Universidade do Minho
Escola de Engenharia

Laboratórios de Informática III

Carlos Diogo Fernandes Pina A95349
Lara Beatriz Pinto Ferreira A95454
João Machado Gonçalves A97321

A95349

A95454

A97321

18 de Novembro de 2023

Índice

1 Introdução

No âmbito da Unidade Curricular de Laboratórios de Informática III da licenciatura em Engenharia Informática, nesta primeira fase de avaliação, foi nos requisitado a implementação de um *parser* em C que validasse as linhas dos quatro ficheiros *.csv* (*users*, *flights*, *passengers* e *reservations*) que nos foram disponibilizados. Foi-nos também solicitado a implementação de pelo menos seis *queries*, que no caso resolvermos implementar as *queries 1,3,4,5,8,9*.

2 Estratégia utilizada

Após a leitura do enunciado optamos por criar catálogos para as *structs users*, *flights*, *reservations* e *passengers*, e percorrê-los de forma iterativa.

2.1 Estruturas de dados utilizadas para representar utilizadores, voos, reservas e passageiros

As estruturas de dados referidas foram criadas de forma a ser possível realizar, o mais eficientemente possível, tanto a nível de performance como de memória, as *queries* que implementamos.

2.1.1 Utilizadores

Utilizamos depois um catálogo *users* construido com base na *struct users* com o objetivo de conseguirmos encontrar e manipular, de modo eficiente, informações sobre um certo *user*, através do seu *ID*, que é a *key* da *hashtable*. Deste modo, decidimos armazenar a informação dos *users* através do uso de *hashtables*, pois achamos que é a maneira mais eficiente de implementar o objetivo descrito.

```
struct users {
    char *id;
    char *name;
    char *email;
    char *phone_number;
    Date birth_date;
    char *sex;
    char *passport;
    char *country_code;
    char *adress;
    Datetime account_creation;
    enum pay_method pay_method;
    enum account_status account_status;

    int flights_total;
    int reservations_total;
    double spent_total;
};
```

Figura 1: Struct Users

- Id : O identificador do *user* é representado como uma *string*
- Name : O Nome do *user* é representado como uma *string*
- Email : O email do *user* é representado como uma *string*
- Phone_number : O número de telemóvel do *user* é representado como uma *string*
- Birth_date : A data de nascimento do *user* é representada como um *datetime*
- Sex : O género do *user* é representado como uma *string*
- Passaport : O númerop do passaporte do *user* é presentado como uma *string*
- Country_code : O código do país de residência
- Address : A morada do *user* é representado como uma *string*
- Account_creation : A data de criação da conta é representada com um *datetime*
- Pay_method : O método de pagamento é representado como um *enum*
- Account_status : O estado da conta é representado como um *enum*
- Flights_total : O número de voos realizados pelo *user* é representado como um inteiro
- Reservations_total : O número de reservas realizadas pelo *user* é representado como um inteiro
- Spent_total : O total de dinheiro gasto pelo *user* é representado como um *double*

2.1.2 Voos

Utilizamos depois um catálogo *flights* construido com base na *struct flights* com o objetivo de conseguirmos encontrar e manipular, de modo eficiente, informações sobre um certo *flight*, através do seu *ID*, que é a *key* da *hashtable*. Deste modo, decidimos armazenar a informação dos *flights* através do uso de *hashtables*, pois achamos que é a maneira mais eficiente de implementar o objetivo descrito.

```
struct flights {
    char *id_flights;
    char *airline;
    char *plane_model;
    int total_seats;
    char *origin;
    char *destination;
    Datetime schedule_departure_date;
    Datetime schedule_arrival_date;
    Datetime real_departure_date;
    Datetime real_arrival_date;
    char *pilot;
    char *copilot;
    char *notes;

    int num_passengers;
    int delay;
};
```

Figura 2: Struct Flights

- Id_flights : O identificador do *Flight* é representado como uma *string*
- Airline : A companhia aérea que realizou o *Flight* é representada com uma *string*
- Plane_model : O modelo do avião usado na *Flight* é representada com uma *string*
- Total_seats: O número de lugares total do avião usado na *Flight* é representada como um inteiro
- Origin : O aeroporto de origem da *Flight* é representada como uma *string*
- Destination : O aeroporto de destino da *Flight* é representada como uma *string*
- Schedule_departure_date : A data e hora planeada para a partida da *Flight* é representado por um *datetime*
- Schedule_arrival_date : A data e hora planeada para a chegada da *Flight* é representado por um *datetime*

- Real_departure_date : A data e hora efetiva da partida da *Flight* é representado por um *datetime*
- Real_arrival_date : A data e hora efetiva da *Flight* é representado por um *datetime*
- Pilot : O nomedo piloto que realizou a *Flight* é representado por uma *string*
- Copilot : O nome do Co-piloto que realizou a *Flight* é representado por uma *string*
- Notes : As observações sobre a *Flight* são representadas como uma *string*
- num_passengers : O número de passageiros que realizou a *Flight* é representado como um inteiro
- Delay : O atraso da *flight*, em segundos, é representado como um inteiro

2.1.3 Reservas

Utilizamos depois um catálogo *reservations* construido com base na **struct reservations** com o objetivo de conseguirmos encontrar e manipular, de modo eficiente, informações sobre um certo *reservation*, através do seu *ID*, que é a *key* da *hashtable*. Deste modo, decidimos armazenar a informação dos flights através do uso de *hashtables*, pois achamos que é a maneira mais eficiente de implementar o objetivo descrito.

```
struct reservations {
    char *id_res;
    char *user_id;
    char *hotel_id;
    char *hotel_name;
    int hotel_stars;
    int city_tax;
    char *adress;
    Date begin_date;
    Date end_date;
    int price_per_night;
    char *includes_breakfast;
    char *room_details;
    char *rating;
    char *comments;

    int nights;
    double total_price;
};
```

Figura 3: Struct Reservations

- Id_res : O identificador da *Reservation* é representado como uma *string*
- User_id : O identificador do *User* que fez a *Reservation* é representado como uma *string*
- Hotel_id : O identificador do hotel é representado como uma *string*
- Hotel_name : O nome do hotel é representado como uma *string*
- Hotel_stars : O número de estrelas do hotel é representado como um inteiro
- City_tax : A percentagem do imposto da cidade é representado como um inteiro
- Begin_date : A data de início da *Reservation* é representado como uma *date*
- End_date : A data de fim da *Reservation* é representado como uma *date*
- Price_per_night : O preço por noite é representado por um inteiro
- Includes_breakfast : A inclusão de pequeno-almoço na *Reservation* é representa como uma *string*
- Room_details : Os detalhes sobre o quarto são representado como uma *string*
- Rating : A classificação atribuída pelo *user* é representada como uma *string*
- Comment : O comentário sobre a reserva
- Nights : O número de noites que abrangidas na *Reservation* é representada como um inteiro
- Total_price : O preço total da *Reservation* é representado como um *double*

2.1.4 Passageiros

Utilizamos depois um catálogo *passengers* construido com base na *struct passengers* com o objetivo de conseguirmos encontrar e manipular, de modo eficiente, informações sobre um certo *passenger*, através do *ID* do *Flight*, que é a *key* da *hashtable*. Deste modo, decidimos armazenar a informação dos *flights* através do uso de *hashtables*, pois achamos que é a maneira mais eficiente de implementar o objetivo descrito.

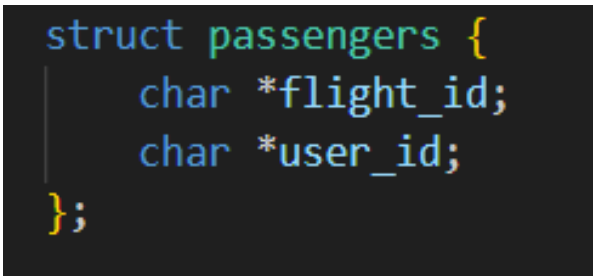

Figura 4: Struct Passengers

- Flight_Id : O identificador da *Flight* é representado como uma *string*
- User_Id : O identificador do *User* é representado como uma *string*

2.2 Estratégia de modularização e reutilização de código

A modularização e reutilização do código, apesar de não serem avaliados nesta fase, promovem uma maior organização, leitura e facilidade de manutenção do código, que por sua vez auxiliam no processo de desenvolvimento, desta forma decidimos implementar os seguintes módulos:

- Users : Representação de um *User* e catalogo dos mesmos
- Flights : Representação de um *Flight* e catalogo dos mesmos
- Reservations : Representação de um *Reservation* e catalogo dos mesmos
- Passengers : Representação de um *Passenger* e catalogo dos mesmos
- Catalog : Agregação dos diversos catalogos e estruturas
- Date : Representação de um *Date* e/ou *Datetime*
- Valid : Agregado de métodos de validação

2.3 Estratégia de encapsulamento e abstração

De forma a manter o código seguro e bem estruturado, todos os módulos escondem a sua implementação de maneira que toda a operação sobre os seus dados pode ser feita com as funções que estes disponibilizam.

3 Queries

Com os dados já validados e organizados, foi nos pedido o processamento dos mesmo usando diferentes queries, estas suportam também a flag 'F', isto é, uma query com esta flag apresenta o seu resultado no output no formato *field: value*.

3.1 Query 1

3.1.1 Descrição

Q1: Listar o resumo de um utilizador, voo, ou reserva, consoante o identificador recebido por argumento. É garantido que não existem identificadores repetidos entre as diferentes entidades. A query deverá retornar as seguintes informações:

- 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)

Comando
1 <ID>
Output
(ver acima)

Figura 5: Comando Query 1

3.1.2 Implementação

Se o identificador conter apenas dígitos, procuramos na hashtable das *flights*, caso contrário, se os primeiros 4 caracteres forem '*Book*', procuramos na hashtable das *reservations*, caso nenhuma das condições anteriores se verifique, pesquisamos na hashtable dos *users*. Em todos os casos é devolvida uma cópia dos valores encontrados ou nada se não existirem exceto no caso dos utilizadores com *account_status* = *"inactive"* que também não devolve nada.

3.2 Query 3

Q3: Apresentar a classificação média de um hotel, a partir do seu identificador.

Comando
3 <ID>
Output
rating

Figura 6: Comando Query 3

3.2.1 Implementação

Procuramos todas as *reservations*, na hashtable das mesmas, que foram efetuadas no hotel com o identificador especificado e obtemos todas classificações. No fim, calculamos a media dessas classificações.

3.3 Query 4

Q4: Listar as reservas de um hotel, ordenadas por data de início (da mais recente para a mais antiga). Caso duas reservas tenham a mesma data, deve ser usado o identificador da reserva como critério de desempate (de forma crescente)

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 7: Comando Query 4

3.3.1 Implementação

Procuramos todas as *reservations*, na hashtable das mesmas, as que foram efetuadas no hotel com o identificador especificado e usamos-las para criar uma lista, de seguida organizamos essa mesma lista consoante os requisitos em cima.

3.4 Query 5

Q5: Listar os voos com origem num dado aeroporto, entre duas datas, ordenados por data de partida estimada (da mais antiga para a mais recente). Um voo está entre ¡begin_date¿ e ¡end_date¿ caso a sua respetiva data estimada de partida esteja entre ¡begin_date¿ e ¡end_date¿ (ambos inclusivos). Caso dois voos tenham a mesma data, o identificador do voo deverá ser usado como critério de desempate (de forma crescente).

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 8: Comando Query 5

3.4.1 Implementação

Procuramos todas as *flights*, na *hashtable* das mesmas, que foram efetuadas numa origem e período especificados e usamos-las para criar uma lista e organizamos essa mesma lista consoante os requisitos em cima.

3.5 Query 8

Q8: Apresentar a receita total de um hotel entre duas datas (inclusive), a partir do seu identificador. As receitas de um hotel devem considerar apenas o preço por noite (price_per_night) de todas as reservas com noites entre as duas datas. E.g., caso um hotel tenha apenas uma reserva de 100€/noite de 2023/10/01 a 2023/10/10, e quisermos saber as receitas entre 2023/10/01 a 2023/10/02, deverá ser retornado 200€ (duas noites). Por outro lado, caso a reserva seja entre 2023/10/01 a 2023/09/02, deverá ser retornado 100€ (uma noite).

Comando
8 <Id> <Begin_date> <End_date>
Output
revenue

Figura 9: Comando Query 8

3.5.1 Implementação

Procuramos todas as *reservations*, na *hashtable* das mesmas, que foram efetuadas num hotel e período especificados, calculamos o preço de cada *reservation* nesse período e somamos à receita total.

3.6 Query 9

Q9: Listar todos os utilizadores cujo nome começa com o prefixo passado por argumento, ordenados por nome (de forma crescente). Caso dois utilizadores tenham o mesmo nome, deverá ser usado o seu identificador como critério de desempate (de forma crescente). Utilizadores inativos não deverão ser considerados pela pesquisa.

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

Figura 10: Comando Query 9

3.6.1 Implementação

Procuramos todas os *users*, na *hashtable* dos mesmos, cujo nome começa com o prefixo especificado e a sua conta não está inativa, usamos-los para criar uma lista e organizamos essa mesma lista consoante os requisitos em cima.

4 Conclusão

O trabalho desenvolvido nesta primeira fase satisfez, de uma forma geral, todos os requisitos presentes para esta fase. Este projeto foi desenvolvido com o objetivo de ser o mais eficiente e rápido possível, tentando sempre cumprir as regras de encapsulamento e modularidade. Embora exista sempre espaço para melhorar o trabalho, encontramos-nos satisfeitos com esta entrega.