Practical Assignment 3

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 **April 14th 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`, `UserManagementList`, `NodeUser and UserManagementTree.`
- `TVseries.cpp`: implementation of the methods of the classes defined in `TVseries.hpp`.
- `series_testTP3.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.
- `user_updateWateched.txt`: text file with information about which series each user saw.

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`, `UserManagementList`, `NodeUser and UserManagementTree`.. 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, the seventh represents a BST tree node with all registered users and, finally, the ninth represents a BST tree with all registered users.

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.

Class `NodeUser`

The objects of the NodeUser class are nodes of the tree(s) of the UserManagementTree class. They have a pointer to an object of the User class representing a user registered on the platform, and two more pointers to the nodes on the left and right in the tree, respectively.

Class `UserManagementTree`

Objects of the UserManagementTree class have a BST tree 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 `UserManagement`

1. `priority_queue<TVSeries> queueTVSeriesCategory (priority_queue <TVSeries>& pq, string cat);`

   From a classification table (priority queue) of series by rating pq, create a new table (priority queue) only for a specific category of cat series.

2. `priority_queue<TVSeries> queueTVSeries(list<TVSeries> listTV, int min);`

   Create a leaderboard (priority heap/queue) for ranking series where at least two episodes were viewable by a minimum number of min users..

> **NOTE**
> How an overload was created for the < operator based on the rating attribute of the TVSeries class
>
> bool TVSeries::operator <(const TVSeries& tv) const
>
> { return this->rating < tv.rating; }
>
> when inserting a series into the priority queue, this priority queue depends on the rating.

Classe `UserManagement`

3. `vector<User*> usersInitialLetter(NodeUser* root, char ch);`
   Creates a vector with users (username) initialized by a specific letter ch (pay attention to upper and lower case letters. for example: the users 'Pedro' and 'pcastro' are two users initialized by the letter p.

4. `list<User*> usersNotFan(NodeUser* root);`
   Create a list of users who don't usually finish series. (users who have more than two series in which they have not seen all the episodes).

5. `vector<int> usersCategoryStatistics(NodeUser* root, string cat, int perc);`
   Creates a vector with some statistics:
   1) In the 1st position of the vector, the total number of users who viewed a given series from a given cat category
   2) In the 2nd position of the vector, the total number of users who saw a minimum number of episodes of a given series of a given cat category. This minimum number is determined by the total percentage (perc) of episodes in each series.
   3) In the 3rd position of the vector, the total number of users who saw a minimum number of episodes of a given series of a given cat category. But also have the cat category as a favorite.

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_testTP3`. 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_testTP3`, when executed, shall present the following result:

```
INICIO DOS TESTES


...verifica_queueTVSeriesCategory: (Categoria incorreta) - retorno é uma fila de prioridade vazia (ok)
...verifica_queueTVSeriesCategory: (Series da categoria comedia) Tamanho da fila de prioridade (=3) e' o esperado (ok)
...verifica_queueTVSeriesCategory: (Series da categoria comedia) 1º elemento da fila de prioridade (=Friends) e' o esperado (ok)
OK: verifica_queueTVSeriesCategory passou


...verifica_queueTVSeries: (lista de series vazia) retorno é uma lista vazia (ok)
...verifica_queueTVSeries: (Lista de Series existe),  Tamanho da fila de prioridade (=5) e' o esperado (ok)
```

```
...verifica_queueTVSeries: (Lista de Series existe) 1º elemento da fila de prioridade  (=Stranger Things) e' o esperado (ok)

OK: verifica_queueTVSeries passou


...verifica_usersInitialLetter: (Nó não existe) o vetor de retorno é vazio  (ok)

...verifica_usersInitialLetter: (Nó existe), retorno(=2) (ok)

...verifica_usersInitialLetter: (Nó existe) Vetor de utlizadores como a carater inicial de m (=mia_davis - mike124) e' o esperado (ok)

OK: verifica_usersInitialLetter passou


...verifica_usersNotFan: (Nó não existe) a lista de retorno é vazia  (ok)

...verifica_usersNotFan: (Nó existe), Tamanho da lista (=4) (ok)

...verifica_usersNotFan: (Nó existe) Lista de utilizadores que não são fans (=emily_c - carlitos - rodrigo8 - teresa_santos) e' o esperado (ok)

OK: verifica_usersNotFan passou


...verifica_usersCategoryStatistics: (Nó não existe) o vetor de estatistica tem todas as posicoes a zero (ok)

...verifica_usersCategoryStatistics: (Percentagem fora dos limites) o vetor de estatistica tem todas as posicoes a zero (ok)

...verifica_usersCategoryStatistics: (Parametros ok), número de utlizadores que assistiram a pelo menos uma série de uma categoria  (=6) (ok)

...verifica_usersCategoryStatistics: (Parametros ok), número de utilizadores que assistiram a uma determinada numero de episodios de pelo menos uma série de uma categoria (=4) (ok)

...verifica_usersCategoryStatistics: (Parametros ok), número de utilizadores que assistiram a uma determinada numero de episodios de pelo menos uma série de uma categoria que gostam (=1) (ok)

OK: verifica_usersCategoryStatistics 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 (MTP3) is given by:

MTP3 = Implementation x oral assessment

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%.

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 submission of the implemented functions must be done in CodeRunner, in the dedicated areas in Moodle.