

Universidade do Minho  
Departamento de Informática  
Licenciatura em Engenharia Informática

## Laboratórios de Informática III

### Fase II – Grupo 23

3 fevereiro 2023

Ana Filipa Cruz Pinto - a96862

Rui Pedro Castro Alves - a100699

Rui Pedro Fernandes Ribeiro - a100816

## Índice

Introdução.....	2
Organização dos Dados .....	2
Enumeração e descrição dos módulos .....	2
Esquematização dos módulos .....	3
Catálogos.....	4
Queries.....	4
Query1 .....	4
Query2 .....	5
Query3 .....	6
Query4 .....	6
Query 5.....	7
Query 6.....	7
Query7 .....	7
Query8 .....	8
Query9 .....	8
Testes .....	9
Conclusão.....	9

## Introdução

O presente relatório acompanha o projeto desenvolvido no âmbito da unidade curricular Laboratórios de Informática III. Este relatório é referente à segunda fase deste projeto, que tem como objetivos concluir as queries propostas e aperfeiçoar a sua implementação através, em parte, da modularização e encapsulamento do seu código.

## Organização dos Dados

### Enumeração e descrição dos módulos

Nesta segunda fase do trabalho, com vista a melhorar a modularidade da implementação da aplicação, alteramos a divisão do projeto, ficando este com os seguintes módulos:

- `catalogos.c` – Este módulo contém o código responsável pela leitura dos ficheiros `.csv` e armazenamento do `user`, `driver` ou `ride` lida no respetivo catálogo e, posteriormente, num catálogo genérico.
- `catalogos.h` - Header do módulo `catalogos.c`.
- `data.c` - Este módulo contém o código responsável pela formatação das datas na forma `dd/mm/aaaa`.
- `data.h` - Header do módulo `data.c`.
- `drivers.c` - Este módulo contém o código responsável pela formatação dos condutores. Ou seja, cria a estrutura de um `driver`.
- `drivers.h` - Header do módulo `drivers.c`.
- `getters.c` - Este módulo contém o código de auxílio à resolução das queries.
- `getters.h` - Header do módulo `getters.c`.
- `main.c` - Este módulo contém o código responsável pela interpretação dos comandos e pela indicação do tempo de execução do programa.
- `menu.c` - Este módulo contém o código responsável pela implementação do modo interativo.
- `menu.h` - Header do módulo `menu.c`.
- `queries.c` - Este módulo contém o código responsável pela implementação das queries.
- `queries.h` - Header do módulo `queries.c`.
- `rides.c` - Este módulo contém o código responsável pela formatação das viagens. Ou seja, cria a estrutura de uma `ride`.
- `rides.h` - Header do módulo `rides.c`.
- `testes.c` - Este módulo contém o código responsável pela invocação das funções de teste do módulo `testaqueries.c`.
- `users.c` - Este módulo contém o código responsável pela formatação dos utilizadores. Ou seja, cria a estrutura de um `user`.
- `users.h` - Header do módulo `users.c`.

- validador.c - Este módulo contém o código responsável por testar a validade de cada driver, user e ride.
- validador.h - Header do módulo validador.h.
- testaqueries.c - Este módulo contém o código responsável por testar as diferentes queries.

### Esquematização dos módulos

O esquema que se segue representa a modularidade deste projeto. Ou seja, representa a forma como os módulos se comunicam entre si.

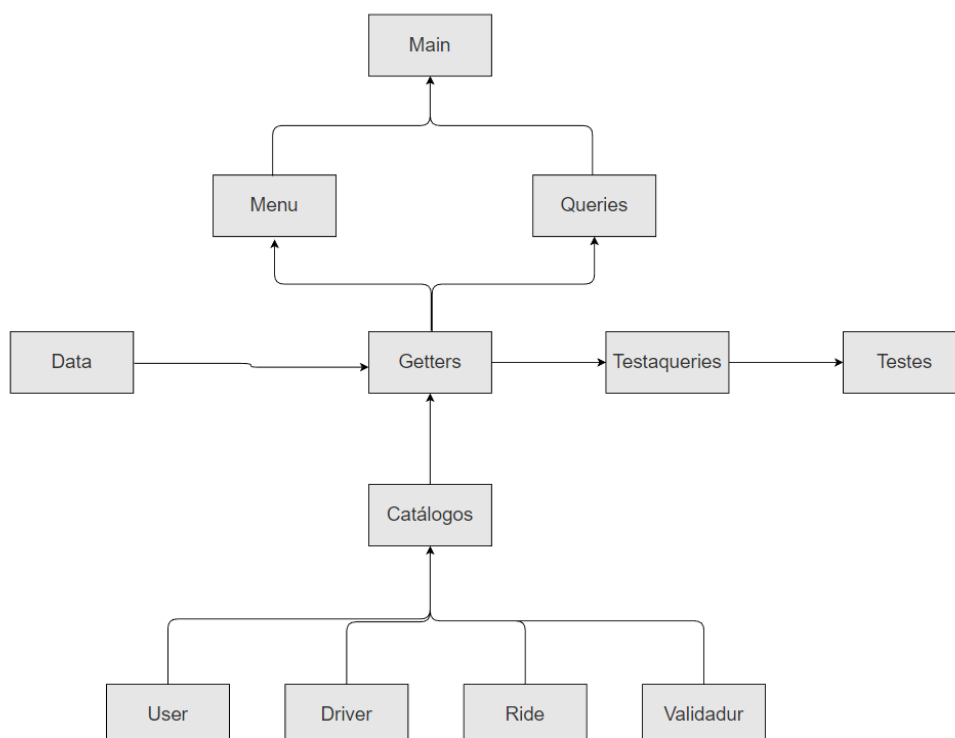
Os quadrados representam os diferentes módulos.

As setas representam os canais de comunicação entre os módulos onde a informação segue a orientação das mesmas. Na sua origem está o módulo que contém a informação e no final o módulo de destino dessa mesma informação.

Os módulos User ,Driver e Ride têm a informação retirada dos .csv. O módulo Validadur valida a informação de cada módulo anterior. Essa informação, parcialmente processada, é transmitida para o módulo Catálogos, que utiliza a informação transmitida dos módulos anteriores para criar e processar o catálogo.

Os módulos Data e Catálogos transmitem a informação por si processada ao módulo Getters que usa essa informação em funções auxiliares para a posterior invocação das queries.

Este último fornece módulo de duas partes distintas do projeto, à parte de testes às queries (Testaqueries e Testes) e à parte do menu interativo (Menu), invocação das queries (Queries) e Main.



## Catálogos

Nesta segunda parte do projeto decidimos modificar parcialmente a forma de armazenar os dados já processados dos ficheiros .csv. Para isso decidimos, no módulo catalogos, juntar os três catálogos, um para os utilizadores, um para os condutores e um para as viagens, numa única estrutura, catalogos. Esta estrutura armazena as três hashTables.

- **Catálogo de Utilizadores** - Este catálogo está implementado na função CatUser parseUser. Aqui a linha é lida e, posteriormente, é guardada, com a estrutura definida no módulo user, na hashTable do catalogoUser. Para isso, usa com key o username do utilizador.

```
g_hash_table_insert(user, get_username_user(a), a);
```

- **Catálogo de Condutores** - Este catálogo está implementado na função CatDriver parseDrivers. Aqui a linha é lida e, posteriormente, é guardada, com a estrutura definida no módulo drivers, na hashTable do catalogoDriver. Para isso, usa como key o id do condutor.

```
g_hash_table_insert(driver, get_id_driver(d), d);
```

- **Catálogo das Viagens** – Este catálogo está implementado na função CatRides parseRides. Aqui a linha é lida e, posteriormente, é guardada, com a estrutura definida no módulo rides, nas hashTables do catalogoRides. Para isso, usa como key o id da viagem.

```
g_hash_table_insert(rides, get_id_Rides(r), r);
```

## Queries

### Query1

- Na implementação desta query decidimos não fazer alterações relativamente à primeira fase do projeto por, na nossa conceção, estar implementada de uma forma correta e eficiente. Por este facto, a sua descrição é idêntica à apresentada no relatório que acompanha a primeira fase de desenvolvimento.
- Esta query tem a sua resolução dividida em 2, a sua especificação para os utilizadores e a sua especificação para os condutores.
- Para resolver a primeira, começamos por, no módulo getters, implementar as funções:
  - get\_name\_list\_user e get\_gender\_list\_user que procuram no catálogo de utilizadores o nome e o género, respetivamente, do utilizador;
  - get\_idade\_list\_User calcula a idade do utilizador, acedendo ao catálogo dos utilizadores e invocando a função build\_data do módulo data;

- `get_avaliacao_media_user` calcula a avaliação média do utilizador, acedendo ao catálogo das viagens e invocando as funções `get_user_Rides` e `get_score_user_Rides` do módulo `rides`;
- `get_numero_viagens_user` calcula o número de viagens do utilizador, acedendo ao catálogo das viagens e invocando a função `get_user_Rides` do módulo `rides`;
- `get_total_gasto` calcula o valor total gasto pelo utilizador, acedendo aos três catálogos e invocando as funções `get_driver_Rides`, `get_user_Rides`, `get_distance_Rides` e `get_tip_Rides` do módulo `rides` e a função `get_car_class_driver` do módulo `drivers`.

Já no módulo `queries`, se o estado da conta do utilizador estiver “active”, a função procura os parâmetros pedidos no catálogo dos utilizadores através das funções definidas anteriormente no módulo `getters` e faz print dos mesmos.

- Para resolver a segunda especificação, começamos por, no módulo `getters`, implementar as funções:
  - `get_name_list_driver` e `get_gender_list_driver` que procuram no catálogo de condutores o nome e o género, respetivamente, do condutor;
  - `get_idade_list_driver` que é implementada da mesma forma que a função `get_idade_list_User` diferindo no facto da procura ser feita no catálogo dos condutores;
  - `get_avaliacao_media` calcula a avaliação média do condutor, acedendo ao catálogo das viagens e invocando as funções `get_driver_Rides` e `get_score_driver_Rides` do módulo `rides`;
  - `get_numero_viagens` calcula o número de viagens do condutor, acedendo ao catálogo das viagens e invocando a função `get_driver_Rides` do módulo `rides`;
  - `get_total_euferido` calcula o valor total ganho, acedendo aos três catálogos e invocando as funções `get_driver_Rides`, `get_distance_Rides` e `get_tip_Rides` do módulo `rides` e a função `get_car_class_driver` do módulo `drivers`.

A implementação no módulo `queries` é, idêntica à da primeira especificação, com a diferença do catálogo consultado ser o dos condutores.

**Petra Pacheco;F;51;2.667;6;78.230**

Figura 1: Resultados Obtidos com `query1()`

## Query2

- Nesta query, começamos por definir, no módulo `getters`, a função `auxquerie2` que começa por criar uma nova `hashTable`. Em seguida, itera, no catálogo, o catálogo de rides. Em cada interação, se o driver da ride que está a ser analisada não for um elemento da nova `hashTable`, cria uma nova estrutura de driver e

acrescenta-o à hashTable. Caso contrário, procura a estrutura, na nova hashTable, em que o driver está guardado e atualiza os seus atributos.

- Posteriormente, através da função pré-definida `g_hash_table_values`, transforma a hashTable numa GList. Esta lista é depois ordenada com recurso à função `sort_function_driver`.
- Esta última função, `sort_function_driver`, altera os dois inputs para estruturas de drivers. Em seguida, faz a média da avaliação de cada driver, dividindo a avaliação pelo contador (ou seja, divide a soma das avaliações pelo número de viagens realizadas) e calcula a data da viagem mais recente de cada input. Por fim, comparando todos estes valores, a função retorna estes dois inputs de forma ordenada pelos critérios pedidos no enunciado da query.
- Numa fase posterior, já no módulo queries, a função `query2` começa por chamar a função `auxquerie2` com o catálogo como input. Isto para calcular a lista ordenada dos condutores. Posteriormente, intera esta lista até ao valor N e a cada interação faz print com os dados pedidos pelo enunciado.

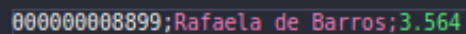


Figura 2: Resultados Obtidos com `query2()`

### Query3

- Nesta query, de forma semelhante com a `query2`, começamos por, no módulo getters, definir a função `auxquerie3`. Esta função cria uma nova hashTable, intera, no catalogo, o catálogo de rides e, em cada interação, se o user não pertencer à hashTable, cria uma estrutura de user e insere-o. Caso contrário, procura a estrutura onde este user está inserido e atualiza-o.
- Em seguida, transforma a hashTable numa GList e ordena-a segundo os critérios do enunciado, usando para isso a função `sort_function_user` em que a sua implementação em tudo se assemelha com a função `sort_function_driver` apenas diferindo na estrutura em que os inputs são, inicialmente, transformados. Ao invés de colocar os inputs numa estrutura de driver, altera-os para uma estrutura de user.
- Finalmente, no módulo queries, a função `query3` começa por chamar a função `auxquerie3` com o input catálogo para calcular a lista ordenada dos utilizadores. Posteriormente, intera essa lista até ao valor de N e a cada interação faz print dos dados pedidos pelo enunciado.

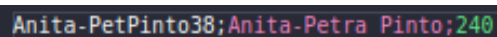


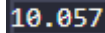
Figura 3: Resultado Obtido com `query3()`

### Query4

- De forma idêntica à `query1`, esta query foi também implementada na primeira fase de desenvolvimento deste projeto. Desta forma, a sua descrição em tudo se assemelha com a apresentada na sua primeira parte.
- Esta query, em semelhança com a query 1, começa por definir, no módulo getters, a função `get_preco_medio_city` que, acedendo aos três catálogos e

invocando as funções `get_driver_Rides`, `get_city_Rides` e `get_distance_Rides` do módulo `rides` e a função `get_car_class_driver` do módulo `drivers`, calcula o preço médio das viagens, sem gorjeta, numa determinada cidade.

- Posteriormente, no módulo `queries`, a função `query4`, acedendo aos catálogos dos condutores e das viagens, chama a função `get_preco_medio_city`, anteriormente definida, e faz print do seu resultado.



10.057

Figura 4: Resultados Obtidos com `query4()`

### Query 5

- Nesta query, começamos por definir, no módulo `getters`, a função `get_preco_medio_data` que, inicialmente, através da função `build_data`, dá as datas inicial e final. Posteriormente, acedendo aos catálogos, invoca as funções `get_driver_Rides`, `get_date_ride` do módulo `rides` e a função `get_car_class_driver` do módulo `drivers`, calcula o preço médio das viagens, sem gorjeta, num determinado intervalo de tempo.
- Em seguida, no módulo `queries`, a função `query5`, acedendo aos catálogos, chama a função `get_preco_medio_data` anteriormente definida e faz print do seu resultado.



10.039

Figura 5: Resultados Obtidos com `query5()`

### Query 6

- Nesta query, começamos por definir, no módulo `getters`, a função `get_distancia_media_city` que, inicialmente, através da função `build_data`, dá as datas inicial e final. Posteriormente, acedendo aos catálogos, invoca as funções `get_date_Rides`, `get_city_Rides` e `get_distance_Rides` do módulo `ride`, calcula a distância média percorrida, numa determinada cidade, num determinado intervalo de tempo.
- Em seguida, no módulo `queries`, a função `query6`, acedendo aos catálogos, chama a função `get_distancia_media_city` anteriormente definida e faz print do seu resultado.



6.896

Figura 6: Resultados Obtidos com `query6()`

### Query7

- Nesta query, começamos por definir, no módulo `getters`, a função `auxquerie7` que começa por criar uma nova `hashTable`. Em seguida, intera, no catalogo, o catálogo de `rides`. Em cada interação, se houver viagens na cidade indicada, se o condutor dessa viagem ainda não tiver sido considerado no problema, ou seja,



ainda não esteja inserido na nova hashTable, insere-o. Caso contrário, atualiza a avaliação desse condutor na hashTable.

- Em seguida, transforma esta nova hashTable numa GList. Para terminar a implementação desta função, através das funções `sort_function_q7` e `g_list_sort`, é feita a ordenação da GList pelos critérios enumerados no enunciado desta query.
- Finalmente, no módulo queries, a função `query7` invoca a função `auxquerie7` para obter a lista ordenada. E em seguida itera essa lista até ao valor N para obter o resultado pretendido pelo enunciado.

```
000000001488;Rita Correia;4.250
```

Figura 7: Resultados Obtidos com `query7()`

### Query8

- Nesta query, começamos por definir, no módulo getters, a função `auxquerie8` que começa por criar uma nova hashTable. Em seguida, itera, no catálogo, o catálogo de rides.
- Iterando, no catálogo, os catálogos de drivers e users, procura o condutor e o utilizador, respetivamente.
- Através de funções do módulo data, cria o ano referência subtraindo ao ano atual a idade de perfil passada como input.
- Em cada interação do catálogo de rides, se condutor e o utilizador forem do género passado como input e tiverem a conta ativa, se o ano de referência for superior ou igual ao da criação da conta, tanto do condutor como do utilizador, guarda os dados destes dois últimos.
- Em seguida, de forma idêntica à query 7, transforma a hashTable numa GList, e ordena essa lista, pelos critérios do enunciado, através das funções `g_list_sort` e `sort_function_q8`.
- Por fim, no módulo queries, a função `query8` invoca a função `auxquerie8`, incrementa-a e faz print de cada elemento.

```
000000009452;Ricardo Figueiredo;ÁlvBatista-Garcia98;Álvaro Batista-Garcia
```

Figura 8: Resultados Obtidos com `query8()`

### Query9

- Nesta query, começamos por definir, no módulo getters, a função `auxquerie9` que começa por criar uma nova hashTable. Em seguida, itera, no catálogo, o catálogo de rides.
- Através de funções do módulo data, transforma os inputs data inicial e data final em formato Data.
- Em cada interação, se a data da viagem em análise está entre a data inicial e a data final, guarda os dados dessa viagem na hashTable.
- De forma análogo às queries 7 e 8, transforma a hashTable em uma GList que depois a ordena, segundo os critérios do enunciado, através das funções `g_list_sort` e `sort_function_q9`.

- Para terminar a sua implementação, no módulo queries, a função query9 invoca a função auxquerie9, incrementa-a e imprime os seus elementos.

```
000000998847;25/12/2021;14;Lisboa;3.000
```

Figura 9: Resultados Obtidos com query9()

## Testes

Nesta segunda fase do projeto adicionamos, também módulos de testes. Estes módulos testam as queries desenvolvidas.

Para isso, o primeiro módulo é composto por um ficheiro, de nome testaqueries.c, e dois Files com ficheiros texto (.txt), um para guardar o output de testaqueries.c e outro com o output correto para cada query.

O ficheiro testaqueries.c, para cada query, invoca-a e guarda o seu resultado no ficheiro de texto de saída. Em seguida, verifica se o resultado é correto comparando o ficheiro de saída com o ficheiro valido, também de texto, que guarda o output esperado para esta invocação da query.

O segundo módulo, de nome testes.c, é implementado através de uma função main que invoca as funções de testaqueries.c e, desta forma, retorna um resultado positivo ou negativo consoante o teste ter sido realizado com sucesso ou não.

## Conclusão

Perante o problema proposto, no nosso ponto de vista, a resolução por nós implementada cumpre os principais objetivos deste projeto.

Para esta segunda fase de desenvolvimento do projeto, decidimos manter a implementação do armazenamento dos dados em hashTables por considerarmos ser um dos métodos mais eficientes e rápidos na procura dos dados para a resolução das queries propostas. Isto visto que as implementações por nós proposta têm muitas interações de procura nos catálogos.

Relativamente à primeira fase, como nos foi proposto na apresentação pelos professores presentes, melhoramos, por exemplo, o parsing dos dados, fazendo uma função genérica, carregaCatalogos, que faz parse e invoca cada um dos catálogos.

Além disso, implementamos, tal como era pedido no enunciado para esta segunda fase de desenvolvimento, a totalidade das queries, o modo de operação interativo, melhoramentos ao nível da modularidade e encapsulamento e implementação de testes.