Practical Assignment 1

Implementation of a managament system for TV series

1. General Information

The Practical Assignment 1 applies concepts of Object-Oriented Programming and aims to implement classes based on vectors, one of the linear data structures.

This assignment must be done autonomously, by each group, until the defined deadline. It is acceptable to browse the different available information sources, but the submitted code must solely be by the group members. Any detected copied work will be penalised accordingly. The inability to explain the submitted code by any of the group members will also be penalised.

The submission deadline (in Moodle) is **March 9th at 23:59h**.

2. Concept

A TV series platform intends to obtain a system to manage the series watched by different users.

3. Assignment implementation

The compressed folder **`ESDA_2024_MTP1.zip`** contains the files needed to perform the assignment, namely:

- `TVseries.hpp`: definition of the classes representing the system (`User`, `TVSeries`, `TVSeriesManagement` and `UserManagement`).
- `TVseries.cpp`: implementation of the methods of the classes defined in `TVseries.hpp`.
- `series_test.c`: includes the main program that invokes and performs basic tests to the implemented functions.
- `user_series.txt`: text file with the information of the series watched by the users.

Important remarks:

1. **The `TVseries.cpp` file is the only file that must be changed, and all that is required is to include the implementation of each function in the code submission through CodeRunner, in Moodle.**
2. **Each attribute and method of the classes defined has additional details next to each of them in `TVseries.hpp`.**

The file `TVseries.hpp` contains the classes `User`, `TVSeries`, `TVSeriesManagement` and `UserManagement`. The first characterises each TV series, the second each user, the third allows the management of the series in the platform, while the latter defines the users registered in the platform.

Class `User`

The objects of the class `User` have the following attributes:

1) username (`username`)
2) user’s name (`name`)
3) *string* with the user’s country (`country`)
4) string vector with the user’s favourite TV series genres (`favoriteGenres`)
5) vector with the series watched by the user (`watchedSeries`)
6) integer vector with the ratings given by the user (`ratings`); i.e., the rating of the series at index 0 in `watchedSeries` is at index 0
7) integer vector with the number of episodes watched of each TV series (`episodesWatched`); i.e., the number of episodes watched of the series at index 0 in `watchedSeries` is at index 0

Class `TVSeries`

The objects of the class `TVSeries` have the following attributes:

1) series title (`title`)
2) current number of seasons (`numberOfSeasons`)
3) integer vector with the current number of episodes of each season (`episodesPerSeason`); i.e., the number of episodes of season 1 is at index 0
4) series genre (`genre`)
5) float with the series rating (`rating`)
6) boolean flag to indicate if the series is or not complete (`finished`)

Class `TVSeriesManagement`

The objects of the class `TVSeriesManagement` have a pointer vector to objects of the class `TVSeries`, which represents all TV series available in the platform.

Class `UserManagement`

The objects of the class `UserManagement` have a pointer vector to objects of the class `User`, representing all the platform's registered users.

The functions to be implemented in this assignment are methods of each class.

Class `User`

1. `void displayUSerInfo() const;`
   Display the information of a user. Includes printing: username, user's name, country, favourite genres, and the series watched, along with the number of episodes watched of each series.

2. `int addRating(TVSeries *series, float rating);`
   Add a new element (rating) to the vector `ratings`*, according to the position of the series pointed by* `series` *int the vector* `watcehdSeries`*. Returns zero if successful, -1 if an error occurs, or -2 if the user has not yet watched the series.*

Class `TVSeries`

3. `int updateRating(const vector<User*>& vectorUser);`
   Update the series rating. Returns the rating calculated or -1 if an error occurs. The series rating corresponds to the average rating given by the users in `vectorUser` *who have watched the series.*

   TIPS
   Use an auxiliary vector to store the ratings given by the users that have watched the series and how many have watched it.

Class `TVSeriesManagement`

4. `int TVSeriesInsert(TVSeries* series);`
   Insert the series pointed by `series` *in the last position available in the vector* `vectorTVSeries`*. If the series already exists (has the same title), it must be updated. Returns zero if the new series is added successfully, 1 if it already exists, or -1 if an error occurs.*

Class `UserManagement`

5. `int updateWatched(string filename, TVSeriesManagement& manager);`
   Fill the vector `vectorUsers` *by reading the content of the text file* `filename`*. Returns zero if successful or -1 if an error occurs. Each line of the text file corresponds to a user and has the format described below.*

   ```
   series_title,username,number_of_episodes_watched
   ```

   Verify if the user and the series already exist. If the user does not exist, add a new element to the vector `vectorUsers` *and fill the attributes* `name` *and* `country` *with "*`Unknown`*".*

   TIPS
   To split the information in each line, use a `stringstream`.
   Other methods of the library can be used in the function implementation.

Note: The input files and test cases which will be used to evaluate the submitted functions might contain different content and include critical cases, such as invalid function parameters.

Thus, it is your own responsibility to ensure that the function parameters are properly tested to only be considered if valid.

4. Testing the function library

The library can be tested by executing the program `series_test`. There is a test per each implemented function which assesses if that function has the expected behaviour. Nevertheless, the tests are not extensive and should be considered as an indicator of an apparent accurate implementation of the expected functionality.

If all functions pass the testcases included, the program `series_test`, when executed, shall present the following result:

```
INICIO DOS TESTES

...verifica_displayUserInfo: Informação do User1 está correta (ok)
...verifica_displayUserInfo: Informação do User3 está correta (ok)
OK: verifica_displayUserInfo passou

...verifica_addRating: Serie não existe, retorno =-2 (ok)
...verifica_addRating: Serie existe, retorno =0 (ok)
...verifica_addRating: Rating da serie =4.5 (ok)
OK: verifica_addRating passou

...verifica_updateRating: Ninguém viu esta serie, retorno =0 (ok)
...verifica_updateRating: retorno de updateRating==6.25 (ok)
...verifica_updateRating: series_echo->getRating()=6.25 (ok)
OK: verifica_updateRating passou

...verifica_TVSeriesInsert: Serie não existe, retorno =-1 (ok)
...verifica_TVSeriesInsert: Inserir Serie, retorno =0 (ok)
...verifica_TVSeriesInsert: Serie foi inserida com sucesso na ultima posicao do vetor
tvseriesManager (ok)
OK: verifica_TVSeriesInsert passou

...verifica_updateWatched: Ficheiro não existe, retorno =-1 (ok)
...verifica_updateWatched: Inserir Serie, retorno =0 (ok)
...verifica_updateWatched:  Numero  de  episodios  do  user  emily_c  foi  inserido
corretamente
=24 (ok)
...verifica_updateWatched:  Numero  de  episodios  do  user  carlitos  foi  inserido
corretamente =10 (ok)

OK: verifica_updateWatched passou

FIM DOS TESTES: Todos os testes passaram
```

5. Development tools

Using an IDE or Visual Studio Code for the assignment development is recommended as it allows for more effective debugging. Information on the usage of Visual Studio Code can be found in a short tutorial in Moodle.

It is possible to implement the functions requested in this assignment directly in CodeRunner, but it is advisable to check the files provided so that the assignment context can be well understood.

6. Evaluation

The classification of this assessment is given by the evaluation of the implementation submitted by the students (automatically calculated in Moodle) and by assessing the capacity of the students to explain their work. The final work classification (MTP1) is given by:

MTP1 = Implementation x oral assessment

Implementation classification is essentially determined by additional automatic testing (for example, using larger test files). If the submitted implementation does not compile, this component will be 0%.

The oral assessment will be divided into 4 levels: 100% master the code; 75% – some flaws 40% - several flaws detected in the explanation; 0% – demonstrated serious gaps.

7. Submission

The submission is solely possible through Moodle and up until the deadline mentioned at the beginning of this document. The implemented functions must be submitted in CodeRunner, in the dedicated areas in Moodle.