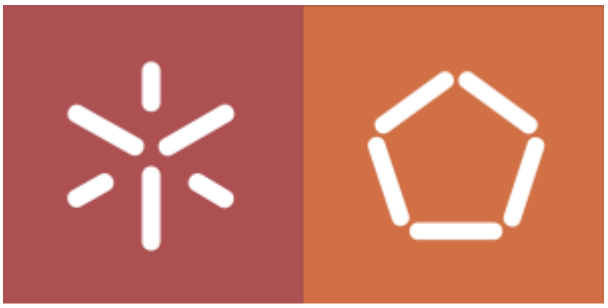

Universidade do Minho
Escola de Engenharia

Laboratório de Informática III

Fase 1

A84675, António Luís Braga Mendes
Lu1sm3ndes

A104611, João Manuel Machado da Cunha
JoaoCunha50

A104612, João Pedro Ribeiro de Sá
joaosa24

Índice

1. Introdução

No âmbito da disciplina de LI3, desenvolvemos o projeto que nos foi apresentado, onde tivemos em conta os conceitos de encapsulamento e modularidade. Usamos diferentes estruturas de dados e estratégias de encapsulamento e modularidade.

2. Estruturas de Dados

Para alocar toda a informação relativa aos usuários, voos, passageiros e reservas decidimos usar structs.

2.1 Users

Os users são identificados pelo seu id, nome, email, número de telemóvel, género, passaporte, código postal, morada, método de pagamento e status da conta usamos diferentes strings. Para a data de nascimento usamos uma struct Data que guarda o dia, mês e ano de nascimento em diferentes int e para o momento de criação de conta usamos uma struct Schedule que contém a mesma struct Data mas para o dia, mês e ano de criação de conta e uma outra Time que guarda as horas, minutos e segundos em diferentes int. Ao nível da verificação implementamos 3 funções importantes. Para verificar se um user é valido usamos *int isUserValid*, para verificar se um email de um user é válido usamos a função *int isEmailValid*, e se uma conta está ativa, *int isAccountStatusValid.* Os dados são destruídos com *void freeUser.*

2.2 Flights

Para os voos, usamos a struct Flight para guardar a informação. Usamos um id e o número total de lugares em formatos int, para a companhia aérea, modelo de avião, origem, destino, piloto, copiloto e observações sobre o voo usamos uma string para cada um deles. Para a data e hora estimada de partida, estimada de chegada, real de partida e real de chegada usamos uma struct Schedule. Já ao nível da validação usamos a função *int isFlightValid* para verificar se um voo é válido ou não. Com a função *void free_flights* os dados são destruídos.

2.3 Passengers

Já no caso dos passageiros, usamos uma struct Passengers para guardar o correspondente id de voo e de usuário em strings. Para verificar se um passageiro é válido usamos a função *int isPvalid.* Se quisermos saber o número de passageiros de voo usamos a função *int CountPassengers.* Com a função *int free_passageiros* os dados são destruídos.

2.4 Reservations

Por fim, para as reservas, usamos duas structs, a struct Data já referida anteriormente e a struct Reservation. Para as reservas guardamos o id da reserva, o id do user, o id do hotel, o nome do hotel, a morada, pequeno almoço incluído ou não, detalhes sobre o quarto e comentários em diferentes strings. Para o número de estrelas do hotel, a taxa da cidade, o preço por noite e a avaliação usamos um int para cada um deles. Para a data de check in e data de check out reaproveitamos a struct Data. Para verificar se uma reserva é válida usamos a função *int isReservationValid.* Com a função *void freeReservation* os dados relativos às reservas são destruídos.

3. Parsing

3.1 Parsing dos dados

Começamos por carregar os ficheiros “users.csv”, “flights.csv”, “passengers.csv” e “reservations.csv”, respetivamente nos módulos “users.h”, “flights.h”, “passengers.h” e “reservations.h”. Usamos vários setters nos ficheiros “users.c”, “flights.c”, “passengers.c” e “reservations.c” para colocar cada dado dos ficheiros .csv no campo correto para posteriormente validação. Usamos também a função *strsep* para separar a string em tokens, o que torna o acesso às informações de cada uma das structs de dados mais direto.

3.1.1 Parsing dos Users

Começamos por carregar o ficheiro “users.csv” e alocar memória para os users. De seguida verificamos se os users são válidos através da função *isUserValid*, libertamos a memória temporária e fechamos o ficheiro “users.csv”. Por fim, retornamos a struct de dados dos users.

3.1.2 Parsing dos Flights

Começamos por carregar o ficheiro “flights.csv” e alocar memória para os voos. De seguida verificamos se os voos são válidos através da função *isFlightValid*, libertamos a memória temporária e fechamos o ficheiro “flights.csv”. Por fim, retornamos a struct de dados dos voos.

3.1.3 Parsing dos Passengers

Começamos por carregar o ficheiro “passengers.csv” e alocar memória para os passageiros. De seguida verificamos se os passageiros são válidos através da função *isPvalid,* libertamos a memória temporária e fechamos o ficheiro “passengers.csv”. Por fim, retornamos a struct de dados dos passageiros.

3.1.4 Parsing das Reservations

Começamos por carregar o ficheiro “reservations.csv” e alocar memória para as reservas. De seguida verificamos se as reservas são válidas através da função *isReservationValid*, libertamos a memória temporária e fechamos o ficheiro “reservations.csv”. Por fim, retornamos a struct de dados das reservas.

4. Queries

Antes de irmos à implementação das querys, decidimos usar um módulo responsável por ler o ficheiro de comandos e executar a respectiva query com os devidos argumentos. O ficheiro “interpreter.c” verifica a query que está a ser pedida e retorna-nos o conteúdo resultante da aplicação de uma query.

4.1 Query 1

Esta query requer informações sobre os users, voos ou reservas, consoante o identificador que é passado. Criamos a função *void query1* para implementar o que nos é pedido. Usamos os getters implementados na parte da estrutura dos dados para implementar o que nos é pedido. Todos os getters estão nos ficheiros “users.c”, “flights.c”, “passengers.c” e “reservations.c”.

4.2 Query 3

Esta query requer informações sobre a classificação média do rating de um hotel, com base no seu identificador. Para obter essas informações, precisamos de aceder às informaçẽs das structs Data e Reservation. Criamos a função *void query3* para implementar o que nos era pedido. Usamos também a função *float media* inserida no ficheiro statistics.c para obter a média das classificações.

4.3 Query 4

Esta query requer informações sobre as reservas de um hotel, ordenadas por data de início, da mais recente para a mais antiga. Implementamos a função *void query4* que retorna a lista de reservas de um hotel, que em caso de empate usa o identificador do próprio hotel como desempate.

4.4 Query 8

Esta query apresenta a receita total de um hotel entre duas datas, a partir do seu identificador. Implementamos a função *int query8* para nos devolver a receita total. Mais uma vez necessitamos da informação contida nas structs Data e Reservation.

4.5 Query 9

Esta query lista todos os utilizadores cujo nome começa com o prefixo passado por argumento, ordenados por nome, de forma crescente. Implementamos a função void query9 que no caso de empate, usa o identificador do nome como critério de desempate, mais uma vez de forma crescente

5. Modularidade e Encapsulamento

Apesar das componentes relacionadas com a modularidade e encapsulamento não serem contabilizadas para a nota desta primeira fase, preferimos dividir o projeto em vários módulos. No ficheiro “files.c” os ficheiros de input são abertos com recurso à função *FILE **open_cmdfiles(int argc, char **argv)*. Já no “parser.c” testam- se erros de abertura e no ficheiro “interpreter.c” os ficheiros do comando são lidos e interpretados, para executar a respectiva query que nos é pedida de momento. Nos ficheiros “users.c”, “flights.c”, ”passengers.c” e “reservations.c, são transferidos os dados dos ficheiros de input “users.csv”, “flights.csv”, ”passengers.csv” e “reservations.csv”, respetivamente, para as estruturas de dados acima referidas.
No ficheiro “parser.c” também existem funções usadas por esses módulos no parsing dos ficheiros de input.

Já para consultar os dados sobre os ficheiros, nas várias queries, são referidas todas as structs necessárias para aceder a essas mesmas informações.

6. A implementar na fase2

Com o objetivo de atacar a abordagem planeada para o projeto, apresentamos algumas considerações adicionais sobre a estrutura e desempenho desejados para a mesma. Pretendemos desenvolver um módulo dedicado à recuperação de novas estruturas de dados produzidos nas queries. Assim, o tempo de resposta para essas queries será otimizado, o que será essencial no momento da implementação do menu de modo interativo. Além disso, tentamos elevar a eficiência do sistema, com o objetivo de garantir uma linha mais dinâmica e eficaz.