Trabalho Prático 1

Implementação de um sistema de gestão de séries televisivas

1. Informação geral

O Trabalho Prático 1 aplica conceitos de Programação Orientada a Objetos e consiste na implementação de classes baseadas em vetores, 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 **09 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_MTP1.zip`** contém os ficheiros necessários para a realização deste trabalho, nomeadamente:

- `TVseries.hpp`: definição das classes para representação do sistema (`User`, `TVSeries`, `TVSeriesManagement` e `UserManagement`
- `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`.**

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`

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 `UserManagement`

Os objetos da classe `UserManagement` possuem um vetor 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 `User`

1. `void` **`displayUSerInfo`**`() const;`
   Apresenta a informação do utilizador. Inclui imprimir: nome de utilizador, nome, país, géneros favoritos, e séries visualizadas com o número de episódios visualizados.

2. `int` **`addRating`**`(TVSeries *series, float rating);`
   Adiciona uma nova entrada (classificação) ao vetor `ratings`*, de acordo com a posição da série apontada por* `series` *no vetor* `watcehdSeries`*. Retorna zero em caso de sucesso, -1 em caso de erro ou -2 caso a série não tenha sido visualizada pelo utilizador.*

Classe `TVSeries`

3. `float` **`updateRating`**`(const vector<User*>& vectorUser);`
   Realiza a atualização da classificação da série, retornando este valor, -1 em caso de erro ou 0 se ninguém vi a serie. O cálculo da classificação é realizado calculando a média baseada nas classificações dadas por todos os utilizadores do vetor `vectorUser` *que visualizaram a série.*

> **SUGESTÕES**
> Utilizar um vetor auxiliar para guardar as classificações atribuídas pelos utilizadores que visualizaram a série, registando quantos a visualizaram.

Classe `TVSeriesManagement`

4. `int` **`TVSeriesInsert`**`(TVSeries* series);`
   Insere a série apontada por `series` *na última posição do vetor* `vectorTVSeries`*. Caso a série já exista (tenha o mesmo título), deve atualizá-la. Retorna zero se a série for inserida com sucesso, 1 caso já exista ou -1 em caso de erro.*

Classe `UserManagement`

5. `int` **`updateWatched`**`(string filename, TVSeriesManagement& manager);`
   Preenche o vetor `vectorUsers` *lendo o conteúdo do ficheiro de texto* `filename`*. Retorna zero em caso de sucesso, -2 se a serie não existe ou -1 se ocorrer um erro. Cada linha do ficheiro corresponde a um utilizador e tem o formato abaixo indicado.*

   `series_title,username,number_of_episodes_watched`

   Verifica se a série existe e se o utilizador já existe. Caso o utilizador não exista e a serie exista, adiciona uma nova entrada ao vetor `vectorUsers` *e preenche os atributos* `name` *e* `country` *com "*`Unknown`*".*

> **SUGESTÕES**
> Para separar a informação presente em cada linha, utilizar uma `stringstream`.
> Outros métodos da biblioteca podem ser utilizados na implementação.

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


...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. 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.