PLU February 2020 Programming Contest

Novice Division

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. Problems will have either no input or will read input from a specified file. All output should be to standard output (the monitor).
3. All input is as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros.
4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

II. Names of Problems

Number	Name
Problem 1	Howl
Problem 2	Copier
Problem 3	Pups
Problem 4	Date
Problem 5	Math
Problem 6	Word
Problem 7	Bomb
Problem 8	Rain
Problem 9	Repeat
Problem 10	Rook
Problem 11	Treasure
Problem 12	Helicopter

1. Howl

Input File: none

On a full moon, one might be able to hear a wolf howl in the distance. You manage to come across a wolf howling at the moon, and you want to capture the beautiful image. Unfortunately, you left your camera at home, so you decide to just create the picture on your computer! Create the image of a wolf howling as shown below.

Input
None.

Output
The wolf art, exactly as shown below, with no extra blank spaces. In particular, a line must not end with a blank space. There is an extra row of numbers added for your convenience, but the row of numbers should NOT be included in your output. The eye of the wolf is a back-tick character (same keyboard key as, ~, tilde).

Output to Screen

```
                     .
                    / V\
                  / `  /
                 <<   |
                 /    |
               /      |
             /        |
           /    \  \ /
          (      ) | |
  ________|   _/_  | |
<__________\______)\__)
012345678901234567890123
```

2. Copier

Input File: copier.dat

Your copier broke down last week, and you need to copy a list of numbers for a class project due tomorrow! Luckily, you can use your computer to copy the numbers for you. Given a list of numbers, each on their own line, print out the number, a space, and then another copy of the number.

Input
The first line will contain a single integer `n` that indicates the number of numbers to follow, each on their own line. The next `n` lines will each contain a single number.

Output
For each of the `n` lines, print out the original number and a copy of the number, with one space of separation.

Example Input File
```
3
7
3
10
```

Example Output to Screen
```
7 7
3 3
10 10
```

3. Pups

Input File: pups.dat

Congratulations, you adopted some little puppies! Now you just need to go grab food for them at the store. Your vet tells you how many pounds of food each pup will eat before your next trip to the store, so you just need to calculate the total amount of food that you will need to buy. You also know how much food costs per pound, so you just need to make sure that you bring the right amount of money to pay for the food. Write a program that, given the number of puppies, the amount of food per puppy, and the price per pound of food, calculates the amount of money that you will need to bring.

Input
The first line will contain a single integer `n` that indicates the number of lines that follow. Each line will contain three non-negative numbers `d, f,` and `p` for the number of dogs, the amount of food per dog in pounds, and the price of the food per pound.

Output
For each data set, output the total amount of money required to buy the food, rounded to the nearest hundredth. Include a dollar sign before the number.

Example Input File
```
3
3 2 1
5 .16 4.54
3 3.7 3.6
```

Example Output to Screen
```
$6.00
$3.63
$39.96
```

4. Date

Input File: date.dat

You are starting to get annoyed with how your phone shows you the date, so you want to put it into a different format. Currently, your phone gives you the date in the format `Month Day, Year` (ex. `July 25, 2018`). Your goal is to shorten the format exactly like this, `MM/DD/YY`, where M is month, D is day, and Y is year (ex. `07/25/18`). Write a program that will calculate this format change.

Input
The first line will contain a single integer `n` that indicates the number of lines that follow. The next `n` lines will be in the format `Month Day, Year`. The full year integer will be between 1 and 300,000,000.

Output
Your output should be the same date as the input, but in the new format `MM/DD/YY`. If the month, day or year is only one digit, pad with a `0`. If the date is not valid (the month is spelled incorrectly, the day is less than 1 or greater than 31), print `Invalid`. You do not need to check that the day fits within the month's days, just that it is not greater than 31. The year format will be the last two digits (or 1 digit padded with a 0) of the input year.

Example Input File
```
3
February 5, 990
December 25, 12018
Decembuary 31, 2000
```

Example Output to Screen
```
02/05/90
12/25/18
Invalid
```

5. Math

Input File: math.dat

While doing your math homework, you decide that it might be easier for your computer to do the work instead of having to do them yourself with a calculator. Write a program that will calculate the answer to an equation.

Input
The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will consist of a single line containing alternating integers and operators. There will be a space between each number and operator, and the only operators will be `+ - * / %`.

Output
Output the integer value solution to the equation. The final answer will always be an integer, and all intermediate steps will be integers. The equation will be solved using the standard mathematical precedence rules.

Example Input File
```
3
5 * 2 + 9 / 3
2 + 2 + 6 % 5
6 + 2 * 10 - 4
```

Example Output to Screen
```
13
5
22
```

6. Word

Input File: word.dat

Word searches are hard. Luckily, you're a programmer, so you can just write a program to find the words! Fortunately, the word searches that you are doing are only looking for one word, and that word is "`word`". Write a program that finds the number of instances of the word "`word`" in a word search.

Input
The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will start with two integers, `r c` with `r` being the number of rows and `c` being the number of columns, respectively. The next `r` lines will contain `c` random letters, creating a word search grid.

Output
Output the number of times that the word "`word`" appears in the word search. The letters can be going in any direction, including backwards.

Example Input File
```
2
4 4
word
aoah
nerd
qpid
4 4
sowk
dyuf
asaf
diub
```

Example Output to Screen
```
2
0
```

7. Bomb

Input File: bomb.dat

Bomberman is having some trouble defeating his enemies, and he has asked you to help him figure out where to place his bomb in order to defeat the most enemies. In this scenario, bomberman's bombs destroy everything in the four cardinal directions to the edge of the map, except for stopping at walls that cannot be broken. Write a program that finds the best spot to place a bomb to destroy the most enemies.

Input
The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will start with two integers, `r c` representing the number of rows and columns in each map, respectively. The next `r` lines will consist of `c` characters, representing the map. `#` denotes a wall, `.` is a blank space, and `@` is an enemy. The bomb can only be placed in a blank space. Rows are numbered `0` to `r-1` and columns are numbered `0` to `c-1`.

Output
Output the coordinate for the bomb, in the format `r, c`, where `r` is the row number and `c` is the column number. The bomb will explode outward in the four cardinal directions, destroying any enemy in its path. The explosion only stops in one direction when it reaches the end of the map or when it hits a wall, in which case any enemy past that wall is safe. Assume that there will always be at least one enemy that a bomb can hit, and there will never be more than one best placement.

Example Input File
```
2
9 17
#.#.#.#.#@#.#.#.#
..@..............
#.#.#.#.#.#.#.#.#
.........@.......
#.#.#.#.#.#.#.#.#
.....@........@..
#.#.#.#.#.#.#.#.#
...........@.....
#.#.#.#.#.#.#.#.#
5 5
#####
#...#
#.#@#
#.@.#
#####
```

Example Output to Screen
```
5, 9
3, 3
```

8. Rain

Input File: rain.dat

In your city, there is a shortage of drinking water so the city planner is finding more ways to trap rainwater to be used for drinking. The city has varying heights of buildings, and rainwater gets trapped between the buildings. The city planner has asked you to write a program that, given the heights of the buildings, can find the largest amount of rainwater that can get trapped between the buildings.

Input
The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will consist of a single line of integers, denoting the heights of the buildings. The building height can be positive or negative.

Output
Output the total amount of rainwater that can be trapped between/above the buildings of varying height.

Example Input File
```
2
5 0 0 0 5
6 3 1 6 4 5 1 0 3 5 3 1 4
```

Example Output to Screen
```
15
24
```

9. Repeat

Input File: repeat.dat

Your teacher has been marking students off on their essays for repeating words and phrases when they could be exploring new syntax and using a thesaurus to spice up their writing. In order to save yourself from being called out for using repeats, you decide to create a way to check your essay for repeated substrings and find the longest one to fix. Write a program that, given a string of characters, finds the greatest repeated substring with no overlapping characters.

Input
The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will consist of one line with a string of letters of unknown length.

Output
Output the integer length of the longest repeated substring, a space, and then the repeated string. If there are multiple with the same length, print out the first one to appear in the string.

Example Input File
```
2
thequickbrownfoxjumpedoverthelazydog
ratsarejustlikebigmicebutratsarecompletelydifferentanimals
```

Example Output to Screen
```
3 the
7 ratsare
```

10. Rook

Input File: none

You have just learned how to output text to the screen and your teacher has challenged you to create an ASCII art of a chess piece. You have decided to make your favorite piece, the rook.

Notes:

- **3 underscores** ___
- **2 underscores** __

Input
None.

Output
The rook art, exactly as shown below, with no extra blank spaces. In particular, a line must not end with a blank space. The extra row of numbers at the top and bottom are added for your convenience, but these rows of numbers should **NOT** be included in your output.

Output to Screen

```
012345678901234567
  ___  ___  ___
  | |__| |__| |
  |           |
   \_________/
    \_______/
     |     |
     |     |
     |     |
     |     |
     |_____|
  __/       \__
 /             \
/________________\
012345678901234567
```

11. Treasure

Input File: treasure.dat

After doing some digging on the beach, you happen to pull up an old treasure chest! It is full of rare artefacts worth a lot of money! Unfortunately, you can only carry so much with you back to your boat before the tide washes the treasure chest away. Write a program that will calculate the highest total item value that you can take back in one trip, given several item weights and values, and your carrying capacity.

Input
The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will start with two integers `i w` that indicate the number items that follow and your carrying capacity, respectively. The next `i` lines will each have two numbers, the value of each item and its weight.

Output
Output the highest value that you can carry in one trip for each data set.

Example Input File
```
2
5 50
70 10
90 32
40 4
150 45
60 46
1 10
500000 11
```

Example Output to Screen
```
200
0
```

12. Helicopter

Input File: none

Write a program that displays the helicopter as seen below.

```
012345678901234567012345678
      ===================
          ____||___
\ /      /       []\
 X=======           \__
/ \      \____________|
            ||  ||
         \-----------/
```

Input
None.

Output
Displays the helicopter, with no extra blank spaces. In particular, a line must not end with a blank space.

Example Output to Screen

```
      ===================
          ____||___
\ /      /       []\
 X=======           \__
/ \      \____________|
            ||  ||
         \-----------/
```