PROJECT Nº2

INTRODUCTION

The aim of the project is to develop an application to manage the information of a **radio station**. A radio station has a name, a list of music tracks to broadcast and a top ten list of the music tracks preferred by the listeners and a list of **radio listeners** (users). Each **music track** is characterized by a unique integer id, its title, artist, author, album, music genre, year, number of "likes", number of "dislikes". A radio listener is someone that has a unique ID, a name, age, gender and a play list (list of music track ids). The play list of a user must have information concerning the music track. The top ten list is sorted, in descending order, by the difference between the number of "likes" and the number of "dislikes" of the music tracks. Information concerning the radio station and the users is kept updated in files. To have access to the radio station facilities a user must register himself at the application (simply fill in the user information mentioned above). The radio station registers the number of times each track is played in a given period and assigns a prize to the user with the hightest number of hits in his/her playlist.

REQUIREMENTS

The application should have the following functionalities:

- Enable a special user (administrator) to manage the list of music tracks (add, signal as unavailable and edit the music tracks).
- Enable a user to register at the radio station.
- Enable a registered user to manage his/her play list (add or remove tracks from the list).
- Enable any registered user to increment the "like"/"dislike" counter of any music track in the radio station list. Each time a "like"/"dislike" is changed the top ten list must be checked/updated.
- Generate, randomly, the set of musics played in a given period. This set may be re-initialized by the administrator.

Any user may

- Consult the top ten list of the radio station.
- Search the music track list of the station by the following "keys": title, year, artist or music genre.
- Display the tracks of a given artist/author/year.
- Display his/her play list.

Data input/output:

- The program should be able to save and load data from CSV files.
- Use the following files to store/recover information concerning the application:
  - **radioStationMusics.csv** – stores the music tracks available at the radio station (one music per line)
  - **topTen.csv** – stores the top ten music tracks (one music per line)
  - **users.csv** – the list of users (one user per line)
  - **playListUserXXX.csv** – the playlist of the user whose ID is XXX.

METHODOLOGY

The following approach is suggested for the development of the project.

1. Identify the relevant entities involved in the domain application.
2. Encode each entity as a C++ class.
3. Define the features of each entity as data fields of the class. Define the behaviors of the entities as class methods. Provide "get methods" to inspect the data fields of each class. Provide "set methods" to change the data fields of the classes.

After this design phase a document should be submitted with your design proposal (see below for the submission details).

CSV FILES

A CSV file is a comma separated values file where the first line contains the name of the fields and the remaining lines contain the data itself. To simplify, considerer that the fields do not contain commas, line breaks or leading/trailing spaces.

GRADING

The grading for this project will take into account several criteria, namely:

- An adequate set of C++ classes to encode the domain of the application.
- An adequate choice for the classes data fields and methods (types, visibility, access methods).
- Use of STL containers and algorithms to reduce code length.
- Code readability in general, namely, identifier names (constants, variables, types, functions,...) , indentation, separation between functions.
- Comments: function header comments and in-line comments.
- Code structure and modularity.
- Correct choice of function parameters. All function parameters must be passed using the appropriate method and 'const' qualifier must be used when justifiable.
- Input validation.
- The program produces all required output; the output produced is correct and is in an acceptable format.
- All project requirements are met.

SUBMISSION

Submission of the 1st part

The first submission concerns the definition of the classes to implement. This document should contain de class declarations and a brief description of their properties and methods. The document should be uploaded to Moodle no latter than **23:55** of the **30th of April/2013**. The file must be in PDF format and attending to the following name format **TxGyy.pdf**, where **x** and **yy** represent the class and team numbers, as specified below for the final submission.

Final Submission

- Create a folder named **TxGyy** (where **x** and **yy** represent, respectively, the number of the class/*turma* and the team number (for example, T2G07, for team 7 of class 2) and copy to the folder the source code of the program (**only the files with extensions .cpp** or **.h**).
- Compact the files into a file named **TxGyy.zip** or **TxGyy.rar** and submit the file through the link available at the course page, at Moodle/FEUP. Add a **bin directory** with the executable file and the **.csv** files required to test the program.
- **The code must compile without errors in the actual development environments available at the classroom.**
- **Each working group must submit only one project.** The group may resubmit the project several times, but only the last submittal will be graded and will be used to determine if the project is late.
- **Submission must be done before 23:55 of the 26th of May/2013.**