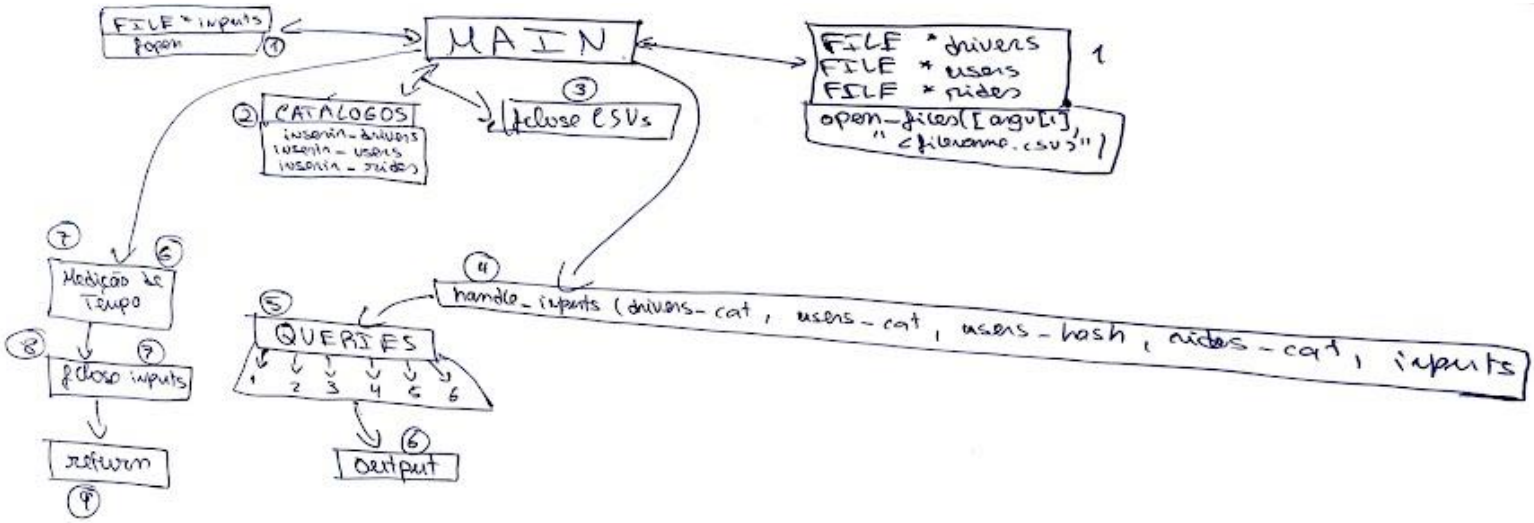
Laboratórios de Informática III – 2022/2023

Grupo 24

João Pedro Baptista (a100705), João Rodrigues (a100896), Mateus Martins (a100645)

1. Introdução e Resumo



No âmbito da cadeira “Laboratórios de Informática III”, o nosso grupo realizou um projeto que pensamos cumprir as metas e objetivos do conteúdo programático.

Começamos por fazer uma função main que dará início ao programa. Após a verificação da quantidade de argumentos, criamos um diretório “Resultados” com a ajuda da função system e o comando mkdir. De seguida são abertos os ficheiros CSV e o input.txt chamando a função open_files.

De seguida são guardados os catálogos nas respetivas variáveis, com ajuda das funções que inserem os dados nos catálogos, sendo que o catálogo dos users é constituído pela array dos mesmos e pela hash table que retorna o índice do username dado na key.

Os dados armazendos serão usados na chamada da função handle_inputs, sendo posteriormente redireconados da devida forma para a query requisitada, com o devido encapsulamento (o funcionamento das queries são explicados nos tópicos abaixo).

O tempo de execução é medido com ajuda da biblioteca “time.h” tanto no programa inteiro (main) como por cada query, sendo o mesmo impresso no ecrã.

2. Estratégias

2.1– Main – “main.c”

Começando pela função main, criamos um módulo (ficheiro .c) dedicado a ela.

A função principal do programa recebe 2 argumentos: o número de argumentos que o programa recebe (int argv), e os próprios argumentos do mesmo, colocados em forma de string (char *argv). Dadas as informações dos docentes, o programa irá receber 2 argumentos: o caminho para a pasta onde estão os ficheiros de entrada e o caminho para um ficheiro de texto que contém uma lista de comandos (queries) a serem executados.

O grupo começou por fazer um “if” que confirma que o argc é igual a 3 (o próprio programa + os dois argumentos que o programa recebe), sendo que se isto não se verificar, imprime no ecrã “Número de argumentos inválido”, seguido do número de argumentos dado entre parênteses, retornando -1 logo de seguida. Caso o argv seja igual a 3, o programa corre nas normalidades.

A próxima parte da main é responsável por abrir os ficheiros que são dados como argumentos: drivers.csv; users.csv; rides.csv e o ficheiro de inputs (usando a função open_files definida por nós). O nosso método para essa etapa consiste no seguinte: para cada um dos ficheiros CSV, criar uma string temporária (cuja memória será libertada no final da abertura do ficheiro), copiando (com o comando strcpy) o argv[1] (diretório onde se encontram os ficheiros CSV) para a string. De seguida, acrescentamos-lhe a string que corresponde ao nome do ficheiro (por exemplo: drivers.csv). Tendo já o caminho do ficheiro que pretendemos abrir, é apenas necessário abri-lo em modo “r” (read), para que nada o possa alterar.

Existe também um comando “mkdir” que cria um diretório na pasta “trabalho-prático”, chamado “Resultados”, dentro do qual serão criados os ficheiros de output.

Chama-se também as funções que inserem os dados dos drivers, users e rides nos respetivos catálogos (insere_drivers / insere_users e insere_rides) para obter na main todas as informações dos mesmos, seguido de fechá-los com a função fclose logo que os catálogos estejam preenchidos. O ficheiro inputs.txt apenas é fechado após a execução da função “handle_inputs”.

2.2 – Catálogos – “cat_drivers.c/.h” , “cat_users.c/.h” , “cat_rides.c/.h”

Para armazenar cada elemento (driver, user e ride), decidimos utilizar Structs que permitem guardar todas as informações dos mesmos, como: data de nascimento, ID, username, entre outros.

Usamos também uma hash table que, com o uso da função g_hash_table_lookup, dada uma key (username), devolve o índice desse user na array do catálogo dos users.

As funções “insere_drivers”, “insere_users” e “insere_rides” (distribuídas por 3 módulos) começam por criar uma string “temp” que irá armazenar cada linha dos ficheiros CSV nos loops.

Para criar os catálogos, decidimos utilizar 3 arrays de structs que memorizam os dados dos 3 tipos de elementos. Para esse efeito, decidimos, para cada tipo de elemento, alocar memória usando a função “malloc” com espaço para as respetivas quantidades + 1 (10001, 100001 e 1000001), pois o índice 0 das arrays não serão utilizados (por pura conveniência).

Posteriormente é feito um ciclo while fgets (para cada tipo de elemento), em que cada linha de cada ficheiro é armazenado na string temporária mencionada acima. Dentro desse ciclo será criada uma struct (também temporária), que depois será copiada para a array. Achamos adequado utilizar a função "sscanf", com vários argumentos "%[^;];" (excepto no último que tem de ser %[^\n]; pois acaba a linha), para rechear a struct temporária. É também preenchida a hash table dentro desse for, no caso da "insere_users".

No final dessa função é chamada a "handle_inputs", passando como argumentos os 3 arrays (catálogos) já preenchidos, a hash table e o ficheiro inputs.

2.3 – Inputs – "inputs.c" & "inputs.h"

Antes da execução da função que lê os ficheiros de input e executa os mesmos, criamos catálogos e a hash table duplicados para dar como argumento às queries, para garantir um melhor encapsulamento.

A função "handle_inputs" irá criar, da mesma forma das funções que inserem nos catálogos, um ciclo while fgets que irá percorrer o ficheiros de inputs, armazenando cada linha numa string temporária "input". Irá também incluir um contador "int i" para o programa "saber" em que linha está (para posteriormente utilizarmos essa informação na criação dos ficheiros de outputs.

Caso o primeiro char da mesma for 1, a função "handle_inputs" chamará a função "query1", caso o primeiro char seja 2, chamará a função query2, com a mesma ordem de ideias. E o mesmo acontecerá para as outras queries.

2.4 – Query 1 – "query1.c" & "query1.h"

Para saber se o resto da string do input (sem o 1 e o espaço) é um ID ou um username, a função "query1" utiliza uma função auxiliar (definida em randomfuncs.c & randomfuncs.h) "isDriver", que retorna um int: 1 se a string dada for um Driver (se todos os elementos dessa string forem dígitos), e, caso contário, 0 se a string for um username.

É chamada depois a função correspondente se for Driver ou User, para garantir uma maior modularidade e encapsulamento.

Caso seja um Driver, buscar-se-á ao catálogo a struct correspondente ao mesmo, através do índice (que é igual ao ID).

Se se verificar que o Driver não existe, ou do mesmo ser inativo, chama-se a função handle_inputs com argumento de string vazia, e dando return void.

Posteriormente, irá ser feito um loop for que percorre o catálogo das rides, e em cada Ride verifica se o ID do Driver é igual ao ID do Driver da Ride. Caso seja, irá proceder a calcular: total de avaliações; número de viagens e total auferido. No final do loop, será dividido o total de avaliações pelo número de viagens de modo a obter a média das avaliações. Se o número de viagens for igual a 0, a avaliação média irá ser preenchida automaticamente com o valor 0.

Todas as informações serão juntas numa única string "output", sendo que a idade será calculada com ajuda da função "age", (que calcula a idade através da data de nascimento).

Caso seja User, a estratégia utilizada foi a mesma, porém tivemos de obter o índice do user a partir da hash table, de modo a obter todas as informações do mesmo acedendo a esse índice no catálogo.

É também verificado a não existência do User e a possibilidade do mesmo ser inativo, chamando logo a função handle_inputs e dando return void.

Posteriormente irá ser utilizada outra função auxiliar (definida em outputs.c & outputs.h) “handle_outputs”, que recebe o contador referido no tópico passado e a string que foi dada como output, para que seja escrita no respetivo ficheiro. Essa função cria uma string temporária “filename” em que, utilizando a função sprintf e o contador, é preenchida com o caminho correto para o novo ficheiro ("Resultados/command%i_output.txt", counter). Para criar esse ficheiro é criado um novo FILE temporário, utilizando depois o “fopen” para o fazer, com o modo write.

Finalmente, é apenas necessário escrever o output no ficheiro recorrendo ao “fprintf”.

2.5 – Query 2 – “query2.c” & “query2.h”

A query 2 recebe como argumento o counter, os catálogos e o N em forma de string.
Com ajuda do malloc/calloc iremos alocar memória para as variáveis necessárias a esta query, sendo essas libertadas no final.

A array de recente_rides (que guarda a viagem mais recente de cada user) é inicializada com “00/00/0000” em todos os seus elementos, para que qualquer data seja mais recente do que esta.

Posteriormente é usado um loop for que percorre todo o catálogo das rides, em que para cada uma delas: verifica se o driver dessa ride está ativo; incrementa o total de avaliações desse driver com a avaliação dessa ride; incrementa em 1 o número de viagens desse driver, e se a data dessa ride for mais recente do que a data que está colocada na array recente_rides, substitui a mesma na array.

Nesta etapa, a array das avaliações médias será preenchida com a divisão do total de avaliações de cada driver com o número de viagens do mesmo.

Com a ajuda de outro for loop, irão ser calculados as N maiores avaliações médias, e o índice dos respetivos drivers serão guardados na array id_maiores (sendo que id_maiores[0] guarda o ID do driver com a maior avaliação média). Caso num_viagens == 0, a avaliação média nesse índice será 0.

De seguida, é feito o desempate diretamente na array id_maiores.

Finalizando, apenas é necessário chamar a função handle_outputs para criar o ficheiro com o output pretendido.

2.6 – Query 3 – “query3.c” & “query3.h”

Relativamente à Q3, a estrutura da mesma é bastante similar à da Q2, sendo que as diferenças são que em vez de avaliação média, temos a distância total, e sendo que a Q3 engloba também os users, usamos a hash table para dar o índice do mesmos. A partir dessa etapa, a estratégia é a mesma do que para os drivers.

2.7 – Query 4 – “query4.c” & “query4.h”

Na Q4, optamos por apenas fazer um for loop que percorria o catálogo das rides. Nesse loop existe a condição que a cidade dada como input, tem de ser igual à cidade de cada Ride. Guarda-se a distância de cada ride numa variável int “distance”; a classe do carro do respetivo

driver numa variável char* class, procedendo a preencher as variáveis taxa_base e taxa_dist, para realizar o cálculo de cada viagem.

Em cada viagem, são incrementadas as variáveis “preco” com o preço total da viagem (sem gorjeta) e “contagem”, em 1. No final, verifica-se se a contagem é igual a 0, e caso seja, chama-se logo a função handle_outputs com os argumentos counter e “”, para realizar o respetivo output. Caso contrário, é realizada a divisão entre o preço total e a contagem de viagens de modo a obter o resultado final, procedendo a dar output do mesmo com ajuda da função sprintf e handle outputs, com 3 casas decimais.

2.8 – Query 5 – “query5.c” & “query5.h”

A Query 5 é muito similar à Q4, com a exceção de que em vez de usarmos a cidade como condição para o for loop, usamos as duas datas. Para sabermos se a data de cada ride está entre essas duas datas (inclusive), usamos a função (definida em randomfuncs.c) most_recent que recebe 2 datas em string como argumento, e retorna 1 se a primeira data for a mais recente, 2 se for a segunda, ou 3 se forem iguais.

No final, é apenas preciso chamar a função handle_outputs com a mesma lógica da Q4.

2.9 – Query 6 – “query6.c” & “query6.h”

Inicialmente, necessitamos de separar o argumento da Query em 3 argumentos diferentes, cidade, data 1 e data 2. Para isso usamos a função memcpy.

Para resolver a Query 6, usamos novamente um for loop que percorre o catálogo das rides, e usa as condições dos loops das Q4 e Q5, pois a Q6 pede a distância média entre duas datas, numa certa cidade.

Para determinar a distância média, usamos dois acumuladores que vão sendo incrementados dentro do loop (distance & contagem). Os seguintes passos são semelhantes à Q4 e Q5.

2. Limitações

Encontramos algumas limitações na realização do projeto, tais como:

- Na abertura inicial dos ficheiros, tentamos colocar a definição dos mesmos num ficheiro “main.h”, de modo a que conseguíssemos acessá-los em qualquer parte do programa, incluindo o main.h no mesmo. Porém, infelizmente deparámo-nos com o erro de “multiple definitions”, e depois de muitas tentativas de resolução sem sucesso, encontrámos a brilhante ideia de passar os catálogos como argumentos para o resto das funções.

3. Aspetos a melhorar

- Em relação ao encapsulamento, temos criar funções para que seja possível ir buscar os dados necessários nas queries, sem as mesmas terem acesso ao resto (estrutura, catálogos, etc.).

- Fazer uma melhor modularização, apesar de pensarmos que a atual é muito boa.

- Tentar melhorar a leitura do ficheiro “rides.csv” (mais de 50% do tempo de execução do programa inteiro é utilizado pela mesma), apesar de parecer um pouco impossível preencher o catálogo das Rides sem percorrer o ficheiro inteiro.

- Adicionar módulo de estatísticas para ser mais rápido ir buscar os resultados dos inputs anteriores, e também conseguindo um melhor encapsulamento.

- Fazer uma melhor documentação.