Trabalho Prático 2

Continuação da Implementação de um sistema de gestão de séries televisivas

1. Informação geral

O Trabalho Prático 2 aplica conceitos de Programação Orientada a Objetos e consiste na implementação de classes baseadas em vetores, listas e filas, uma das estruturas de dados lineares.

Este trabalho deverá ser feito de forma autónoma, por cada grupo, até à data limite estabelecida. A consulta de informação nas diversas fontes disponíveis é aceitável. No entanto, o código submetido deverá ser apenas da autoria dos elementos do grupo e quaisquer cópias detetadas serão devidamente penalizadas. A incapacidade de explicar o código submetido por parte de algum elemento do grupo implicará também uma penalização.

O prazo limite para submissão (através do Moodle) é o dia **23 de março às 23:59h**.

2. Conceito

Uma plataforma de visualização de séries televisivas pretende obter um sistema que permita gerir as séries visualizadas por diferentes utilizadores.

3. Implementação do trabalho

O arquivo comprimido **`ESDA_2024_MTP2.zip`** contém os ficheiros necessários para a realização deste trabalho, nomeadamente:

- `TVseries.hpp`: definição das classes para representação do sistema (`User`, `TVSeries`, `TVSeriesManagement`, `UserManagement`, `TVSeriesManagementList e UserManagementList`
- `TVseries.cpp`: implementação dos métodos relativos às classes definidas em `TVseries.hpp`.
- `series_test.c`: inclui o programa principal que invoca e realiza testes básicos às funções implementadas.
- `user_series.txt`: ficheiro de texto com a informação das séries visualizadas pelos utilizadores.

Notas importantes:

1. **Apenas deverá ser alterado o ficheiro `TVseries.cpp`, sendo somente necessário incluir a implementação de cada função na submissão do código em CodeRunner, no Moodle.**
2. **Cada atributo e método das classes definidas apresenta detalhes adicionais junto a cada um deles em `TVseries.hpp`.**
3. **A amarelo estão as alterações verificadas nas classes em relação ao primeiro trabalho**

O ficheiro contém as classes `User`, `TVSeries`, `TVSeriesManagement` e `UserManagement`. A primeira permite caracterizar cada série televisiva, a segunda caracteriza cada utilizador, a terceira permite a gestão das séries da plataforma e a última define os utilizadores inscritos na

plataforma.

Classe `User`

Os objetos da classe `User` têm os seguintes atributos:

1) nome de identificação utilizador (`username`)
2) nome do utilizador (`name`)
3) *string* com o país do utilizador (`country`)
4) vetor de inteiros com os géneros de séries favoritos do utilizador (`favoriteGenres`)
5) vetor com as séries visualizadas pelo utilizador (`watchedSeries`)
6) vetor de inteiros com as classificações atribuídas pelo utilizador (`ratings`); por exemplo, no índice 0, encontra-se a classificação atribuída à série na primeira posição do vetor `watchedSeries`
7) vetor de inteiros com o número de episódios visualizados de cada série (`episodesWatched`); por exemplo, no índice 0, encontra-se o número de episódios visualizados da série na primeira posição do vetor `watchedSeries`
8) fila com as series que pretendemos ver, por ordem de interesse(wishSeries)

Classe `TVSeries`

Os objetos da classe `TVSeries` têm os seguintes atributos:

1) título da série (`title`)
2) número atual de temporadas da série (`numberOfSeasons`)
3) vetor de inteiros com o número atual de episódios de cada temporada (`episodesPerSeason`); por exemplo, no índice 0, encontra-se o número de episódios da temporada
4) *string* com a identificação do género da série (`genre`)
5) *float* com a classificação da série (`rating`)
6) booleano que indica se a série se encontra ou não terminada (`finished`)

Classe `TVSeriesManagement`

Os objetos da classe `TVSeriesManagement` possuem um vetor de apontadores para objetos da classe `TVSeries`, representando todas as séries disponíveis na plataforma.

Classe `TVSeriesManagementList`

Os objetos da classe `TVSeriesManagementList` possuem uma lista ligada de apontadores para objetos da classe `TVSeries`, representando todas as séries disponíveis na plataforma.

Classe `UserManagement`

Os objetos da classe `UserManagement` possuem um vetor de apontadores para objetos da classe `User`, representando todos os utilizadores registados na plataforma.

Classe `UserManagementList`

Os objetos da classe `UserManagementList` possuem uma lista ligada de apontadores para objetos da classe `User`, representando todos os utilizadores registados na plataforma.

As funções a implementar neste trabalho correspondem a métodos definidos em cada classe.

Classe `TVSeriesManagementList`

1. `list<TVSeries*>` **`seriesByCategory`**`(string cat) const;`
   Criar uma lista com todas as series de uma determinada categoria. Essa categoria é determinada pelo parâmetro de entrada cat Em caso de erro devolve uma lista vazia.

2. `int` **`TVSeriesDelete`** `(string title,` **`UserManagementList& userManagementlist)`**

3. *Remove da lista de series uma determinada serie. Essa serie é determinada pelo titulo da serie (parâmetro de entrada title).* ***Imprtante****: também têm que ir remover a serie ao vetor* `watchedSeries dos` *utilizadores (logicamente que também têm que remover nos vetores* `episodesWatched e ratings)` *. Retorna zero em caso de sucesso, -1 em caso de erro a serie não existir na lista.*

4. `list<TVSeries*>` **`suggestsSeries`**`(string username, string userWhoSuggests ) const;`
   Cria uma lista de series para um determinado user (username). serie. Essa lista é sugerida por outro user (userWhoSuggests). Essa sugestão tem que seguir os seguintes critérios: 1) o userWhoSuggests só pode sugerir series que tenha visto; 2) o userWhoSuggests só pode sugerir series de categorias em que o username tenha visto pelo menos uma serie dessa categoria; 3) o userWhoSuggests só pode sugerir series que o username ainda não tenha visto. 4) no caso de não existir o userWhoSuggests, vamos usar a lista toda de series para sugerir ao user. Retorno de lista vazia no caso de username não existir.

Classe `User`

5. `int` **`numberOfEpisodesToSee`**`(string title, list<TVSeries*> listTVSeries );`
   Função que responde à pergunta "Quantos episódios tenho de ver antes de chegar a determinada serie, seguindo a ordem da fila de interesse?". Essa serie é determinada pelo titulo da serie (parâmetro de entrada title). Retorna o numero de episódios que preciso ver, retorna -1 em caso de erro e em caso da serie não existir na fila de interesse.

Classe `UserManagementList`

4. `list<User*>` **`seeAll`**`(TVSeries* series);`

   Criar uma lista com todos os utilizadores que viram todos os episódios de uma determinada series. Essa serie é determinada pelo parâmetro de entrada serie. Em caso de erro devolve uma lista vazia.

Nota: Os ficheiros de entrada e casos de teste em que serão avaliadas as funções submetidas poderão apresentar conteúdo diferente e incluir casos limite (por exemplo, argumentos de funções com gamas não previstas). Como tal, é sua responsabilidade garantir que os argumentos são devidamente testados de forma a aceitá-los apenas quando válidos.

4. Teste da biblioteca de funções

A biblioteca pode ser testada executando o programa `series_test`. Existe um teste por cada função a implementar e que determina se essa função tem o comportamento esperado. Note que os testes não são exaustivos. Por isso, os testes devem ser considerados apenas como um indicador de uma aparente correta implementação das funcionalidades esperadas.

Se as funções passarem nos testes unitários incluídos, o programa `series_test`, quando executado, deverá apresentar o seguinte resultado:

```
INICIO DOS TESTES
FIM DOS TESTES: Todos os testes passaram
```

5. Ferramenta de desenvolvimento

A utilização de um IDE ou do Visual Studio Code é aconselhável no desenvolvimento deste trabalho, uma vez que permite fazer depuração de uma forma mais eficaz. Poderá encontrar informações sobre a utilização do Visual Studio Code num breve tutorial disponibilizado no Moodle.

É possível implementar as funções solicitadas diretamente no CodeRunner, sendo aconselhável consultar os ficheiros fornecidos, de modo a compreender todo o contexto do trabalho a ser realizado.

6. Avaliação

A classificação do trabalho é dada pela avaliação feita à implementação submetida pelos estudantes, sendo automaticamente atribuída no Moodle, e à capacidade de os estudantes explicarem o código submetido. A classificação final do trabalho (MTP1) é dada por:

MTP1 = Implementação x avaliação oral

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 avaliação oral será dividida em 4 patamares: 100% domina o código ; 75% –algumas falhas 40 % - várias falhas detetas na explicação; 0% – demonstrou graves lacunas.

7. Submissão da resolução

A submissão é apenas possível através do Moodle e até à data indicada no início do documento. A submissão da implementação das funções deverá ser realizada através do CodeRunner, nos espaços preparados no Moodle.