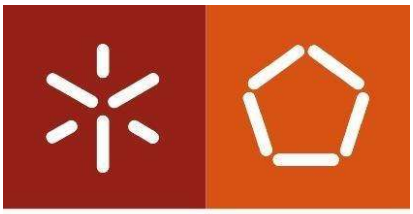

Universidade do Minho

Licenciatura em Engenharia Informática

LI3 – Relatório:1ª Fase
Grupo 77

António Pedro (A100821) João Magalhães (A100740) Rodrigo Gomes (A100555)

Ano Letivo 2022/2023

Índice

Introdução e principais desafios 1

Grandes estruturas de dados 1

Módulos e pequenas estruturas de dados 2

Módulo: users 2

Módulo: drivers 3

Módulo: rides 4

Módulo: mpreco 5

Módulo: main_parse 6

Módulos: query1,query2 e query4 6

Módulo: Main 6

Encapsulamento de dados 7

Teste de desempenho 7

Conclusão 8

Grafo de dependências 9

Introdução e principais desafios

Este projeto, realizado no âmbito da Unidade Curricular de Laboratórios de Informática III, do segundo ano da Licenciatura em Engenharia Informática, teve até agora um grande foco na análise de grandes ficheiros de dados (neste caso, com o formato .csv) recorrendo à linguagem de programação C. Sendo um projeto que se debruça sobre a programação em escala amplificada, foi necessário recorrer a métodos mais avançado de tratamento de dados, como o uso de estruturas de dados dinâmicas para armazenar e consultar grandes quantidades de informação. Ainda mais, foi-nos incutido desde o início a enorme importância de garantir sempre a modularidade e encapsulamento dos dados. Todas as estruturas de dados usadas neste trabalho foram ou idealizadas e definidas pelos membros do grupo ou importadas de bibliotecas, como é o caso da GHashTable, proveniente da biblioteca Glib.

O maior obstáculo ao qual fomos submetidos nesta fase foi encontrar uma forma competente de armazenar todos os dados, aos quais teríamos posteriormente de aceder para resolução das *queries* , de maneira a que toda a informação necessária fosse rapidamente acessível. Logo que as estruturas foram definidas e a informação dos ficheiros guardada nos respetivos campos das mesmas, a construção do trabalho prático foi relativamente simples.

Já na reta final da entrega desta primeira fase, deparamo-nos com uma situação que não nos foi possível resolver ainda, de grande importância, que recai sobre a libertação da memória alocada para as *structs* dos nossos catálogos de dados e das *Hash Tables*.

Grandes estruturas de dados

Com o intuito de guardar organizadamente números elevadíssimos de *structs driver, user e mpreco* (abordadas em baixo), empregamos estruturas de *Hash Tables* já definias

na Glib, permitindo-nos responder, até à data, às *queries* 1 e 4. Realizamos a inserção com g_hash_table_insert, a procura com g_hash_table_lookup e a substituição com g_hash_table_replace.

Para realizar a ordenação por avaliação média de todos os condutores ativos, implementamos um *array* constituído por apontadores para a *struct* que guarda os condutores (*struct driver*). Após a sua ordenação por ordem crescente, tendo como base um mecanismo *standard* de *insertion sort*, é possível responder à *query* 2, simplesmente imprimindo as informações pedidas do número de elementos do array requsitados, da última posição em direção à primeira.

Módulos e pequenas estruturas de dados

Recorrendo a estratégias de modularidade abordadas nas aulas teórico-práticas da UC, acabamos por concluir a primeira fase com o projeto dividido nos seguintes módulos, a explicar:

Módulo: users

Para guardar toda a informação relativa aos utilizadores foi criada uma estrutura de dados, designada por *struct user* (Figura 1).

Após a abertura do ficheiro users.csv, cujo *path* para a respetiva pasta é passado como argumento na função main, através de um ciclo *while*, lemos linhas individuais do ficheiro e atualizamos os dados de cada *user*, previamente inicializado. Para os restantes parâmetros, remetentes a informações obtidas somente no ficheiro rides.csv (que iremos abordar mais à frente) e ainda com o seu valor de inicialização, foi necessária a leitura deste último de forma bastante similar ao anterior, atualizando agora os valores relativos à ocorrência de viagens com um determinado *user*.

A estrutura foi posteriormente guardada num campo de uma *Hash Table*, já criada (GHashTable* hashUser), a qual recebe como *key* o (char*)username desse mesmo utilizador e como *value* um apontador para a *struct user*, do tipo USER.

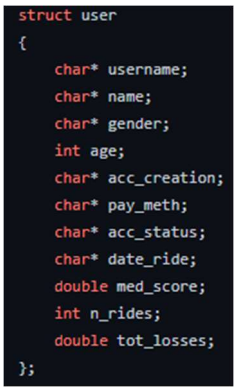

Figura 1: Definição da struct utilizada para armazenar dados do ficheiro users.csv.

Módulo: drivers

Para guardar toda a informação relativa aos condutores foi criada uma estrutura de dados, designada por *struct driver* (Figura 2).

Assim como no módulo dos utilizadores, ocorre a abertura do ficheiro respetivo aos dados que queremos armazenar para posterior *parsing*, neste caso drivers.csv, cujo *path* é passado como argumento na função main. Através de um ciclo *while*, lemos linhas individuais do ficheiro e atualizamos os dados de cada *driver*, previamente inicializado. Para os restantes parâmetros, remetentes a informações obtidas somente no ficheiro rides.csv e ainda com o seu valor de inicialização, procede-se à leitura deste último, atualizando agora os valores relativos à ocorrência de viagens com um determinado *driver*.

A estrutura foi posteriormente guardada num campo de uma *Hash Table*, já criada (GHashTable* hashDriver), a qual recebe como *key* o (char*)id desse mesmo condutor e como *value* um apontador para a *struct driver*, do tipo DRIVER.

```c
struct driver
{
    char* id;
    char* name;
    int age;
    char* gender;
    char* car_class;
    char* city;
    char* acc_creation;
    char* acc_status;
    char* date_ride;//Última viagem
    double med_score;
    int n_rides;
    double tot_profit;
};
```

Figura 2: Definição da struct utilizada para armazenar dados do ficheiro drivers.csv.

Módulo: rides

Para guardar toda a informação relativa às viagens foi criada uma estrutura de dados, designada por *struct ride* (Figura 3).

Assim como nos módulos descritos anteriormente, ocorre a abertura do ficheiro respetivo aos dados que queremos armazenar para posterior *parsing*, neste caso rides.csv, cujo *path* é passado como argumento na função main. Através de um ciclo *while*, lemos linhas individuais do ficheiro e atualizamos os dados de cada *ride*, previamente inicializada.

Quanto ao armazenamento, não foi ainda pertinente criar nenhuma estrutura que guarde todas as *struct ride* (*Hash Table, array* ou *BinTree*), sendo que só precisamos de ler os dados uma vez e atualizar nos utilizadores, condutores e preços médios. Assim sendo, a cada viagem lida, são chamadas três funções para transmitir os dados obtidos às restantes estruturas, nomeadamente *userTable*, *drive* e *cityTable*.

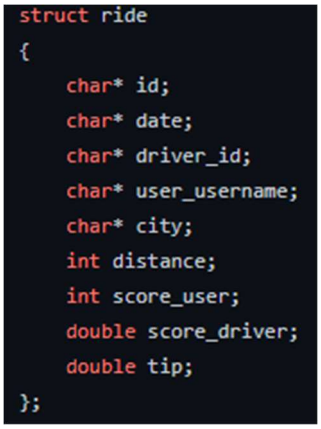

Figura 3: Definição da struct utilizada para armazenar dados do ficheiro rides.csv.

Módulo: mpreco

De forma a responder de forma rápida e eficiente à query 4, optamos por criar uma estrutura de dados para auxiliar o cálculo do preço médio das viagens numa determinada cidade, designada por *struct mpreco* (Figura 4).

Aquando da leitura do ficheiro rides.csv, é inicializado um *mpreco* relativo à cidade em que ocorreu a viagem lida no momento, o qual é inserido numa *Hash Table* constituída por *structs mpreco*, do tipo (GHashTable* hashCity). Se for a primeira ocorrência da cidade em questão, é inserido diretamente. Caso contrário, os dados são recalculados e o antigo *mpreco* substituído.

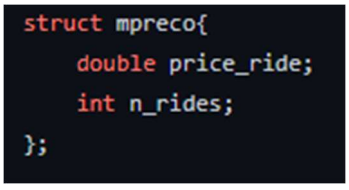

Figura 4: Definição da struct utilizada pra calcular o preço médio das viagens numa determinada cidade.

Módulo: main_parse

O módulo main_parse é o responsável pelo dinamismo e fluxo da resposta às *queries*, sendo que contém a função que trata o input passado pela main, recorrendo à *struct args*, definida em baixo (Figura 5). Após analisar os argumentos recebidos, redireciona os dados e as estruturas necessárias a cada *query* para o seu respetivo módulo, pelo chamamento da função da *query* em questão.

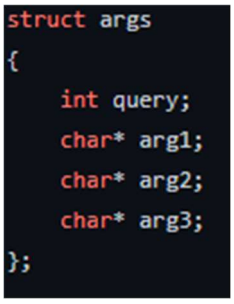

Figura 5: Definição da struct utilizada para tratar os comandos provenientes do ficheiro de inputs (inputs.txt, p. e.).

Módulos: query1,query2 e query4

Nestes módulos situam-se as funções responsáveis pela criação dos ficheiros de *output* na pasta "Resultados" e pela escrita dos valores pedidos, os quais correspondem a 100% aos testes realizados não só pelo grupo, mas também pela equipa docente, de forma automatizada. Nestas três *queries* que optamos por resolver na primeira fase, encontram-se grandes bases para o que nos é pedido nas restantes, sendo que consideramos expectável que a resolução das que faltam seja rápida, fácil e eficiente, permitindo-nos avançar quase diretamente para a construção do modo interativo.

Módulo: Main

Por último, o módulo Main, o módulo principal, é o que controla todo o programa. Neste é feito a abertura dos ficheiros .csv necessários à execução do executável e feito

o *parsing* de todos os dados, antes da leitura do ficheiro de comandos das *queries*. Desse modo, toda a informação necessária para dar resposta ao que possa ser pedido já se encontra pronta e rapidamente acessível. No final, é chamado o main_parse e, após a escrita ter sido feita, são destruídas as *Hash Tables* usadas (g_hash_table_destroy).

Encapsulamento de dados

Tendo como base o que nos foi transmitido em aula sobre a abstração de dados e a sua importância, terminamos a primeira fase deste projeto com todas as nossas estruturas de dados devidamente encapsuladas. Consequentemente, todos os acessos a dados de *structs* são realizados no ficheiro .c em que estas estão definidas, individualmente. Qualquer necessidade de dados por parte de ficheiros ditos externos à *struct* é resolvida com utilização de *getters*, ou seja, funções que acedem à *struct* em questão e devolvem o valor requerido, permitindo um bom funcionamento do programa e promovendo dinamismo entre diferentes módulos, sem nunca corromper a abstração.

Teste de desempenho

```
ts.txt
        1,45 real         1,39 user         0,05 sys
           205553664  maximum resident set size
                   0  average shared memory size
                   0  average unshared data size
                   0  average unshared stack size
               12671  page reclaims
                   4  page faults
                   0  swaps
                   0  block input operations
                   0  block output operations
                   0  messages sent
                   0  messages received
                   0  signals received
                   5  voluntary context switches
                  41  involuntary context switches
         16832645265  instructions retired
          4525449000  cycles elapsed
           204359616  peak memory footprint
```

Figura 6: Desempenho do programa-principal.

Como é possível observar pela Figura 6, na totalidade, o programa demorou somente 1.45 segundos a fazer o *parsing* completo dos ficheiros e a executar 35 testes, tendo

como armazenamento total usado aproximadamente 20.5 MB, o que poderia ser melhorado executando mais libertações de memória, estrategicamente, ao longo do programa.

Conclusão

De um ponto de vista pedagógico, o desenvolvimento deste projeto foi extremamente interessante, pois permitiu-nos aprofundar o nosso conhecimento da linguagem C, a nível da criação de novas estruturas de dados, da utilização de bibliotecas já existentes e da manipulação e desenvolvimento de diversas funções, de forma a otimizar o tempo de processamento e a memória utilizada.

Considerando os objetivos propostos para esta fase, pensamos ter um projeto bem estruturado e construído, que não só cumpre com os requisitos, como o faz com grande eficiência concluindo testes às três *queries* por nós selecionadas em menos de 1,5 segundos. Para além disso, permitirá um contínuo e facilitado trabalho no desenvolvimento da segunda fase, já que as funções e estruturas existentes estão preparadas para receber dados de tamanho desconhecido, sem que seja necessário alterar as mesmas. Ainda, o desenvolvimento de soluções para a resolução das restantes *queries* irá assentar em lógicas e arquiteturas semelhantes às utilizadas, sendo certo o uso das funções já definidas por nós.

A nível da escrita do código, acreditamos ter feito um excelente trabalho, sendo um código simples, elegante e de fácil legibilidade e compreensão, do qual são resultado funções com as mesmas características, o que não só origina facilidade na modularização e encapsulamento, mas também na leitura, alteração de funções existentes e criação de novas, não só por nós, mas por qualquer pessoa familiarizada com a linguagem C. Aqueles que não estejam familiarizados, conseguirão na mesma compreender o propósito das funções devido ao rigor da nossa documentação, que se provou de bastante utilidade no contexto de trabalho de grupo.

Por todos estes motivos, o balanço que fazemos do projeto, até ao momento, é bastante positivo, tendo sido um prazer para os elementos do grupo trabalhar conjuntamente.

Grafo de dependências

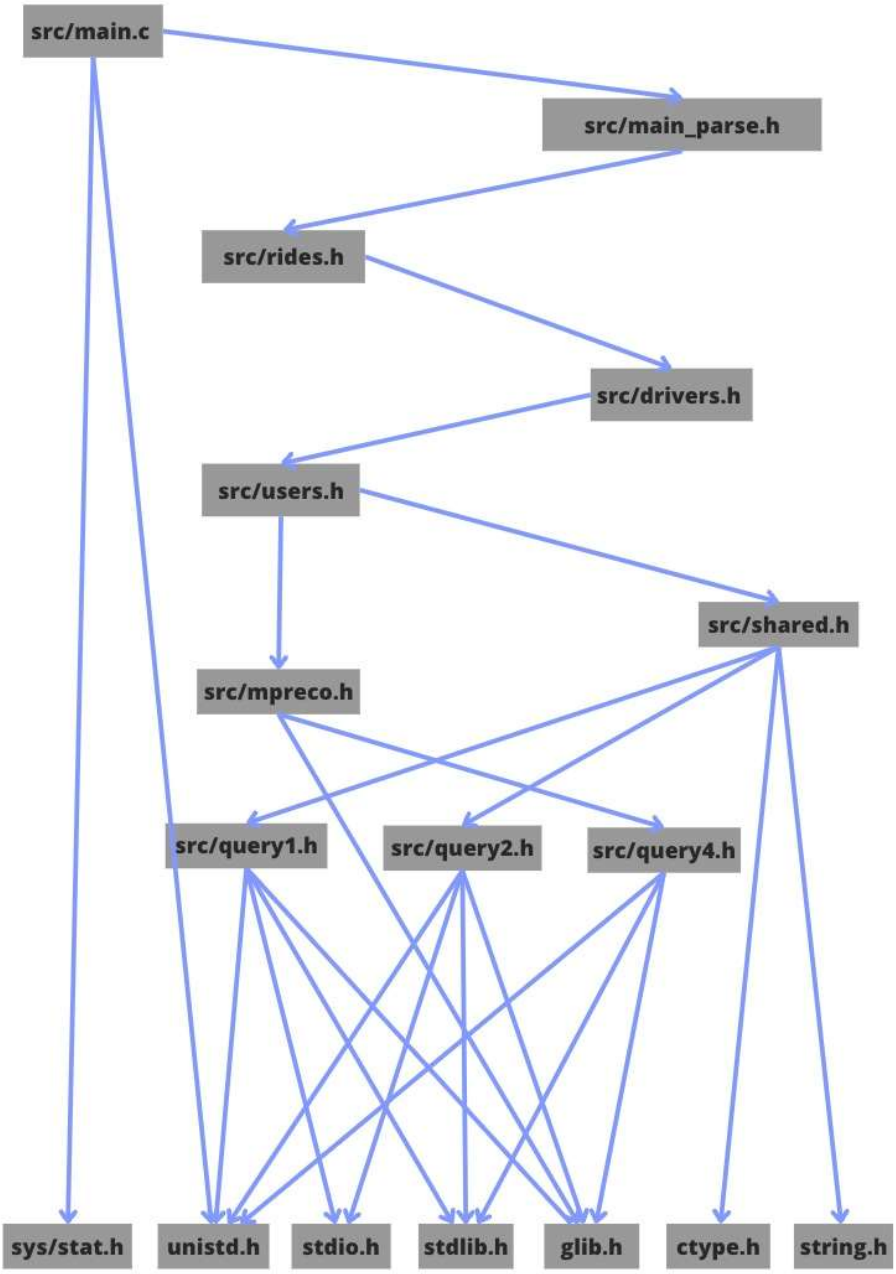

Figura 7: Grafo de dependências.