Practical Assignment 5

Implementation of a TV series management app

1. General information

The Practical Assignment 5 applies concepts of Object-Oriented Programming and aims to implement classes based in various data structures, which shall be chosen by the students to achieve the best code efficiency possible.

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 **May 19th at 23:59h**.

2. Concept

A TV series platform intends to obtain an app to manage TV series.

3. Assignment implementation

The compressed folder **`EDA_2024_TP5.zip`** contains the files needed to perform the assignment, namely:

- `TVseries.hpp`: definition of the classes to represent the app (`TitleBasics`, `TitlePrincipals`, `TitleEpisode` e `TVSeriesAPP`).
- `TVseries.cpp`: implementation of the methods of the classes defined in `TVseries.hpp`.
- `testTP5.cpp` includes the main program that invokes and performs basic tests to the implemented functions.
- `.tsv`: text files to test the implemented functions.

Important remarks:

1. **The `TVseries.hpp` and `TVseries.cpp` files are the only files that must be changed.**
2. **Each attribute of the classes defined has additional details next to each of them in `TVseries.hpp`.**

The file `TVseries.hpp` contains the classes:

1. `TitleBasics` – to characterise each TV series,
2. `TitleEpisode` – to characterise each episode of each TV series,
3. `TitlePrincipals` – to characterise who is a part of each episode (crew or cast),
4. `TVSeriesAPP` – to define the app for the series management.

Class `TitleBasics`

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

1) alphanumeric unique identifier of the title (`tconst`)
2) type/format of the title (`titleType`); e.g., “movie”, “short”, “tvseries”, “tvepisode”, ”video”, etc
3) most popular title (`primaryTitle`)
4) original title, in the original language (`originalTitle`)
5) boolean to identify if it’s a non-adult title – 0 – or an adult title – 1 – (`isAdult`)
6) release year (`startYear`); in the case of a TV series, it represents the series start year
7) end year (`endYear`); for all other title types, represented by “\N”
8) primary runtime of the title, in minutes (`runtimeMinutes`)
9) string vector with up to three genres associated with the title (`genres`)

Class `TitleEpisode`

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

1) alphanumeric unique identifier of the episode (`tconst`)
2) alphanumeric unique identifier of the TV series the episode belongs to (`parentTconst`)
3) season number the episode belongs to (`seasonNumber`)
4) episode number of the series (`episodeNumber`)

Class `TitlePrincipals`

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

1) alphanumeric unique identifier of the episode the person was a part of (`tconst`)
2) number to uniquely identify rows for a given title (`ordering`)
3) alphanumeric unique identifier of the person (`nconst`)
4) name by which the person is most often credited (`primaryName`)
5) birth year (`birthYear`)
6) category of the job the person was in (`category`)
7) job title (`job`); if not applicable, it is represented by “\N”
8) name of the character played (`characters`); if not applicable, it is represented by “\N”

Class `TVSeriesAPP`

The class `TVSeriesAPP` has the methods to represent the application.

The functions to be implemented in this assignment are methods of the previous class described.

Important Remark

All classes already defined can be manipulated to add new attributes and methods. New classes can also be implemented. The data structures to be used in the implementation of the `TVSeriesAPP` class must be **chosen** to achieve the best code efficiency in the built solution.

The methods `addTitleBasics`, `addTitleEpisodes` and `addTitlePrincipal` must be implemented to be used in the functions `parseTitleBasics`, `parseTitleEpisodes` e `parseTitlePrincipals`, respectively, which allow filling the defined classes according to the information of the text files.

Classe **`TVSeriesAPP`**

Auxiliary functions (that must be implemented):

1. **`TVSeriesAPP`**`();`
   Constructor. Creates an object of the class `TVSeriesAPP`*.*

2. **`~TVSeriesAPP`**`();`
   Destructor. Destroys na object of the class `TVSeriesAPP`*, erasing it form memory.*

3. `void` **`addTitleBasics`**`(const TitleBasics& title);`
   Adds a new element (title) to `TVSeriesAPP`*.*

4. `void` **`addTitleEpisodes`**`(const TitleEpisodes& episode);`
   Adds a new element (episode associated with a certain title) to `TVSeriesAPP`*.*

5. `void` **`addTitlePrincipal`**`(const TitlePrincipal& principal);`
   Adds a new element (person associated to a certain episode of a certain series) to `TVSeriesAPP`*.*

Functions evaluated for code efficiency:

6. `vector<string>` **`getUniquePrincipals`**`(const string& seriesTconst) const;`
   Finds all people who have been in the episodes of a certain series, i.e., searches all different elements of the class `TitlePrincipals` *of the series with ID* `seriesTconst`*. Returns a vector with the names of the people (*`primaryName`*) found, alphabetically sorted. Returns an empty vector if an error occurs.*

7. `string` **`getMostSeriesGenre`**`() const;`
   Finds the most popular genre in the class `TVSeriesAPP`*. In case several genres have the same count of series associated with it, the name with less characters is selected. Returns the genre found or an empty string if an error occurs.*

8. `string` **`getPrincipalFromCharacter`**`(const string& character) const;`
   Finds the person who has played most times a certain character (`character`*) in the episodes. If the count results in a tie, the person's name with higher alphabetical order*

is selected. Returns the name of the person (`primaryName`) found, or an empty string if an error occurs.

9. `vector<string>` **`principalsWithMultipleCategories`**`(const string& seriesTconst) const;`
   Finds all the people who have taken different categories in the job performed in the episodes they were in of the series with ID `seriesTconst` Returns a vector with the names of the people (`primaryName`) found, alphabetically sorted. Returns an empty vector if an error occurs.

10. `int` **`principalInMultipleGenres`**`(vector<string> vGenres) const;`
    Determines the number of people who have been in the series with genres correspondent to the values in `vGenres`, and returns it.

11. `vector<string>` **`principalsInAllEpisodes`**`(const string& seriesTconst) const;`
    Finds all people who have been in all the episodes of the series with ID `seriesTconst`. Returns a vector with the names of the people (`primaryName`) found, alphabetically sorted. Returns an empty vector if an error occurs.

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 `testTP5.cpp`. 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 included testcases, the program `testTP5`, when executed, shall present the following result:

```
INICIO DOS TESTES

Base de dados pequena (5 series)

...verifica_getUniquePrincipals5: (Serie - Voetbal Inside) - Os elementos (=Chris
Woerts, Danny Vera, Gertjan Verbeek, Hans Kraay Jr., Jan Boskamp, Johan Derksen, Johnny
de Mol, Marcel van Roosmalen, René van der Gijp, Roelof Luinge, Ronald Koeman, Sensi
Lowe, Simon Zijlemans, Valentijn Driessen, Walter Trout, Wilfred Genee, Willem Feenstra,
Wim Kieft) são os esperados (ok)
...verifica_getUniquePrincipals5: (Serie - Weird Norwegian) - Os elementos (=Albert
Cerný, Antonín Hrabal, Jeroným Subrt, Victor Sotberg) são os esperados (ok)
OK: verifica_getUniquePrincipals5 passou

...verifica_getMostSeriesGenre5: (Genero com mais series) - A serie (=Talk-Show) é o
esperado (ok)
OK: verifica_getMostSeriesGenre5 passou

...verifica_principalsWithMultipleCategories5: (Serie 'Voetbal Inside') - Não existe
nenhuma pessoa (ok)
```

```
...verifica_principalsWithMultipleCategories5: (Serie 'Wish ko lang') - As pessoas
(=Jeffrey Hidalgo) são o esperado (ok)
OK: verifica_principalsWithMultipleCategories5 passou

...verifica_principalsInAllEpisodes5: (Serie: Voetbal Inside) - As pessoas (=Johan
Derksen, René van der Gijp, Wilfred Genee) são o esperado (ok)
...verifica_principalsInAllEpisodes5: (Serie: Wish ko lang) - As pessoas (=) são o
esperado (ok)
OK: verifica_principalsInAllEpisodes5 passou

...verifica_principalInMultipleGenres5: (Genero: Sport, Talk-Show) - O total (=18) são
o esperado (ok)
...verifica_principalInMultipleGenres5: (Genero: Crime, Documentary, Drama) - O total
(=175) são o esperado (ok)
OK: verifica_principalInMultipleGenres5 passou

...verifica_getPrincipalFromCharacter5: (Pessoa que interpetou de 'Self' mais vezes) -
A pessoa (=Johan Derksen) é o esperado (ok)
...verifica_getPrincipalFromCharacter5: (Pessoa que interpetou de 'Raul' mais vezes) -
A pessoa (=Kristof Garcia) é o esperado (ok)
OK: verifica_getPrincipalFromCharacter5 passou

Fim dos testes da base de dados pequena


Base de dados grande

...verifica_getUniquePrincipals: (Serie - Corruption) - Os elementos  (=Alex Butcher,
Andrew Katz, Anthony Rosario, Brett Smith, Curtis Maloney, Dan Kelly, David Kern, Emily
Dunlop, Jack Dalton, Jack Moynihan, Jenna Phelan, John DiSabito, John O'Connor, Katelyn
Johnson, Leo Dibbern, Mahak Kanjolia, Matt O'Brien, Rachel Michaelson, Rose Stella,
Ryan Imelio, Sammie Dibbern, ) são os esperados (ok)
OK: verifica_getUniquePrincipals passou

...verifica_getMostSeriesGenre: (Genero com mais series) - A serie (=Talk-Show) é o
esperado (ok)
OK: verifica_getMostSeriesGenre passou

...verifica_principalsWithMultipleCategories: (Serie Ted and Johnny) - Não existe
nenhuma pessoa (ok)
...verifica_principalsWithMultipleCategories: (Serie 'Call Me Katie') - As pessoas
(=Chris Greenwood, Dave Baines, David Foulkes, Di Evans, Donnovan Harris, Joseph
Hassell, Laura Balfour, Matt Jackson, Matt Ward) são o esperado (ok)
OK: verifica_principalsWithMultipleCategories passou

...verifica_principalsInAllEpisodes: (Serie: Lakshmi Vanthachu) - As pessoas (=S.
Sekilar, Saran Rajesh, Suresh Krishna, Vani Bhojan) são o esperado (ok)
...verifica_principalsInAllEpisodes: (Serie: Secret Diaries) - As pessoas (=Divya
Aggarwal, ) são o esperado (ok)
OK: verifica_principalsInAllEpisodes passou

...verifica_principalInMultipleGenres: (Genero: Sport, Talk-Show) - O total (=52) são
o esperado (ok)
...verifica_principalInMultipleGenres: (Genero: Crime, Documentary, Drama) - O total
(=97) são o esperado (ok)
OK: verifica_principalInMultipleGenres passou

...verifica_getPrincipalFromCharacter: (Pessoa que interpetou de 'Self' mais vezes) -
A pessoa (=Aaron Elliott) é o esperado (ok)
```

```
...verifica_getPrincipalFromCharacter: (Pessoa que interpetou de 'Judy' mais vezes) - 
A pessoa (=Elora Españo) é o esperado (ok)
OK: verifica_getPrincipalFromCharacter passou

FIM DOS TESTES: Todos os testes passaram
```

Recommendation: There are two databases available to test the implemented functions (one with five TV series and the other with 1000). We suggest that, to save time during implementation, only the one with five series is used. Thus, the tests for the 1000-database should be commented in the file `testTP5.cpp`, where the place to start (/*) and end the comment (*/) is indicated.

5. Development tools

Using an IDE or Visual Studio Code for assignment development is recommended as it allows for more effective debugging. A short tutorial in Moodle provides information on using Visual Studio Code.

6. Evaluation

The classification of this assessment is given by the evaluation of the implementation submitted by the students, also considering its code efficiency. The final classification of the assessment (TP5) is given by:

TP5 = (0.75 Implementation + 0.25 Efficiency) × OA

The classification of the implementation is mainly determined by additional **automatic testcases** (e.g., using larger test files). If the submitted implementation does not compile, this component will be 0%.

The **code efficiency** of the submitted solution will be evaluated considering the **runtimes** of all the implementations submitted by the students, which will be sorted by time in ascending order and divided into 5 levels: 100% – first 20%; 80% – second 20%; 60% – third 20%; 40% – fourth 20%; 20% – last 20%.

The oral assessment (OA) will be divided into 4 levels: 100% – masters the code; 75% – some flaws; 40% – several flaws detected in the explanation; 0% – demonstrates serious gaps.

7. Testing in server

Soon, a server will be available to test the code during development. The code submitted in this server **WON'T BE EVALUATED**. Only the submission in Moodle is valid for the evaluation.

8. Submission

The submission is solely possible through Moodle and up until the deadline mentioned at the beginning of this document. It shall be submitted as a *zip* file containing:

- the files `TVseries.hpp` and `TVseries.cpp` with the implemented functions;

- a file `autores.txt` with the name and number of the group elements.

Important remark: Only submissions with the following filename will be accepted: `T5_G<group_number>.zip`. For example: `T5_G9999.zip`.

9. Plagiarism

The submitted implementations will be analysed through a plagiarism detection program. Any copies identified will be penalised accordingly.