COMP281 2017 – Assignment 1

- 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 Feb 15, 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 1013

Title: Record marks

Description

You are in charge of recording marks for a group of students. Input a list of marks. Input ends with 0 (0 itself is not someone's mark). Output the number of students who scored 1) greater than or equal to 85; 2) between 60 and 84; 3) strictly less than 60.

Make sure you use good coding standards: i.e., create separate functions for processing the input and writing the output, and be consistent in the way you name variables (also see the "COMP 281 Detailed Marking Guidelines").

Input

Output

Sample Input

```
88 71 68 70 59 81 91 42 66 77 83 0
```

Sample Output

```
>=85:2
60-84:7
<60:2
```

HINT

Remember that there are different ways of obtaining input. For this assignment "scanf" might be convenient, but make sure to use the correct syntax (it requires a 'pointer' as a second argument.)

2. Problem 1078

Title: Recursive Reverse Omitting Prime Positions

Description

Write a program that reverses the input. There are two complications:

1) You need to implement this by making use of recursion.
2) Characters on positions of the input that correspond to prime numbers must be omitted completely.

Think about good coding standards and divide your code in logical functions.

Input

An arbitrary string of ASCII, ended by EOF.

Output

The reverse of the input. Characters on prime positions (in the forward ordering, i.e, characters that are on position that are prime in the input) are not included. Note:

- for purposes of this assignment positions in the input string begin counting at 0.
- 0 and 1 are considered prime.
- do not output the 'EOF' character.

Sample Input (1)

```
12345678
9012345678901
2345
```

Sample Output (1)

```

543
108654209
75
```

Sample Input (2)

```
this input has
no newline before the end of input
```

Sample Output (2)

```
tuni odneeht eoeb enlwe n
 a tun
```

HINT

Note that all characters in the input should be treated equally including newline characters. E.g., the sample output (1) above starts with a newline because the last character of the input is a newline. The only exception is the 'EOF' character, that should not be written to the output.

3. Problem 1079

Title: Football Statistics II

Description

In this assignment, you are asked to process a list of outcomes of football games and process the results.

You can assume that there are only 30 teams, but not much else:

- not every teams need to play against every other team
- some teams might not play a single match
- the number of matches between two teams need not be the same as that between other teams
- the number of home games might be different from the number of away games
- the total number of match results that you may need to process can be small, or *very large.*

The last point means that you cannot store the entire list of match results in memory, only aggregate statistics.

Since the statistics are fractional, make sure that you process the data using double precision.

Input

The input consists of:

- The number of matches played
- List of results for each match

Each row in the list has the form

```
home_team_id away_team_id goals_home_team goals_away_team
```

where the IDs are integers in the range [0, 29]. That is, you can assume that there are maximally 30 teams.

For instance:

```
3 8 1 0
```
encodes a 1-nil win for home team 3 against away team 8, and

```
16 1 0 0
```
encodes a draw between teams 16 and 1.

Output

For each team that played at least one match, output a row containing:

1. team id
2. win ratio (= #wins / #plays)
3. win ratio on home games. This should be -1 in case of no home games.
4. average goal difference. E.g., "-0.25" would indicate that on average the team lost by 0.25 goals per game.

All fractional output data should be truncated to 3 digits behind the decimal point.

Sample Input (1)

```
4
0 1 5 0
2 3 0 5
20 21 3 0
```

```
21 20 4 0
```

Sample Output (1)

```
0 1.000 1.000 5.000
1 0.000 -1 -5.000
2 0.000 0.000 -5.000
3 1.000 -1 5.000
20 0.500 1.000 -0.500
21 0.500 1.000 0.500
```

Sample Input (2)

```
6
0 1 5 0
1 0 3 1
0 2 2 2
2 0 4 2
1 2 2 3
2 1 8 0
```

Sample Output (2)

```
0 0.250 0.500 0.250
1 0.250 0.500 -3.000
2 0.750 1.000 2.750
```