**Realization of an offline social network using C++**

**Author: Marco Sento**

This project consists in the implementation of an offline social network as requirements for the Algorithm Exam at Politecnico di Torino.

This is the core of a social network, able to store many different information about the users with the minimum amount of resources, adopting an extensive use of the Standard Template Library in its 4000+ lines of code.

The social network is able to manage three different typologies of users, performing almost 30 different operations that spans from users’ management to statistical data analysis.

Many checks about the data inserted have to be performed in order to guarantee the integrity of the system.

Text

Description automatically generated

**Social Network requirements:**

The goal is to realize a social network able to satisfy the following requests in the most possible flexible way.

It must provide to the user the capability of:

1. View the ID of all the users, for all the types that are:
   1. Simple user (semplici): able to have friends, acquaintance, parents, partner, son, jobs and they can be part of some groups.

They can also post messages and express like and dislike.

* 1. Enterprise user (azienda): able to have employees and corporates, can also post messages and express like and dislike.
  2. Group user (gruppi): able to have subscribers.

1. View detailed information of each user:

2.1) The ID, the name and surname, the b-day and all the additional info.

2.2) The number of like and dislike expressed.

2.3) The number of friends and acquaintances.

2.4) The relationship status.

2.5) The names of the parents and of the sons.

2.6) The employee status.

2.7) The ownership of a social network groups.

1. Manage individual users:

3.1) Add, delete or change the information of a specific user on the base of its ID

1. Manage relationships between users:

4.1) Add or remove a friend.

4.2) Add or remove an acquaintance.

4.3) Add or remove a parent.

4.4) Add or remove a partner.

4.5) Add or remove a son.

4.6) Add or remove a job.

4.7) Add or remove a group membership.

1. Manage messages and opinions:

5.1) add or delete a message of a user.

5.3) modify a message and its reactions of a user.

5.4) view the messages of a user and their reactions.

1. View statistical functions about:

6.1) Number of total users, number of simple users, company and group.

6.2) Number of friends and relatives for each simple user.

6.3) Number of users born after a specific date of birth.

6.4) Number of employees and number of subsidiaries for each company.

6.5) Number of users for each group.

6.6) Company with the largest number of employees.

6.7) News with the highest number of appreciations.

6.8) News with the greatest number of indignations.

6.9) User with the most friends.

6.10) User with the most knowledge.

6.11) Average age of simple users.

1. Perform researches:

7.1) Get a user's family trees.

7.2) Get family trees of all users

7.3) Search for 'lone wolves', simple users who do not actively participate in social life.

7.4) Sort companies by best ratio of positive and negative reactions to their news.

7.5) Sort companies by worst ratio of positive and negative reactions to their news.

In doing this, continuously checks are performed on the format of the data provided to guarantee their validity.

As input, 3 files must be provided, also empty, in order to store and acquire system’s data.

The name are not relevant as the program is able to identify them on the base of their format, otherwise the name are requested to the user.

The input files are:

1. A user file containing all the users in the system at the startup, provided in the following syntax:

* <ID>,utente semplice,{data di nascita:<gg-mm-yyyy>,Cognome:<surname>,Nome:<name>,<additional field>:<value>,....}
* <ID>,utente azienda,{Nome:<company name>,<additional field>:<value>,....}
* <ID>,utente gruppo,{Nome:<group name>,<additional field>:<value>,....}

1. A news file containing all the messages posted by the users, stored with the owner ID, the post time and the number of likes and dislikes, with the IDs of those like or dislike it.

* <ID>,<message>,data di nascita:<gg-mm-yyyy>,like:{ID1,..},dislike:{ID2,…}

1. A relations file that express the relationships between two user in the following form:

* <ID1>,<ID2>,<relationships>

The reserved character are: { } , :

Every user must have a unique ID, composed of an arbitrary sequence of characters.

All the modifications update these files.

**Social network structure:**

Graphical user interface, text, application

Description automatically generated

Let’s have a bottom-up inspection.

The social network can handle three different types of users: Semplice (a simple user, a person), Azienda (an enterprise) and Gruppo (a Groups of users).

Those are all inherited from the class Utente (User), that own the basic methods useful for all the other derived classes.

**Class Utente:**

Text

Description automatically generated

As it is possible to see, this class provides many useful set and get methods to:

Publish messages related to a specific user, remove it, set like and dislike to the messages of the other users and keep track of the liked or disliked messages and also manages the user’s information.

Most of them are overridden in the derived classes.

**Class** **Semplice**:

Text

Description automatically generated

It has many methods in order to manage all the requirement for a simple user.

It has to handle: friendship relations, parental relations (sons and parents, no more of 2 parents of course), partner relations (no more of one wife or husband), groups membership, work place, like and dislike (Opinions) and many other methos in order to print and remove its information.

Other methods are the get and sets to provide these functions.

**Class Azienda:**

Text

Description automatically generated

It is quite simpler, it has the capabilities to manage consociates enterprises, employees, track it’s employees and give back the number of employees or consociates the enterprises has.

It can also express opinions on someone posts.

**Class Gruppo:**

Text, timeline

Description automatically generated

Basically it manage the members of a group, naturally each group has its own name.

**Class GestioneUtenti:**

Text

Description automatically generated

This class offer the possibility to manage all the users belonging to a single type.

They are organized in a list of users object, but here are NOT instantiated so, all the functions provided are in the form of template.

It offer the possibility of manage reletions between users, of each type they will be eg. Azienda-Semplice, add and remove users in the list , get the total number of user of the instantiated type and get a particular information of a certain user.

It is interesting to notice that different functions uses the parent class Utente, in order to offer the possibility to apply a cast to the inherited classes.

It leverages on the template, no declarations are implemented.

**Class Logging:**

**Text

Description automatically generated with low confidence**

This is a crucial class.

It instantiates and manage the lists of users types as:



The provided methods perform controls after the start up but also at any changes applied to the users.

It controls the input files (that can be provided in any order, it has the capability to recognise them by its self), that have to respect the format reported in the previous input constrains.

If empty, the file are filled at run time.

It check at the start-up and at each changes if the number of parents are correct, if the information provided for each user are meaningful (there are some mandatory information and other optional inserted by the user without number limitation) and every user must have a unique ID, composed of an arbitrary sequence of characters.

It checks also if some anomalies occur, like having a parents younger of its son or two legal partners.

**Class Management:**

This is a huge class, It is the control centre of the social network, for sure it should be divided into at least two subclasses for simplicity.

Its first application is to provide an interface for the user, presenting to him the various menus and printing the results of its choices.

As a second scope, to be brief, it manage at higher level all the specification provided in the introduction; it execute all the user requests.

It deeply exploits the standard template library and the mutability concept, allowing the reuse of its functions for different parameters.

A picture containing text

Description automatically generated

One very peculiar focus should be intended to the “genealogicalTree” function.

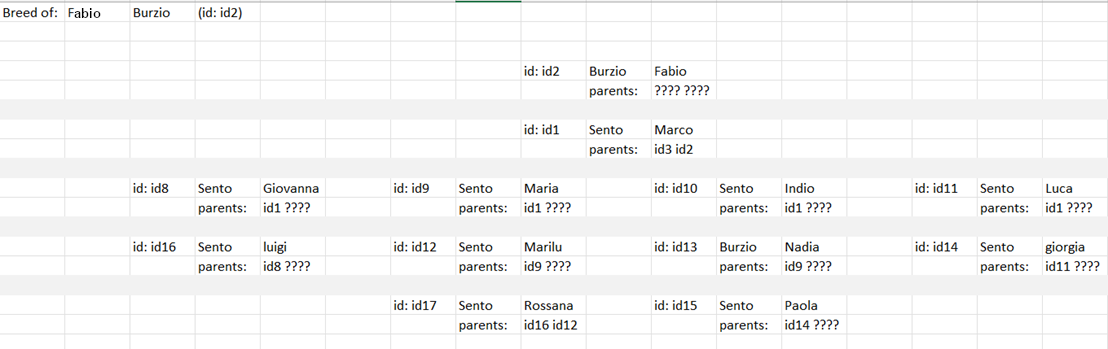
As the name suggest it is able to provide the genealogical tree of a user, iterating recursively on the users relatives it provide the complete tree from the further ancestors to the descendants.

This project is for a C++ only course and no GUI is provided but the genealogical tree is provided in a csv file for each request.

It is also notable that if a tree is already printed, the tree of a relative in the same tree won’t be printed, as it would be a copy.

The csv output involves a fancy algorithm in order to print the further ancestors at the begin (with some infos) and then going down to the descendants. Notice that the siblings are on the same level.

In absence of information about a relative “???” are inserted.



**Class Date:**

**Text

Description automatically generated**

This class provide the capability to verify the validity of a date, compare two date and define which is newer and which older, to provide the ability to make a wide range of assumptions and controls in the system.

A huge flexibility characterizes also this class, as a date can use every type of separator as wanted indeed both 11/01/2000 and 11-01-2000 are both valid.

**Main.cpp:**

For completeness the main is reported.

Graphical user interface, text, application

Description automatically generated

**Makefile:**

Text

Description automatically generated