1. Print Triangle Waves 2

In this problem you are to generate several double-sided triangular waveforms according to some specified amplitude values.

Input:

The input is a string composed by *N* (1 ≤ *N* ≤ 20) wave amplitude values. Each wave *Amplitude* value is represented by a single digit odd number (3 ≤ *A* ≤ 9).

Output:

For the output of your program, you will be printing double-sided waveforms between each wave amplitude. The horizontal "height" of each wave equals the amplitude. The waveform itself should be filled with integers on each line which indicate the "height" of that line.

Sample Input:

```
353
```

Sample Output:

```
  1
 333
  1
 333
55555
 333
  1
 333
  1
```

Hints:

Be careful with the center position of each wave.

2. Sorting Points 2

There are N points in 2D Cartesian coordinate denoted as $P_i = (x_i, y_i)$ for i-th point. Write a program to convert given data into a **Linked List** data structure and reorder those points from bottom-left to top-right (sort x and y in ascending order **prioritizing on the x value**).

Input:

Input data is provided in `main.c` and with variables named `data` and `size`. The variable `data` is a 2D array consists of several records, and the size is defined by the variable `size`. Each record is a 1D array consists of 2 floating numbers representing values of axis x and y.

Output:

Show the reordered result. Each number should be rounded to 2 decimal places.

Sample Input:

```
int size = 5;
float data[][2] =
{
    {1.2, 3.7}, {5.98, 7.75},
    {10.13, 7.75}, {-7.51, -3.7},
    {1.2, -6.23}
};
```

Sample Output:

```
-7.51 -3.70
1.20 -6.23
1.20 3.70
5.98 7.75
10.13 7.75
```

Hints:

Please implement the `convert` and `sort` function in `func.h`. Any modification in `main.c` will be discarded during compilation.

3. CSV Parser

"A comma-separated values (CSV) file is a delimited text file that uses a comma to separate values. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format." (Comma-separated values, WIKIPEDIA)

Now we are given a student score list, and it's our responsibility to found out some meaningful values from these numbers. But first, let us parse the CSV data into structural information.

Write a program to convert a given CSV data into a **Linked List** data structure, then show the whole list and calculate the average score.

Input:

Input data is provided in `main.c` with the variable named `data` contains student score list data in CSV format. Although `data` is a large string, one can still get student records line by line with the new-line (`\n`) symbol. Each record consists of 5 fields, name of a student and 4 scores of English, Math, History and Physics. The maximum length of name is 50, and the scores are all integers.

Output:

List the name and scores (English, Math, History and Physics) of all students and an average total score (rounded to 2 decimal places) in the following format.

```
{name0}: {English0}, {Math0}, {History0}, {Physics0}
{name1}: {English1}, {Math1}, {History1}, {Physics1}
...
Average score: {avg}
```

Sample Input:

```
char* data =
    "Caspar Murray,50,25,100,80\n"
    "Rory Gordon,50,50,100,60\n"
    "Winnie Randolph,50,100,50,20\n";
```

Sample Output:

```
Caspar Murray: 50, 25, 100, 80
Rory Gordon: 50, 50, 100, 60
Winnie Randolph: 50, 100, 50, 20
Average score: 245.00
```

Hints:

Please implement the `convert` and `average` function in `func.h`. Any modification in `main.c` will be discarded during compilation.

4. Maze Traversal 2

For a given maze, there's a simple algorithm for walking through the maze that guarantees finding the exit (assuming there's an exit). If there's not an exit, you'll arrive at the starting location again. Place your **right** hand on the wall to your right and begin walking forward. Never remove your hand from the wall. If the maze turns to the right, you follow the wall to the right.
As long as you do not remove your hand from the wall, eventually you'll arrive at the exit of the maze.

Input:

1. The input is a 15-by-15 maze.
2. The maze is guaranteed to have an entry at the left and an exit on the right (if there exists one exit).
3. The # symbols represent the walls where you cannot walk through.
4. The X symbols represent the possible paths.

Sample Input 1:

```
###############
#XXXXXXXXX#XXX#
#X#X#####X###X#
#X#X#XXXXXXX#X#
#X#######X###X#
#X#X#XXXXXXXXX#
###X###X#######
#XXXXXXX#X#XXX#
#X#######X#X###
XX#XXXXX#XXX#XX
#X#X###X#X#X#X#
#XXX#XXXXX#XXX#
#X#####X#####X#
#XXXXX#XXXXX#X#
###############
```

Sample Input 2:

```
###############
#X#XXXXXXXXX#X#
#X#X#X###X#X#X#
#XXX#X#XXX#XXX#
#########X###X#
XXXXXXXXXXXX#X#
###X#######X###
#XXXXXXXXX#XXX#
#X#X#####X###X#
#X#X#XXXXXXX#X#
#X#######X###X#
#X#X#XXXXXXXXX#
###X###X#######
#XXX###XXXXXXX#
###############
```

Output:

1. Please follow the given algorithm to solve and show the solution of the maze.
2. Other solutions, for example, a shortest path is not acceptable.
3. Put R on the path that is taken.
4. If the maze does not have a solution, you still need to show the path and then print "This maze has no solution"

Sample Output 1:

```
###############
#XXXXXXXXX#XXX#
#X#X#####X###X#
#X#X#XXXXXXX#X#
#X#######X###X#
#X#X#XXXXXXXXX#
###X###X#######
#XXXXXXX#X#XXX#
#X#######X#X###
RR#RRRRR#RRR#RR
#R#R###R#R#R#R#
#RRR#RRRRR#RRR#
#R#####R#####R#
#RRRRR#RRRRR#R#
###############
```

Sample Output 2:

```
###############
#R#RRRRRRRRR#R#
#R#R#R###R#R#R#
#RRR#R#RRR#RRR#
#########R###R#
RRRRRRRRRRRR#R#
###R#######R###
#RRRRRRRRR#RRR#
#R#R#####R###R#
#R#R#RRRRRXX#R#
#R#######R###R#
#R#X#RRRRRRRRR#
###X###R#######
#XXX###RRRRRRR#
###############
This maze has no solution
```

Hints:

1. Please follow the given algorithm to solve the maze.
2. If you return to the starting point, that means the maze does not have solutions.
3. To read the maze, one can use `getchar()` or string:

```
while (input != '\n') {
    input = getchar();
    ...
}
```

4. Using two-dimensional array can help to visualize the maze and the coordinate system.

5. Find the Telephone

In some places is common to remember a phone number associating its digits to letters. In this way the expression "`MY LOVE`" means `69 5683`. Of course, there are some problems, because some phone numbers cannot form a word or a phrase and the digits `1` and `0` are not associated to any letter. Your task is to read an expression and find the corresponding phone number based on the table below. An expression is composed by the capital letters (`A-Z`), hyphens (`-`) and the numbers `1` and `0`.

Letters	Number
ABC	2
DEF	3
GHI	4
JKL	5
MNO	6
PQRS	7
TUV	8
WXYZ	9

Input:
The input consists of at most 2000 lines. The first line contains a number n, indicating that there will be n more lines of expressions. Each expression is in a line by itself and has *C* characters, where $1 \le C \le 30$.

Output:
For each expression you should print the corresponding phone number, the number of capital letters, and the number of hyphens.

Sample Input:

```
2
1-HOME-SWEET-HOME
MY-MISERABLE-JOB
```

Sample Output:

```
1-4663-79338-4663 13 3
69-647372253-562 14 2
```

6. List of Conquests

In Act I, Leporello is telling Donna Elvira about his master's long list of conquests:

"This is the list of the beauties my master has loved, a list I've made out myself: take a look, read it with me. In Italy six hundred and forty, in Germany two hundred and thirty-one, a hundred in France, ninety-one in Turkey; but in Spain already a thousand and three! Among them are country girls, waiting-maids, city beauties; there are countesses, baronesses, marchionesses, princesses: women of every rank, of every size, of every age." (Madamina, il catalogo questo)

As Leporello records all the "beauties" Don Giovanni "loved" in chronological order, it is very troublesome for him to present his master's conquest to others because he needs to count the number of "beauties" by their nationality each time. You are to help Leporello to count.

Input:
The input consists of at most 2000 lines. The first line contains a number n, indicating that there will be n more lines. Each following line, with at most 75 characters, contains a country (the first word) and the name of a woman (the rest of the words in the line) Giovanni loved. You may assume that the names of all countries consist of only one word.

Output:
The output consists of lines in alphabetical order. Each line starts with the name of a country, followed by the total number of women Giovanni loved in that country, separated by a space.

Example Input:

```
3
Spain Donna Elvira
England Jane Doe
Spain Donna Anna
```

Example Output:

```
England 1
Spain 2
```