Trabalho Prático 4

Implementação de um sistema de gestão da Fórmula 1

1. Informação geral

O **Trabalho Prático 4** aplica conceitos de **Programação Orientada a Objetos** e consiste na implementação de classes baseadas em diferentes estruturas de dados, que serão escolhidas pelos estudantes de modo a obter a melhor eficiência possível.

Este trabalho deve ser desenvolvido de forma **autónoma**, por cada grupo, e entregue até à data-limite estabelecida.

- É permitida a consulta de informação em diversas fontes, mas o **código submetido deve ser exclusivamente da autoria do grupo**.
- Cópias detetadas serão penalizadas.
- Todos os elementos do grupo devem ser capazes de **explicar e modificar o código submetido**, caso contrário, haverá penalização.
- A submissão deve ser feita via **Moodle**, até **25 de maio, às 23:59h**.

2. Conceito

Os proprietários da **Fórmula 1** pretendem obter um sistema para gerir os pilotos, as equipas (construtores), circuitos e as corridas.

3. Implementação do trabalho

O arquivo comprimido **`ESDA_2025_T4.zip`** contém os ficheiros necessários para a realização deste trabalho, nomeadamente:

- `F1.hpp`: definição das classes para representação do sistema (`Driver`, `Constructor`, `DriverManagement`, `ConstructorManagement`, `Circuit`, `CircuitManagement`, `Race`, `RaceManagement`, `RaceManagementTree` e `ConstructorGraph`) e as estruturas `DriCons`, `DriResult`, `Qualifying`, `NodeRace` e `TransferData`.
- `F1.cpp`: implementação dos métodos relativos às classes definidas em `F1.hpp`.
- `F1_test.cpp`: programa principal para testes das funções implementadas.
- `alldrivers.csv`: ficheiro de texto com a informação de todos os pilotos.
- `allconstructors.csv`: ficheiro de texto com a informação de todos os construtores.
- `numbers.csv`: ficheiro de texto com a informação de todos números utilizados pelos pilotos.
- `allConstruDrivers.csv`: ficheiro de texto com a informação da associação de pilotos e construtores por ano.
- `allCircuits.csv`: ficheiro de texto com a informação de todos os circuitos.
- `allraces.csv`: ficheiro de texto com a informação de todas as corridas (de 2004 a 2009).
- `allresults.csv`: ficheiro de texto com a informação de todos os resultados das corridas (de 2004 a 2009).
- `qualify.csv`: ficheiro de texto com informação dos tempos de todas as voltas de

qualificação de um determinado grande prémio.

Notas importantes:

1. **Ambos os ficheiros F1.hpp e F1.cpp podem ser alterados.**
2. **Cada atributo e método das classes definidas apresenta detalhes adicionais junto a cada um deles em `F1.hpp`.**

O ficheiro `F1.hpp` contém as classes `Driver`, `Constructor`, `DriverManagement`, `ConstructorManagement`, `Circuit`, `CircuitManagement`, `Race`, `RaceManagementTree` e `ConstructorGraph` e as estruturas `DriCons`, `DriResult`, `Qualifying`, `NodeRace` e `TransferData`. A primeira permite caracterizar cada o piloto, a segunda caracteriza cada construtor, a terceira permite a gestão de todos os pilotos, a quarta permite a gestão de todos os construtores, a quinta permite caracterizar cada circuito, a sexta faz a gestão dos circuitos em lista, a sétima caracteriza uma corrida, a oitava permite a gestão em lista das corridas, a nona faz na mesma a gestão das corridas usando uma árvore e a última permite criar um grafo com as informações entre transferência de pilotos por construtores. As estruturas: a primeira associa um intervalo de tempo a um piloto, a segunda contém a informação do resultado de um piloto para uma determinada corrida, a terceira associa um piloto a um tempo de volta, a quarta é um nó da árvore que contém uma corrida e apontadores para a esquerda e para a direita e a última tem a informação (estatística) da transferência dos pilotos de um determinado construtor para outro.

Classe `Driver`

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

3. Identificador único do piloto (`driverId`)
4. Código de 3 caracteres do piloto (`code`)
5. Nome do piloto (`name`)
6. Vetor de inteiros com os números já utilizados pelo piloto (`numbers`)
7. Data de nascimento do piloto (`dateOfBirth`)
8. Nacionalidade do piloto (`nationality`)

Classe `Constructor`

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

1) Identificador único do construtor (`constructorId`)
2) Nome do construtor (`name`)
3) Nacionalidade do construtor (`nationality`)
4) Vetor de apontadores para a estrutura `DriCons`, pilotos que já representaram o construtor (`Drivers`)

Classe `DriverManagement`

Os objetos da classe `DriverManagement` possuem um vetor de apontadores para objetos da classe `Driver`, representando todos os pilotos.

Classe `ConstructorManagement`

Os objetos da classe `ConstructorManagement` possuem um vetor de apontadores para objetos da classe `Driver`, representando todos os construtores.

Classe `Circuit`

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

1) Identificador único do circuito (`circuitId`)
2) Nome do circuito (`name`)
3) Local do circuito (`location`)
4) País do circuito (`country`)
5) Altitude em que se encontra o circuito (`alt`)

Classe `CircuitManagement`

Os objetos da classe `CircuitManagement` possuem uma lista de apontadores para objetos da classe `Circuit`, representando todos os circuitos.

Classe `Race`

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

1) Identificador único da corrida (`raceId`)
2) Ano (`season`)
3) Etapa (`round`)
4) Apontador para o circuito (`circuit`)
5) Data (`date`)
6) Lista com os resultados da corrida (`listRaceResults`)

Classe `RaceManagement`

Os objetos da classe `RaceManagement` possuem uma lista de apontadores para objetos da classe `Race`, representando todos os resultados.

Classe `RaceManagementTree`

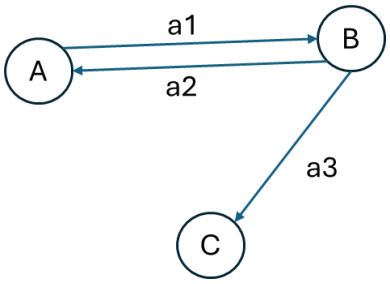
Esta classe representa uma Árvore Binária de Pesquisa (BST) organizada pelo ano (`season`) e pela etapa (`round`) das corridas. Em cada nó da árvore, todos os nós à esquerda possuem um ano (`season`) menor ou, em caso de igualdade de ano, uma etapa (`round`) menor. Da mesma forma, todos os nós à direita possuem um ano maior ou, em caso de igualdade de ano, uma etapa maior.

Os objetos da classe `RaceManagementTree` possuem um apontador para a raiz de uma árvore, onde cada nó é uma estrutura do tipo `NodeRace`. Essa árvore binária de pesquisa (BST) representa todos os resultados das corridas, organizados conforme o ano (`season`) e a etapa (`round`).

Classe `ConstructorGraph`

A classe `ConstructorGraph` representa um grafo dirigido em que os nós, são construtores e uma aresta de A para B representa informação sobre as transferências que o construtor A recebeu do construtor B. Cada aresta contém a seguinte informação:

- Diferença de pontos entre os pontos ganhos pelos pilotos no construtor B (época anterior) e no construtor A (época seguinte)
- Diferença das vitórias entre as vitórias ganhas pelos pilotos no construtor B (época anterior) e no construtor A (época seguinte)
- Número de pilotos que se transferiram do construtor B para o construtor A.



A aresta a1 corresponde às transferências que o Construtor A recebeu os pilotos vindos do Construtor B

A aresta a2 corresponde às transferências que o Construtor B recebeu os pilotos vindos do Construtor A

A aresta a3 corresponde às transferências que o Construtor B recebeu os pilotos vindos do Construtor C

Classe F1APP

A classe `F1APP` é a classe que têm de implementar.

As funções a implementar neste trabalho correspondem aos métodos definidos nas classes **`ConstructorGraph e F1APP`**.

Nota Importante

Todas as classes já definidas menos a ConstructorGraph podem ser manipuladas para adicionar novos atributos e métodos. Novas classes podem também ser implementadas. As estruturas de dados a utilizar na implementação da classe `F1APP` devem ser **escolhidas** de modo a obter a melhor eficiência na solução criada. Só podem escolher as estruturas de dados dadas nas aulas (**vectores, listas, filas, pilhas, filas de prioridade, árvores,grafos e tabelas de dispersão**). **Não podem usar** os containers do STL **maps, sets e tuples.**

Classe `ConstructorGraph`

1. `Constructor* worstVictories(Constructor* constructorA);`
   Determina o construtor que apresentou a pior diferença de vitórias com pilotos transferidos para o construtor A.

Em caso de empate no total da diferença de vitórias, é selecionado o construtor com a pior diferença de pontos.
Se o empate persistir, escolhe-se o construtor cujo nome seja alfabeticamente menor.
Retorna o nome do construtor selecionado ou NULL em caso de erro ou se não existirem transferências de pilotos para o construtor A.

2. `Constructor* moreDrivers(Constructor* constructorB);`

Determina o construtor que mais pilotos recebeu de um determinado construtor B.
Em caso de empate no total de pilotos, é selecionado o construtor com a melhor diferença de pontos.
Se o empate persistir, escolhe-se o construtor com a melhor diferença de vitórias.
Se o empate persistir, escolhe-se o construtor cujo nome seja alfabeticamente menor.
Retorna o nome do construtor selecionado ou NULL em caso de erro ou se não existirem pilotos provenientes do construtor B.

3. `vector<string> noConnection ( Constructor* constructorA );`

Determina os construtores que não tiveram qualquer ligação com um determinado construtor.
Ou seja, identifica os construtores que não receberam pilotos provenientes do construtor A e nem enviaram pilotos para esse construtor.
Retorna um vetor com os nomes desses construtores, ordenados alfabeticamente.
Retorna NULL em caso de erro ou se não existirem transferências de pilotos para ou a partir do construtor A.

4. `int updateTransfersOfYear(int year, RaceManagement &RaM);`

Cria os nós e as arestas correspondentes às transferências de pilotos num determinado ano.

-

Considera-se uma transferência no ano X se um piloto estava num construtor no ano X-1 e mudou para outro no ano X. Não existem trocas de construtor a meio de uma época.
Retorna o número total de transferências identificadas nesse ano, ou -1 se o ano for inválido.
Nota: Podem usar a estrutura SeasonData para implementar esta função.

Classe **F1APP**

Funções que não podem alterar e que têm de implementar:

F1APP();
Construtor. Cria um objeto da classe **F1APP**.

As seguintes funções são as funções que além de serem avaliadas quanto à sua correçãotambém serão avaliadas quanto à sua eficiência:

5. `void updateF1APP(DriverManagement &drM, ConstructorManagement &coM, CircuitsManagement &ciM, RaceManagement &raM);`
Preenche a classe F1APP indo buscar informação às diversas classes disponíveis.

6. `Driver* mostRaceFinish(int yearA, int yearB);`
   Determina o piloto que completou o maior número de corridas consecutivas sem abandonos, num determinado intervalo de tempo (entre os anos YearA e YearB, inclusive).
   Em caso de empate, é selecionado o piloto cujo nome seja alfabeticamente menor.
   Retorna um apontador para o piloto selecionado, ou NULL se os anos forem inválidos.

7. `Constructor * mostRaceNotPole(int yearA, int yearB);`
   Determina o construtor que venceu mais corridas sem partir da pole position, num determinado intervalo de tempo (entre os anos YearA e YearB, inclusive).
   Em caso de empate, é selecionado o construtor cujo nome seja alfabeticamente menor.
   Retorna um apontador para o construtor selecionado, ou NULL se os anos forem inválidos.

8. `string poleToWin();`
   Determina o nome do circuito com maior rácio entre de vitórias conseguidas partindo da pole e o número de corridas. Em caso de empate, é selecionado o circuito cujo nome seja alfabeticamente menor.
   Retorna o nome do circuito selecionado.

9. `list<string> pointsWidthoutWon(Circuit* cir);`
   Identifica os pilotos que, num determinado circuito, nunca venceram uma corrida, mas pontuaram sempre que terminaram.
   (O sistema de pontuação encontra-se definido no Trabalho 2.)
   Retorna uma lista com os nomes dos pilotos, ordenada alfabeticamente por ordem crescente.
   Retorna uma lista vazia se o circuito for inválido.

10. `vector<pair<Driver*,int>> classificationBySeason(int season);`
    Cria a classificação final do campeonato mundial de pilotos para uma determinada época.
    Retorna um vetor ordenado por pontuação, do maior para o menor, onde cada elemento é um par composto por um apontador para o piloto e a respetiva pontuação.
    Retorna um vetor vazio se a época (season) for inválida.

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 `F1_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 `F1_test`, quando executado, deverá apresentar o seguinte resultado:

```
INICIO DOS TESTES

Importou: 861 drivers
Importou: 212 constructors
```

```
Importou: 77 cuircuits
Importou: 107 races
Importou: 2213 resultados

(0) McLaren -> BMW Sauber( 45,4,3) | Williams( -24,-2,2) | Toro Rosso( 57,10,4)
(1) BMW Sauber -> McLaren( -12,-1,1) | Williams( -47,0,3) | Toro Rosso( 18,7,2)
(2) Williams -> McLaren( -34,-6,2) | Renault( -6,1,2)
(3) Renault -> Toro Rosso( 68,17,6) | McLaren( -23,-7,4)
(4) Toro Rosso -> Williams( -68,-7,2)

...verifica_worstVictories(nó McLaren ):  Construtor (=Toro Rosso) (ok)
OK: verifica_worstVictories passou


...verifica_moreDrivers(nó Williams ):  Construtor (=BMW Sauber) (ok)
OK: verifica_moreDrivers passou


...verifica_noConnection(nó BMW Sauber ):  vetor  (=Renault) (ok)
OK: verifica_noConnection passou


...verifica_updateTransfersOfYear( ano 2009):  Total de Construtores com transferências  (=4)
(ok)
...verifica_updateTransfersOfYear(      ano      2009):           A      Braw      recebeu      da
(construtor:pontos:vitorias,numerode pilotos)(=Honda:-153:-7:2) (ok)
OK: verifica_updateTransfersOfYear passou


...verifica_mostRaceFinish(2004-2010): Piloto (=Nick Heidfeld) (ok)
OK: verifica_mostRaceFinish passou


...verifica_pointsWidthoutWon(Albert    Park    Grand    Prix    Circuit):    Pilotos    (=Fernando
Alonso-Sébastien Buemi) (ok)
OK: verifica_pointsWidthoutWon passou


...verifica_poleToWin(): Circuito (=Circuit de Barcelona-Catalunya) (ok)
OK: verifica_poleToWin passou


...verifica_mostRaceNotPole(2004-2010): Construtor (=McLaren) (ok)
OK: verifica_mostRaceNotPole passou


...verifica_classificationBySeason(2005):  Classificação  (=Fernando  Alonso:133  pontos--Kimi
Räikkönen:112  pontos--Michael  Schumacher:62  pontos--Juan  Pablo  Montoya:60  pontos--Giancarlo
Fisichella:58 pontos--Ralf Schumacher:45
pontos--Jarno  Trulli:43  pontos--Rubens  Barrichello:38  pontos--Jenson  Button:37  pontos--Mark
Webber:36   pontos--Nick   Heidfeld:28   pontos--David   Coulthard:24   pontos--Felipe   Massa:11
pontos--Christian    Klien:9    pontos--Jacques    Villeneuve:9    pontos--Tiago    Monteiro:7
pontos--Alexander Wurz:6 pontos--Narain Karthikeyan:5 pontos--Christijan Albers:4 pontos--Pedro
de  la  Rosa:4  pontos--Patrick  Friesacher:3  pontos--Antônio  Pizzonia:2  pontos--Takuma  Sato:1
pontos--Vitantonio  Liuzzi:1  pontos--Anthony  Davidson:0  pontos--Ricardo  Zonta:0  pontos--Robert
Doornbos:0 pontos) (ok)
OK: verifica_classificationBySeason passou


Importou: 861 drivers
Importou: 212 constructors
Importou: 77 cuircuits
Importou: 1125 races
Importou: 26668 resultados

Fim do update
fim question 6
fim question 7
fim question 8
fim question 9
fim question 10
Tempo : 0.203

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.

6. Avaliação

A classificação do trabalho será baseada em:

1. **Implementação**: avaliada automaticamente.
    - Se não compilar, a nota será **0**.
    - Testes adicionais serão aplicados.
2. A **eficiência** das soluções submetidas será avaliada tendo em consideração os **tempos** das funções 5 até à 10 que serão organizadas por ordem crescente de tempo e divididas em 5 patamares:
    - 100% – primeiros 20%;
    - 80% – segundos 20%;
    - 60% – terceiros 20%;
    - 40% – quartos 20%;
    - 20% – últimos 20%.

    Nota: tempo total = função 5 + 65 x função 6 + 65 x função 7 + 100 x função 8 + 77 x função 9 + 144 x função 10

3. **Avaliação Oral**: explicação e capacidade de modificar o código.
    - **100%**: Domina totalmente o código
    - **75%**: Algumas falhas
    - **50%**: Mais ao menos
    - **25%**: Muitas falhas
    - **0%**: Graves lacunas

A nota final (**MTP4**) é dada por:

MTP4= (0,75 x Implementação + 0,25 x Eficiência) x Avaliação oral

7. Submissão da resolução

A submissão é apenas possível através do Moodle e até à data indicada no início do documento. Deverá ser submetido um ficheiro *zip* contendo:

- os ficheiros `F1.hpp` e `F1.cpp` com as funções implementadas;
- um ficheiro `autores.txt` indicando o nome e número dos elementos do grupo.

Nota importante: Apenas as submissões com o seguinte nome serão aceites: `T4_G<numero_do_grupo>.zip`. Por exemplo: `T4_G9999.zip` (turma 4 grupo 9999)

8. Plágio

As implementações submetidas serão analisadas num programa para deteção de plágio. Quaisquer cópias identificadas serão devidamente penalizadas.