Udacity Capstone Project

Céline Giesser, June 2022

Project

The CoffeeRoulette chooses randomly groups of two persons in a team, that will then take a coffee together. The idea behind is to bring people together, who maybe wouldn't meet otherwise.

Structure

The structure consists essentially in three steps:

Participants input

The input comes from the ParticipantsList.txt file, which is an output from an invitation system. It contains the last and first names of the persons, an answer if the person wants to participate to the CoffeeRoulette and information from the invitation system. The format in the file is the following:

```
From;Subject;Received;Size;Categories
BOHR Niels;Yes: Coffee roulette KW49;Mi. 12:08;26 KB;
LAPLACE Pierre;No: Coffee roulette KW49;Mi. 11:26;26 KB;
…
```

If the application has already been run, another .txt file, called NameList.txt, exists. It contains a participant number, their first and last names and a history of persons (number) already met in the previous CoffeeRoulette. The format of this file is the following:

```
…
13 Niels BOHR 14 11 21
14 Robert BROWN 7 4 0
…
```

These files readings generates in the code an object ListOfPersons, which is a vector of objects Person with number, first and last names, a history of person numbers already met and a Boolean, telling if the person is participating or not.

Generation of the groups of persons

From the list of persons coming from the files, it generates a list of persons participating. In this list, it chooses randomly a first person. Based on this person's history of persons already met, it will generate a new list with the persons not met and randomly choose a second person in that list.

If the first person already met all the other participants, the “oldest” person in the history will be set as second person.

If the number of persons is odd, the last remaining person will be added to the last pair, creating a group of three persons.

The history of persons met is updated for the participating persons.

In the code, this generates a list of persons by group, being a succession of the generated pairs.

Groups output

Finally, the groups are written in GroupList.txt on the following manner:

```
…
Niels BOHR meets Pierre LAPLACE
Anders CELSISUS meets Galilei GALILEE
…
```

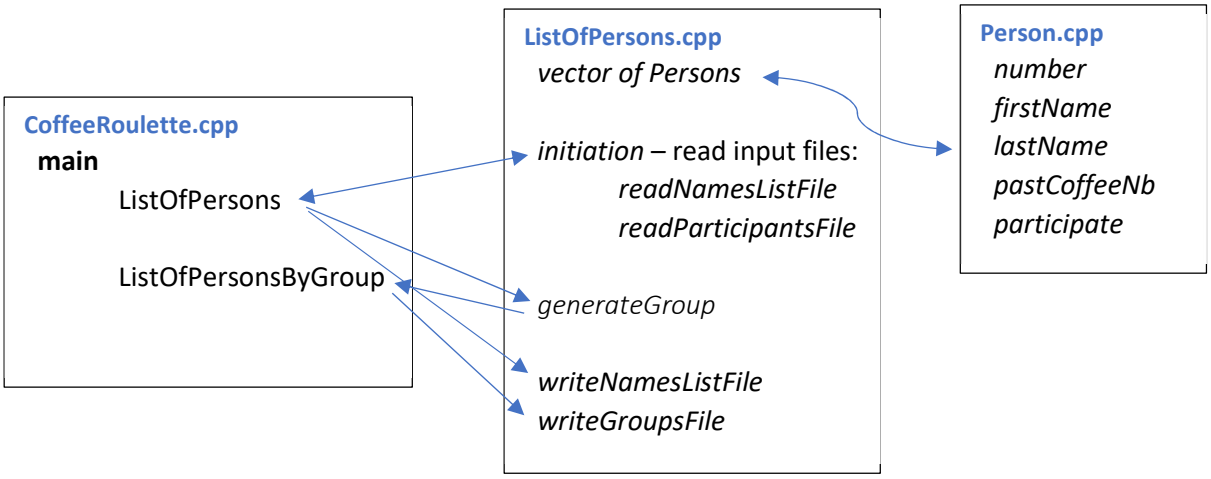
A file NamesList.txt is generated or updated with the new list of persons and the updated history of the persons already met. If the length of the history is higher than 80% of the total number of persons, only the three last persons met will be written into the file. This avoids that the persons always meet in the same row (given by the history list) once they have met most of the persons.

Code walkthrough

In CoffeeRoulette.cpp, the main function:

- initiates an object of ListOfPersons, which reads the input files
- calls the ListOfPersons’s function generateGroup, which generates the groups of persons
- calls the two ListOfPersons’s functions, which update the two output files

The code contains two classes: ListOfPersons in ListOfPersons.cpp and Person in Person.cpp.



Build instructions

- Make a build directory in the top directory: `mkdir build && cd build`
- Compile: `cmake .. && make`
- Run it: `./CoffeeRoulette`

Rubric points

- Loops, functions:
  - The project demonstrates an understanding of C++ functions and control structures
  - The project reads data from a file and process the data, or the program writes data to a file.
- Object oriented programming
  - The project uses Object Oriented Programming techniques.
  - Classes use appropriate access specifiers for class members.
  - Class constructors utilize member initialization lists.
  - Classes abstract implementation details from their interfaces.
  - Classes encapsulate behavior.
- Memory management – most of the points
  - The project makes use of references in function declarations.
  - The project uses scope / Resource Acquisition Is Initialization (RAII) where appropriate.
  - The project follows the Rule of 5.
  - The project uses move semantics to move data, instead of copying it, where possible.