? - ? - -
- - ? - - - ? -
- - - - K - - -
- - ? - - - ? -
- - - ? - ? - -
- - - - - - - -

- - - - - ? - ?
- - - - ? - - -
- - - - - - K -
- - - - ? - - -
- - - - - ? - ?
- - - - - - - -
- - - - - - - -
- - - - - - - -

- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- ? - - - - - -
- - ? - - - - -
K - - - - - - -
```

Card Sort

Input file: cardsort.dat

In some card games, it is useful to sort a hand of cards by suit and number. You will write a program to help with sorting.

Input:
The first line will contain the number of hands to follow. Each subsequent line will contain a hand of cards. Each hand will consist of some number of cards, separated by commas and single spaces. Each card is represented by a suit (heart, diamond, spade, or club) followed by a number 2 to 10, jack, queen, king, or ace (represented by J, Q, K, or A, respectively).

Output:
Sort each hand by suit (in alphabetical order) and then by number (in the order: ace, 2, … 10, jack, queen, king). Each output should be on its own line.

Example Input:
```
2
heart 3, club 10, club K
club 7, diamond 7, spade 7, heart Q, diamond 2
```

Example Output:
```
club 10, club K, heart 3
diamond 2, diamond 7, club 7, heart Q, spade 7
```

Scavenger Hunt

Input file: scavengerhunt.dat

In a scavenger hunt, players go to locations to collect items. Each item collected is worth a different number of points. Whichever player has the most points at the end of the game wins. In this version, the player can only go right or down, and the player must start at the top-left corner and end up at the bottom-right corner. On their way, they must collect the items that are worth the most points.

Input:
The first 8 lines will each contain a row of 8 numbers. Each number in this 8 by 8 grid represents the number of points a player would receive if they went to that location and collected that item.

Output:
Output the 8 by 8 grid where the optimal path has been cleared. (The collected items' numbers have been removed.) On the following line, output "___ points collected!" where the blank is the total number of points collected.

Example Input:
```
0 3 4 0 1 0 3 4
3 1 1 4 1 3 1 1
1 2 3 3 4 3 1 1
1 1 5 1 1 3 1 1
2 1 3 0 0 2 1 3
3 1 1 4 1 0 3 3
1 2 3 3 4 2 2 2
3 1 1 4 1 5 3 2
```

Example Output:
```
      0 1 0 3 4
3 1   4 1 3 1 1
1 2   3 4 3 1 1
1 1   1 1 3 1 1
2 1   0 0 2 1 3
3 1     1 0 3 3
1 2 3       2 2
3 1 1 4 1
39 points collected!
```