# CAMBRIDGE INTERNATIONAL UNIVERSITY

## COURSE OF COMPUTER SCIENCE
## *LABORATORY PRACTICE n. 11*

### Exercise 1 (1 point):

Write a program which:

- Reads four integer (*not* necessarily positive) values from the keyboard, representing two fractions in the form numerator/denominator
- Computes the fraction resulting from the sum of such fractions.

Use a structure for representing every fraction. The minimization of the resulting fraction is also required.

***Example***: Assume that the four values introduced by the user are 2, -3, -1 and -6, which represent the fractions -2/3 and 1/6, respectively. Their sum is (-4+1)/6 = -3/6, and thus the program must display the result -1/2.

### Exercise 2 (2 points):

A text file contains the description of at most 100 points in the Cartesian space, one point per line, with format:

`<coord_X> <coord_Y> <coord_Z>`

Write a program in order to:

- Load the set of point into an array of structures.
- Compute the distances of all such points from the origin.
- Sort the points according to such distances.
- Write the sorted array into a second text file, one point per line, according to the format:

`<coord_X> <coord_Y> <coord_Z> <distance>`

The names of the two files should be received as command line parameters.

### Exercise 3 (2 points):

The sales of a small market have been recorded and stored into a text file made up of an unknown number of rows (the order in which the rows have been stored is also unknown). Each such row has the following format:

`<product> <category> <price>`

where:

- `<product>` is the name of the purchased product (a string with at most 30 characters).
- `<category>` is a code of exactly 3 characters, indicating the category of the product which has been bought (e.g., food, clothes, etc.). Assume a maximum of 100 distinct categories.
- `<price>` is the amount of money expended for the purchase (real value).

Write down a C program able to generate a second text file with some statistics on the purchases, divided by the products' categories (i.e., every category must appear in the file only once). More specifically, every line of this file should have the following format:

`<category> <number> <total> <average>`

where:

- `<number>` is the total number of purchases of products for the given category.
- `<total>` indicates the overall money expended in all the purchases for that category.

- `<average>` is the average amount of money used for a purchase of that kind of products.

***Example***: Assume the following contents for the input file:

```
pastaPPP fod 0.82
jeansJJJ clo 39.90
oilLLL fod 6.76
shampooSSS mis 4.30
trousersTTT clo 23.45
breadBBB fod 1.29
```

Then, the output file contents should be (the order in which the categories appear is not relevant):

```
fod 3 8.87 2.96
clo 2 63.35 31.68
mis 1 4.30 4.30
```

## Exercise 4 (2 points):

A registry office has stored into two text files (named "`births.txt`" and "`deaths.txt`", respectively) the dates of the births and deaths occurred in a small village. The format of the two files is identical:

- For each person, two consecutive file lines are used.
- The first line reports the name of the person.
- The second line gives the date (specified with format `dd/mm/yyyy`) in which that person was born or died, respectively.

Write a program which, starting from the contents of the two files, generates a third file, named "`lives.txt`", in which every line reports all the data about one person, with format:

`<name> <birth_date> <death_date>`

Assume that the number of people in the two files is limited to 1000 and that every person appearing in one file also appears in the other file. No sorting of the two files can be assumed.

***Example***: Assume the following contents for the two input files:

```
births.txt          deaths.txt

Greens              Greens
01/01/1932          15/02/1988
Whites              Blacks
19/06/1940          10/02/2000
Blacks              Whites
22/12/1942          21/04/1996
```

Then, the following file should be generated (the order of the lines is not relevant):

```
Greens  01/01/1932  15/02/1988
Blacks  22/12/1942  10/02/2000
Whites  19/06/1940  21/04/1996
```

## Exercise 5 (2 points):

The flights performed by an airline are stored in a file, one flight per file line, with the following format:

`<code> <source> <destination> <departure> <arrival>`

where:

- `<code>` is a string consisting of 5 characters, uniquely identifying the flight.

- `<source>` and `<destination>` are the cities connected by the flight (strings made up of at most 20 characters).
- `<departure>` and `<arrival>` indicate the flight departure and arrival times, and they are represented as real values (e.g., 7.15 corresponds to 7 hours and 15 minutes).

It is necessary to write a Python program which, once the contents of the previous file has been loaded, reads from the keyboard the name of two cities and prints the full list of flights connecting the two cities *directly or even with one intermediate stop*. In the latter case, the take-off time of the second flight must be compatible with (i.e., sub-sequent to) the landing time of the first flight.
The overall number of flights in the file is limited to 150.

***Example***: let the contents of the input file be:

```
AZ0A3 TOR ROM 17.00 18.00
AZ0A4 TOR ROM 19.00 20.00
AZ0B3 ROM PAL 17.30 18.30
AZ0B4 ROM PAL 19.30 20.30
AZ0C3 TOR PAL 17.45 19.45
AZ0C4 TOR PAL 19.45 21.45
```

Assuming that the two cities specified by the user are `TOR` and `PAL`, the program must then display the following messages:

```
Flight with one stop:
AZ0A3 TOR ROM 17.00 18.00
AZ0B4 ROM PAL 19.30 20.30
Direct flight:
AZ0C3 TOR PAL 17.45 19.45
Direct flight:
AZ0C4 TOR PAL 19.45 21.45
```