# Relatório da Fase 1 – LI3 – Grupo 13

Fernando Jorge da Silva Pires (a77399)
Edgar Carvalho Ferreira (a99890)
João Gomes Dias de Faria (a100553)

Novembro 2023

# Índice


## 1 - Introdução

## 2 – Desenvolvimento

2.1 - Modularidade, encapsulamento e memory leaks

2.2 - Modelos de dados

2.2.1 - Utilização da biblioteca glib

2.3 - Execução inicial

2.4 - Leitura e inserção de dados

2.5 - Queries

2.5.1 - Queries desenvolvidas

2.6 - Estrutural geral

2.7 - Desempenho geral

2.8 - Trabalho futuro e melhorias

## 3 – Conclusões

# 1 - Introdução

Este relatório foi elaborado no âmbito da unidade curricular de Laboratórios de Informática III no ano letivo 2023/2024. Os objetivos desta fase são realizar o *parsing* dos ficheiros de entrada (*users.csv*, *passengers.csv*, *reservations.csv* e *flights.csv*), realizando validação campo a campo, implementação do modo *batch* e realização de, no mínimo, 6 das 10 *queries* existentes. Tudo isto, respeitando o encapsulamento, a modularidade e sem *memory leaks*.

# 2 – Desenvolvimento

## 2.1 - Modularidade, encapsulamento e memory leaks

No que toca a modularidade, procedemos à criação de módulos. Para tal, foram criadas as pastas ***include*** e ***src***. Dentro de cada uma delas temos diferentes módulos, como ***catalogs***, ***data_manager*** , ***program***, ***queries*** e ***stats.*** Além disto, através do make, copiamos automaticamente para a pasta desejada o **programa-principal**.

Já o encapsulamento, que tem como principal razão a separação das interfaces e das respetivas implementações de forma que apenas se seja possível aceder à interface e não às implementações, é garantido pela não possibilidade de acesso ao conteúdo das estruturas fora do ficheiro de implementação. Assim sendo, foram criados ***setters*** e ***getters*** para os campos do modelo de dados que seja pedido.

No que toca a *memory leaks*, nesta fase, o programa apresenta 0.019Mb, valores que se devem à utilização da biblioteca *glib* e algo que não nos é possível alterar.

## 2.2 - Modelos de dados

De forma a armazenar em memória os diferentes dados foram criados 5 catálogos com as respetivas distribuições:

- ***flights –*** Armazenadas informações relativas a voos e a passageiros dos mesmos;
- ***reservations*** – Armazenadas informações relativas a reservações de hotéis
- ***users –*** Armazenados atributos relativos a utilizadores
- ***airports*** – Armazenadas informações relativas a aeroportos utilizados
- ***hotels*** – Armazenadas informações relativas a hotéis

Os dois últimos modelos de dados foram implementados de forma a facilitar acessos a dados requeridos em algumas queries, como iremos abordar posteriormente.

## 2.2.1 - Utilização da biblioteca *glib*

Sendo possível a inclusão desta biblioteca no projeto, as suas estruturas de dados já otimizadas foram o maior atrativo para a sua utilização. Além destas, já se encontravam predefinidas funções de manipulação das tais estruturas, algo que nos facilitou a transformação do nosso processo de desenvolvimento mental do projeto em código.

## 2.3 - Execução inicial

No que toca à execução, são recebidos na ***main*** os argumentos necessários para passar para o ***batch_mode***: o caminho para os ficheiros e os comandos necessários das *queries.*

No ***batch_mode***, são inicializadas as estruturas de dados que servem para guardar a informação, a *GHashTable* (provenientes da biblioteca *glib*), uma para cada um dos modelos de dados acima abordados. São abertos todos os ficheiros *CSV* e os ficheiros das *queries*. Tudo isto é passado como argumento ao ***parser***, que tratará da leitura da informação dos ficheiros *CSV* e da respetiva inserção em cada uma das tabelas.

## 2.4 - Leitura e inserção de dados

Esta fase é desenvolvida e tratada pelo ***parser*** que tem como função ler os ficheiros CSV que contêm os dados para inserir em memória, todos lidos linha a linha e posteriormente validados no ***validater***.

Caso todos os campos da linha lida do *CSV* se encontrarem devidamente corretos de acordo com as informações dadas para a validação, a informação será condensada num dos modelos de dados e posteriormente adicionada às ***GHashTables*** que irão guardar tal informação em memória.

## 2.5 - Queries

Esta fase é iniciada no ***batch_mode***, após o *parse* da informação e inserção nas tabelas. Aqui é chamada a função ***query_manager***, função implementada noutro ficheiro e que recebe como argumento todas as tabelas e o ficheiro com as queries. Por fim, no ***batch_mode*** são fechados todos os ficheiros abertos e dada free à memória ocupada por cada uma das tabelas.

Já no ***query_manager***, será obtida, linha a linha, a informação contida no ficheiro passado e tratada de forma a ser passada para cada uma das *queries* consoante o conteúdo. Cada *query* terá um ficheiro .c e .h, de forma a conservar o encapsulamento.

## 2.5.1 - Queries desenvolvidas

Para esta fase do projeto foram implementadas 6 *queries* que passaremos a explicar:

- **Query 1 – Resumo de utilizador, reserva ou voo**

Esta *query* faz uso da rapidez de acesso a informação de cada *hashtable*. Verifica o id que é passado como input, identificando a qual dos modelos de dados corresponde e a existência. Após isso, escreve num ficheiro, o resumo consoante o pedido e de acordo com o previamente estabelecido de cada um dos modelos de dados (*user*, *reservation* e *flight*)

- **Query 2 – Listagem de voos ou reservas de um utilizador**

Esta *query,* contrariamente à anterior, pode receber mais que um argumento, sendo necessário tratar o input obtido. O primeiro argumento será o utilizador ao qual será necessário buscar a segunda parte do input, ou *flight* ou *reservation,* caso não seja passado um segundo argumento, serão apresentadas as duas hipóteses. Em ambos os casos, serão percorridas os *GArrays*, existentes (previamente ordenados) no modelo de dados *users*, que contêm a informação pedida e escritos num ficheiro.

- **Query 3 - Classificação média de um hotel**

A *query* 3 irá fazer uso da tabela dos hotéis, por nós criada de forma a mais fácil obtenção de resultados. O argumento dado será o *id* do hotel, chave da tabela dos hotéis, que servirá para obtermos, e escrevermos no ficheiro de resultado, o *rating* (ou seja, a classificação média) do hotel cujo id foi passado, algo previamente calculado e obtido através do ficheiro das *reservations*.

- **Query 4 – Listagem das reservas de um hotel**

Esta *query*, tal como a anterior, vai fazer uso da tabela dos hotéis e das suas características. O argumento vai, novamente, ser o id do hotel cujas informações queremos obter. Será obtido esse hotel através da chave, o id, e percorrida o *GArray* ordenada das reservas desse hotel e vai escrever, por ordem das reservas, as informações requeridas, ou seja: o *id* do hotel, a data de início e fim da reserva, o *id* do utilizador que fez tal reserva, a avaliação dada e o preço pago por noite.

- **Query 7 – Listagem de aeroportos com as N maiores medianas de atraso**

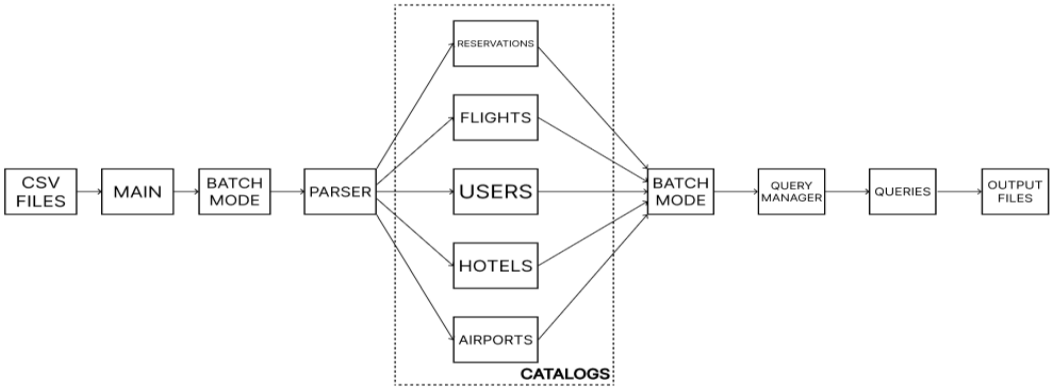
A *query* 7 irá fazer uso da tabela dos aeroportos e das suas características, insidindo no campo do atraso mediano. A mediana de atraso de cada aeroporto é calculada a partir da diferença entre a hora estimada e a real de partida de cada voo. Essa informação é guardada e ordenada de forma a fazer o cálculo da mediana. Tendo essa informação inserida na tabela de aeroportos, a tabela será percorrida de forma a obter os N (valor passado como argumento) aeroportos com a maior mediana de atrasos, sendo escrito o nome e a mediana no ficheiro de saída.

- **Query 9 – Listagem de utilizadores cujo nome começa com o prefixo dado**

A *query* 9, e última desta fase, incide sobre a tabela de utilizadores. Esta será percorrida de forma a verificar, entrada a entrada, se o argumento passado é, ou não, prefixo do nome de cada uma das entradas. Posteriormente, estes valores serão ordenados por ordem alfabética e, em caso de empate, pelo valor de id de cada um. Os resultados (*id* e nome do utilizador) são, posteriormente, colocados num ficheiro.

## 2.6 - Estrutura geral

Graficamente, quanto a estrutura geral do projeto, podemos dividir da seguinte maneira




## 2.7 - Desempenho geral

No que toca ao desempenho geral, foram feitos 10 testes e retirados os 2 com valores mais desenquadrados em 3 diferentes computadores e os resultados foram os seguintes:

| | COMPUTADOR 1 | COMPUTADOR 2 | COMPUTADOR 3 |
|---|---|---|---|
| CPU | Intel i7-6500U CPU @ 2.50GHz | Intel i5-1135G7, 11th Gen @ 4.200GHz | Intel i7-11800H, 11th Gen @ 2.30GHz x 8 |
| RAM | 23.9 Gb | 7.56 GiB | 15.3 Gb |
| OS | WSL 2.0 / Ubuntu 22.04.2 | 6.6.1-Arch1-1 | Linux Mint 21.2 Cinnamon |
| TEMPO LEITURA DE FICHEIROS E INSERÇÃO NAS TABELAS | 1.345 segundos | 0.712 segundos | 0.563 segundos |
| TEMPO EXECUÇÃO | 0.105 segundos | 0.019 segundos | 0.013 segundos |
| TOTAL | 1.450 segundos | 0.731 segundos | 0.576 segundos |

É possível verificar que grande parte do tempo de execução do programa se trata da leitura dos ficheiros, da validação e inserção nas diferentes tabelas. Tempo que se justifica porque, relativamente ao código, é feita ordenação de algumas das componentes dos diferentes modelos de dados aquando cada inserção.

## 2.8 - Trabalho futuro e melhorias

No que diz respeito ao trabalho futuro, além da implementação das restantes queries, do modo interativo e do modo de testes, vai ser necessário limar alguns detalhes no que toca a leitura de ficheiros e respetivas operações de inserção de informação nas tabelas (como o *sorting* de *arrays* de cada modelo de dados). Poderá também haver a necessidade de alterar alguns dos campos dos modelos de dados caso, com o aumento de tamanho dos datasets, se atinja valores demasiado grandes e cujo impacto assim o justifique.

# 3 - Conclusões

Em jeito de conclusão, podemos afirmar que este projeto ajudou a estabelecer conceitos como encapsulamento e modularidade, além de nos incentivar à exploração de bibliotecas e suas funcionalidades numa linguagem como o C. Características como o controle direto sobre o hardware e manipulação de apontadores tornam esta linguagem bastante boa para perceber melhor temas como o performance e gestão de memória.

Apesar disto, trabalhar com C pode resultar em alguns problemas relacionados com *memory leaks* algo que foi difícil de lidar durante o desenvolvimento do projeto.