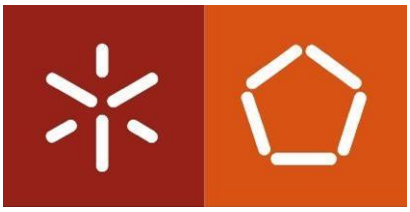

UNIVERSIDADE DO MINHO

Licenciatura em Engenharia Informática

LI3 - Sistema de organização e escrita de dados

Grupo 21

Gonçalo Cruz (A104346) Gonçalo Cunha (A104003)
Luís Freitas (A104000)

Ano Letivo 2023/2024

Índice

Capítulo 1

Introdução e complicações iniciais

Este projeto consiste numa organização de dados inicialmente fornecidos em 4 ficheiros diferentes (voos, utilizadores, reservas e passageiros). Posteriormente são expostas *queries*, nas quais é necessário apresentar os dados que nos são pedidos e que foram previamente guardados.

Foi um trabalho feito com a linguagem de programação C, no qual tivemos de recorrer à utilização de estruturas de dados que fossem o mais eficientes possível, visto que foi necessário armazenar uma grande quantidade de informação, que teve de ser manuseada mais à frente, com as *queries*.

Decidimos utilizar *arrays* dinâmicos, todos eles criados por nós, pois achámos que seria uma maneira rápida e eficiente para quando fosse preciso aceder às informações guardadas inicialmente.

Uma das complicações que nos surgiu no começo do projeto foi tentar arranjar a melhor forma de armazenar uma quantidade de informação tão elevada. Até começámos por utilizar listas ligadas, mas chegámos à conclusão de que, como seria necessária a utilização de apontadores auxiliares para percorrer as listas até chegarmos aos dados que queremos, essa abordagem não seria tão eficiente como é o manuseamento dos *arrays* dinâmicos, já para não falar de que seria escusadamente mais complicado ir por esse método.

Mais à frente surgiram novos obstáculos de como poderíamos tornar o programa sempre cada vez mais rápido. Apesar disso ser um aspeto a ter mais em conta na segunda fase do projeto, decidimos começar a implementar maneiras de diminuir o tempo de execução desde cedo, o que acabou por resultar numa redução do mesmo de cerca de 5 segundos nesta primeira fase.

Capítulo 2

Ficheiros fornecidos

2.1 - Voos

Para armazenar todos os parâmetros de voos usamos *arrays* dinâmicos. Não é nada mais do que um *array* de *structs* (estruturas), no qual cada elemento é composto por todas as informações de um voo. À medida que uma linha do ficheiro de voos é lida, todos os seus parâmetros são guardados no respetivo campo, e no final é feita a validação dessa linha consoante o exigido no guião do projeto. Dependendo do resultado da validação, as informações dessa linha lida serão guardadas num elemento do *array* de voos válidos ou então no de inválidos. O armazenamento de informação deste modo permite realizar eficientemente operações pedidas pelas *queries*.

2.2 - Utilizadores

Para armazenar todos os parâmetros de utilizadores usamos também *arrays* dinâmicos. Funciona de forma muito semelhante aos voos, na medida em que cada elemento é composto por todas as informações de um utilizador. À medida que uma linha do ficheiro de utilizadores é lida, todos os seus parâmetros são guardados no respetivo campo e, no final, é feita a validação dessa linha. Dependendo do resultado da validação, as informações dessa linha lida serão então guardadas num elemento do *array* de utilizadores válidos ou inválidos.

2.3 - Passageiros

Do mesmo modo que nos anteriores, para armazenar todos os parâmetros de passageiros usamos *arrays* dinâmicos. Cada elemento é composto por todas as informações de um passageiro. À medida que uma linha do ficheiro de passageiros é lida, todos os seus parâmetros são guardados no respetivo campo e, no final, é feita a validação dessa linha consoante o exigido no guião do projeto. Dependendo do resultado da validação, as informações dessa linha lida serão guardadas num elemento do *array* de passageiros válidos ou inválidos. O armazenamento de informação deste modo permite realizar as operações pedidas pelas *queries*.

2.4 - Reservas

Finalmente, para armazenar todos os parâmetros de reservas, também usamos *arrays* dinâmicos. Cada elemento é composto por todas as informações de uma reserva. À medida que uma linha do ficheiro de utilizadores é lida, todos os seus parâmetros são guardados no respetivo campo, e, no final, é feita a validação dessa linha consoante o exigido no guião do projeto. Dependendo do resultado dessa validação, as informações da linha lida serão guardadas num elemento do *array* de reservas válidas ou então no de inválidas. O armazenamento de informação assim permite realizar operações pedidas pelas *queries*.

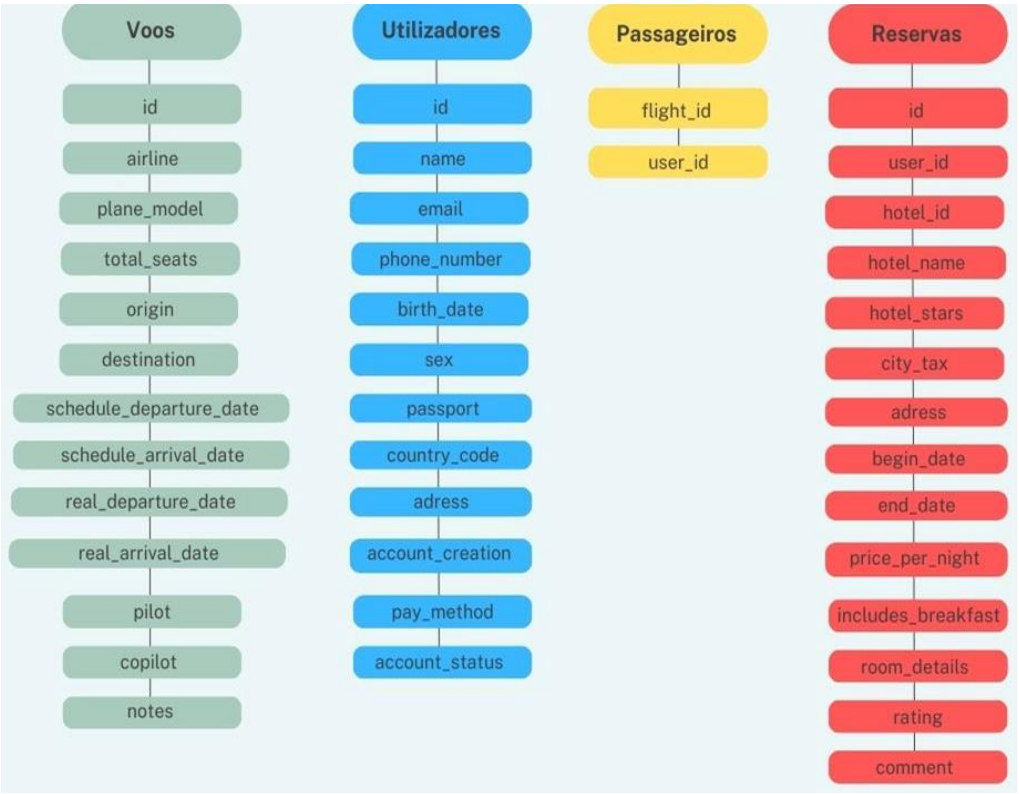

Figura 1: Campos de informações dos quatro ficheiros fornecidos

Capítulo 3

Queries

3.1 – Funcionamento geral das *queries*

O programa lê a linha do ficheiro do *input* (com as *queries*) e identifica qual é o número da *query* pretendida. Conforme o número, chama a função *query* correspondente (*ex.* query1), com os seguintes argumentos: a linha inteira, lida do ficheiro fornecido; um inteiro i, que é o índice onde acaba o identificador da *query* (para a função saber se tem a *flag* "F" ou não); e um inteiro n, que é o número da linha lida do ficheiro.

3.2 – *Query* 1

Para a *query* 1, que "lista o resumo de um utilizador, voo ou reserva, consoante o identificador recebido por argumento", o nosso programa guarda o argumento (*id*) e cria o ficheiro de *output*. De seguida, identifica a que tipo de dados se refere o *id* (se a um voo, a uma reserva ou a um utilizador) e, para cada um dos casos, procura-o no *array* dinâmico correspondente. Se encontrar e se o utilizador correspondente for válido (se o seu *account_status* for *active*), vai armazenar as informações necessárias, que neste caso são um resumo dos dados do tipo correspondente (utilizador, voo ou reserva), e escrevê-las no ficheiro de *output*.

- Utilizador
  nome;sexo;idade;código_do_país;passaporte;número_voos;número_reservas;total_gasto
  (name;sex;age;country_code;number_of_flights;number_of_reservations;total_spent)
- Voo
  companhia;avião;origem;destino;partida_est;chegada_est;número_passageiros;tempo_atraso
  (airline;plane_model;origin;destination;schedule_departure_date;schedule_arrival_date;passengers;delay)
- Reserva
  id_hotel;nome_hotel;estrelas_hotel;data_início;data_fim;pequeno_almoço;número_de_noites;preço_total
  (hotel_id;hotel_name;hotel_stars;begin_date;end_date;includes_breakfast;nights;total_price)

Figura 2: Funcionamento da *query* 1 (*output*)

Comando
1 <ID>
Output
(ver acima)

Figura 3: Funcionamento da *query* 1 (*input*)

3.3 – *Query* 2

Para a *query* 2, que "lista os voos ou reservas de um utilizador", o nosso programa guarda o argumento (*id* do utilizador) e cria o ficheiro de *output*. Depois, procura esse *id* no *array* dinâmico dos utilizadores. Se não encontrar, significa que o *input* da *query* era inválido. Caso contrário, vai guardar os dados importantes, que neste caso são os voos e reservas do utilizador ou somente um deles, consoante o que tenha sido especificado no *input*. De seguida, ordena estes dados por data (da mais recente para a mais antiga) para que, por fim, escreva esta informação no ficheiro de *output*.

Comando
2 <ID> [flights|reservations]
Output
id;date[;type]
id;date[;type]
...

Figura 4: Funcionamento da *query* 2 (*input* e *output*)

3.4 – *Query* 3

Para a *query* 3, que "apresenta a classificação média de um hotel, a partir do seu identificador", o nosso programa vai percorrer o *array* dinâmico de reservas, somando (para uma variável) a respetiva avaliação atribuída por cada utilizador (apenas quando o identificador dele for igual ao que foi dado no *input*) para, por fim, fazer a divisão e ter a respetiva média de classificações do hotel.

Comando
3 <ID>
Output
rating

Figura 5: Funcionamento da *query* 3 (*input* e *output*)

3.5 – *Query* 4

Para a *query* 4, que "lista as reservas de um hotel, ordenadas por data de início (da mais recente para a mais antiga)", o nosso programa guarda o argumento (*id* do hotel) e cria o ficheiro de *output*. Depois, procura esse *id* no *array* dinâmico das reservas e, se encontrar, vai guardar os dados necessários, que neste caso são o *id* da reserva, a data de início, a data de fim, o *id* do utilizador, o *rating* (classificação) e o preço total. De seguida, ordena estes dados por data de início e, por fim, escreve esta informação no ficheiro de *output*.

Comando
4 <ID>
Output
id;begin_date;end_date;user_id;rating;total_price
id;begin_date;end_date;user_id;rating;total_price
...

Figura 6: Funcionamento da *query* 4 (*input* e *output*)

3.6 – *Query* 5

Para a *query* 5, que "lista os voos com origem num dado aeroporto, entre duas datas, ordenados por data de partida estimada (da mais antiga para a mais recente)", o nosso programa guarda os argumentos (aeroporto de origem do voo, data de início e data de fim) e cria o ficheiro de *output*. Depois, procura a *origin* (origem) no *array* dinâmico dos voos e, se encontrar, vai guardar os dados necessários dos voos (apenas daqueles que têm *schedule_departure_date* entre as datas fornecidas no *input*), que neste caso são o *id* do voo, a data estimada de partida, o aeroporto de destino, a companhia aérea e o modelo do avião. De seguida, ordena estes dados por data de partida estimada e, por fim, escreve esta informação no ficheiro de *output*.

Comando
5 <Name> <Begin_date> <End_date>
Output
id;schedule_departure_date;destination;airline;plane_model
id;schedule_departure_date;destination;airline;plane_model
...

Figura 7: Funcionamento da *query* 5 (*input* e *output*)

3.7 – *Query* 9

Para a *query* 9, que "lista todos os utilizadores cujo nome começa com o prefixo passado por argumento, ordenados por nome (de forma crescente)", o nosso programa vai percorrer o *array* dinâmico dos utilizadores e, para cada um deles, verificar se o seu nome começa com o prefixo dado no *input*. Caso comece, vai guardar o respetivo *id* e nome num *array* dinâmico, para, no final, ordenar e escrever estes dados no ficheiro de *output*.

Comando
9 <Prefix>
Output
id;name
id;name
...

Figura 8: Funcionamento da *query* 9 (*input* e *output*)

Capítulo 4

Melhorias

4.1 – Melhorias já implementadas

Tal como foi referido na introdução deste relatório, através de ideias que foram surgindo ao longo da execução e análise do projeto/guião, foi conseguida uma redução do tempo de execução do programa em cerca de 5 segundos.

Reparamos que no ficheiro dos passageiros, as linhas seguem um padrão constante. Primeiro aparecem todos os passageiros do primeiro voo, depois todos os do segundo, e assim por diante (tal como se pode verificar na figura 9 abaixo). Sendo assim, e visto que cada linha do ficheiro dos passageiros é composta pelo *id* de um voo e pelo *id* de um utilizador, a partir do momento em que se verifica que o voo com um *id* é inválido, essa mesma linha do ficheiro será inválida, não havendo necessidade de verificar a validação do utilizador. Para além disso, se a linha seguinte desse ficheiro tiver o mesmo *id* de voo que a anterior e se tivesse verificado que esse *id* era inválido, então automaticamente essa nova linha lida também será inválida.

```
0000000001;SanLeite
0000000001;FNunes
0000000001;Carminho-FlReis153
0000000001;KyaNascimento-Domingues470
0000000001;JulianTorres
0000000001;GonçalMagalhães-Guerreiro166
0000000001;BárbaAmorim1683
0000000001;CrisBrito1393
0000000001;IriMartins1718
0000000001;CésaMiranda
0000000001;CristMarques721
0000000001;JMarques336
0000000001;LeRibeiro
0000000001;LetíPinho
0000000001;GasSoares
0000000001;CarlCastro
0000000001;Rafaela-Pinto
0000000001;DiogMorais1934
0000000001;LAmaral286
0000000001;IvVieira
0000000001;CristianVicente61
```

Figura 9: Exemplo de linhas no ficheiro dos passageiros com o mesmo *id* de voo

Outra melhoria que decidimos implementar foi optarmos por não criar um *array* dinâmico para cada ficheiro (ou seja um para voos, um para reservas, um para utilizadores e outro para passageiros). Em vez disso, chegamos à conclusão que seria muito mais rápido e eficiente criarmos dois *arrays* dinâmicos para cada arquivo: um para armazenar linhas válidas e outro para linhas inválidas. Isto é feito simultaneamente à validação de cada linha: se uma linha do ficheiro lido for válida, é inserida no respetivo *array* válido; caso contrário, é inserida no *array* inválido.

Esta última melhoria ajuda imenso na diminuição do tempo de execução do programa, já que as *queries* apenas trabalham com argumentos válidos. Por isso, para verificar se um parâmetro de uma *query* é válido, basta percorrer o *array* de linhas inválidas apropriado (que é significativamente menor que o *array* de dados válidos) e caso se verifique que não se encontra neste, sabe-se que é válido.

4.2 – Melhorias futuras

Algo que temos em mente para aperfeiçoar já após a entrega da primeira fase do projeto é a subsequente organização do código nos seus respetivos módulos, demonstrando, assim, conhecimentos a respeito de modularidade e encapsulamento de código.

Inicialmente temos apenas 3 módulos (ficheiros .c): o *parser.c*, o *validation.c* e o *queries.c*. Trataremos depois de melhorar a organização do programa, para termos módulos com menos linhas e, assim, tornar o código mais organizado, percetível e legível, contribuindo também para uma maior agilidade na procura de futuros erros.

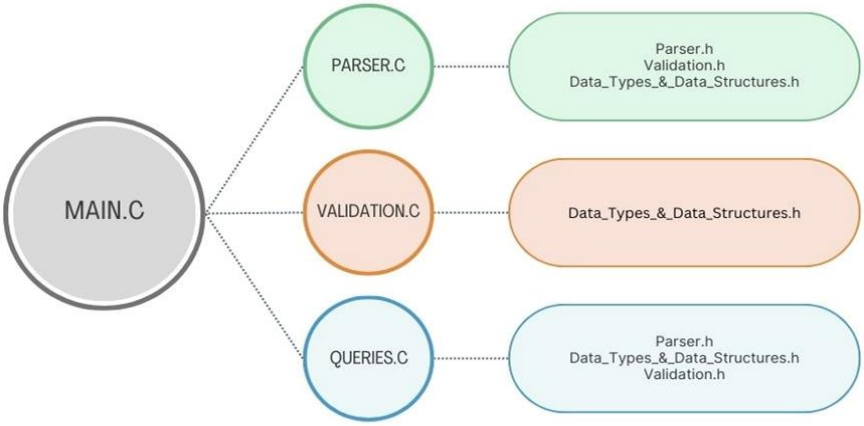

Figura 10: Arquitetura da aplicação

Capítulo 5

Conclusão

No geral, e como já foi abordado anteriormente neste relatório, consideramos que temos um projeto bem conseguido. A forma como decidimos organizar a quantidade de dados dos ficheiros é definitivamente um método bastante eficiente e, juntamente com todas as melhorias de procura nos *arrays* implementadas, conseguimos obter um tempo de execução baixo e um código funcional.

Achamos também que o modo como organizámos as informações e, posteriormente, as utilizámos para dar resposta às *queries* é relativamente simples de entender, devido à simplicidade inerente à procura de dados num *array*.

Algo que foi necessário ter sempre em conta e que nos foi gerando alguns impasses ao longo do projeto foi a correta libertação de memória, visto que os *arrays* são dinâmicos, e não estáticos. Felizmente, também conseguimos dar resposta a este problema, executando, assim, o código sem existirem *memory leaks*.