Practical Assignment 2

Continued 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 in vectors, lists and queues (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 25th 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 **`EDA_2024_T1.zip`** contains the files needed to perform the assignment, namely:

- `TVseries.hpp`: definition of the classes to represent the system (`User`, `TVSeries`, `TVSeriesManagement`, `UserManagement`, `TVSeriesManagementList` e `UserManagementList`).
- `TVseries.cpp`: implementation of the methods of the classes defined in `TVseries.hpp`.
- `series_testTP2.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`.**
3. **In yellow are the changes made to the classes in relation to the first work**

The file `TVseries.hpp` contains the classes `User`, `TVSeries`, `TVSeriesManagement`, `UserManagement`, `TVSeriesManagementList and UserManagementList`. The first characterises each TV series, the second each user, the third allows the management of the series in the platform, the fourth defines the users registered in the platform, the fifth and sixth do the same as the second and third but in lists.

Class `User`

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

1) usrname (`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
8) wish queue with the series we want to see, in order of arrival (wishSeries)

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 `TVSeriesManagementList`

The objects of the TVSeriesManagementList class have a linked list of pointers to objects of the TVSeries class, representing all series available on the platform.

Class `UserManagement`

The objects **of** the class `UserManagement` have a pointer vector to objects of the class `User`, which represents all the registered users of the platform.

Class `UserManagementList`

The objects of the UserManagementList class have a linked list of pointers to objects of the User class, representing all users registered on the platform.

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

Class `TVSeriesManagementList`

1. `list<TVSeries*> seriesByCategory(string cat) const;`

   Create a list with all series in a given category. This category is determined by the cat input parameter. In case of error, it returns an empty list.

4. `int TVSeriesDelete (string title, list<User*>& vectorUser);`
   Removes a given series from the series list. This series is determined by the series title (title input parameter). Important: you also have to remove the series from the users' watchedSeries vector (of course, you also have to remove it from the episodesWatched, ratings and wishSeries vectors). Returns zero in case of success, -1 in case of error if the series does not exist in the list.

5. `list<TVSeries*> suggestsSeries(string username, string userWhoSuggests ) const;`

   Creates a list of series for a given user (username). This list is suggested by another user (userWhoSuggests). This suggestion must meet the following criteria: 1) userWhoSuggests can only suggest series that it has seen; 2) userWhoSuggests can only suggest series of categories that are from the username's favorite categories; 3) userWhoSuggests can only suggest series that the username has not yet seen. 4) if userWhoSuggests does not exist, the total list of series is used to suggest to the user. Returns an empty list if username does not exist.

Class `UserManagementList`

2. `ist<User*> seeAll(TVSeries* series);`

   Create a list of all users who have seen all episodes of a given series. This series is determined by the series input parameter. In case of error, it returns an empty list.

Class `User`

3. `int numberOfEpisodesToSee(string title, list<TVSeries*> listTVSeries );`

   Function that answers the question "How many episodes do I have to watch before reaching a certain series, following the order of the wish queue?". This series is determined by the series title (title input parameter). Returns the number of episodes I need to see, returns -1 in case of error and if the series does not exist in the queue of interest. Note: the queue of interest must remain intact at the end of the function.

Note: The input files and testcases 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_testTP2`. 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_testTP2`, when executed, shall present the following result:

```
INICIO DOS TESTES


...verifica_seriesByCategory: (Categoria incorreta) - retorno é uma lista vazia (ok)

...verifica_seriesByCategory: (Series da categoria comedia) Tamanho da lista (=3) e' o esperado (ok)

...verifica_seriesByCategory: (Series da categoria comedia) Lista das series  (=The Office - Friends - The Big Bang Theory) e' o esperado (ok)

OK: verifica_seriesByCategory passou


...verifica_seeAll: (Serie não existe) retorno é uma lista vazia (ok)

...verifica_seeAll: (Serie existe), Tamanho da lista (=2) (ok)

...verifica_seeAll: (Serie exite) Lista das series  (=mia_davis - mike124) e' o esperado (ok)

OK: verifica_seeAll passou


...verifica_numberOfEpisodesToSee: (Serie não existe) retorno=(-1) (ok)

...verifica_numberOfEpisodesToSee: (Serie Sherlock), retorno(=(=512) (ok)

OK: verifica_numberOfEpisodesToSee passou


...verifica_TVSeriesDelete: (Serie não existe) retorno=(-1) (ok)

...verifica_TVSeriesDelete: (Serie existe), Tamanho da lista (=16) (ok)

...verifica_TVSeriesDelete: (Serie exite) Foi apagada da lista (ok)

...verifica_TVSeriesDelete: (Serie exite) Foi apagada do vetor de wateched de um utilizador (ok)

OK: verifica_TVSeriesDelete passou


...verifica_suggestsSeries: (Utlizador sugestor não exite) Lista de series sugeridas  (=Money Heist - Narcos - Sherlock - Peaky Blinders) e' o esperado (ok)

...verifica_suggestsSeries: (Utlizador sugestor exite) Lista de series sugeridas  (=Money Heist - Narcos) e' o esperado (ok)

OK: verifica_suggestsSeries 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 a 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 classification of the assessment (MTP2) is given by:

MTP2 = 0.8 Implementação + 0.2 Memória

The classification of the implementation is mainly determined by additional automatic testcases A classificação da implementação é essencialmente determinada por testes automáticos adicionais (por exemplo, recorrendo a ficheiros de teste de maiores dimensões). No caso de a implementação submetida não compilar, esta componente será 0%.

A gestão de memória também será avaliada, sendo considerados 3 patamares: 100% – nenhum *memory leak*; 50% – alguns *memory leaks*, mas pouco significativos; 0% – muitos *memory leaks*.

7. Submission

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