COMP281 2017 – Assignment 2

- In the following, you will find the problems that constitute Assignment 1. They will be also available on the *online judging (OJ)* system available at http://intranet.csc.liv.ac.uk/JudgeOnline/

- You need to write a valid C program that solves each of these problems – it must read the input, as specified in the problem description then print the solution to the given problem for that input.
  - Note that code is "correct" only if it **correctly implements a solution to the problem stated** in the assignment, not "if online judge accepts it".
  - That is, even if OJ accepts your code, it could be wrong. Read the problems carefully.

- Input is read from the standard input, in the same way that you read input from the keyboard as shown in lectures (e.g., using scanf). Output is also printed to the standard output, as you have seen (e.g., using printf).

- When you are satisfied that your programs work correctly, you must submit them through the departmental submission system, as described below.
  - Even if the program is not correct, still submit whatever you have! You can still earn points if certain parts of the code are done right.

- You must also include a brief report describing your solutions to the problems. This should be maximum one side of A4 paper (although there is no penalty for a longer report) and should give a description of how each of your solutions works. This should include describing the algorithm used to reach the solution, describing your use of any C language features (that were not discussed in lectures) and identifying any resources that you have used to help you solve the problems.

- This assignment is worth 30% of the total mark for COMP281.
  - All problems are weighted equally.
  - For each problem, you can earn a total of 20 points
    - 10 points for "Functionality and Correctness" awarded for programs that **correctly** solve the problem for all test cases.
    - 8 points for "Programming style, use of comments, indentation and identifiers" awarded depending on the style, comments and efficiency of the solution
    - 2 points for the quality and depth of the accompanying report
  - The final grade results from normalizing the earned points to a scale of 100.
  - See separate "COMP 281 Detailed Marking Guidelines" for more details.

Submission Instructions

- Name each of your c files according to the problem number; i.e., problem 10xx should be in a file 10xx.c (where 'xx' is replaced by the actual number).
- Place all of your C files, one per problem, and your report (in .pdf format) into a single zip file.
    - Please use the standard (pkzip) zip file format:
      https://en.wikipedia.org/wiki/Zip_%28file_format%29
      which is supported by winzip, winrar, etc. on windows/mac os X, and 'zip' on linux
    - test your zip file before submitting.
- Before you submit your solutions, please first test them using the online judge. **You are required to include the "Result" and the "RunID" in your C code as comments.** The OJ provides a RunID. **RunIDs are not problem IDs**.
    - Example:
        - the problem is 10xx
        - The solution has been accepted by the OJ, and the runID is 2033.
        - You add to your code: /* 2033 Accepted */ to 10xx.c
    - Result is one of the following: Accepted, Wrong Answer, Presentation Error, Time Limit Exceeded, Memory Limit Exceeded, Output Limit Exceeded, Runtime Error, Compile Error
- Submit this zip file using the departmental submission system at
  http://www.csc.liv.ac.uk/cgi-bin/submit.pl
  Only the file submitted through this link will be marked.
- The **deadline** for this assignment is **Wed March 1, 2017, 4:30pm**
- Penalties for late submission apply in accordance with departmental policy as set out in the student handbook, which can be found at:
  http://intranet.csc.liv.ac.uk/student/ug-handbook.pdf

1. Problem 1043

Title: Sort strings

Description

Input n and then followed by n strings, one string per line. Subsequently this is followed by input m, and then followed by m strings (one per line). Strings consist of lowercase English letters only.

The maximum string length is 100 (at most 99 letters, plus '\0'). However, most strings are very short. That is, you should dynamically allocate just enough memory to store each string. Clearly, since it is not possible to predict the length of each string before having read it, you should use an array of length 100 to be able to read the next string. Subsequently, however, you should store the string read in a dynamically allocated array of exactly the right size.

You should also dynamically allocate just enough space for storing n+m string pointers.

Please sort these n+m strings based on their lengths in descending order. String lengths are unique. There will be no ties. Output the sorted list of strings.

Input

n and then followed by n strings, m and then followed by m strings.

Output

Sorted list, one string per line

Sample Input

```
5
abcdefghijk
abcde
a
abcdefgaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
zzzzzzzzzzzzzzzzzzzzz
3
bb
cccc
hhhhhhhhhhhhh
```

Sample Output

```
abcdefgaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
zzzzzzzzzzzzzzzzzzzzz
hhhhhhhhhhhhh
abcdefghijk
abcde
cccc
bb
a
```

2. Problem 1080

Title: Normalize potentially large vectors

Description

The length, or 'norm', |v| of an n-dimensional vector v = (x0, x1, x2,… xn-1) is given by:

$|v| = [\text{sum}_{i=0}^{n} (x_i^2)]^{1/2}$

That is, it is given by the sum of the squared elements of the vector and subsequently taking the square root of this sum. Given this norm, a normalized version v' of this vector v can be computed as follows:

$x'_i = x_i/|v|$

Write a program that normalizes a vector v.

Some further requirements are:

-use the sqrt() function from the math.h library. So also include: #include <math.h>

-use doubles to represent the normalized vector

-make sure you only use as much memory as you need by dynamically allocating precisely enough memory using malloc. (do not use 'variable length arrays')

-as always: make your code modular by using functions

Input
n numbers that represent the normalized vector (n could be small or large).

Output
Print the numbers with three digits precision.

Sample Input

```
2
4 3
```

Sample Output

```
0.800 0.600
```

3. Problem 1081

Title: Game of Life

Description

In this assignment, you will code a simulation called "Game of Life". It is a very famous 'game' and the formed the basis of an entire field of simulation models called "cellular automata". Before continuing reading this assignment, I suggest you read a bit more about Game of Life at:

https://en.wikipedia.org/wiki/Conway's_Game_of_Life
or
http://www.math.com/students/wonders/life/life.html
(the latter also has a nice Java applet of the game that is quite fun to play with!)

You should now know about the basics of game of life: it is a 'simulation' of cells that are 'dead' or 'alive' in discrete time steps, at every step all of the cells will get computed a new value in parallel (i.e., based on the values of the neighbours at the *previous* time step - this will require a little bit of thinking!), and the rules to determine if a cell is dead or alive are:

-Any live cell with fewer than two live neighbours dies, as if caused by underpopulation.
-Any live cell with two or three live neighbours lives on to the next generation.
-Any live cell with more than three live neighbours dies, as if by overpopulation.
-Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

You now need to write a program that reads in an initial 'board' configuration, and then performs a specified number of simulations.

As always, try and keep your program clean, modular, and only allocate as much memory as you need (using malloc).

Input

The first line will specify 3 numbers:

-the number of rows
-the number of colums
-the number of steps you need to simulate

The remainder of the input file comprises the initial configuration, which will have exactly the specified number of rows and columns. Dead cells are indicated with '.' while live cells are marked with 'X'.

Output

As output, you will need to write the board configuration that results from the specified number of simulations, in exactly the same format as you read the input.

Sample Input

```
6 6 20
.X...X
X.X.X.
X...X.
X..XX.
..XX.X
...X.X
```

Sample Output

```
...X..
..X.X.
..X.X.
...X..
......
......
```

Sample Input 2 (the 'blinker')

```
5 5 40000
.....
.....
.XXX.
.....
.....
```

Sample Output 2

```
.....
.....
.XXX.
.....
.....
```

Sample Input 3

```
10 10 20
..........
..........
...X...X..
..X.X.X...
..X...X...
..X..XX...
....XX.X..
.....X.X..
..........
..........
```

Sample Output 2

```
..........
..........
..X.......
.X.X......
..X.......
..........
..........
..........
..........
..........
```