



**Universidade do Minho**  
Escola de Engenharia

# LABORATÓRIOS DE INFORMÁTICA III

## Relatório da 1ª Fase

Grupo 09

a104184, Duarte Ribeiro Barbosa

a103993, Júlia Bughi Corrêa da Costa

a104613, Luís Pinto da Cunha

# Índice


- Objetivos, Introdução e Metodologia	2
- Análise do Estado Atual	3
- Parsing e Organização de dados	4
- Validação	6
- Resolução das Queries	7

# Análise do Estado Atual

Nesta primeira etapa de avaliação, o nosso grupo realizou por completo toda a etapa de validação do dataset e parsing do ficheiro de inputs de queries. Fomos capazes de realizar 9 queries em modo batch, com exceção à query 10, que ainda se encontra em construção.

O nosso trabalho está estruturado e dividido em módulos com funções adaptadas aos objetivos do programa. Projetamos módulos destinados ao tratamento dos tipos de dados, à leitura dos ficheiros de input e da execução das queries.

Acreditamos que os principais aspetos a melhorar são a otimização de recursos (tanto funções quanto gasto de memória) e a otimização do tempo de execução.

Repo	Log	PDF	Test	Make	Exec	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Total	Val.	Leaks MB*	Time S	Mem MB
09		0/2	×	✓	✓	100%	100%	100%	100%	100%	100%	100%	100%	100%	0%	94%	100%	0.019	0.9	28

# Objetivos, Introdução e Metodologia

O objetivo do projeto proposto é a consolidação dos conhecimentos da linguagem C, mas também da aprendizagem e da utilização de ferramentas essenciais a todo programador, como, por exemplo, o encapsulamento e a modularidade, o uso de estruturas dinâmicas de dados e medição de desempenho.

A primeira fase do projeto consistiu na elaboração de um programa em modo batch que resolve uma série de comandos, relacionados ao dataset fornecido.

O nosso grupo concluiu que a melhor forma de abordar a proposta de trabalho seria dividir as tarefas em três grandes grupos: parsing dos ficheiros, validação dos dados e realização das queries.



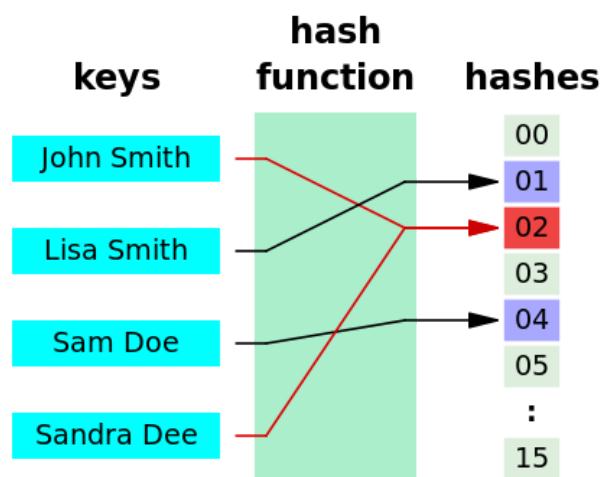
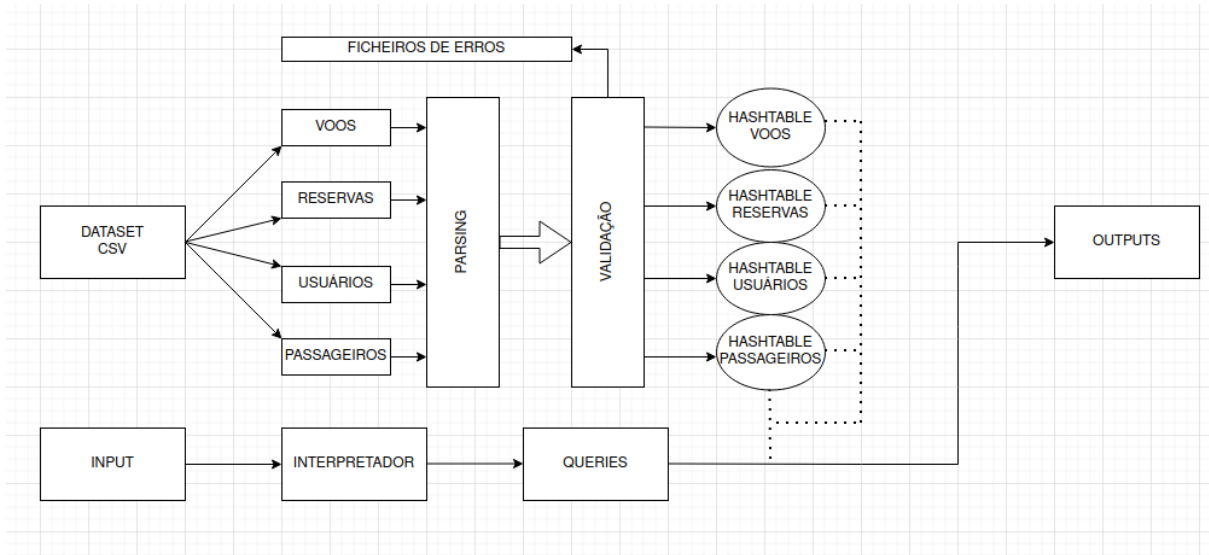
# Parsing e Organização de dados

Em primeiro lugar, decidimos que, por enquanto, iremos distribuir os dados por hashtables. Futuramente, se necessário, podemos incluir outras formas de organização de dados no nosso trabalho.

O parsing do dataset consiste na criação de 4 catálogos, um para os usuários, um para os voos, um para as reservas e um para os passageiros. Depois de se aceder aos ficheiros de dados, através do caminho fornecido pelo utilizador, criamos uma função específica para cada tipo de dados. Cada uma dessas funções ramifica-se em muitas outras que têm o objetivo de ler o ficheiro de dados, validar se os dados são corretos e guardá-los na hashtable para que possam depois ser usados na realização das queries.

Nas hashtables dos users, reservas e voos são guardadas structs com a informação específica de um user, reserva, voo respetivamente, sendo também adicionadas outros dados que possam ser úteis e/ou necessárias na resolução das queries. A hashtable dos passageiros é um pouco diferente dado que são guardadas structs com o id de um voo, uma lista ligada dos passageiros desse voo e o número de passageiros.

Optamos por usar os id de forma a gerar as keys das hashtables, uma vez que tornaria mais fácil a procura de um user/voo/reserva específico através da função predefinida `g_hash_table_lookup`.



# Validação

O método de validação varia com o tipo de dados. Porém, algumas funções auxiliares podem ser utilizadas em diversos contextos, como é o caso da “verify\_Date”, por exemplo.

A estrutura, no entanto, é sempre a mesma. As funções recebem uma linha do ficheiro de dados e guardam apenas as informações necessárias (de acordo com a demanda das queries), campo a campo, evitando um uso excessivo de memória. A validação é feita durante essa leitura e caso algo seja errado, é acionado um campo que, no final do loop, indica que a linha contém informação inválida.

No caso de ser inválido, copiamos a linha para o ficheiro de erros e damos free à memória. Caso contrário, adicionamos a struct à hashtable, como já foi explicado.

# Resolução das Queries

## Query 1

Para resolver esta query, primeiro determinamos se o id recebido era de user, reserva ou voo. De seguida fizemos lookup do id na respetiva hashtable que permitiu aceder às suas informações guardadas na struct. Bastou depois imprimir os valores pedidos usando as funções de get para os aceder.

## Query 2

Nesta query determinamos logo no parsing se era para imprimir voos, reservas ou ambos. Já na função da query fazemos o lookup do user e percorremos as listas de voos e de reservas do user que estão na struct, e usando a lookup para as procurar damos append deles numa GList. No final ordenamos os elementos da GList por data e imprimimos os valores necessários de cada um.

## Query 3

Para realizar esta query, percorremos a hashtable das reservas através do recurso “g\_hash\_table\_iter” e ao encontrar um id de hotel compatível com aquele recebido pelo input, realiza uma acumulação dos valores dos ratings e da quantidade dos mesmos. No final da passagem pela hashtable, é realizada uma divisão destes valores, obtendo assim a média.

## Query 4

Nesta query, como não temos forma de identificar diretamente todas as reservas de um certo hotel, percorremos a hashtable das reservas usando uma “g\_hash\_table\_iter” e todas as que são do hotel especificado são colocadas numa GList que é posteriormente ordenada por data. No final só é preciso percorrer a GList e imprimir os valores necessários.



## Query 5

Na realização desta query também percorremos a hashtable dos flights usando uma “g\_hash\_table\_iter” e verificamos que voos estão entre a begin\_date e a end\_date usando a função auxiliar difference\_dates2. Caso cumpra os requisitos é adicionada a uma GList que é depois ordenada por data e percorrida para se imprimir os voos as respectivas informações

## Query 6

Nesta query, usamos uma GList de structs auxiliares que contêm o aeroporto e o número de passageiros que lá passaram durante o ano dado. Percorremos a hashtable usando uma “g\_hash\_table\_iter” e adicionamos o número de passageiros desse voo às structs dos aeroportos de partida e chegada do voo, caso a data estimada de partida do voo seja no ano passado à função. Por fim, ordenamos a GList por número de passageiros e imprimimos os N aeroportos e os respectivos passageiros.

## Query 7

Para fazer esta query, usamos uma GList de structs auxiliares que têm o aeroporto, uma lista dos atrasados dos voos desse aeroporto e o número de voos atrasados. Percorremos a hashtable usando uma “g\_hash\_table\_iter” e para cada voo adicionamos o seu atraso(caso haja) à struct do aeroporto de partida. A GList é depois ordenada por mediana de atrasados que é calculada com uma função auxiliar, e são depois impressos os N aeroportos com maior mediana de atrasados com a respetiva mediana.

## Query 8

Para realizar a query 8, estabelecemos uma função que percorre a hashtable das reservas e realiza uma determinada função caso o id fornecido pelo input for igual à chave da função “g\_hash\_table\_iter”. É realizada uma verificação quanto às datas de

início e fim das reservas e então é executado um cálculo adequado à resolução do problema.

### **Query 9**

Nesta query, percorremos a hashtable dos users usando um “g\_hash\_table\_iter” e usamos uma função auxiliar para determinar se o user\_id começa com o prefixo passado como argumento e se começar é adicionado a uma GList. A GList é posteriormente ordenada alfabeticamente e impressa.