A+ Computer Science
Computer Science Competition
Hands-On Programming Set

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. All problems have a value of 60 points.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Point Values and Names of Problems

Number	Name
Problem 1	Most Value
Problem 2	Register
Problem 3	Card Sort
Problem 4	Octagon Side Length
Problem 5	Rearrange
Problem 6	Cave Adventure
Problem 7	Lost City
Problem 8	Camel
Problem 9	Bumps
Problem 10	Safest Path
Problem 11	Forrest
Problem 12	Album Length

For more Computer Science practice tests and materials, go to www.apluscompsci.com

1. Most Value

Program Name: most_value.java **Input File: most_value.dat**

Write a program to find the most often occurring number in a data set and then display total value that number represents in the data set's total value.

Input

There will be an unknow number of inputs, each on its own line. Each input will consist of an unknow number of integer values separated by spaces.

Output

For each input display the most often occurring number, the total of the most often occurring number and the total of the data set, in the following format.

mostOftenNumber accounts for **mostOftenToyal** of the sets total value of **totalValue**

Example Input File

```
4 5 4 5 8 5 6 5 8
1 1 1 2 2 1 1
78 55 55 88 147 9 6 3 5 18 88 52 88
```

Example Output to Screen

```
5 accounts for 20 of the sets total value of 50
1 accounts for 5 of the sets total value of 9
88 accounts for 264 of the sets total value of 692
```

2. Register

Program Name: register.java **Input File: register.dat**

You have been hired to write a program that will verify register funds. The program will take in the amount that the register should have and the quantity of each unit of currency. The program will then report if the register is over, under or correct.

Input

The input file will contain of three inputs, each on their own line. The format for each input is:

ExpectedValue #twenties #tens #fives #ones #quarters #dimes #nickles #pennies

Output

For each input display if it is over, under or correct.

When correct display:
Correct

When under display:
Missing $#.##

When over display:
Over $#.##

Notes:

- Cash values always have 2 decimal places
- Never display a negative number

Example Input File

```
50.32 1 1 1 1 1 1 1 1
20.00 1 0 1 0 0 0 0 1
10.32 0 1 0 0 0 3 0 2
```

Example Output to Screen

```
Missing $13.91
Over $5.01
Correct
```

3. Card Sort

Program Name: card_sort.java **Input File: card_sort.dat**

You have designed a card game, where each card has a name, attack value and defense value. You game has a hidden stat called power which is the combined attack and defense power. When designing card sets you find it helpful to the cards sorted in the following manor:

Sorting Rules:

- Cards should be sorted in descending order by power
- For cards that have the same power, sort them in descending order by attack
- For cards that have the same power and attack, sort them ascending order by name

Input

There will be an unknown number of inputs, each on their own line. The format for each input will be:

cardName/attack/defense

Output

Display the sorted cards, with each card on its own line. The format for displaying the card will be:

power (**cardName/attack/defense**)

Example Input File

```
mog/3/6
robot/3/6
grappler/4/5
locker/3/4
poki/9/6
rac/5/5
```

Example Output to Screen

```
15 (poki/9/6)
10 (rac/5/5)
9 (grappler/4/5)
9 (mog/3/6)
9 (robot/3/6)
7 (locker/3/4)
```

4. Octagon Side Length

Program Name: octagon.java **Input File: octagon.dat**

A sign company has hired you to write a program that will find the side length of an octagon, given its area.

Formulas:

$A = 2 (1+\sqrt{2}) s^2$
(A – area, s – side length)

Input

The first line will contain a single integer n that indicates the number of lines that follow. The next n lines will each contain the area of an octagon.

Output

Display the side length of each octagon with 2 decimal places, each on its own line.

Example Input File

```
4
78.2541
1
25.34
9.999
```

Example Output to Screen

```
4.03
0.46
2.29
1.44
```

5. Rearrange

Program Name: rearrange.java **Input File: rearrange.dat**

Your teacher wants you to write a program that will rearrange the letters in words that are 3 more letters long, words of length 2 and 3 will not be changed. When a word is of odd length the first and last letters are swapped. For words of even length, the middle left letter swaps with the first letter and the middle right letter swaps with the last letter.

Input

The input will consist of an unknow number lines with an unknown number of words on each line.

Output

Display each converted line of text, on its own line.

Example Input File

```
I am testing this for Alex
Alex likes snakes and so do I
James is coming over to play
```

Example Output to Screen

```
I am gestint htsi rof lAxe
lAxe sikel anssek dna so do I
sameJ is mocgni vore to lpya
```

6. Cave Adventure

Program Name: cave_adventure.java **Input File: cave_adventure.dat**

Write a program to determine if a maze is solvable. Mazes will contain a starting point, ending point, floors, walls and gaps. The player can jump over a single gap, if they land on a floor. The player can only move orthogonally (up/down/left/right). A cave is solvable if the end can be reached from the start.

Map Key:

- ‘W’ – Wall
- ‘-‘ – Floor
- ‘G’ – Gap
- ‘S’ – Start
- ‘E’ - end

Input

There will be an unknown number of inputs. Each input will be contain an 8 by 8 grid that represents a maze. There will a line containing a single ‘-‘ separating each maze.

Output

For each maze, print “Solvable” or “No Solution”.

Example Input File

```
WWWWWWWW
WS-G-WWW
WWWWGWWW
WWWWEWWW
WWWWWWWW
WWWWWWWW
WWWWWWWW
WWWWWWWW
-
W--WWWWW
WEGWWWWW
WW-WWWWW
W-----WW
W-WWW-WW
W-WWW-WW
WS----WW
WWWWWWWW
-
S--WWWWW
---WWWWW
WW-GG--W
WWWWW--W
WWWWW--E
WWWWWWWW
WWWWWWWW
WWWWWWWW
-
WWWWWWWW
WWWWWWWW
WWSGGEWW
WWWWWWWW
WWWWWWWW
WWWWWWWW
WWWWWWWW
WWWWWWWW
```

Example Output to Screen

```
Solvable
Solvable
No Solution
No Solution
```

7. Lost City

Program Name: lost_city.java **Input File: lost_city.dat**

You are an explorer and must travel light. When you discover a new tomb, you want to carry the highest value in artifacts that you can carry. Write a program that will calculate how much value you can leave the tomb with. The program will take the maximum weight you can carry and the weight/value of each artifact.

Input

There will be an unknow number of inputs, each on its own line. The format for each input will be:
carryCapacity-itemWeight1:**itemValue1**/**itemWeight2**:**itemValue2**/…**itemWeightX**:**itemValueX**

Output

For each input display the maximum loot value, in the following format:

You grabbed $**value** worth of artifacts.

Note: value will need to be formatted to 2 decimal places.

Example Input File

```
5-3:4.5/2:1.86/3:6.78
10-8:8060/10:9999/1:5555/1:5555/2:10000
1-8:8060/10:9999/5:5555/3:5555/2:10000
```

Example Output to Screen

```
You grabbed $8.64 worth of artifacts.
You grabbed $21110.00 worth of artifacts.
You grabbed $0.00 worth of artifacts.
```

8. Camel

Program Name: camel.java **Input File: none**

For art you have been assigned the task of creating an abstract image of an animal. You have decided that you will make a camel in ascii.

Input

None.

Output

The camel art, exactly as shown below. There is an extra row of numbers added for your convenience, but the row of numbers should **NOT** be included in your output.

Output to Screen

```
 __
/ "\
\__ \       ___      ___
   \ \     /   \    /   \
    \ \___/     \__/     \_______
     \_                        \ \
       |                        | \
        \ ___   _______  ___   /   \_
        / /  | |      / /   | |
       / /   | |     / /    | |
      / /    ( |     / /    ( |
      | |     | |    | |    | |
      | |     | |    | |    | |
      /_/     /_/    /_/    /_/
012345678901234567890123456789012345678
```

9. Bumps

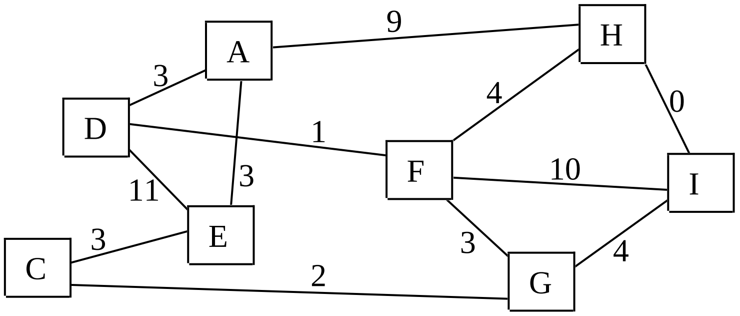
Program Name: bumps.java **Input File: bumps.dat**

Unfortunately, your city has neglected to keep up the roads and they are riddled with potholes. You are worried about damage to your car and want to create an algorithm that will get you to your destination, while driving over as few potholes as possible.

Notes:

- Each location in your city is denoted by a single letter
- A will be the starting location
- Each edge will have an integer weight, in potholes.
- There will only be one solution with that contains the fewest potholes

Here is an example:



Input

The first line of input will be a single letter, denoting the destination:

Each additional line describes a road between two locations and the number of potholes on that road The format for each road is:

```
LocationA/LocationB - numberOfPotHoles
```

Output

Display the path that has the fewest number of potholes, listing the locations in the order they are visited.

Example Input File

```
I
A/H – 9
A/D – 3
A/E – 3
D/E – 11
D/F – 1
E/C – 3
C/G – 2
F/H – 4
F/I – 10
F/G – 3
H/I – 0
G/I - 4
```

Example Output to Screen

```
ADFHI
```

10. Safest Path

Program Name: safest_path.java **Input File: safest_path.dat**

You are planning a journey with your adventuring party and want to take to the safest path. You have access to map that includes bridges, roads, wilderness, goblin sightings, troll caves and bandit camps. You have come up with the following key to determine how dangerous a map quadrant is based on what is found there:

Key:

- 'S' – Starting Point (0)
- 'E' – Ending Point (0)
- 'R' – Road (1)
- 'B' – Bridges (2)
- 'W' – Wilderness (4)
- 'G' – Goblins (5)
- 'T' – Trolls (9)

The danger value of a route will be the total of all the quadrants traveled through. Write a program that will find danger value of the safest route.

Notes:

- **Traveling over 7 road locations is more dangerous than traveling over 1 goblin location.**
- **Diagonal movement is not allowed**

Input

There will be an unknown number of inputs. Each input will contain an 8 by 8 grid that represents a map. There will line containing a single '-' separating each map.

Output

Display the lowest danger value for each map.

Example Input File

```
WWWWWWWW
WWWWTWWW
WWWTRWEW
WRRRRWWW
WWWWRWWW
WRRRRWWW
WRWWWWWW
WSWWWWWW
-
WWWWWWWW
WWWWWWWW
WWSRRRWW
WWRBWRWW
WWRWWTWW
WWRWWRWW
WWRBBEWW
WWRRRRWW
```

Example Output to Screen

```
12
8
```

11. Forest

Program Name: forest.java **Input File: forest.dat**

You are building a program that will draw a text representation of a forest. Each tree will be 6 characters wide and contain the following parts:

Top:

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

Middle levels:

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

Bottom:

```
  ||
  ||
```

Each tree will consist of a bottom, top and some number of middle levels and each tree will be separated by a single space. When part of the forest has no tree, then there will be 6 spaces where the tree and those 6 spaces will be separated from other forest location by 1 space, just like a tree.

Tree height will be denoted with a number from 0 to 10. Zero means there is not a tree. A tree will always have a base, 1 top and its height-1 number of middle levels.

Input

The input will be on a single line. The first number will define how many tree locations your forest will contain, followed by the tree height of each tree in the forest. All the values will be separated by spaces.

Output

Output the forest so that the top line of the output will have the tip of the tallest tree. The trees will be drawn in a horizontal line and each tree will be separated by a single space. There is an extra row of numbers added for your convenience, but the row of numbers should **NOT** be included in your output.

Example Input File

```
6 3 7 3 0 0 1
```

Example Output to Screen

```
         /\
        /  \
       /_  _\
        /  \
       /_  _\
        /  \
       /_  _\
        /  \
  /\   /_  _\   /\
 /  \   /  \   /  \
/_  _\ /_  _\ /_  _\
 /  \   /  \   /  \
/_  _\ /_  _\ /_  _\                 /\
 /  \   /  \   /  \                 /  \
/_  _\ /_  _\ /_  _\               /_  _\
  ||     ||     ||                   ||
  ||     ||     ||                   ||
012345678901234567890123456789012345678901023456789
```

12. Album Length

Program Name: album_length.java **Input File: album_length.dat**

Write a program that will find the length of an album, given the length of all its songs. The length of each song will be given in minutes and seconds. The total album length will always be less than 1 hour.

Input

The first line of input will contain how many songs are on the album. Each remaining line will describe a song and will in the following format:

songName, **minutes**:**seconds**

Output

Display the length of the album in the following format:

The album length is **minutes**:**seconds**

The seconds place must be two digits!

Example Input File

```
7
songs of the bird, 3:14
bird calls, 5:10
moching bird, 3:57
robin rage, 8:46
sad blue bird, 5:13
hungry hawk, 2:22
follow that bird, 4:56
```

Example Output to Screen

```
The album length is 33:38
```