



UNIVERSIDADE DO MINHO
Departamento de Informática

FASE 2

Laboratórios de Informática III

Grupo 07

Flávio Alexandre Silva (A97352)
João Pedro Malheiro da Costa (A84701)
José Pedro Torres Vasconcelos (A100763)

25 de janeiro de 2024

Índice

1	Introdução	2
2	Desenvolvimento	3
2.1	Arquitetura de aplicação	3
2.2	Validação de dados	4
2.3	<i>Parsing</i> e inserção nas estruturas de dados	5
2.4	Modularização e Encapsulamento	5
2.5	Queries	5
3	Otimizações	6
4	Modo interativo	7
5	Testes e desempenho do programa	9
6	Dificuldades sentidas	10
7	Conclusão	11

1 Introdução

Este projeto está a ser desenvolvido no âmbito da Unidade Curricular de Laboratórios de Informática III, no presente ano letivo de 2023/2024, e tem como objetivos a consolidação de conhecimentos essenciais da linguagem C e de Engenharia de Software, nomeadamente, modularidade e encapsulamento, estruturas dinâmicas de dados, validação funcional, e medição de desempenho, assim como a consolidação do uso de ferramentas essenciais ao desenvolvimento de projetos em C, nomeadamente, compilação, linkagem, definição de objetivos de projeto com base nas suas dependências, depuração de erros, avaliação de desempenho e consumo de recursos, e gestão de repositórios colaborativos.

Nesta segunda fase do projeto, foi-nos proposto que implementássemos o modo *interativo*, as *queries* não implementadas na primeira fase, o *programa-testes*, que o nosso programa corra sem *memory leaks*, tenha uma boa estratégia de *modularização* e *encapsulamento*, assim como estruturas de dados e algoritmos adequados.

No final, pretende-se que o nosso programa responda, com sucesso, a todos estes requisitos, com alta performance.

2 Desenvolvimento

2.1 Arquitetura de aplicação

Após a apresentação da primeira fase, decidimos realizar as mudanças que nos foram sugeridas pela equipa docente, em particular, implementar um gestor que conhece e tem acesso a todos os catálogos, nos quais estão guardados todas as funções das respetivas entidades.

Com o intuito de se ter uma visão geral do funcionamento da nossa aplicação, foi criada o seguinte diagrama:

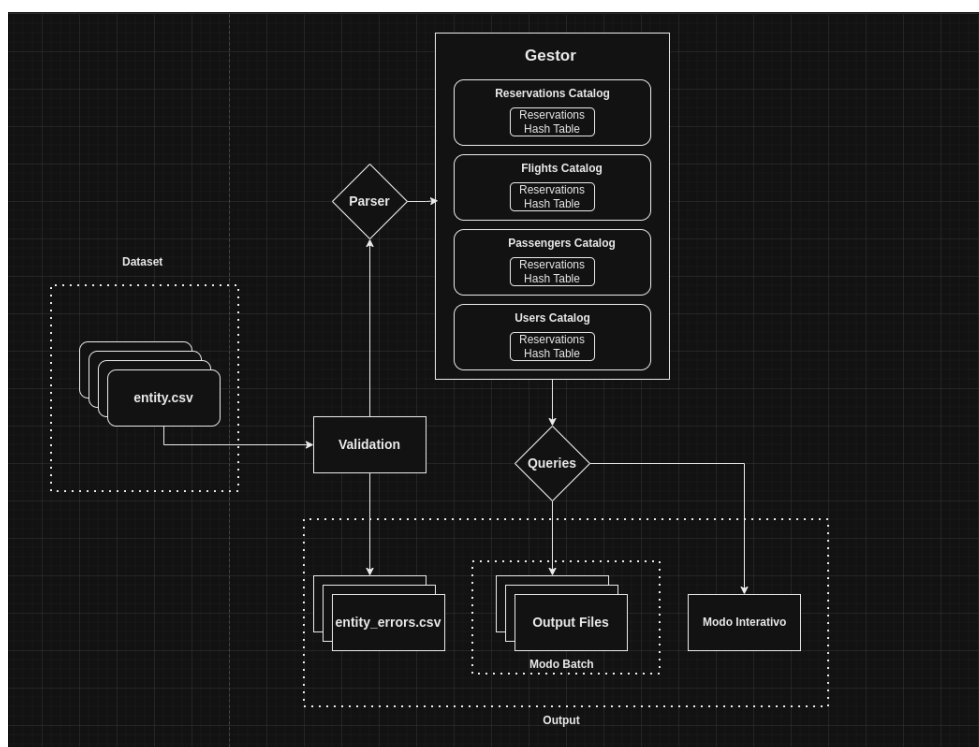


Figura 1: Diagrama da arquitetura da nossa aplicação.

Cada módulo novo ou que sofreu alterações nesta fase do projeto, assim como o funcionamento do modo *interativo*, serão explorados com mais detalhe nos próximos capítulos deste relatório.

2.2 Validação de dados

De acordo com o enunciado que nos foi fornecido, tal como na primeira fase do projeto, tivemos que ter o cuidado de verificar a validade de cada campo dos ficheiros *csv*. Assim, criamos um módulo responsável por esta validação, para o *user*, *flight*, *passenger*, *reservation*. Este integra diversas funções genéricas de validação, abrangendo *strings*, *ints*, *doubles* e *dates*, que são tipos específicos de dados essenciais para a filtragem. As funções *invalid_flights*, *invalid_users*, *invalid_passengers* e *invalid_reservations* percorrem cada linha do respetivo ficheiro *.csv*, dividindo-a em tokens separados por ";" usando *strtok*. Ao validar cada campo da linha com a função respectiva ao seu formato, a variável *checker* é inicializada com o valor 1 mas pode se tornar 0 se um termo for inválido ou se a linha não tiver o número correto de campos, resultando na escrita da linha no arquivo *file.errors.csv* correspondente.

Para verificar se determinados usuários ou voos inválidos estão presentes nas reservas ou nos passageiros e precisam ser validados, utilizaremos duas listas: *invalid_uids* e *invalid_fids*, que armazenarão os valores dos IDs dos usuários e dos voos inválidos, respectivamente. Essas listas serão usadas para verificar a presença desses IDs nas listas de passageiros (*passengers*) e reservas (*reservations*).

Se a sua presença for confirmada, atualizamos o valor do verificador (*checker*) para 0.

2.3 Parsing e inserção nas estruturas de dados

Tal como na primeira fase do projeto, depois de chegarmos à conclusão que uma dada linha é válida, verificamos se já existe algum registo, na respetiva *Hashtable*, com o *id* especificado. Se já existir, eliminamos o novo registo, se não, inserimos na *Hashtable*. Após a apresentação da primeira fase, concluímos que a utilização de *hash tables* para cada uma das entidades, foi uma boa opção. Sendo assim, mantivemos esta decisão na segunda fase do projeto.

2.4 Modularização e Encapsulamento

Com o objetivo de melhorar a modularização e encapsulamento do nosso programa, foi criado um gestor principal que tem acesso a cada catálogo das entidades, que contém a respetiva *hashtable*. A execução do nosso programa passa sempre pelo gestor, que por sua vez, chama as funções necessárias para a execução do programa.

Foram definidos os *gets* e *setters* no catálogo de cada entidade, para que, ao correremos o programa, estas sejam chamadas, evitando assim a quebra do encapsulamento, algo que acontecia na primeira fase do projeto. A utilização destas funções aumentou os *memory leaks* do nosso programa, pelo que os tivemos que resolver.

Para além disto, passamos as *structs* de cada entidade, do respetivo ficheiro *.h* para o ficheiro *.c*, contribuindo para o encapsulamento da aplicação.

2.5 Queries

Na primeira fase, implementamos as *queries* 1,2,3,4,5,7,8 e 9, ficando a faltar a implementação das *queries* 6 e 10 para esta segunda fase do projeto. As principais mudanças que ocorreram nas *queries* implementadas na primeira fase, foram realizadas com o intuito de estas não quebrarem o encapsulamento do nosso programa. Apesar de termos previsto conseguir implementarmos a *queries* que faltavam, isto acabou por não se concretizar.

3 Otimizações

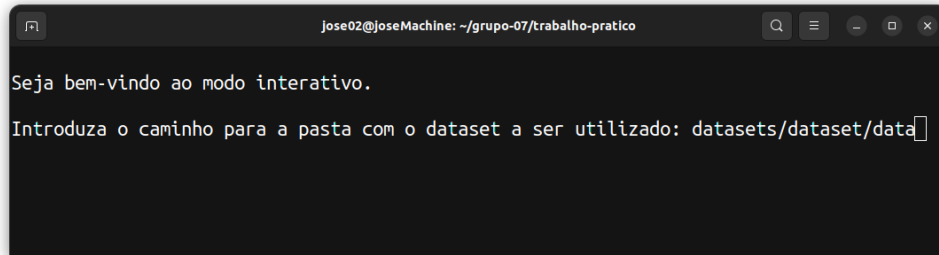
Logo após o início da segunda fase do trabalho prático ter começado, otimizamos o nosso programa de forma a reduzir o tempo de execução do mesmo, passando de cerca de 6 segundos para, sensivelmente, 1,2 segundos, ainda com o *dataset* da primeira fase. Esta redução deveu-se ao facto de termos trocados as estruturas de dados auxiliares de *GLists* para *GPtrArrays*, melhorando assim, não só a performance, mas também a memória utilizada pelo programa.

Após a implementação dos *gets* e *setters*, em busca de melhorarmos o encapsulamento do nosso programa, tivemos que refactorar o nosso código, passando a utilizar estas funções. Com o objetivo de diminuirmos o número de chamadas destas funções nas *queries* desenvolvidas, utilizamos variáveis auxiliares para guardar as informações relevantes para o nosso programa.

4 Modo interativo

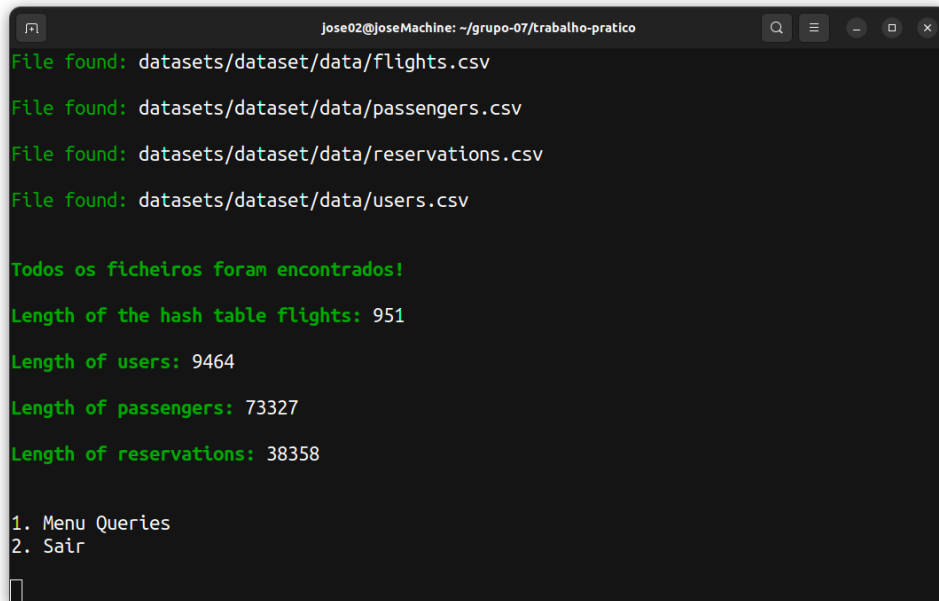
Nesta fase do projeto, um dos requisitos principais foi a implementação de um modo interativo, com interface gráfica no terminal. Através do modo interativo, o utilizador é capaz de:

- Escolher a *query* que deseja executar;
- Ler o enunciado antes da execução de uma determinada *query*;



```
Jose02@JoseMachine: ~/grupo-07/trabalho-pratico
Seja bem-vindo ao modo interativo.
Introduza o caminho para a pasta com o dataset a ser utilizado: datasets/dataset/data
```

Figura 2: Página inicial do modo interativo.



```
Jose02@JoseMachine: ~/grupo-07/trabalho-pratico
File found: datasets/dataset/data/flights.csv
File found: datasets/dataset/data/passengers.csv
File found: datasets/dataset/data/reservations.csv
File found: datasets/dataset/data/users.csv

Todos os ficheiros foram encontrados!
Length of the hash table flights: 951
Length of users: 9464
Length of passengers: 73327
Length of reservations: 38358

1. Menu Queries
2. Sair

```

Figura 3: *Output* do *parsing* dos ficheiro csv.

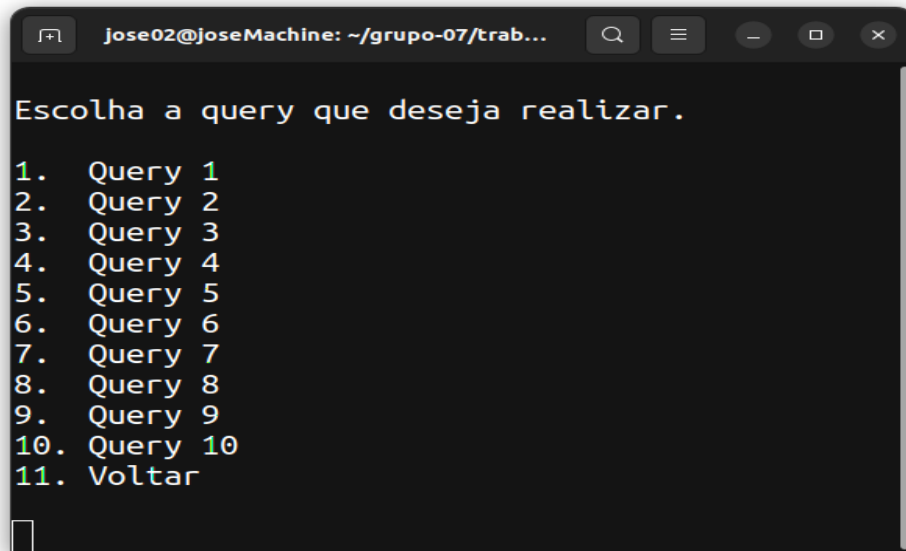


Figura 4: Menu *Queries*.



Figura 5: Exemplo de execução de uma *query*.

5 Testes e desempenho do programa

	Máquina 1	Máquina 2	Máquina 3
CPU	Intel Celeron N4000	Intel I7-8750H	———
Cores/Threads	2/2	6/12	———
RAM	4 GB DDR4 2400	24 GB DDR4	———
Disco	64 GB SSD M.2	1TB SSD M.2	———
OS	Ubuntu Linux	Manjaro Arch Linux	———
Compilador/Versão	gcc 11.4.0	gcc 11.4.0	———
Pico de memória	39MB	39MB	———
Tempo execução <i>queries</i>	2,14s	1.15s	———

6 Dificuldades sentidas

Apesar de sentirmos que concluímos com sucesso os requisitos de modularização e encapsulamento, e parcialmente o do modo interativo, achamos que deveríamos ter trabalhado de forma mais regular, já que não conseguimos concluir a implementação das *queries* 6 e 10 e a do *programa-testes*. Para além disto, sentimos dificuldades, especialmente quando tivemos que refactorar o nosso código para implementar o encapsulamento do nosso programa, sendo que, de cada vez que acedíamos diretamente a uma estrutura, tivemos que alterar por uma função *get*.

De cada vez que uma função *get* é chamada, é feita e devolvida uma cópia do campo da respetiva estrutura, através da função *strdup*, presente na biblioteca *string.h*. Isto fez com que o tempo de execução do nosso programa aumentasse bastante, sendo que, com mais um pouco de tempo e experiência, poderíamos ter resolvido melhor esta questão. Ainda em relação ao tempo de execução do programa, se a informação de cada *query* fosse reutilizada de uma melhor forma, acreditamos que ajudaria a otimizar, não só o tempo , mas a memória utilizada pelo programa.

7 Conclusão

Apesar de sentirmos que podíamos e devíamos ter feito mais e melhor nesta segunda fase do trabalho, este projeto proporcionou-nos uma valiosa experiência na manipulação eficiente de dados estruturados. A implementação bem-sucedida de algoritmos de *parsing* e processamento, juntamente com a atenção à eficiência do código e boas práticas de documentação, modularização e encapsulamento, resultou numa solução robusta e funcional. A experiência contribuiu para o aprimoramento das habilidades práticas em C, preparando o nosso grupo para desafios futuros no desenvolvimento de *software* e destacando a importância da precisão, eficiência, organização na criação de soluções escaláveis e gestão de memória.