

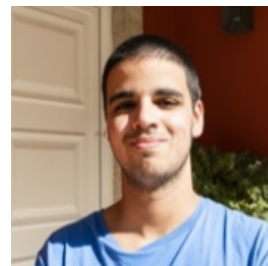
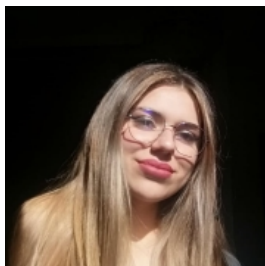


Universidade do Minho
Escola de Engenharia

Laboratórios de Informática III
Relatório de Projeto: FASE 2

Afonso Xavier Cardoso Marques 94940
Inês Daniela Alves Ferreira 97372
Luís Tiago Pereira Borges 96357

2022/2023



Contents

1	Introdução	3
2	Estrutura do projeto	4
3	Resolução das queries	7
4	Testes funcionais e de desempenho	11
5	Modo interativo	12
6	Conclusão	13
7	Anexos	14

1 Introdução

O presente relatório tem como propósito continuar a exposição do trabalho desenvolvido no âmbito da disciplina de Laboratórios de Informática III, mais especificamente o trabalho relativo à segunda fase do projeto.

Tal como já tinha sido mencionado o projeto consiste no desenvolvimento de um Sistema de Consulta de Ficheiros CSV (CSV Manager) na linguagem de programação C. Para desenvolver a aplicação foi necessário recorrer a técnicas de encapsulamento e modularidade havendo a necessidade de implementar estruturas de dados eficientes para armazenar e consultar grandes quantidades de informação.

Nesta segunda fase foram novamente usadas estruturas criadas por nós em conjunto com as pertencentes à biblioteca GLib da GNOME, nomeadamente as Hash Tables.

Serve este relatório para expor as nossas dificuldades nesta fase final e quais as decisões tomadas pelo grupo.

2 Estrutura do projeto

O trabalho que realizamos está dividido em diferentes módulos onde cada um serve um determinado propósito mantendo assim o critério de modularidade que é pedido no enunciado.

Alguns dos módulos que criamos utilizam estruturas de dados que elaboramos especificamente para guardar informação retirada dos ficheiros CSV.

Estão listadas de seguida as estruturas que criamos:

- **User:**
A estrutura é composta por sete campos, todos string (char*).
Os campos são: username, name, gender, birth_date, account_creation, pay_method e account_status.
- **Driver:**
A estrutura é composta por nove campos, todos string (char*).
Os campos são: id, name, birth_date, gender, class, plate, city, account_creation e status.
- **Ride:**
A estrutura é composta por dez campos, todos string (char*).
Os campos são: id, date, driver, user, city, distance, score_user, score_driver, tip e comment.

Estão listadas de seguida os módulos criados:

- **Users** - Módulo que serve para armazenar todos os utilizadores que são retirados do ficheiro 'users.csv'.
Usamos uma hashtable de tamanho dinâmico, que a cada username (uma string) faz corresponder uma estrutura de dados User.
A API associada a este módulo permite criar um novo utilizador, fazer uma cópia de um utilizador já existente, imprimir a informação relativa a um utilizador tanto no terminal como num ficheiro de texto, obter o username e nome de um utilizador e a função de parse.
Compete também a este módulo fazer a verificação de dados relativa ao ficheiro 'users.csv', a partir do uso de funções auxiliares.
A estrutura associada a este módulo foi criada por nós, neste caso 'User'.
Este módulo está representado pelo ficheiro 'parse-users.c'.
- **Drivers** - Módulo que serve para armazenar todos os condutores que são retirados do ficheiro 'drivers.csv'.
Semelhante aos utilizadores, usamos uma hashtable de tamanho dinâmico, que a cada id (uma string) faz corresponder uma estrutura de dados Driver.
Neste módulo, a API implementada permite criar um novo condutor, imprimir a informação relativa a um condutor tanto no terminal como num ficheiro de texto, obter o id, nome e classe de um condutor e a função de parse.
Compete também a este módulo fazer a verificação de dados relativa ao ficheiro 'drivers.csv', a partir do uso

de funções auxiliares.

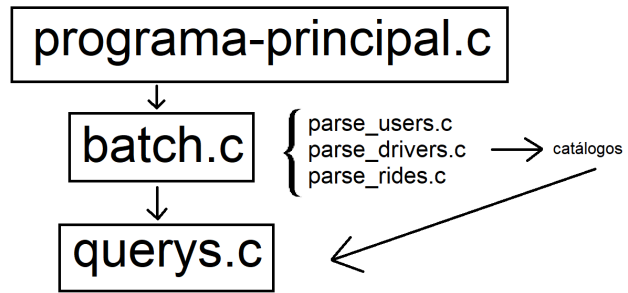
A estrutura 'Driver', criada por nós, está associada a este módulo.

A API está no ficheiro 'parse-drivers.c'.

- **Rides** - Neste módulo armazenamos todas as viagens (rides) que são retiradas do ficheiro 'rides.csv'. Semelhante aos utilizadores e condutores, usamos uma hashtable de tamanho dinâmico, que a cada id faz corresponder uma estrutura de dados Ride.
A API construída para este módulo permite criar uma nova viagem, imprimir a informação relativa a uma viagem tanto no terminal como num ficheiro de texto, obter o id, data, condutor, utilizador, cidade, distância e gorjeta de uma viagem e a função de parse.
Compete também a este módulo fazer a verificação de dados relativa ao ficheiro 'rides.csv', a partir do uso de funções auxiliares.
A estrutura 'Ride' é a que está associada a este módulo.
A API está no ficheiro 'parse-rides.c'.
- **Batch** - Este módulo é relativo ao modo batch da aplicação que estamos a desenvolver.
Aqui é possível criar catálogos de utilizadores, condutores e viagens. No entanto o mais relevante é a implementação da função 'parse-BATCH' que é responsável pela leitura de um ficheiro de texto que contém comandos para executar queries.
A API deste módulo está contida no ficheiro 'batch.c'.
- **Queries** - Este módulo contém toda a lógica por detrás da resolução das queries que são feitas à nossa aplicação.
É composto por nove funções, cada uma relativa a uma query, e funções auxiliares para ajudar na resolução.
A API deste módulo está presente no ficheiro 'querys.c'.

No que toca ao funcionamento do programa a nível geral, começamos por gerar os catálogos, guardando toda a informação relativa às viagens, aos condutores e aos utilizadores em HashTables correspondentes. Seguidamente, identificamos quais as queries pedidas no input, e quais os seus argumentos. Por fim, utilizamos as funções relativas às queries para obter os resultados pedidos.

Serve a imagem abaixo como demonstração visual simples do funcionamento do programa.



3 Resolução das queries

Nesta secção vamos fazer uma descrição detalhada da resolução de cada query implementada na fase final deste projeto. Também iremos retificar quaisquer alterações feitas às queries que já tinham sido expostas no relatório da fase anterior.

Query 1:

Na query 1, é-nos dado um input que pode corresponder a um utilizador ou a um condutor e pretende-se que devolvamos o resumo do perfil correspondente. Inicialmente, criamos uma GList (lista_rides) com a informação de todas as viagens. Após isso, dividimos a função, de forma a ter uma porção do código que trabalhe quando o input corresponde a um condutor, outra porção que responda quando o input é o de um utilizador, e uma última porção que devolva um resultado quando se trata de um input inválido.

Na parte do condutor, começamos por calcular a sua idade e inicializamos uma série de variáveis. Logo após, percorremos a lista_rides e, para cada viagem, averiguamos se o condutor em causa é o condutor sobre o qual estamos a construir o resumo. Se sim, obtemos a distância e o score dessa viagem, que são adicionados às variáveis. Adicionamos também 1 ao número de viagens e o respetivo valor ao preço, conforme a classe do carro, a distância e a gorjeta.

Finalizada a análise de todas as viagens, dividimos o score total pelo número de viagens, obtendo o score médio. Tendo em conta que temos toda a informação necessária, resta apenas imprimir os valores para o ficheiro de output.

Na parte do utilizador, a estratégia é semelhante: calculamos a idade, inicializá-mos as variáveis, obtemos a informação necessária analisando a lista de viagens, elemento a elemento, e imprimimos os valores no ficheiro de output.

Por fim, caso o input não corresponde a nenhum utilizador ou condutor que tenha uma conta ativa na base de dados, devolvemos um ficheiro vazio.

Query 2:

Na query 2 somos desafiados a listar os N condutores com a maior avaliação média.

Logo no início dividimos a função em 2 opções: Quando N é diferente de 0 ou quando N é igual a 0.

Quando N é diferente de 0, criamos a lista_rides, que, tal como acima, constitui uma lista com informação sobre todas as rides. Criamos também uma lista_nova, que será preenchida futuramente, e transformamos N num inteiro. Seguidamente, criamos uma hashtable chamada driver_info_table.

Percorrendo a lista de viagens, vamos adicionando à hashtable informação relevante sobre todos os condutores com uma conta ativa. Ao chegar ao fim da lista, teremos, para cada condutor, o seu ID, o seu score total, o seu número de viagens e a sua viagem mais recente. Estes valores estão guardados em estruturas Driver_q2.

Após este processo, passamos toda a informação obtida para a GList lista_driver_info. Seguidamente, percorremos essa lista e vamos adicionando elementos à lista_nova, que criamos no início. De forma a diminuir o número de comparações e assim aumentar a eficiência da função, limitamos a lista_nova a um tamanho igual a N. Assim, cada elemento da lista_driver_info é comparado, primeiramente, ao último elemento da lista_nova. Caso a comparação conclua que deve estar numa posição anterior, na lista, o elemento é inserido na lista de forma ordenada, utilizando a função compara_2. No entanto, no caso contrário, o elemento é logo descartado.

Importa aqui fazer um parêntese para abordar a função auxiliar compara_2. Trata-se, simplesmente, de uma função que compara a informação de duas estruturas Driver_q2, imaginemos, as estruturas A e B. Se, no output, A deve aparecer antes, devolve -1, caso contrário devolve 1.

Estando a lista concluída, resta apenas, através de um loop for, imprimir a informação para o output.

No caso em que N corresponde a 0, a função imprime apenas um ficheiro vazio.

Query 3:

Na query 3, o objetivo é listar os N utilizadores com maior distância viajada. O raciocínio aqui utilizado é semelhante ao da query 2.

Começamos por dividir a função consoante N. Quando N é diferente de 0, obtemos a lista_rides, criamos a lista_nova e transformamos N num inteiro.

Logo após, criamos uma hashtable chamada user_info_table, onde guardamos informação sobre os utilizadores com conta ativa, em estruturas User.q3.

Concluída a hashtable, transferimos a sua informação para uma GList lista_user_info, que percorremos de forma semelhante à da query 2 de forma a preencher a GList lista_nova com os N elementos com maior distância viajada. Por fim, imprimimos a informação relevante no ficheiro de output.

Já quando N é igual a 0, a função devolve apenas um ficheiro vazio.

Query 4:

Na query 4, é pedido o preço médio de cada viagem, para uma certa cidade.

Na nossa resolução, inicialmente, são criados dois doubles (preco e nr). É também criada uma GList cujos elementos são estruturas do tipo ride com os dados de uma certa viagem. Para cada viagem, verificamos, primeiramente, se se realizou na cidade pedida.

Se sim, criamos uma string "driver_id", para onde copiamos o id do condutor, que será usado para, acendendo à tabela correspondente aos condutores, obter a classe do carro, guardada na string class.

Seguidamente, identificamos o tipo de carro e adicionamos ao preço o montante correspondente. Adicionamos também 1 ao double nr.

No fim, temos o total da soma dos preços e o número total de viagens, pelo que resta apenas dividir estes dois valores, obtendo o preço médio, guardado no double preco_medio, e adicionar esse valor ao ficheiro de output.

Query 5:

Na query 5, é pedido o preço médio de cada viagem feita num certo intervalo de tempo.

Esta resolução é extremamente semelhante à da query 4. Inicialmente, são criados dois doubles (preco e nr).

Convertemos os valores das datas em int, através da seguinte fórmula:

$$\text{dia/mes: } AB = A*10 + B$$

$$\text{ano: } ABCD = A*1000 + B*100 + C*10 + D$$

É também criada uma GList cujos elementos são estruturas do tipo ride com os dados de uma certa viagem. Para cada viagem, convertemos a data em que aconteceu, utilizando a fórmula acima, e verificamos se essa data se encontra dentro do intervalo de tempo dado pelo input.

Se sim, criamos uma string "driver_id", para onde copiamos o id do condutor, que será usado para, acendendo à tabela correspondente aos condutores, obter a classe do carro, guardada na string class.

Seguidamente, identificamos o tipo de carro e adicionamos ao preço o montante correspondente. Adicionamos também 1 ao double nr.

No fim, temos o total da soma dos preços e o número total de viagens, pelo que resta apenas dividir estes dois valores, obtendo o preço médio, guardado no double `preco_medio`, e adicionar esse valor ao ficheiro de output.

Query 6:

Na query 6, é pedida a distância média percorrida, numa certa cidade, num certo intervalo de tempo.

Inicialmente, criamos dois double: `dist`, que representa a distância total, e `nr`, que representa o total de viagens.

Convertemos os valores das datas em int, através da fórmula mencionada na descrição da query 4.

Criamos também uma GList `lista_6`, onde guardamos elementos correspondentes, cada um, a uma estrutura do tipo `ride`.

Analisemos essa lista: Para cada um dos seus elementos, ou seja, para cada viagem, verificamos se se dá na cidade pedida. Se sim, transformamos a data dessa viagem pela fórmula acima, e, através da função auxiliar `compara3Datas`, verificamos se a data da nossa viagem se encontra entre o intervalo de datas referido.

Em caso positivo, adicionamos a distância da viagem ao double `dist`, adicionamos 1 ao número de viagens.

Por fim, resta apenas dividir esses dois valores, obtendo a distância média, guardada no double `r` e adicionar esse valor ao ficheiro output.

Query 7:

Na query 7, o objetivo é imprimir os top N condutores numa determinada cidade.

Para isso, começamos por criar uma GList com informação sobre todas as viagens (`lista_rides`), criar outra GList, que, para já, deixamos como NULL e transformar N num inteiro.

Seguidamente, criamos uma `driver_info_table`. Percorrendo a lista de viagens, será guardada nessa hashtable toda a informação necessária sobre todos os condutores (essa informação refere-se apenas a viagens feitas na cidade indicada), em que a chave é o id do condutor e o valor será informação contida numa struct `Driver_q7`.

Logo após, passamos essa informação para uma GList (`lista`), que percorremos de forma a adicionar, ordenadamente, elementos à `lista_nova`, da mesma forma que fizemos nas queries 2 e 3.

Por fim, agora que temos uma lista ordenada com os top N condutores numa determinada cidade, imprimimos os valores para o output

Query 8:

Na query 8 é-nos pedido para listar todas as viagens em que tanto o condutor como o utilizador são do mesmo género, passado como parâmetro, e ambos têm X ou mais anos.

Como em muitas das outras queries, começamos por criar uma GList com todas as viagens. Convertemos também o valor correspondente a X num inteiro.

Percorrendo a lista das viagens, vamos retirando informação sobre o condutor e o utilizador. No caso de todos os parâmetros estarem corretos (o utilizador e o condutor terem o género pedido, terem X ou mais anos e uma conta ativa), guardamos a sua informação numa estrutura `Driver_user_q8` e adicionamos esses dados, de forma ordenada, através da função `compara_8` à `lista_nova`.

Por fim, imprimimos o output.

Query 9:

Na query 9, o objetivo é listar, ordenadamente, as viagens em que o passageiro deu gorjeta, num certo intervalo de tempo.

Primeiramente, começamos por converter as datas-limite em int, utilizando a fórmula mencionada na descrição da query 5.

São criadas duas GList: lista_9 e lista_nova_9. Na primeira são guardados os dados de cada viagem, em estruturas Ride.

Para cada um dos elementos da lista_9, ou seja, para cada viagem, verifica-se, primeiramente, se houve gorjeta. Se sim, converte-se a data no formato acima mencionado e verifica-se se a viagem decorreu no intervalo de tempo mencionado no input. Em caso positivo, adiciona-se, ordenadamente, com ajuda da função auxiliar compara_9, a informação dessa viagem à lista_nova_9.

Por fim, resta apenas passar os valores das viagens para o ficheiro output.

4 Testes funcionais e de desempenho

O módulo responsável pelos testes exibe um teste para cada uma das queries, averiguando se o resultado produzido pelas queries é correto e obtido em tempo em tempo útil (no âmbito deste trabalho, 10 segundos).

Para cada uma das queries, temos uma função responsável pelo teste (por exemplo, a função `test_q1` é responsável pelo teste da query 1). Medimos o tempo que a query demora a produzir o resultado, comparámo-lo com o resultado esperado e imprimimos a informação obtida.

A função `teste_queries` é chamada na main e serve apenas para agrupar todas as funções `query_x`. Foi usado o seguinte input:

- 1 AfoCastro81
- 2 100
- 3 100
- 4 Braga
- 5 19/03/2015 16/01/2016
- 6 Faro 14/01/2012 12/11/2013
- 7 100 Porto
- 8 F 12
- 9 13/10/2021 13/10/2021

Os resultados estão expostos nas tabelas abaixo e foram obtidos numa máquina com um processador i7-10510U 1.8GHz e 16GB de memória RAM, usando os databases normal e large, respetivamente.

Query	Tempo(s) [Database normal]	Tempo(s) [Database Large]
Query1	0.095275	0.373817
Query2	0.807171	12.024293
Query3	1.256805	15.264327
Query4	0.140410	1.447584
Query5	0.146444	1.533095
Query6	0.096105	0.924077
Query7	0.197949	2.228296
Query8	0.824512	11.686860
Query9	0.133126	2.390169

Table 1: Desempenho temporal das queries

Concluimos também que, com o input mencionado acima e a database normal, a memória usada é de 615Mb. Já com a database large, a memória usada é de 6460Mb.

5 Modo interativo

Tal como pedido no enunciado do trabalho prático, foi implementado um modo de funcionamento interativo na aplicação desenvolvida. Para ter acesso a este modo deve-se correr o ficheiro executável *programa-principal* sem argumentos adicionais. Foi decidido que a interface seria imprimida no terminal, não recorrendo por isso a bibliotecas de terceiros.

No início é pedido ao utilizador que indique a pasta onde se localizam os ficheiros de entrada necessários para a criação dos catálogos. É essencial que a pasta passada ao programa tenha no mínimo os três ficheiros com os dados necessários, isto é, *users.csv*, *drivers.csv* e *rides.csv*.

Após a introdução do caminho da pasta de entrada são criados os três catálogos que irão ser mantidos durante a duração do programa no modo interativo. Quando se fecha o programa, os catálogos são destruídos. Nos testes realizados pelo grupo utilizaram-se múltiplas pastas de entrada, cada uma contendo os diversos databases disponibilizados pelos docentes.

De seguida surgem as quatro opções disponíveis. As primeiras três são relativas a testes que consistem em imprimir para um ficheiro de texto todo o conteúdo de um catálogo, permitindo assim verificar se estes foram de facto criados e se a validação de dados foi aplicada corretamente.

A última opção consiste no modo interativo propriamente dito. Aqui o utilizador tem a opção de escolher qual query vai querer executar. Para cada opção é dado ao utilizador uma breve descrição da funcionalidade de cada query e quais parâmetros deve indicar. Deve ser inserido apenas um parâmetro por cada linha e clicar *Enter* para que este seja lido.

Nos casos em que se introduz um ou mais parâmetros inválidos, isto é, um parâmetro que é impossível de responder (por exemplo, na query 1, indicar um *ID* que não existe em nenhum catálogo), o programa pode reagir de duas formas:

- devolve absolutamente nada como forma de resposta nas queries que envolvem listagens;
- devolve um *ERRO* nas queries cuja resposta é singular.

Realçamos que o tratamento aplicado aos parâmetros inválidos não implica que a aplicação deixe de funcionar.

Nos anexos apresentamos três exemplos onde foram executadas as queries 1, 3 e 6. Nestes exemplos todos os parâmetros passados eram válidos pelo que houve sempre uma resposta.

6 Conclusão

Num panorama geral, e tendo em conta o que foi explicado ao longo deste relatório, achamos que temos um projeto final bem conseguido mesmo que este não cumpra com um dos requisitos desta segunda fase.

Reconhecemos que a gestão de memória não está ao nível que é esperado e que como consequência o programa não tem o melhor desempenho e performance ao lidar com o maior database que foi disponibilizado pela equipa docente (referimos-nos ao *database-large*). Notamos que databases mais pequenos funcionam dentro dos padrões que são expectáveis.

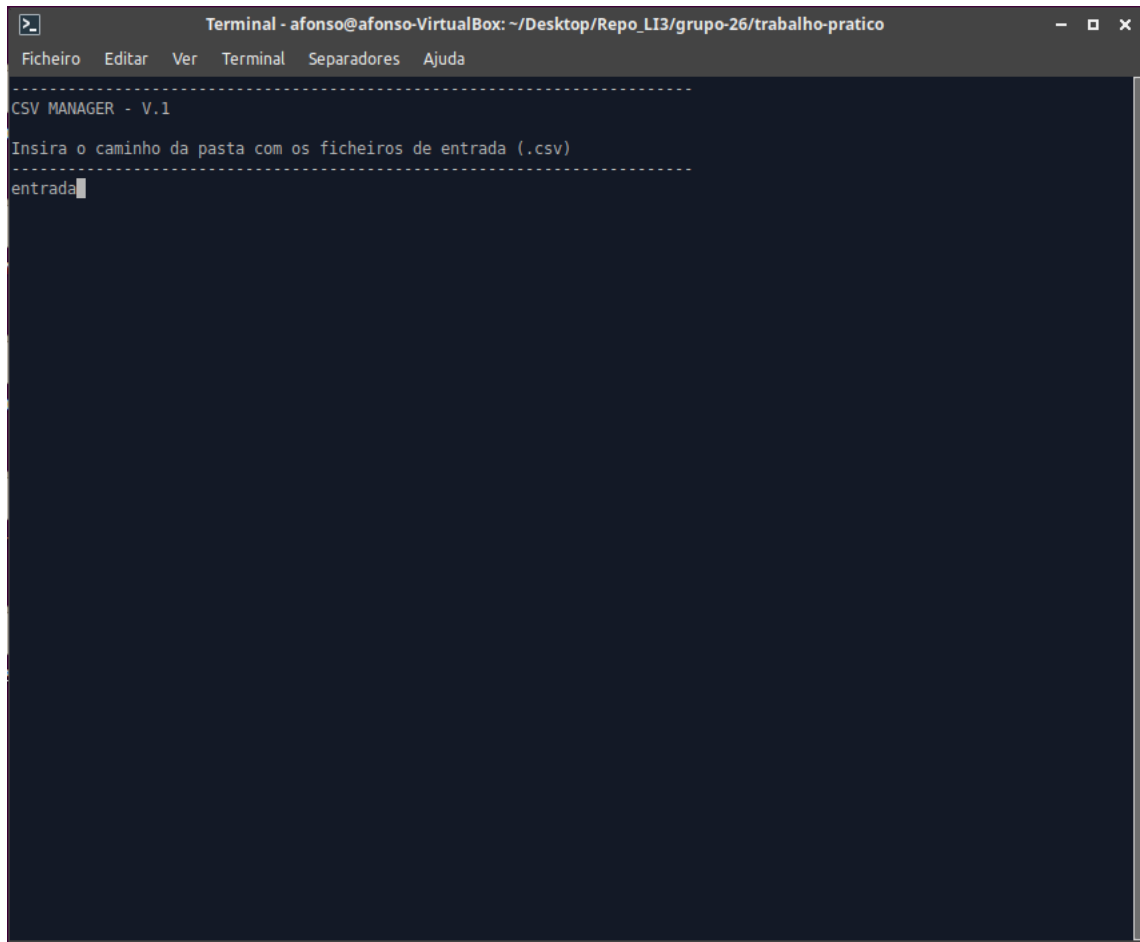
Optamos também por manter os catálogos em estruturas de Hashtables, visto ser mais acessível para o grupo trabalhar com o que já tinha sido feito da primeira fase. O compromisso foi de fazer uma melhor gestão das mesmas. Como forma de trabalho futuro seria interessante aplicar uma nova estrutura para os catálogos como, por exemplo, estruturas de árvores.

Como ponto positivo as queries estão todas a funcionar dentro do esperado, tanto no modo batch como no modo interativo. Para auxiliar na gestão da memória foram criadas estruturas específicas para a resolução das mesmas.

Realçamos também as melhorias feitas ao encapsulamento e à modularidade, nomeadamente a remoção das estruturas de dados de *User*, *Driver* e *Ride* dos ficheiros de *header* e a criação de métodos que permitem retirar informação dos catálogos sem interagir diretamente com o conteúdo dos mesmos.

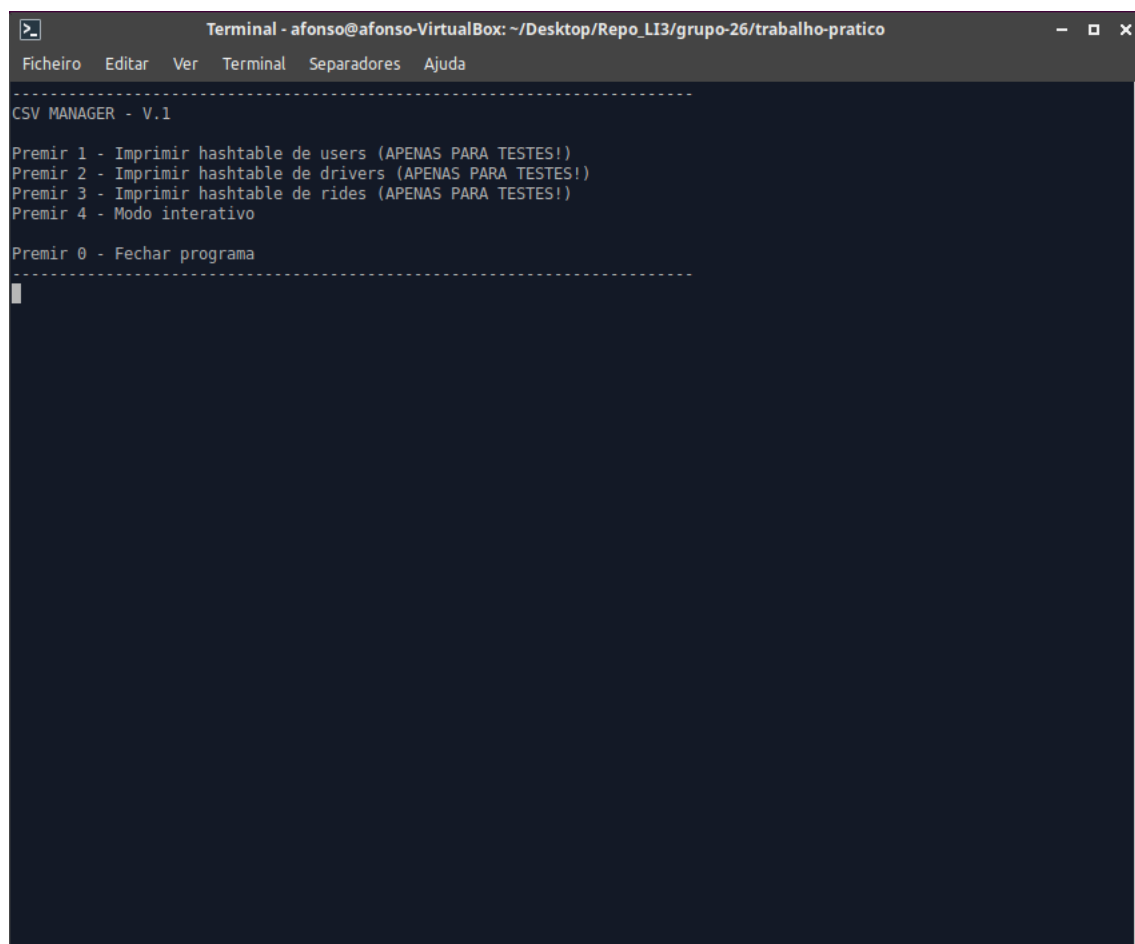
Como ponto final referimos também a adição de métodos que servem para validar os dados que são lidos dos ficheiros de entrada. Aqui não houve falhas que tivessem sido notadas pelo grupo.

7 Anexos



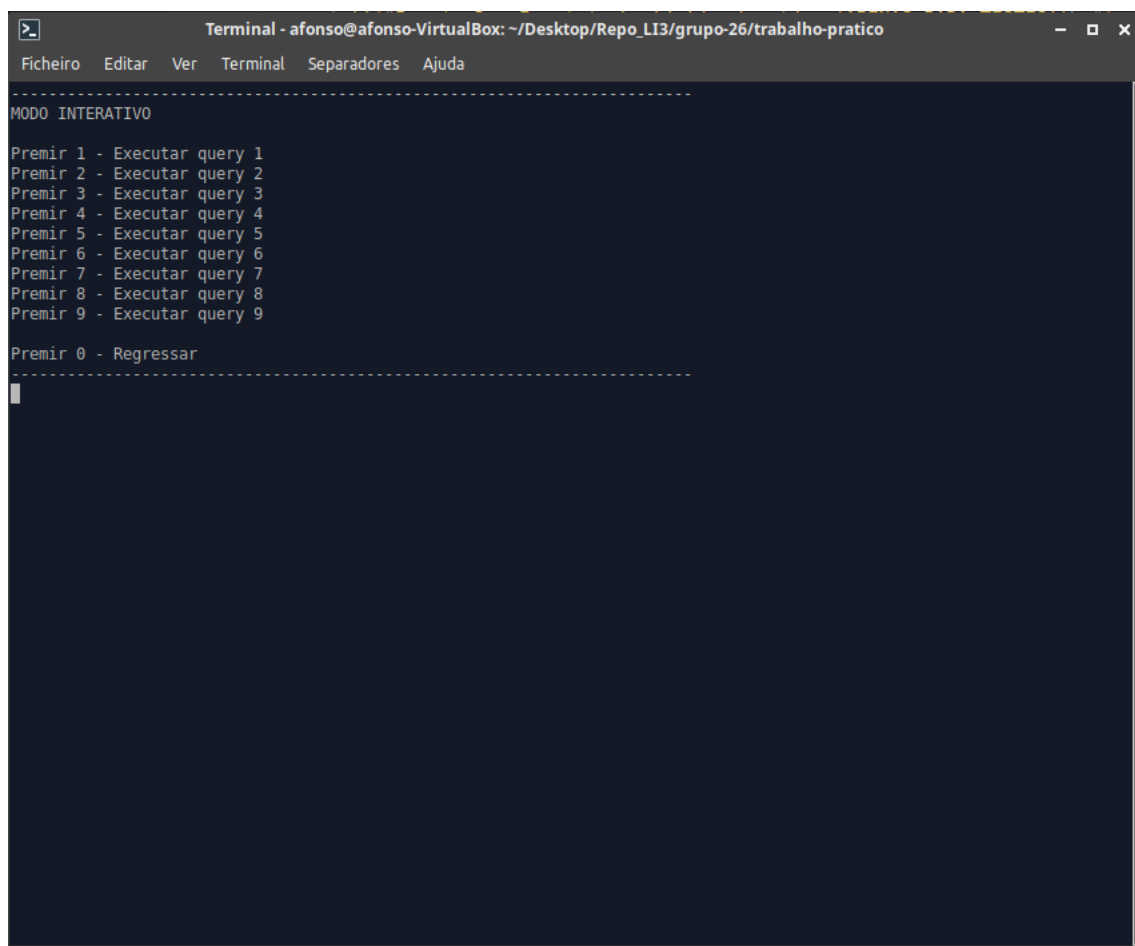
```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/Repo_LI3/grupo-26/trabalho-pratico
Ficheiro  Editar  Ver  Terminal  Separadores  Ajuda
-----
CSV MANAGER - V.1
Insira o caminho da pasta com os ficheiros de entrada (.csv)
-----
entrada
```

Figure 1: Indicação da pasta dos caminhos de entrada



```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/Repo_LI3/grupo-26/trabalho-pratico
Ficheiro  Editar  Ver  Terminal  Separadores  Ajuda
-----
CSV MANAGER - V.1
Premir 1 - Imprimir hashtable de users (APENAS PARA TESTES!)
Premir 2 - Imprimir hashtable de drivers (APENAS PARA TESTES!)
Premir 3 - Imprimir hashtable de rides (APENAS PARA TESTES!)
Premir 4 - Modo interativo
Premir 0 - Fechar programa
-----
█
```

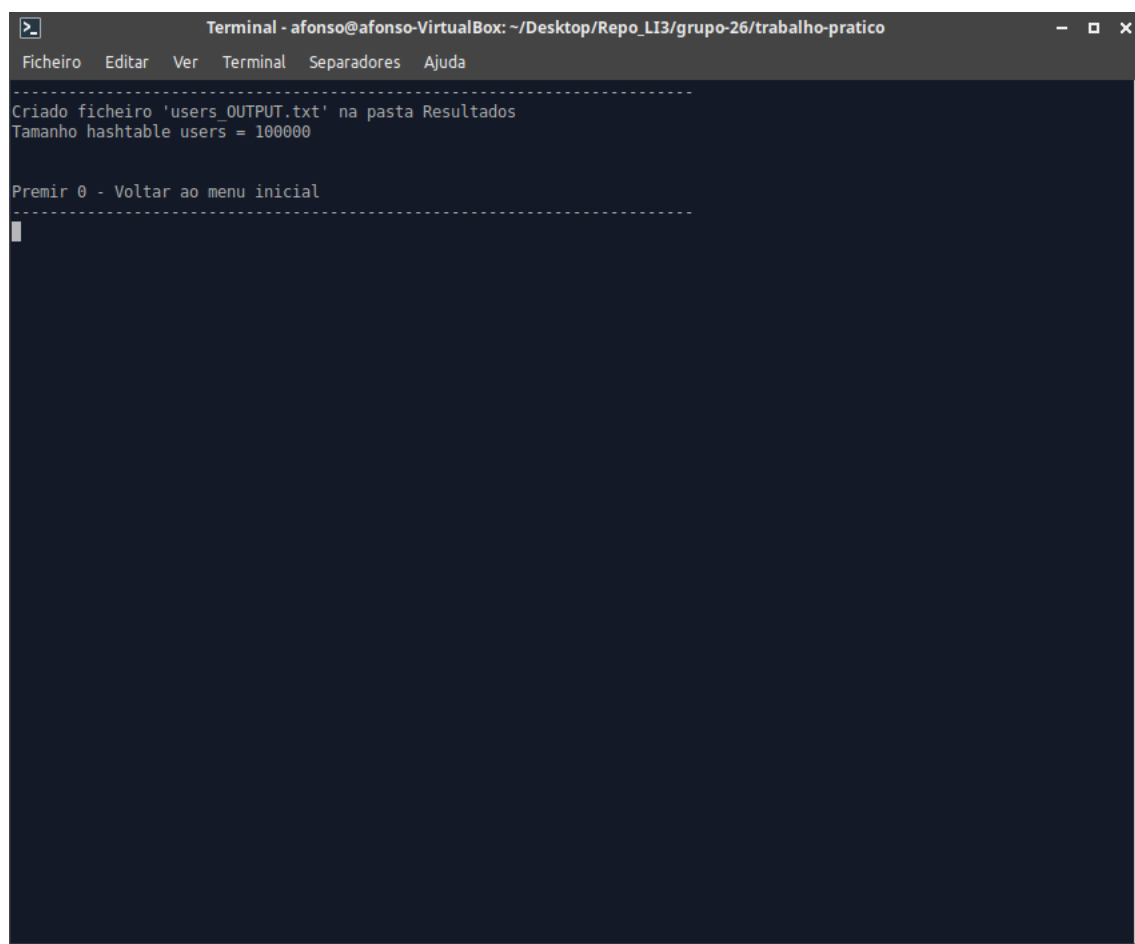
Figure 2: Menu principal

A terminal window titled "Terminal - afonso@afonso-VirtualBox: ~/Desktop/Repo_LI3/grupo-26/trabalho-pratico". The window has a menu bar with "Ficheiro", "Editar", "Ver", "Terminal", "Separadores", and "Ajuda". The terminal content shows a dashed line separator, followed by "MODO INTERATIVO", and a list of options: "Premir 1 - Executar query 1", "Premir 2 - Executar query 2", "Premir 3 - Executar query 3", "Premir 4 - Executar query 4", "Premir 5 - Executar query 5", "Premir 6 - Executar query 6", "Premir 7 - Executar query 7", "Premir 8 - Executar query 8", "Premir 9 - Executar query 9", and "Premir 0 - Regressar". Another dashed line separator follows. A cursor is visible on the line after the second separator.

```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/Repo_LI3/grupo-26/trabalho-pratico
Ficheiro  Editar  Ver  Terminal  Separadores  Ajuda
-----
MODO INTERATIVO
Premir 1 - Executar query 1
Premir 2 - Executar query 2
Premir 3 - Executar query 3
Premir 4 - Executar query 4
Premir 5 - Executar query 5
Premir 6 - Executar query 6
Premir 7 - Executar query 7
Premir 8 - Executar query 8
Premir 9 - Executar query 9
Premir 0 - Regressar
-----

```

Figure 3: Menu das queries

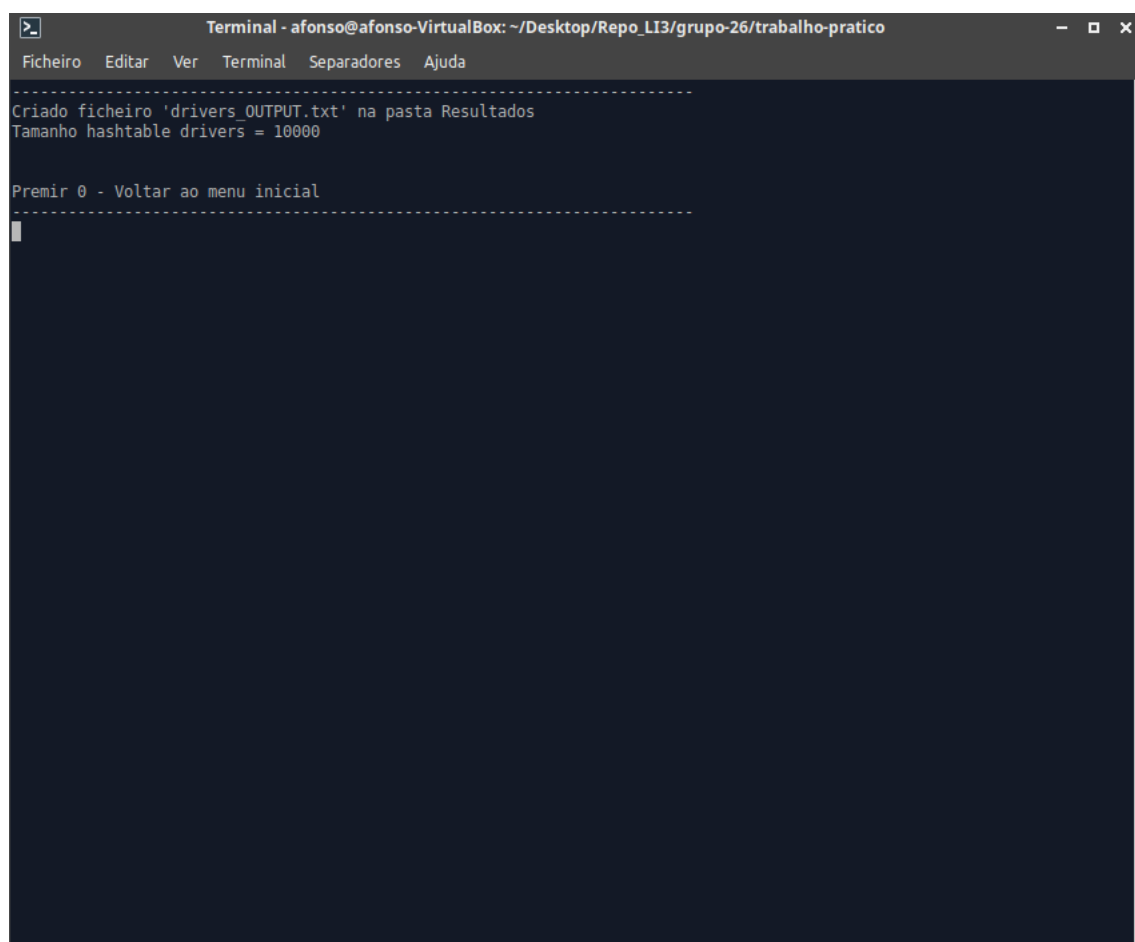


A terminal window titled "Terminal - afonso@afonso-VirtualBox: ~/Desktop/Repo_LI3/grupo-26/trabalho-pratico". The window has a menu bar with "Ficheiro", "Editar", "Ver", "Terminal", "Separadores", and "Ajuda". The terminal output shows a dashed line, followed by the text "Criado ficheiro 'users_OUTPUT.txt' na pasta Resultados" and "Tamanho hashtable users = 100000". Below this is "Premir 0 - Voltar ao menu inicial", another dashed line, and a cursor at the start of a new line.

```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/Repo_LI3/grupo-26/trabalho-pratico
Ficheiro  Editar  Ver   Terminal  Separadores  Ajuda
-----
Criado ficheiro 'users_OUTPUT.txt' na pasta Resultados
Tamanho hashtable users = 100000

Premir 0 - Voltar ao menu inicial
-----
|
```

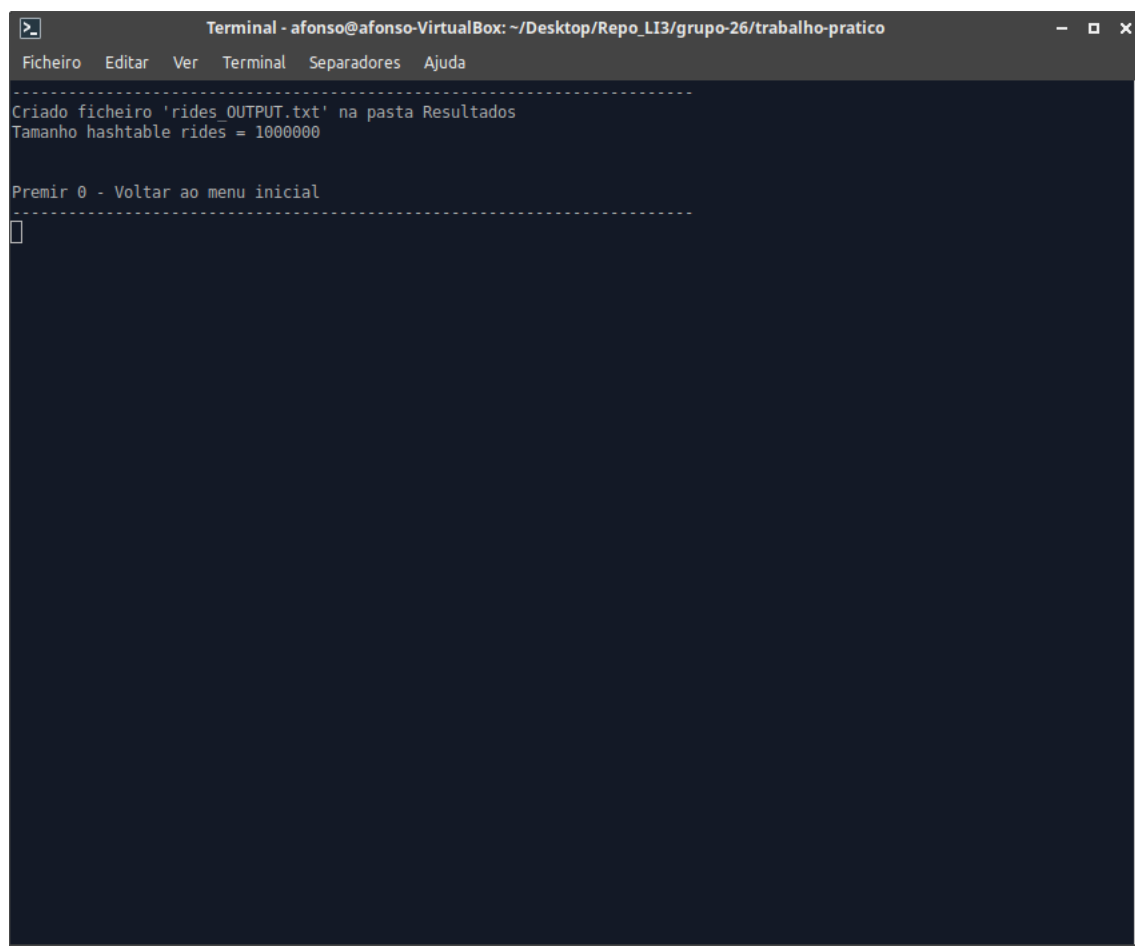
Figure 4: Teste da criação do catálogo dos *users*



A terminal window titled "Terminal - afonso@afonso-VirtualBox: ~/Desktop/Repo_LI3/grupo-26/trabalho-pratico". The window has a menu bar with "Ficheiro", "Editar", "Ver", "Terminal", "Separadores", and "Ajuda". The terminal output shows a dashed line, followed by the text "Criado ficheiro 'drivers_OUTPUT.txt' na pasta Resultados" and "Tamanho hashtable drivers = 10000". Below this is another dashed line, then the text "Premir 0 - Voltar ao menu inicial", and a final dashed line. A cursor is visible on the line following the last dashed line.

```
-----  
Criado ficheiro 'drivers_OUTPUT.txt' na pasta Resultados  
Tamanho hashtable drivers = 10000  
  
-----  
Premir 0 - Voltar ao menu inicial  
-----  
|
```

Figure 5: Teste da criação do catálogo dos *drivers*



A terminal window titled "Terminal - afonso@afonso-VirtualBox: ~/Desktop/Repo_LI3/grupo-26/trabalho-pratico". The window has a menu bar with "Ficheiro", "Editar", "Ver", "Terminal", "Separadores", and "Ajuda". The terminal output shows the creation of a file named "rides_OUTPUT.txt" in the "Resultados" folder, with a message indicating the hashtable size for rides is 1000000. Below this, it says "Premir 0 - Voltar ao menu inicial" and a dashed line. A cursor is visible on the line following the dashed line.

```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/Repo_LI3/grupo-26/trabalho-pratico
Ficheiro  Editar  Ver   Terminal  Separadores  Ajuda
-----
Criado ficheiro 'rides_OUTPUT.txt' na pasta Resultados
Tamanho hashtable rides = 1000000

Premir 0 - Voltar ao menu inicial
-----
█
```

Figure 6: Teste da criação do catálogo das *rides*

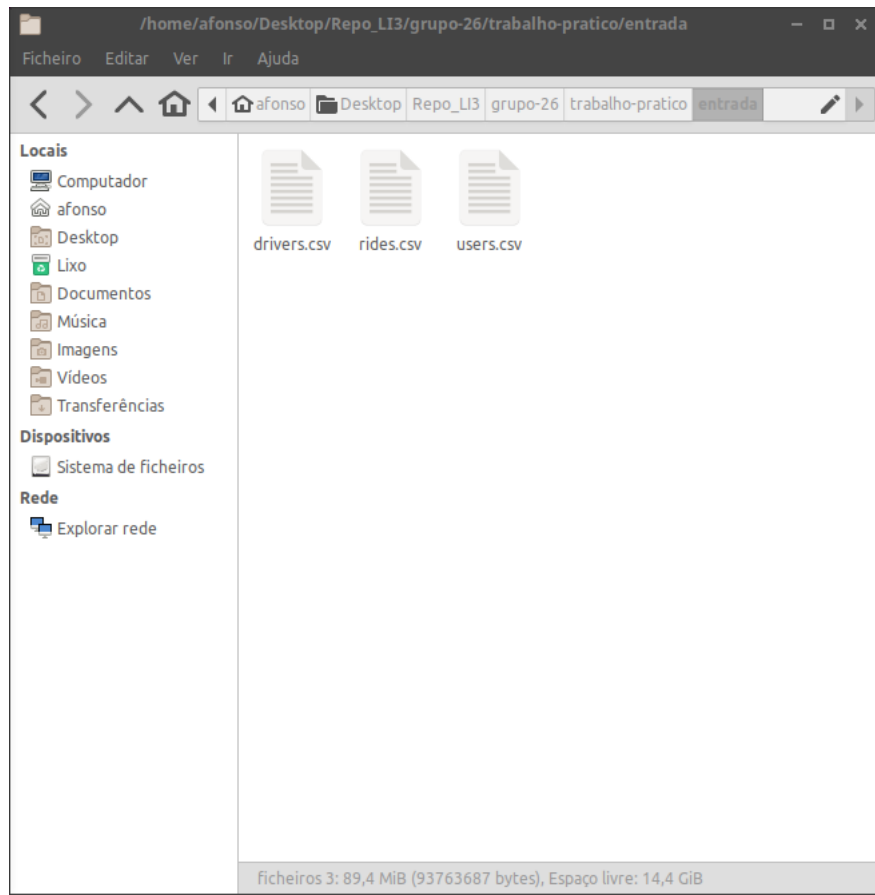


Figure 7: Exemplo da pasta com ficheiros de entrada

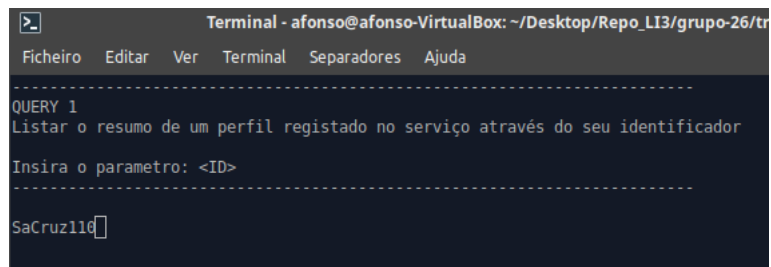


Figure 8: A query 1 com o parâmetro 'SaCruz110'

```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/Repo_LI3/grupo-26/
Ficheiro  Editar  Ver  Terminal  Separadores  Ajuda
-----
Executou Query 1 ---> parâmetros: SaCruz110
-----
Nome (Utilizador) ---> Santiago Cruz
Género ---> M
Idade ---> 31
Avaliação Média ---> 3.200
Total Viagens ---> 5
Total Gasto ---> 50.340
-----
Premir 0 - Voltar ao menu anterior
-----
█
```

Figure 9: Resultado da query 1 para o parâmetro '*SaCruz110*'

```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/Repo_LI3/grupo-26/
Ficheiro  Editar  Ver  Terminal  Separadores  Ajuda
-----
QUERY 3
Listar os N utilizadores com maior distância viajada
Insira o parametro: <N>
-----
4█
```

Figure 10: A query 3 com o parâmetro '*4*'

```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/Repo_LI3/grupo-26/tr
Ficheiro  Editar  Ver  Terminal  Separadores  Ajuda
-----
Executou Query 3 ---> parâmetros: 4
-----

Username ---> Anita-PetPinto38
Nome ---> Anita-Petra Pinto
Distância Total ---> 240

Username ---> REsteves70
Nome ---> Raquel Esteves
Distância Total ---> 234

Username ---> LiMarques
Nome ---> Lia Marques
Distância Total ---> 232

Username ---> ConMelo
Nome ---> Constança Melo
Distância Total ---> 219

-----
Premir 0 - Voltar ao menu anterior
-----
█
```

Figure 11: Resultado da query 3 para o parâmetro '4'

```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/Repo_LI3/grupo-26/trab
Ficheiro  Editar  Ver  Terminal  Separadores  Ajuda
-----
QUERY 6
Distância média percorrida, numa determinada cidade, num dado intervalo de tempo

Insira os parametros: <city> <data A> <data B>
-----

Porto
01/01/2021
01/02/2021█
```

Figure 12: A query 6 com os parâmetros 'Porto' '01/01/2021' '01/02/2021'

```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/Repo_LI3/grupo-26
Ficheiro  Editar  Ver  Terminal  Separadores  Ajuda
-----
Executou Query 6 ---> parâmetros: Porto 01/01/2021 01/02/2021
-----
Distância Média ---> 6.896
-----
Premir 0 - Voltar ao menu anterior
-----
```

Figure 13: Resultado da query 6 para os parâmetros 'Porto' '01/01/2021' '01/02/2021'

```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/Repo_LI3/grupo-26/trabalho-pratico
Ficheiro  Editar  Ver  Terminal  Separadores  Ajuda
-----
Executou Query 5 ---> parâmetros: asdaddawda 1213121
-----
ERRO
-----
Premir 0 - Voltar ao menu anterior
-----
```

Figure 14: Demonstração de um erro na query 5

```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/Repo_LI3/grupo-26/trabalho-pratico
Ficheiro  Editar  Ver  Terminal  Separadores  Ajuda
-----
Executou Query 7 ---> parâmetros: 3 Valpaços
-----
-----
Premir 0 - Voltar ao menu anterior
-----
█
```

Figure 15: Demonstração de um erro na query 7