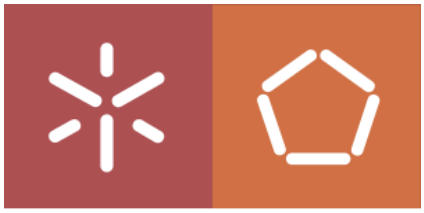

Universidade do Minho
Escola de Engenharia

Trabalho Prático
Laboratórios de Informática III
Fase 2

Grupo 82

Filipa Oliveira da Silva A104167

Maria Cleto Rocha A104441

Mário Rafael Figueiredo da Silva A104182

Índice

Introdução

Este projeto foi desenvolvido no âmbito da unidade curricular de Laboratórios de Informática III no ano letivo 2023/2024, unidade esta que pretende dar a consolidação aos alunos de conhecimentos essenciais da linguagem C, e de Engenharia de Software, mais precisamente modularidade e encapsulamento.

Após a conclusão da primeira fase, que consistia em realizar o *parsing* dos dados e o modo *batch, nã*o conseguimos completar todos os requisitos do programa, mas implementamos mais algumas *queries*, realizamos testes de desempenho, verificação de dados e criação de um modo interativo.

O modo interativo consiste numa interface gráfica na qual o programa é executado sem argumentos e onde cada comando é interpretado e executado individualmente, com o respetivo resultado apresentado no terminal.

Nesta fase, foi introduzido um novo *dataset* com uma dimensão muito maior do que a do *dataset* anterior, o que fez com que tivéssemos de nos adaptar melhor no que toca ao tipo de estruturas utilizadas.

Arquitetura e Estruturas

Na primeira fase do projeto, tínhamos o *parsing* dos ficheiros a ser feito individualmente e optamos por armazenar os dados em *arrays dinâmicos*, o que não era de todo eficiente, e para além disso contradizia os princípios de encapsulamento que são de grande importância na elaboração deste projeto.

Após a apresentação do projeto inicial aos docentes na primeira fase, apercebemo-nos de que a nossa implementação precisava de bastantes melhorias. Assim, nesta segunda fase, adaptamos o nosso código a um tipo de estruturas diferente que permite não só um acesso mais rápido dos dados, mas também a sua melhor organização, as *hash tables*. No entanto, continuamos a usar *arrays dinâmicos* para processar os *passengers*, para que não fosse excedido um limite de memória, já que estão associados a um menor número de parâmetros e há uma grande quantidade de passageiros para serem lidos no *dataset grande*.

Posto isto, com o início da segunda fase, colocamos o nosso foco na correção do código para atender ao encapsulamento e, depois disso, tentamos fazer o maior número de *queries* possível.

Depois das alterações necessárias no que toca ao novo tipo de estruturas utilizadas, apresentamos o diagrama da nossa arquitetura atual:

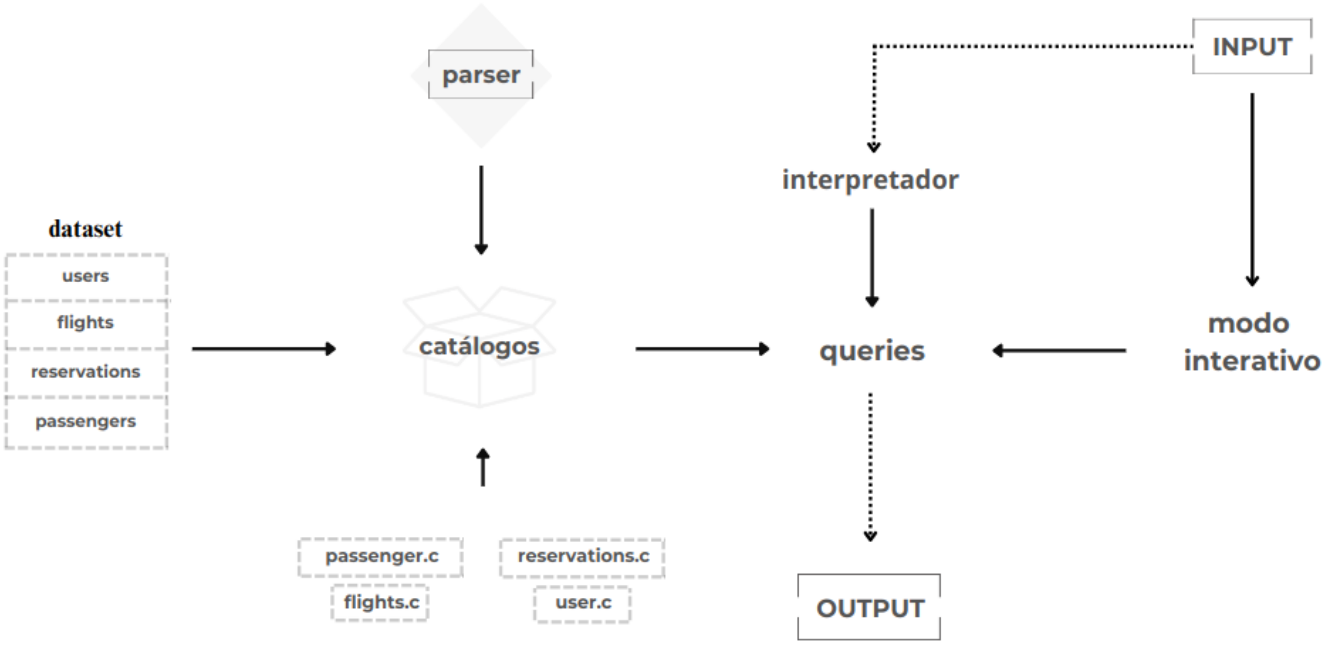

Fig.1: Diagrama da arquitetura utilizada no projeto

Queries

Na primeira fase, tínhamos como objetivo realizar 6 *queries*, o que não foi possível, acabando apenas por conseguir fazer as *queries* 1,3,4 e 9. Nesta fase, devido à mudança do tipo de estrutura, redefinimos as que tínhamos feito anteriormente e acrescentamos à lista a 2,5 e 7, uma vez que não tivemos sucesso na realização das restantes.

Query 2: consiste em *"listar os voos ou reservas de um utilizador, se o segundo argumento for flights ou reservations, respetivamente, ordenados por data (da mais recente para a mais antiga)"*, efetuando a ordenação pelo identificador em caso de empate. Para isto, criamos uma nova estrutura "*CombinedInfo*" que serve para armazenar informações consolidadas sobre voos e reservas, permitindo que estes sejam processados e armazenados juntos apesar de virem de fontes diferentes. Usamos também duas funções auxiliares: "*compareDates*" que compara duas datas facilitando depois a sua ordenação; e "*compareCombinedInfo*" que resolve o caso de empate, fazendo comparações entre *IDs*. Estas implementações permitem uma abordagem mais eficaz, facilitando operações complexas como a filtragem e ordenação, e simplificam a lógica de saída.

Query 5: consiste em *"listar os voos com origem num dado aeroporto, entre duas datas, ordenados por data de partida estimada (da mais recente para a mais antiga)"*, sendo que no caso de dois voos terem a mesma data, o identificador do voo é usado como critério de desempate. Criamos então duas funções auxiliares: "*comparaDatas*" que compara as datas e é, por isso, fundamental na filtragem dos voos dentro do intervalo de datas especificado; e "*comparaVoos*" que resolve o caso de empate, fazendo comparações entre os *IDs* dos voos. Tal como na *query* anterior, estas funções ajudam na eficiência do código e tornam mais fácil a organização dos dados.

Query 7: consiste em *"listar o top N aeroportos com a maior mediana de atrasos, calculados, em segundos, a partir da diferença entre a data estimada e a data real de partida, para os voos com origem nesse aeroporto"* e no caso de empate são usados os nomes dos aeroportos como critério. Criamos também funções auxiliares para tornar o código mais eficaz: "*compararInt*" para facilitar a comparação na ordenação pela função *qsort*; "*encontrarAeroporto*" para ajudar na procura do aeroporto; "*calcularMediana*" que é útil para calcular o valor dos atrasos; e "*compareDelays*" responsável por ordenar os aeroportos dando prioridade aos de maior mediana e usando o nome do aeroporto como critério de desempate.

Modo Interativo

A implementação do modo interativo foi um dos requisitos principais desta fase, com uma interface gráfica no terminal e, para isso, utilizamos a biblioteca *ncurses*, uma vez que já estamos mais familiarizados de projetos de anos anteriores e já que facilita a criação de um modo gráfico que seja capaz de:

-> processar as *queries* e devolver os seus resultados;

-> imprimir os resultados em várias páginas caso estes sejam grandes demais;

Desenvolvemos, por isso, um *menu* para que seja possível escolher a *query* que se deseja obter uma resposta, constituído da seguinte forma:

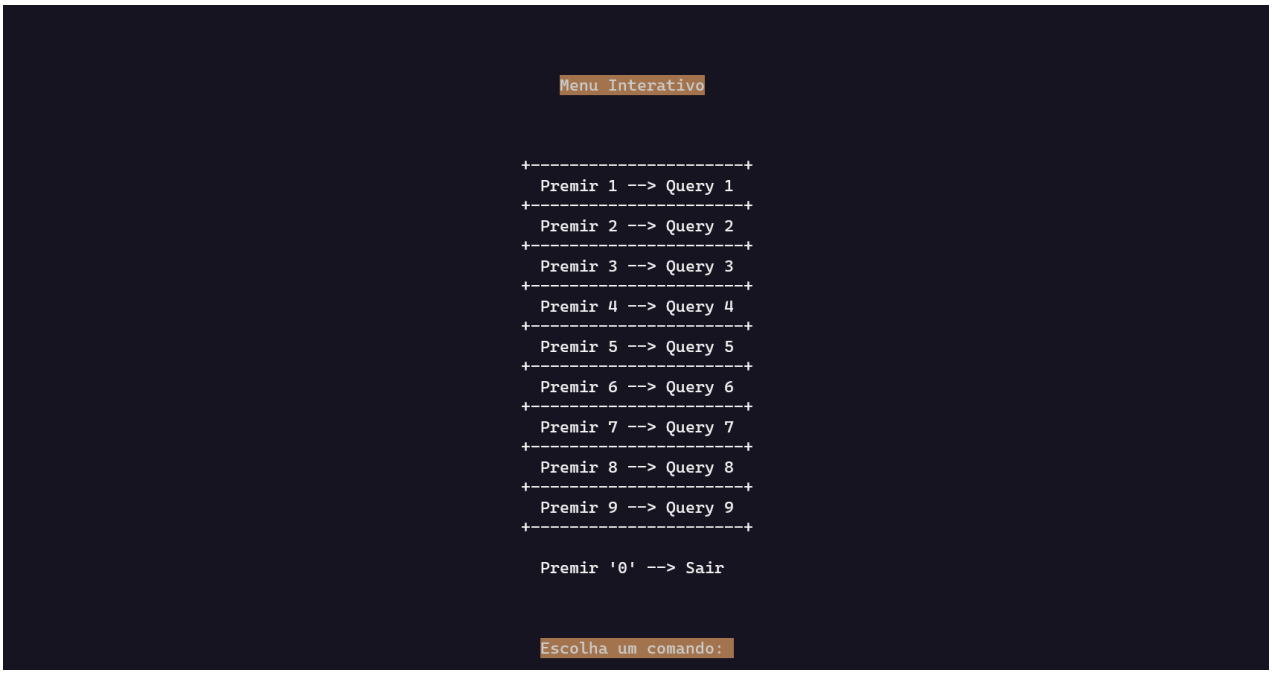

Fig.2: Menu principal onde se encontram as *queries* para escolha

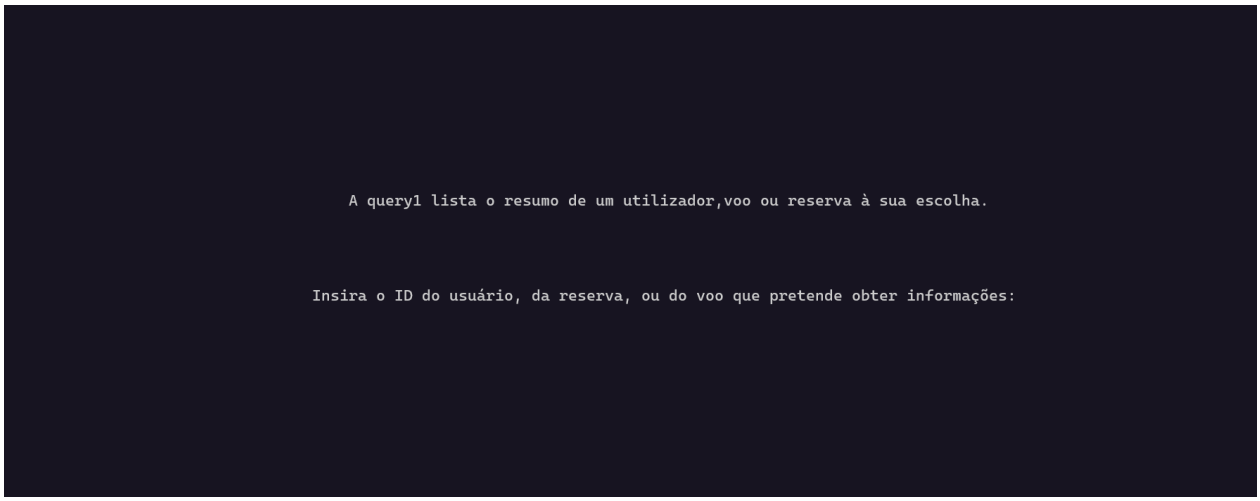

Fig.3: Exemplo da implementação da query1

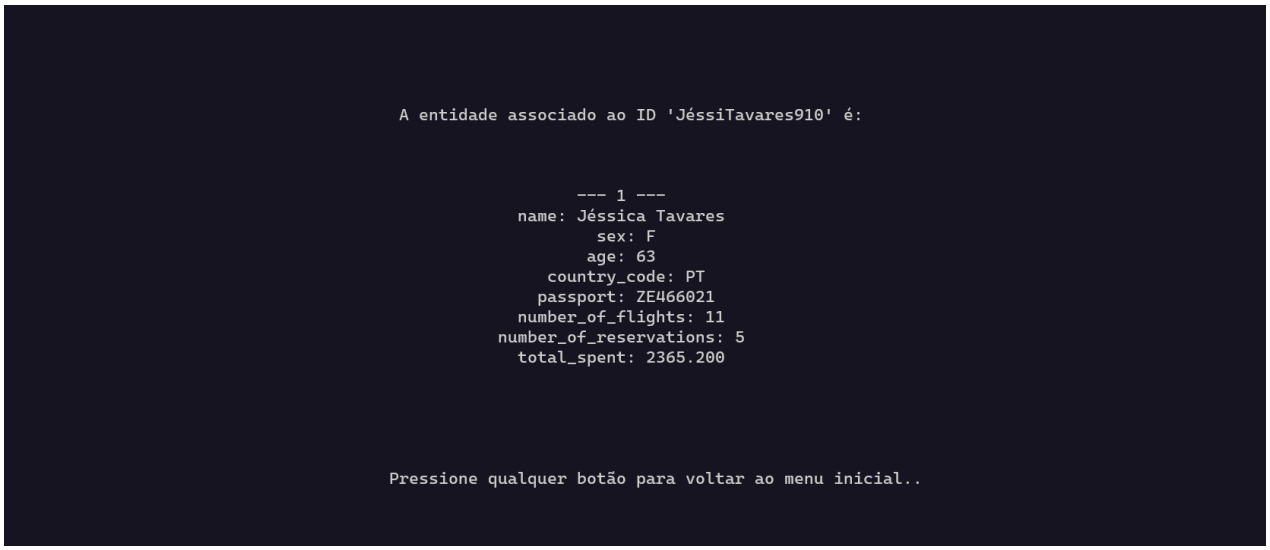

Fig.4: Exemplo de *output* da query1 com *formatFlag "F"*

Desempenho e testes

Para testar a performance do nosso programa escolhemos o *dataset* pequeno, uma vez que era aquele que tínhamos mais certezas que funcionasse corretamente. Realizamos estes testes 3 vezes para cada *query* de modo a tornar os valores mais fidedignos e reais, tendo obtido os seguintes resultados:

→ ***Query 1:***

- Caso das reservas - 0,0000345 segundos, 0kb de memória, (como temos uma função que encontra a reserva pelo *Id.* Para além disso só interage com a *hash table* dos *users* o que ajuda a ter um menor tempo);
- Caso dos voos - 0,0007373 segundos, 0kb de memória (este caso é similar, uma vez que temos uma função que encontra o voo pelo *Id*, no entanto como também interage com os passageiros *(arrays)* requer mais tempo);
- Caso dos *users - 0,002344 segundos, 0kb de memória (Aqui temos uma função com a mesma finalidade que os outros casos*, mas demora mais tempo porque não interage só com os passageiros, mas também com as reservas);

→ ***Query 2:*** 0.0018441 segundos, 0 kb de memória (como temos uma função que encontra o utilizador através do *Id,* facilita a procura, diminuindo assim o tempo);

→ ***Query 3:*** 0.0015038 segundos; 0 kb de memória;

→ ***Query 4:*** 0,003990 segundos, 0kb de memória

→ ***Query 5:*** 0,0003256 segundos, 0kb de memória;

→ ***Query 6:*** 0,22374383 segundos, 50,6kb de memória;

→ **Query 7:** 0,0008915 segundos, 0kb de memória;

→ **Query 9:** 0,000627375 segundos, 0kb de memória.

Implementamos também o modo de "testes", tendo optado por expor os resultados no próprio terminal, que apresenta o tempo total de execução do programa assim como a memória total utilizada.

Dificuldades sentidas

Nesta fase, tal como na primeira, sentimos grandes dificuldades, especialmente porque tínhamos bastantes parâmetros que precisavam de ser mudados para ir de acordo com as regras de encapsulamento. Por isso, fizemos disso a nossa prioridade antes de começarmos a tratar dos objetivos da segunda fase propriamente dita.

Depois de termos conseguido a maior parte dos módulos divididos e das estruturas definidas corretamente, continuamos a executar o nosso programa utilizando os *arrays dinâmicos* da mesma forma que na primeira fase. Ao longo do tempo fomo-nos apercebendo, também devido ao tamanho do novo *dataset*, que não era a estratégia ideal para trabalharmos e, apesar de o tempo não ter estado do nosso lado, decidimos mudar o tipo de estruturas de armazenamento de dados para *hash tables*, tal como referido acima.

Relativamente a esta parte, sentimos dificuldade também na definição das *queries* e na sua adaptação ao novo tipo de estruturas, o que resultou num insucesso no que diz respeito à sua realização completa. Para além disso, apesar de termos conseguido uma classificação de 100% em algumas *queries* quando implementadas ao *dataset* pequeno, com o *dataset* grande desta segunda fase já não foi possível, mas ficamos satisfeitos com o facto de termos conseguido ter pelo menos o programa a rodar com o novo *dataset* sem que fossem excedidos limites de tempo ou memória, apesar de incompleto.

O modo interativo foi também um desafio, uma vez que tivemos alguns problemas em conseguir implementá-lo a todas as *queries*. Uma prova disso é o facto de não termos conseguido fazê-lo em todas, deixando apenas naquelas que achamos mais simples, o que excluí as *queries* que obrigavam a que os *outputs* fossem apresentados em diferentes páginas, não cumprindo também com outro dos objetivos que o modo interativo exigia.

Conclusão

Como é notável, esta fase foi muito mais desafiante do que a anterior pelo que não obtivemos os melhores resultados nem, de todo, aquilo que tencionávamos já que ainda ficou muito por fazer e aperfeiçoar. No entanto, estamos satisfeitos com o nosso esforço e com a nossa melhoria relativamente à primeira fase, já que, a nosso ver, conseguimos melhorar bastantes conceitos que anteriormente não foram bem sucedidos.