

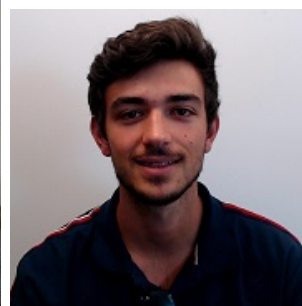
Universidade do Minho  
Licenciatura em Engenharia Informática

## Laboratórios de Informática III

Grupo 30

Filipa Rebelo - A90234      Pedro Pacheco - A61042  
Pedro Pinto - A87983

Fevereiro 2023



# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>4</b>
<b>2</b>	<b>Estruturas de dados</b>	<b>5</b>
2.1	user . . . . .	5
2.2	ride . . . . .	5
2.3	driver . . . . .	5
2.4	catalogoDrivers, catalogoRides e catalogoUsers . . . . .	6
2.5	funcAux . . . . .	6
2.6	loaders . . . . .	6
2.7	queries . . . . .	6
2.8	stats . . . . .	7
2.9	interpretador . . . . .	7
2.9.1	modo interativo . . . . .	7
2.9.2	modo batch . . . . .	7
<b>3</b>	<b>Alterações à 1ª Fase</b>	<b>8</b>
<b>4</b>	<b>Queries</b>	<b>8</b>
4.1	Query 1: Listar o resumo de um perfil registado no serviço através do seu identificador. . . . .	8
4.2	Query 2: Listar os N condutores com maior avaliação média. Em caso de empate, o resultado deverá ser ordenado de forma a que os condutores com a viagem mais recente surjam primeiro. Caso haja novo empate, deverá ser usado o id do condutor para desempatar (por ordem crescente). . . . .	8
4.3	Query 3: Listar os N utilizadores com maior distância viajada. Em caso de empate, o resultado deverá ser ordenado de forma a que os utilizadores com a viagem mais recente surjam primeiro. Caso haja novo empate, deverá ser usado o username do utilizador para desempatar (por ordem crescente). . . . .	9
4.4	Query 4: Preço médio das viagens (sem considerar gorjetas) numa determinada cidade. . . . .	9
4.5	Query 5: Preço médio das viagens (sem considerar gorjetas) num dado intervalo de tempo. . . . .	10
4.6	Query 6: Distância média percorrida, numa determinada cidade, num dado intervalo de tempo. . . . .	11
4.7	Query 7: Top N condutores numa determinada cidade, representada por city (no ficheiro rides.csv), ordenado pela avaliação média do condutor. Em caso de empate, o resultado deverá ser ordenado através do id do condutor, de forma decrescente. A avaliação média de um condutor numa cidade é referente às suas viagens nessa cidade, e não na cidade que está no seu perfil (ou seja, o mesmo condutor poderá ter médias diferentes dependendo da cidade). . . . .	11

4.8	Query 8: Listar todas as viagens nas quais o utilizador e o condutor são do género passado como parâmetro, representado por gender e têm perfis com X ou mais anos, sendo X representado por X. O output deverá ser ordenado de forma que as contas mais antigas apareçam primeiro, mais especificamente, ordenar por conta mais antiga de condutor e, se necessário, pela conta do utilizador. Se persistirem empates, ordenar por id da viagem (em ordem crescente). . . . .	12
4.9	Query 9: Listar as viagens nas quais o passageiro deu gorjeta, no intervalo de tempo (data A, data B), sendo esse intervalo representado pelos parâmetros data A e data B, ordenadas por ordem de distância percorrida (em ordem decrescente). Em caso de empate, as viagens mais recentes deverão aparecer primeiro. Se persistirem empates, ordenar pelo id da viagem (em ordem decrescente). . . . .	12
<b>5</b>	<b>Paginação</b>	<b>14</b>
<b>6</b>	<b>Conclusão e Trabalho Futuro</b>	<b>15</b>

## Lista de Figuras

1	Ilustração de uma página . . . . .	14
---	------------------------------------	----

# 1 Introdução

O presente relatório diz respeito à segunda fase de um projeto que nos foi proposto no âmbito da unidade curricular de Laboratórios de Informática III e consiste na implementação de um programa que seja capaz de processar grandes volumes de dados de forma eficiente, utilizando os princípios de modularidade e encapsulamento. Este programa serve efectivamente para guardar em memória informação relativa a Utilizadores, Condutores e Viagens numa aplicação ao estilo 'Uber'.

Para a realização deste projeto foram fornecidos três ficheiros com um considerável volume de dados - *rides.csv*, *drivers.csv* e *users.csv*, que contêm a informação sobre a qual o programa irá funcionar, o que nos levou a uma preocupação na escolha das estruturas de dados a utilizar e da sua eficiência na resposta a cada uma das *queries* propostas.

Numa fase inicial foi nos desafiado que implementássemos um *parsing* dos três ficheiros de *input* e escolhêssemos estruturas de dados que suportariam essa informação em memória. Após algum debate, a decisão caiu no uso de *hashtables* porque esta estrutura é uma maneira rápida de procurar, criar e remover informação.

Mais especificamente, as *hashtables* são tipicamente mais eficientes a encontrar informação pretendida do que árvores de procura, que têm uma complexidade linear, ou árvores binárias de procura, que têm uma complexidade logarítmica. Portanto, independentemente do tamanho do *input*, as *hashtables* têm tipicamente um tempo constante nas três mecânicas principais - procura, criação e remoção de dados.

Sabendo que o modo de interpretação de comandos nesta fase iria ser feito através da leitura de um ficheiro de texto contendo as *queries* pretendidas, foram também desenvolvidas formas de interpretar cada uma e executar a respetiva *query* com os argumentos indicados.

Nesta última fase foram nos fornecidos três ficheiros de *input* com um tamanho superior aos da primeira, o que se tornou um grande desafio pois, para que a execução do código fosse o mais rápido possível e/ou ocupasse a menor quantidade de memória, obrigou-nos a pensar em algumas alterações que teriam que ser feitas ao trabalho desenvolvido na fase anterior e que iremos abordar com mais detalhe.

## 2 Estruturas de dados

### 2.1 user

O módulo 'User' é onde definimos a estrutura que guarda a informação referente a um Utilizador mencionada no enunciado do Projeto.

Para cada utilizador, para além dos campos contidos no ficheiro *users.csv* acrescentamos os campos de 'score\_medio', 'score\_total', 'num\_viagens', 'data\_ultima\_viagem' e 'kms\_percorridos' que dizem respeito à avaliação média, avaliação total, número de viagens, data da última viagem e quilómetros percorridos de cada utilizador, de modo a facilitar a resolução das *queries* propostas no enunciado. É neste módulo que estão implementadas as funções para criação, validação e impressão de um utilizador. O módulo 'User' contém também as funções relativas aos *gets*, atualização de vários dos campos da estrutura e libertação de memória.

### 2.2 ride

A estrutura 'Ride' guarda a informação pertinente a uma Viagem. Para cada viagem, além dos campos contidos no ficheiro *rides.csv*, adicionamos o campo 'car\_class' que diz respeito à classe do veículo envolvido na viagem, para que se possa aceder a esta informação sem ter que percorrer a base de dados de Condutores e assim facilitar o algoritmo de resolução de algumas *queries*.

Este módulo possui funções relativas a *gets*, atualização de campos e libertação de memória da estrutura 'ride'. Contém ainda funções para criação, validação, impressão de uma viagem e para cálculo do custo da mesma.

### 2.3 driver

Finalmente, o terceiro módulo pilar da aplicação é aquele que guarda a informação relativa ao Condutor. Cada condutor tem, para além dos campos contidos no ficheiro *drivers.csv* e à semelhança dos outros módulos elementares, outros campos que facilitam a resolução das *queries*. Tal como os módulos supracitados, neste também são definidas várias funções para tratamento da informação e e gestão de memória relacionada com o condutor.

## 2.4 catalogoDrivers, catalogoRides e catalogoUsers

Estes três módulos são o agrupamento das estruturas principais acima mencionadas em *hashtables* e funcionam como uma base de dados em memória para a informação contida nos três ficheiros de *input*.

Estes módulos contêm funções de procura, inserção e inicialização feitas com o auxílio de funções disponibilizadas na biblioteca 'GLib'.

## 2.5 funcAux

Módulo com funções auxiliares de carácter mais geral, que fazem o tratamento de dados relativos a datas, idade e verificação de strings numéricas e que são utilizadas para mais que uma *query* e por mais do que um módulo. Contém a função *isNumber* que verifica se uma String tem apenas números, a função *stringdatavalida* que verifica se uma data está bem formatada e é válida. Além disso, a função *comparaDatas* que recebe duas datas e verifica qual é a mais antiga e outras funções que nos auxiliaram nas queries e no parsing.

## 2.6 loaders

O objetivo deste módulo é a leitura e tratamento dos dados dos três ficheiros de *input* (*rides.csv*, *drivers.csv* e *users.csv*) disponibilizados pela equipa docente. Aqui estão definidas três funções, uma por cada ficheiro, responsáveis pela leitura e validação dos dados.

## 2.7 queries

Este módulo contém funções relativas à resolução das queries propostas no enunciado. Para podermos proceder à realização de algumas *Queries*, foi necessário criar mecanismos de para se guardar e aceder aos dados que iriam posteriormente ser devolvidos como resultado. Como tal apresentam-se de seguida essas estruturas.

- **ArrayQuery**: uma estrutura usada nas *queries* 2, 3 e 7 para melhor guardar os top N condutores/utilizadores. A variável central desta *struct* é um *GArray\** no qual o *output* é guardado, mas existem outras que auxiliam o bom funcionamento da função.
- **Catalogos**: à semelhança do *ArrayQuery*, esta *struct* é composta por variáveis que ajudam ao funcionamento das *queries* 8 e 9. Neste caso, a peça central é a estrutura *Viagens* que se apresenta de seguida.
- **Viagens**: Estrutura composta por um *GList\** onde irão ser guardadas as viagens que cumpram os requisitos válidos do input das *queries* 8 e 9.

## 2.8 stats

Este módulo tem como objetivo obter estatísticas para Utilizadores, Condutores e Viagens. A determinação das diversas estatísticas calculadas foi uma escolha do grupo, e portanto foram escolhidas algumas componentes que achamos mais pertinentes. No que diz respeito aos Utilizadores, é calculado o utilizador com mais quilómetros percorridos, mais dinheiro gasto e mais viagens.

Para as Viagens, é mostrada a viagem mais longa, a mais cara e o número de viagens para cada classe de carro.

Já nos Condutores, e em espelho com os utilizadores, calculamos o condutor com mais viagens e com mais dinheiro auferido.

## 2.9 interpretador

O módulo interpretador está dividido em duas partes distintas - uma que contém as funções para o 'modo interativo' e outra para o 'modo batch'.

### 2.9.1 modo interativo

Neste modo, o utilizador é apresentado com uma série de mensagens visuais através do *standard\_output*, e de forma sucinta, tem que inserir a localização dos ficheiros de *input* e posteriormente escolher qual a informação que pretende que lhe seja mostrada.

De notar que é apenas através deste modo que é possível obter a informação relativa às estatísticas.

### 2.9.2 modo batch

Neste modo, o programa é executado com dois argumentos, o primeiro é o caminho para a pasta onde estão os ficheiros de entrada. Já o segundo corresponde ao caminho para um ficheiro de texto que contém uma lista de comandos (*queries*) a serem executados. O resultado da execução de cada comando é escrito num ficheiro individual criado pelo programa e que fica localizado na pasta 'resultados' da raiz da pasta 'trabalho-pratico'.

Este modo é executado à custa de duas funções - 'interpretadorBatch' e 'runQueries'. A primeira recebe os caminhos mencionados no parágrafo anterior assim como os apontadores iniciados para as *hashtables* de cada tipo. Esta procede ao preenchimento do restante dos caminhos para cada ficheiro de *input* e passa estas novas variáveis, assim como as restantes, à função 'runQueries'. Por sua vez é nesta função que é feito a maioria do trabalho, começando por ler os ficheiros de entrada e preencher as *hashtables*. É depois aberto o ficheiro de comandos e, para cada entrada, criado um ficheiro de *output* na pasta dedicada a estes, à medida que é lida cada linha e se vai chamando a função necessária à resolução da *query* pedida. O resultado é por fim escrito no ficheiro de saída.



### 3 Alterações à 1ª Fase

Após a apresentação e discussão com os professores do trabalho desenvolvido na fase anterior surgiu a necessidade de fazer alguns ajustes tais como:

- A reformulação das *queries* 4, 5 e 6 de forma a não usar a função 'getAllRides', evitando um acesso direto e perda do encapsulamento.
- De forma a evitar utilização desnecessária de memória foram feitas as seguintes modificações: alterados alguns tipos de variáveis como por exemplo as datas que passaram de struct tm para char\*; removida a variável 'comments' presente nas 'rides' uma vez que esta não estava a ser utilizada; alterações no que diz respeito à ordem das variáveis das *structs* tendo estas sido agrupadas por tipos (*Structure Padding*).

### 4 Queries

#### 4.1 Query 1: Listar o resumo de um perfil registado no serviço através do seu identificador.

```
char* query1(char* chave, CatDrivers drivers, CatUsers users);
```

Esta *query* recebe como argumento um identificador (*username* no caso de um Utilizador ou ID no caso de um Condutor), assim como apontadores para as *hashtables* de Utilizadores e Condutores onde o identificador será procurado.

O algoritmo de resolução passa por detectar se o identificador é composto apenas por números (condutor) ou por caracteres alfanuméricos (utilizador) e para cada caso procurar pelo mesmo na respectiva *hashtable*. Caso o encontre e o estado da conta seja "active" então é calculada a idade e impressa numa *string* de *output* a informação pedida. No caso de o identificador ser inválido, a *string* de *output* será vazia.

#### 4.2 Query 2: Listar os N condutores com maior avaliação média. Em caso de empate, o resultado deverá ser ordenado de forma a que os condutores com a viagem mais recente surjam primeiro. Caso haja novo empate, deverá ser usado o id do condutor para desempatar (por ordem crescente).

```
ArrayQuery query2(int n, CatDrivers drivers);
```

Para solucionar esta *query* desenvolveu-se uma função auxiliar que é aplicada a todos os 'drivers'. Para cada 'driver' é calculada a pontuação média chamando a função 'getDriverAvaliacaoMedia()' e é verificado se o status está 'active' chamando a função 'getDriverStatusConta()'. Se o número de 'drivers' no array resultante for menor que

'n' a função anexa o driver ao array e atualiza o driver com a pontuação mais baixa, se necessário. Se o número de motoristas no array for igual a n, a função apenas anexará o motorista se sua pontuação média for maior ou igual à pontuação mais baixa atual. Nesse caso, a função também remove o motorista com menor pontuação e atualiza as informações sobre o motorista com menor pontuação. Se dois motoristas tiverem a mesma pontuação média, aquele com a viagem mais recente é considerado como tendo a pontuação mais baixa. Se dois motoristas tiverem a mesma pontuação média e a mesma data para a viagem mais recente, considera-se que o motorista com o ID mais alto tem a pontuação mais baixa. As comparações necessárias das datas são feitas à custa da função comparaDatas().

### 4.3 Query 3: Listar os N utilizadores com maior distância viajada. Em caso de empate, o resultado deverá ser ordenado de forma a que os utilizadores com a viagem mais recente surjam primeiro. Caso haja novo empate, deverá ser usado o username do utilizador para desempatar (por ordem crescente).

```
ArrayQuery query3 (int n, CatUsers users);
```

Para resolver esta *query* recorreremos a duas funções auxiliares sendo estas a 'comparaUsers' e 'query3aux'. A 'comparaUsers' compara dois 'users' com base na distância percorrida, data da última viagem e nome do 'user' (nessa ordem de prioridade), e a 'query3aux' é uma função aplicada a cada user.

À semelhança da estratégia usada para a *query* 2, os utilizadores são adicionados livremente a um ArrayQuery dinâmico - qarray - enquanto este tiver menos que n users, tendo sempre o cuidado de manter atualizadas as variáveis que identificam aquele que é, ao momento, o mais baixo no array. Assim que estiver cheio, é necessário trocar aquele que está na posição mais baixa pelo novo, se se verificar que o novo é superior. Caso haja troca, procede-se à procura do novo Utilizador mais baixo no array.

### 4.4 Query 4: Preço médio das viagens (sem considerar gorjetas) numa determinada cidade.

```
char* query4(char *city, CatRides r);
```

Para solucionar esta *query* começamos por inicializar uma variável que vai contar o preço médio das viagens, `precM` com o valor 0.0 . É alocada memória para o qarray. A cidade passada como argumento é colocada no apontador `info`. Os apontadores `lowest` e `size` do qarray irão guardar respectivamente o preço total de todas as viagens naquela cidade, e quantas viagens ocorreram na mesma cidade. São ambos inicializados a 0. Foram utilizadas estas variáveis do qarray porém o nome não corresponde ao que esta guarda uma vez que estas variáveis já se encontravam declaradas para evitar declarar outras e consequentemente gastar mais memória. De seguida foi usado um `foreachRide` que invoca uma função auxiliar `'query4aux'` para todas as `'rides'`. Nesta função é obtida a cidade onde ocorreu a ride `r` que é comparada com o apontador `info` do qarray, onde está guardada a cidade passada por argumento. Se forem iguais é adicionado o valor da função `'CalculaCustoViagem'` aplicada à ride `r`, à variável `lowest` e é incrementado o `size`. No final liberta-se a memória da variável `cidade`. Voltando à função `'query4'` é calculado o valor final de `precM`, dividindo o preço total (`lowest`) pelo número de viagens (`size`). O print de `precM` é colocado na variável `info`. No final liberta-se a memória alocada no qarray e é devolvida a variável `info`.

#### 4.5 Query 5: Preço médio das viagens (sem considerar gorjetas) num dado intervalo de tempo.

```
char* query5(char* data_inicial , char* data_final , CatRides rides );
```

A solução desta *query* é inicializada na criação de uma função `'query5'` onde se aloca memória para o qarray. De seguida utiliza-se a função `'comparaDatas'` contida no módulo `funcAux`, para verificar se a data final é mais recente que a data inicial. Se se verificar que é verdade, a data inicial é colocada num apontador do qarray chamado `dataA` e a data final é colocada na `dataB`. O número de viagens, representado no apontador `size` do qarray, é definido como 0. O preço total, representado no apontador `lowest` do qarray, é definido como 0.0.

De seguida foi usado um `foreachRide` que invoca função auxiliar `'query5aux'` para todas as `'rides'`. Esta função verifica se a data da viagem atual é mais antiga que a data final e mais recente que a data inicial, se isto se verificar é incrementado o número de viagens e adicionado à variável `lowest` o preço da viagem atual. De volta à `'query5'` é calculada a média do preço das viagens e o print desta é colocado na variável `info`. No final liberta-se a memória alocada no qarray e é devolvida a variável `info`.

#### 4.6 Query 6: Distância média percorrida, numa determinada cidade, num dado intervalo de tempo.

```
char* query6(char* data_inicial, char* data_final, char *city,
CatRides r);
```

Esta *query* é uma espécie de junção das duas anteriores e como tal a sua solução é muito parecida.

É alocado espaço para um `ArrayQuery`, validadas as duas datas passadas por argumento, guardadas nos apontadores `dataA` e `dataB` do `ArrayQuery`, assim como a cidade no `info`. Chama-se então a função `query6aux` para cada *ride*. Aqui é verificada se a cidade da *ride* é aquela que estamos à procura e se a sua data está dentro do intervalo pretendido. Caso estas condições se verifiquem actualizam-se o acumulador `lowest` e contador `size` com os valores da distância da viagem e o numero total de viagens.

No final, são divididos um valor pelo outro, e devolvido o resultado.

#### 4.7 Query 7: Top N condutores numa determinada cidade, representada por *city* (no ficheiro `rides.csv`), ordenado pela avaliação média do condutor. Em caso de empate, o resultado deverá ser ordenado através do *id* do condutor, de forma decrescente. A avaliação média de um condutor numa cidade é referente às suas viagens nessa cidade, e não na cidade que está no seu perfil (ou seja, o mesmo condutor poderá ter médias diferentes dependendo da cidade).

```
ArrayQuery query7(int n, char* cidade, CatDrivers drivers);
```

A função `query7` cria uma estrutura `ArrayQuery` e inicializa os seus valores. Depois, é chamada a função `"foreachDriver"` que executa a função `"query7aux"` para cada `'driver'`.

A função `query7aux`, verifica o status do condutor e se o mesmo tem viagens na cidade pretendida. Caso tenha, e o `ArrayQuery` ainda tenha menos que `n` condutores, o mesmo é adicionado ao array, tendo o cuidado de ir mantendo actualizado o apontador para o condutor com o valor de avaliação média mais baixa. Assim que o array estiver cheio, para cada condutor novo que verifique as condições e tenha uma avaliação média superior àquele que está indicada como sendo o mais baixo já guardado no array, então, é removido o mais baixo, e adicionado o novo. Esta inserção é seguida de uma nova procura no array pelo condutor com a avaliação média mais baixa, de modo a manter este apontador actualizado.

O array, é depois sujeito a reordenação pelos critérios estabelecidos no enunciado. No final é devolvido o `ArrayQuery` devidamente preenchido com os `n` condutores com avaliação média mais alta.

**4.8 Query 8:** Listar todas as viagens nas quais o utilizador e o condutor são do género passado como parâmetro, representado por `gender` e têm perfis com `X` ou mais anos, sendo `X` representado por `X`. O output deverá ser ordenado de forma que as contas mais antigas apareçam primeiro, mais especificamente, ordenar por conta mais antiga de condutor e, se necessário, pela conta do utilizador. Se persistirem empates, ordenar por id da viagem (em ordem crescente).

```
Catalogos query8(char* genero , int anos ,  
CatDrivers drivers , CatRides rides , CatUsers users );
```

Para a resolução deste problema, aplicámos a segunda estrutura que criamos - o `Catalogos`. A função percorre todas as viagens na hashtable `'rides'` e para cada viagem, ela procura o `'driver'` da viagem na hashtable `drivers`. Em seguida, verifica se o género do `'driver'` é igual ao género procurado e se a conta está ativa. Caso essas condições sejam satisfeitas, a função procura o passageiro da viagem na hashtable `users` e verifica se o género do passageiro é igual ao género procurado e se a conta está ativa. Se essas condições também forem satisfeitas, a função insere ordenadamente a viagem numa lista de resultados, que é parte da struct `'catalogos'`. Esta inserção é feita à custa da função `'indexaInserir'` que calcula através da procura binária, qual o índice em que deve inserir a viagem na lista resultado, tendo como critérios de procura e ordenamento, aqueles estipulados no enunciado da *query*.

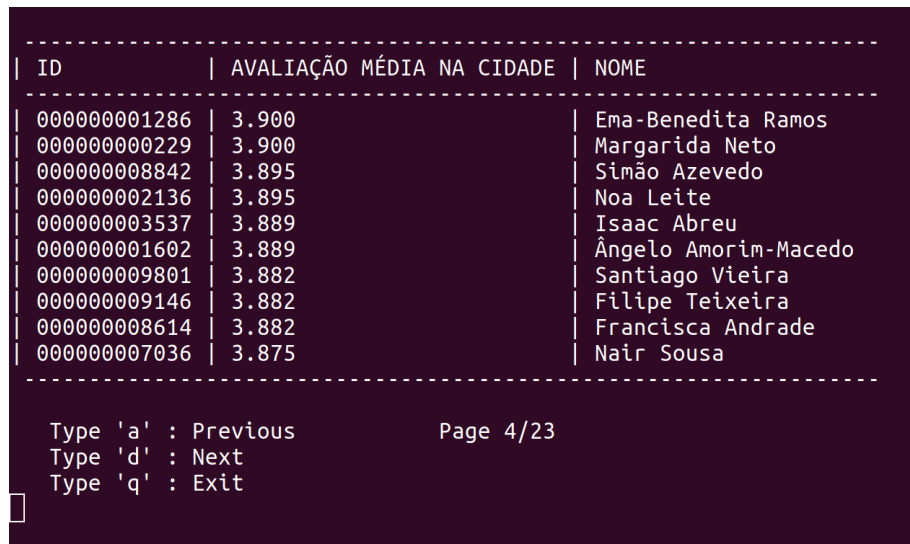
**4.9 Query 9:** Listar as viagens nas quais o passageiro deu gorjeta, no intervalo de tempo (`data A`, `data B`), sendo esse intervalo representado pelos parâmetros `data A` e `data B`, ordenadas por ordem de distância percorrida (em ordem decrescente). Em caso de empate, as viagens mais recentes deverão aparecer primeiro. Se persistirem empates, ordenar pelo id da viagem (em ordem decrescente).

```
Catalogos query9(char* data_inicial , char* data_final ,  
CatRides rides , CatDrivers drivers , CatUsers users )
```

A função verifica se as datas inicial e final são válidas e, se forem, compara a data inicial com a final para determinar se a primeira é anterior à segunda. Em seguida, a função percorre todas as viagens contidas no catálogo de viagens usando a função 'foreachRide'. Para cada viagem, a função 'query9aux' é chamada. A função 'query9aux' verifica se a viagem tem uma gorjeta e se a data da viagem está dentro do intervalo de datas especificado na consulta. Se ambas as condições forem verdadeiras, a viagem é inserida na lista de resultados. À semelhança da *query* anterior, o índice da posição em que a viagem deve ser inserida é determinado pela função 'indexainserir'. No final, a função 'query9' retorna o catálogo de viagens com as viagens que satisfazem as condições especificadas na consulta.

## 5 Paginação

Tendo em conta a quantidade de resultados que uma *query* pode gerar, surgiu a necessidade de apresentar os mesmos de forma mais organizada, simples e intuitiva. Para tal foram criadas funções paginação para o modo interativo do programa que permitem mostrar os resultados sob a forma de páginas, com dez entradas cada, que suporta a funcionalidade de avançar e recuar de página, bem como voltar ao menu principal.



ID	AVALIAÇÃO MÉDIA NA CIDADE	NOME
000000001286	3.900	Ema-Benedita Ramos
000000000229	3.900	Margarida Neto
0000000008842	3.895	Simão Azevedo
0000000002136	3.895	Noa Leite
0000000003537	3.889	Isaac Abreu
0000000001602	3.889	Ângelo Amorim-Macedo
0000000009801	3.882	Santiago Vieira
0000000009146	3.882	Filipe Teixeira
0000000008614	3.882	Francisca Andrade
0000000007036	3.875	Nair Sousa

Type 'a' : Previous      Page 4/23  
Type 'd' : Next  
Type 'q' : Exit

**Figura 1.** Ilustração de uma página

## 6 Conclusão e Trabalho Futuro

A realização deste projeto permitiu-nos aprender e colocar em prática conceitos aprendidos na unidade curricular ao longo do semestre, como modularidade e encapsulamento, bem como o uso das funções da biblioteca 'Glib' para tratamento de dados. Pudemos também cultivar a nossa capacidade de trabalho cooperativo, assim como de organização e delegação de tarefas de um projeto.

Além disso foi possível aprender a manipular grandes volumes de dados, tentando ao máximo aperfeiçoar o código de modo a este correr sem problemas de excesso de memória ou tempo.

Dado por concluído este projeto, consideramos importante fazer algumas ressalvas e realizar uma análise crítica, considerar possibilidades para trabalho futuro, e ainda, realizar uma avaliação final do trabalho realizado.

À semelhança da realização da primeira fase deste projeto, nesta segunda fase conseguimos implementar as 9 *queries* na totalidade havendo algumas dificuldades ao longo do desenvolvimento que conseguiram ser superadas. Foi possível ainda adicionar o modo interativo, paginar os resultados das *queries* e criar mecanismos de verificação de entradas inválidas nos ficheiros de *input*.

Por contraponto, nesta segunda fase foi muito complicado fazer uma boa gestão da memória e do tempo de execução. Para isto tínhamos em mente aplicar uma reestruturação da forma como guardamos algumas variáveis, nomeadamente os identificadores das viagens e dos condutores, que à data de término do trabalho se encontram no formato *char\** mas que assumimos que, se mudadas para *int* iriam diminuir o espaço ocupado. De forma semelhante, havia a intenção também de criarmos uma árvore binária balanceada de apontadores para viagens, ordenada pelas datas das mesmas que iria contribuir substancialmente para a diminuição do tempo de execução de algumas funções, pois ao invés de termos que percorrer toda a *hashtable* à procura das viagens entre duas datas em particular, bastaria pesquisar uma fracção da árvore. Porém, e de forma lamentável pelo grupo, a gestão do nosso tempo em relação a esta e outras unidades curriculares não nos permitiu esta implementação.

Devido também a esta intempérie temporal, ficou também aquém das expectativas o desenvolvimento de um programa de testes que começou por ser implementado, mas nunca terminado. A ideia passaria no entanto por criar uma nova *main* na qual seriam chamadas individualmente as funções das *queries* com um *input* fornecido interactivamente pelo utilizador, em que para cada uma seria feita uma contagem do tempo de execução. O resultado das *queries* seria guardado no ficheiro de *output*, e o tempo de execução apresentado no *std\_output*. Haveria também a possibilidade de, tal como no modo *batch*, chamar este programa de testes para um ficheiro no qual seriam previamente descritos quais os *inputs* de cada *query* e calculados os seus tempos de execução.



Uma palavra final, para a vontade que havia também de melhorar-mos esteticamente o código implementado em alguns dos módulos de modo a poder agrupar algumas repetições em funções mais genéricas como por exemplo as funções de paginação ou as de indexação das estruturas ArrayQuery.

Em maneira de conclusão, e embora não estejamos completamente satisfeitos com os resultados apresentados, consideramos que foi obtido um balanço positivo do trabalho, apesar das melhorias que poderiam ser efetuadas.