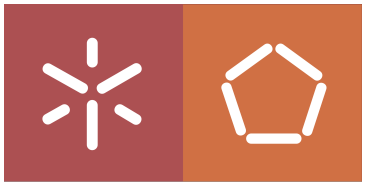

Universidade do Minho
Escola de Engenharia

Trabalho Prático

Laboratórios de Informática III

2 de Outubro, 2023

1 Objetivos

- 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 (computacional, consumo de memória, etc);
- 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.

2 Realização e avaliação do trabalho desenvolvido

O desenvolvimento deste projeto deve ser realizado colaborativamente com o auxílio de *Git* e *GitHub*. Os docentes serão membros integrantes da equipa de desenvolvimento de cada trabalho e irão acompanhar semanalmente a evolução dos projetos. A entrega do trabalho será realizada também através dessa plataforma. Os elementos do grupo serão avaliados individualmente de acordo com a sua contribuição no repositório e na apresentação e discussão do mesmo.

A estrutura do repositório deverá ser mantida, assim como deverão ser escrupulosamente observadas as regras descritas neste enunciado, de forma a que o processo de avaliação e execução dos trabalhos possa ser uniforme entre os grupos e tão automático quanto possível. Deverão seguir a seguinte estrutura:

```
trabalho-pratico
|- include
|  |- ...
|- src
|  |- ...
|- Resultados
   |- ...
|- relatorio-fase1.pdf
|- relatorio-fase2.pdf
|- programa-principal (depois do make)
|- programa-testes (depois do make)
```

Tenham, em particular atenção, ao conjunto de instruções a seguir:

- O trabalho terá de ser desenvolvido por **todos os elementos do grupo de trabalho** e todos deverão registar as suas contribuições individuais no respetivo repositório *git*;
- O trabalho desenvolvido por cada grupo será avaliado em **duas fases** com base no conteúdo da pasta "`trabalho-pratico`" do repositório *GitHub*. A primeira fase de avaliação considerará o conteúdo do repositório no dia 2023/11/11 (23:59). A segunda fase considerará o conteúdo do repositório no dia 2024/01/15 (23:59).
- Cada fase será acompanhada de um **relatório** (máximo de 10 páginas de conteúdo, com páginas adicionais para capas/anexos, em formato PDF) que deverá ser disponibilizado na raiz da pasta "`trabalho-pratico`" na mesma data da entrega da respetiva fase do trabalho. Os ficheiros correspondentes terão os nomes "`relatorio-fase1.pdf`" e "`relatorio-fase2.pdf`", respetivamente. O conteúdo do relatório deverá centrar-se no desenho e implementação da aplicação, identificando as estratégias seguidas e eventuais limitações, bem como eventuais aspetos a melhorar no futuro. Cada relatório deverá ser sempre acompanhado de uma **descrição textual e visual da arquitetura geral da aplicação**, ilustrando a interação entre os diversos módulos;
- O projeto terá de gerar o necessário ficheiro executável (com o nome **"`programa-principal`"**) com base na preparação de um ficheiro *Makefile* (ambos na raiz da pasta "`trabalho-pratico`") e por invocação do comando *make*. Da mesma forma, deverá limpar todos os ficheiros desnecessários ao projeto através da execução do comando *make clean*. Deverá gerar ainda um executável **"`programa-testes`"**, para proceder à realização de testes funcionais e de desempenho;
- O desenvolvimento da aplicação deverá ser feito com uso exclusivo da biblioteca padrão do C (i.e., *libc*). As únicas exceções são a possibilidade de recorrer à *glib2* para a manipulação das coleções de dados, e *ncurses*/*termcap*/*terminfo* e *GNU readline/history* para implementações do modo interativo (a utilização destas bibliotecas adicionais é opcional, não sendo necessárias à realização do projeto);
- Os executáveis deverão assumir a existência de ficheiros de entrada com os nomes ***users.csv***, ***flights.csv***, ***passengers.csv***, e ***reservations.csv***, numa pasta cujo caminho é passado como argumento;
- A aplicação deverá assumir dois modos de execução, diferenciados pelo número de argumentos recebidos ao executá-lo. Os modos são:
  - *Batch*: Neste modo, o programa é executado com **dois argumentos** de linha de comando. O primeiro é o caminho para a pasta onde estão os ficheiros de entrada. Já o segundo corresponde ao caminho para um ficheiro de texto que contém uma lista de comandos (*queries*) a serem executados. O resultado da execução de cada comando deverá ser escrito num ficheiro individual localizado na pasta "`Resultados`" da raiz da pasta "`trabalho-pratico`". O formato dos ficheiros de comandos e de resultados são descritos na Secção 4.

  - *Interativo*: Neste modo, o programa é executado **sem argumentos**. Nele, o grupo disponibilizará um menu interativo contendo toda a informação (instruções) necessária para a execução de cada comando (*query*). Aqui, cada comando é interpretado e executado individualmente, com o respetivo resultado apresentado no terminal. Para isso, o programa deverá inicialmente perguntar ao utilizador qual o caminho do dataset a processar.[1] Este modo deverá fornecer ainda uma funcionalidade de paginação para permitir navegar mais facilmente quando o *output* é longo.

- Em ambas as fases, o trabalho realizado por cada grupo será avaliado em **sessões de discussão** com a equipa docente, posteriormente à entrega da componente em avaliação.

2.1 Fases e critérios de avaliação

Para uma melhor organização do desenvolvimento do trabalho e para potencializar um melhor resultado, cada fase da avaliação estará centrada num conjunto pré-definido de componentes.

2.1.1 Fase 1

Esta primeira fase terá um peso de 40% da nota final. Focar-se-á, sobretudo, numa avaliação funcional da aplicação, com base nos resultados de uma plataforma de testes automáticos. Tem, no entanto, ainda uma componente não funcional relacionada com a qualidade geral do código e com o relatório elaborado. Para a sua realização, terão que considerar os seguintes pontos:

- Parsing dos ficheiros de entrada;
- Modo de operação *batch*;
- 60% das *queries* descritas na Secção 4. As *queries* a implementar ficam ao critério do grupo;
- Validação do dataset, tal como descrito na Secção 5;
- Execução do programa sem *memory leaks*.

Cada grupo deverá ter ainda o cuidado de certificar-se que o seu programa compila e corre na **plataforma de testes automáticos** da UC, disponível em `https://li3.di.uminho.pt`. O ambiente usado para os testes pode ser consultado no Anexo A. *Queries* não implementadas deverão ser ignoradas sem *crashar* o programa, de modo a que toda a aplicação possa ser testada.

Apesar das componentes relacionadas com a **modularidade e encapsulamento** não serem contabilizadas para a nota desta primeira fase, o grupo deverá tê-las sempre em consideração no desenho e implementação da aplicação e na discussão do trabalho – oral e no relatório que terá que preparar –, de modo a receber o necessário *feedback* da equipa docente para potenciais melhorias na segunda fase do projeto.

[1] Por conveniência, poderá ainda assumir um caminho *default* caso o utilizador não especifique nenhum. Sugere-se, no entanto, que esse caminho *default* seja relativo à pasta de execução do programa.

2.1.2 Fase 2

A segunda fase terá um peso de 60% da nota final, onde terão de considerar os seguintes pontos:

- Mesmos requisitos da fase 1;
- Totalidade das *queries* descritas na Secção 4;
- Modo de operação *interativo*, incluindo o menu de interação com o programa e um módulo de paginação para apresentação de resultados longos;
- Análise e discussão sobre o desempenho da solução desenvolvida, incluindo testes de tempo de CPU e memória (Secção 6);
- Testes funcionais (Secção 6);
- Evolução dos aspetos relacionados com a modularidade e encapsulamento;
- Adequação da aplicação a um dataset com uma ordem de grandeza superior.

As estratégias de **modularidade e encapsulamento** são conceitos fundamentais desta unidade curricular. Nesse sentido, a demonstração de uma correta aplicação destes conceitos é de caráter **obrigatório**, sendo fundamental para uma avaliação positiva em Laboratórios de Informática III.

2.1.3 Critérios de avaliação

A avaliação do trabalho estará concentrada, principalmente, nos seguintes critérios e pesos (indicativos):

- Fase 1:
  - Compilação / Makefile (Peso = 5%);
  - Número de *queries* corretamente implementadas (modo *batch*) (Peso = 40%);
  - Validação do dataset (Peso = 20%);
  - Execução sem *memory leaks* (Peso = 10%);
  - Qualidade geral do código e documentação (Peso = 15%);
  - Relatório (Peso = 10%).
- Fase 2:
  - Estratégia de modularização e reutilização do código (Peso = 20%);
  - Estratégia de encapsulamento e abstração (Peso = 15%);
  - Desempenho da aplicação/Adequação das estruturas e algoritmos (Peso = 20%);
  - Número de *queries* corretamente implementadas (modo *batch* e *interativo*) (Peso = 15%);

- Testes funcionais e de desempenho (Peso = 10%);
- Execução sem *memory leaks* (Peso = 5%);
- Qualidade geral do código e documentação (Peso = 5%);
- Relatório e análise de desempenho (Peso = 10%).

3 Arquitetura da aplicação

Com o objetivo de auxiliar o desenho da solução, a Figura 1 apresenta uma arquitetura exemplo da aplicação a desenvolver. Observe, com particular atenção, a divisão entre módulos de leitura de dados (i.e., *batch* e *interativo*), interpretação de comandos, escrita de dados, execução de *queries*, catálogos, tipos de dados, estruturas de dados, e módulos de utilidade.

- Parsing dos dados (*Parser*): módulo no qual é realizada a leitura dos ficheiros de entrada CSV e é efetuado um *parsing* genérico;
- Interpretação dos comandos (*Interpreter*): módulo responsável por ler o ficheiro de comandos, interpretar cada um, e executar a respetiva *query* com os argumentos indicados (se existirem);
- Execução das interrogações (*Queries*): módulo responsável por implementar a lógica das interrogações, delegando responsabilidades aos respetivos catálogos conforme necessário;
- Output dos dados (*Output*): módulo no qual é realizada a escrita dos dados para a respetiva saída (consola ou ficheiro);
- Catálogos de dados (*Catalogs*): módulos responsáveis por armazenar e processar informações das diversas entidades, como utilizadores, voos, etc.;
- Tipos (*Data Types*) e estruturas de dados (*Data Structures*): Tipos e estruturas de dados necessários para a representação e armazenamento de dados, respetivamente;
- Estatísticas (*Statistics*): módulo que efetua relações entre as várias entidades, proporcionando um acesso mais rápido a dados e resultados pedidos nas *queries* do programa;
- Módulos de utilidade (*Utilities*): eventuais módulos extra necessários para o desenvolvimento da aplicação.

Para a estruturação adequada destes módulos, será crucial analisar as *queries* que a aplicação deverá implementar. É de realçar que não devem assumir que o programa irá trabalhar sempre sobre os mesmos ficheiros de dados. Ao conceber os módulos de dados seguindo a estratégia sugerida, obtém-se a primeira arquitetura de base para o projeto, sendo desde já de salientar que esta arquitetura (ou algo equivalente) irá ser construída por fases, módulo a módulo, testando cada fase e cada módulo até se ter a garantia da sua corretude. Só depois destes testes, deverão juntar todas as unidades do projeto na aplicação final.

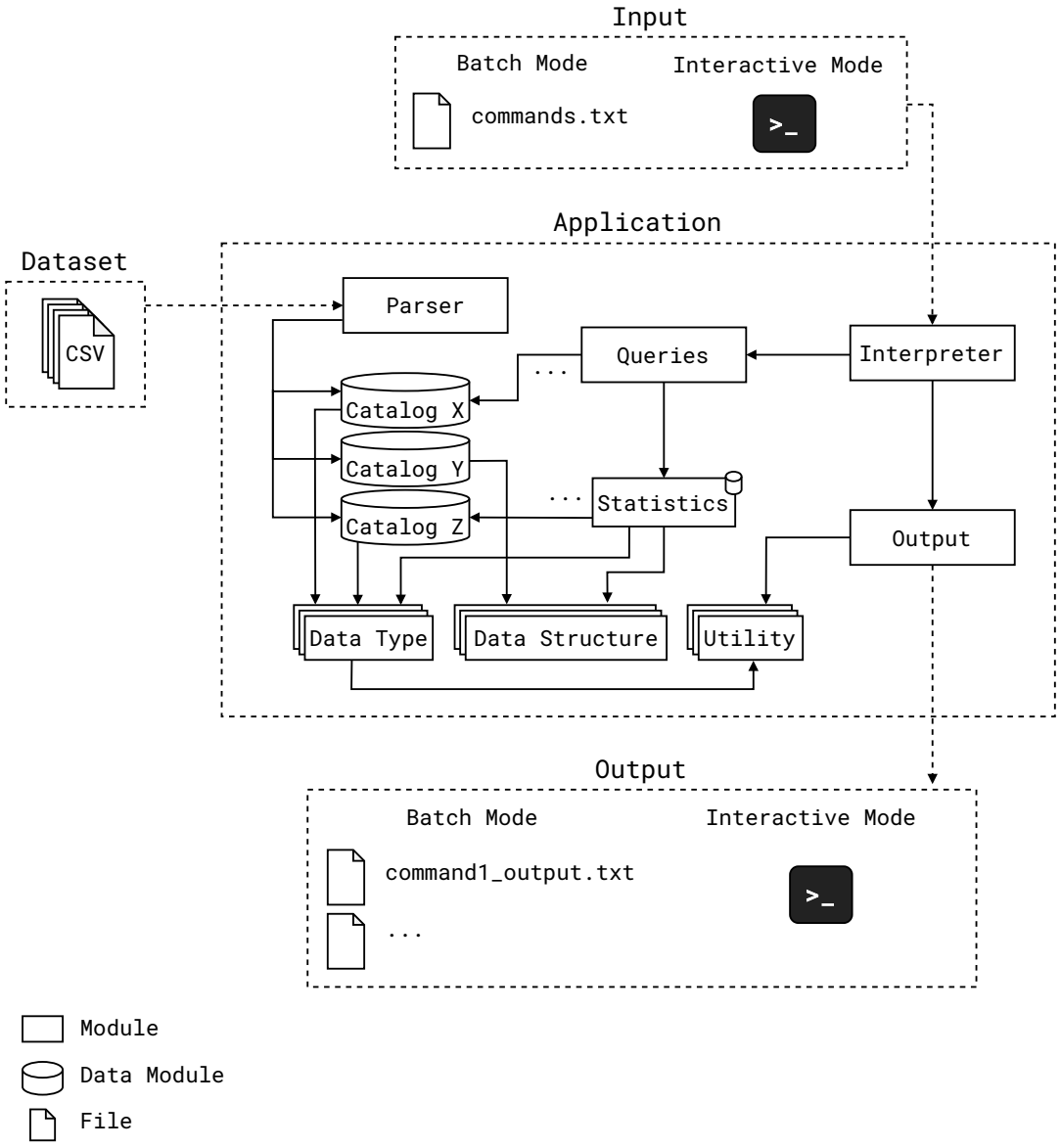

Figura 1: Arquitetura de referência para a aplicação a desenvolver.

4 Funcionamento da Aplicação

De forma a convenientemente estruturar, gerir, e expandir o projeto, pretende-se que na sua conceção tenham em conta os princípios de Modularidade e Encapsulamento abordados nas aulas. A aplicação destes conceitos é obrigatória e será objeto central de avaliação no momento da apreciação e discussão do trabalho.

Com vista a avaliar e validar o funcionamento e a eficiência do armazenamento e gestão da informação em memória, pretende-se que o trabalho prático dê resposta a um conjunto de interrogações (*queries*) sobre os dados. A interação do utilizador com o programa será única e exclusivamente através da sua invocação via linha de comandos, seguindo os requisitos de cada modo de operação (i.e., *batch* ou *interativo*). No modo *batch*, as *queries* que o utilizador pretende executar serão espe-

cificadas como linhas de um ficheiro de texto, cujo nome é passado como argumento ao programa (ver Secção 2). O formato para a especificação de *queries* (dentro do ficheiro de texto) é o seguinte:

```
<query-id>[format-flag] [arg1...argN]
```

O ficheiro de *queries* deverá então especificar uma *query* por linha. Cada uma destas linhas corresponde a um comando. Exemplo do conteúdo de um ficheiro de comandos:

```
2 U000000001
2F U000000001
4 H000000001
5 OPO "2023/10/01 00:00:00" "2023/10/01 23:59:59"
```

No exemplo acima, são definidos quatro comandos: Os dois primeiros referem-se à *query* 2, o terceiro refere-se à *query* 4, e o último à *query* 5. Notar que diferentes *queries* podem ter diferentes números de argumentos, e que uma *query* pode ser executada várias vezes ao longo do programa.

Ainda no modo de operação *batch*, o resultado de cada comando deverá ser escrito num ficheiro de texto com nome que seguirá o formato *commandX_output.txt*[2] e que deverá ser armazenado na pasta "`Resultados`" da raiz da pasta "`trabalho-pratico`". Um exemplo de ficheiro de resultado para a primeira linha seria[3]:

command1_output.txt

```
F000000123;2023/10/06;flight
R000000456;2023/10/02;reservation
F000000121;2023/10/01;flight
```

Caso uma *query* no modo batch contenha a flag '`F`', o seu resultado deverá ser apresentado com o formato `field: value`, para além da indicação do número do registo no seu início (através de `---` n `---`, onde n é o número do registo). Por exemplo, o *output* do comando 2 seria:

command2_output.txt

```
--- 1 ---
id: F000000123
date: 2023/10/06
type: flight

--- 2 ---
id: R000000456
date: 2023/10/02
type: reservation

--- 3 ---
...
```

[2]Onde *X* é o número da linha do respetivo comando no ficheiro de comandos.

[3]Para *queries* com *inputs* inválidos, o ficheiro de *output* deverá ser criado, mas sem conteúdo.

No modo de operação *interativo*, o grupo é responsável por definir o formato de entrada das *queries* e o *layout* dos respetivos resultados. Por exemplo, pode dar ao utilizador a opção de escolher o *output* com um formato CSV, tabular, campo a campo, entre outros. São valorizadas interfaces que sejam intuitivas e que lidem graciosamente com eventuais erros do utilizador (e.g., passar texto para uma *query* de top N, indicar uma pasta inválida para o dataset, ...).

Descrição dos ficheiros de entrada

Considere agora os ficheiros disponibilizados na BB, cuja descrição é apresentada de seguida:

- **Utilizadores** (*users.csv*):
  - *id* – identificador do utilizador;
  - *name* – nome;
  - *email* – email;
  - *phone_number* – número de telemóvel;
  - *birth_date* – data de nascimento;
  - *sex* – sexo;
  - *passport* – número do passaporte;
  - *country_code* – código do país de residência;
  - *address* – morada;
  - *account_creation* – data de criação da conta;
  - *pay_method* – método de pagamento;
  - *account_status* – estado da conta.
- **Voos** (*flights.csv*):
  - *id* – identificador do voo;
  - *airline* – companhia aérea;
  - *plane_model* – modelo do avião;
  - *total_seats* – número de lugares totais disponíveis;
  - *origin* – aeroporto de origem;
  - *destination* – aeroporto de destino;
  - *schedule_departure_date* – data e hora estimada de partida;
  - *schedule_arrival_date* – data e hora estimada de chegada;
  - *real_departure_date* – data e hora real de partida;
  - *real_arrival_date* – data e hora real de chegada;
  - *pilot* – nome do piloto;
  - *copilot* – nome do copiloto;
  - *notes* – observações sobre o voo.
- **Passageiros** (*passengers.csv*):
  - *flight_id* – identificador do voo
  - *user_id* – identificador do utilizador

- **Reservas** (*reservations.csv*):
  - *id* – identificador da reserva;
  - *user_id* – identificador do utilizador;
  - *hotel_id* – identificador do hotel;
  - *hotel_name* – nome do hotel;
  - *hotel_stars* – número de estrelas do hotel;
  - *city_tax* – percentagem do imposto da cidade (sobre o valor total);
  - *address* – morada do hotel;
  - *begin_date* – data de início;
  - *end_date* – data de fim;
  - *price_per_night* – preço por noite;
  - *includes_breakfast* – se a reserva inclui pequeno-almoço;
  - *room_details* – detalhes sobre o quarto;
  - *rating* – classificação atribuída pelo utilizador;
  - *comment* – comentário sobre a reserva.

Considere ainda que:

- As datas devem seguir o formato *aaaa/MM/dd*;.
- As datas com tempo devem seguir o formato *aaaa/MM/dd hh:mm:ss*;
- Os resultados representados por números decimais deverão ser **arredondados** a três casas na parte decimal;[4]
- No modo de operação *interativo*, caso os resultados excedam a capacidade de uma página, deverá existir um **menu de navegação** para consultar as diferentes páginas de resultados;
- Deverá ser considerada a data **2023/10/01** como a data atual do sistema, devendo estar especificada no código através de um ou mais `#define`;
- Deverá ser usado o tipo de dados **`double`** (e não `float`) para a representação de valores decimais;
- Caso a *query* não retorne nenhum resultado (e.g., id inexistente na Q1), o ficheiro resultante no modo *batch* **deverá ser gerado sem conteúdo** (não deverá ser colocado um *newline*);
- O custo total de uma reserva deve ser calculado com a seguinte fórmula: preço_por_noite $\times$ número_de_noites $+ \frac{\text{preço\_por\_noite} \times \text{número\_de\_noites}}{100} \times$ imposto_da_cidade;
- O total gasto por um utilizador é calculado a partir da soma dos seus gastos em reservas;

[4]Notar que um valor médio de, e.g., 3.2, deverá ser apresentado como 3.200. Sugestão: o formato `%.3f` faz o arredondamento e coloca os zeros extra caso necessário.

- O número de noites de uma reserva é calculado a partir da diferença entre a datas de início e fim. E.g., o número de noites para uma reserva com início a 2023/10/01 e fim a 2023/10/02 é 1.
- Para simplificar as *queries*, numa reserva, a data de início e fim são garantidamente sempre no mesmo mês. Da mesma forma, as datas estimadas e reais dos voos são também sempre no mesmo mês;
- Atrasos de um avião são calculados a partir da diferença entre a data estimada de partida (*schedule_departure_date*) e a data real de partida (*real_departure_date*);
- Argumentos de *queries* podem, opcionalmente, ser rodeados por aspas, quando são formados por espaços. Por exemplo, o comando *5 OPO "2023/01/01 00:00:00" "2023/12/31 23:59:59"* refere-se à *query* 5 e é formado por 3 argumentos, *OPO*, *2023/01/01 00:00:00*, e *2023/12/31 23:59:59*.

Queries

De seguida, é apresentado o conjunto de interrogações, ou *queries*, que devem ser suportadas pela aplicação. Para cada *query*, são apresentados os respetivos *inputs* e *outputs*. O formato <*x*> delimita valores obrigatórios; o formato *[x]* delimita campos opcionais; o formato *x*|*y*|*z* delimita possíveis alternativas.

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)

Não deverão ser retornadas informações para utilizadores com *account_status* = "*inactive*".

Comando
1 <ID>
Output
(ver acima)

Q2: Listar os voos ou reservas de um utilizador, se o segundo argumento for *flights* ou *reservations*, respetivamente, ordenados por data (da mais recente para a mais antiga). Caso não seja fornecido um segundo argumento, apresentar voos e reservas, juntamente com o tipo (*flight* ou *reservation*).

Para os voos, *date = schedule_departure_date* (contudo, deverá ser descartada a componente das horas/minutos/segundos no *output*), enquanto que para as reservas, *date = begin_date*.[5] Em caso de empate, ordenar pelo identificador (de forma crescente). Tal como na Q1, utilizadores com *account_status* = "*inactive*" deverão ser ignorados.

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

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

Comando
3 <ID>
Output
rating

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
...

Q5 Listar os voos com origem num dado aeroporto, entre duas datas, ordenados por data de partida estimada (da mais recente para a mais antiga). 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
...

Q6: Listar o top N aeroportos com mais passageiros, para um dado ano. Deverão ser contabilizados os voos com a data estimada de partida nesse ano. Caso dois aeroportos tenham o mesmo valor, deverá ser usado o nome do aeroporto como critério de desempate (de forma crescente).

[5]Para comparar datas de voos com datas de reservas, considera-se que o tempo da data de reserva é *00:00:00*.

Comando
6 <Year> <N>
Output
name;passengers
name;passengers
...

Q7 Listar o top N aeroportos com a maior mediana de atrasos. Atrasos num aeroporto são calculados a partir da diferença entre a data estimada e a data real de partida, para voos com origem nesse aeroporto. O valor do atraso deverá ser apresentado em segundos. Caso dois aeroportos tenham a mesma mediana, o nome do aeroporto deverá ser usado como critério de desempate (de forma crescente).

Comando
7 <N>
Output
name;median
name;median
...

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 2023/10/02, deverá ser retornado 100€ (uma noite).

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

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
...

Q10 Apresentar várias métricas gerais da aplicação. As métricas consideradas são: número de novos utilizadores registados (de acordo com o campo *account_creation*); número de voos (de acordo com o campo *schedule_departure_date*); número de passageiros; número de passageiros únicos; e

número de reservas (de acordo com o campo *begin_date*). Caso a *query* seja executada sem argumentos, apresentar os dados agregados por ano, para todos os anos que a aplicação tem registo. Caso a *query* seja executada com um argumento, *year*, apresentar os dados desse ano agregados por mês. Finalmente, caso a *query* seja executada com dois argumentos, *year* e *month*, apresentar os dados desse ano e mês agregados por dia. O *output* deverá ser ordenado de forma crescente consoante o ano/mês/dia.

Comando
10 [year [month]]
Output
(year|month|day);users;flights;passengers;unique_passengers;reservations
(year|month|day);users;flights;passengers;unique_passengers;reservations
...

5 Validação dos ficheiros de dados

Para a implementação da aplicação, deverão ainda considerar um conjunto de validações a executar sobre os dados recebidos (apenas referente aos CSVs com a informação de utilizadores, voos, e reservas; comandos são sempre válidos[6]). Caso alguma entrada seja inválida, deverá ser ignorada. Apenas certos campos de certas linhas é que poderão ser inválidos. Contudo, os CSVs em si continuarão a ter sempre o formato válido (i.e., mesmo número de colunas para todas as linhas). Caso, por exemplo, um utilizador seja inválido, é necessário descartar todas as suas entradas no ficheiro de passageiros e no ficheiro de reservas, bem como descartar o utilizador em si. Da mesma forma, se um voo for inválido, é necessário também descartar todas as entradas no ficheiro de passageiros. Considerar sempre que o ficheiro de passageiros contém identificadores de voo e utilizadores existentes, e que as reservas referem-se a utilizadores existentes (ou seja, na validação de, por exemplo, um voo, não é necessário verificar se os utilizadores associados existem).

Para além de filtrar os dados inválidos para executar as *queries*, é necessário ainda escrever num ficheiro todas as entradas inválidas, tal como aparecem nos CSVs originais. Os ficheiros deverão ser guardados na pasta "`Resultados`", com o nome *file*`_errors.csv`, onde *file* é o nome da respetiva entidade (e.g., `users_errors.csv`), juntamente com o *header* original na primeira linha. A ordem das linhas inválidas não é relevante.

De seguida, apresentam-se as validações que têm que considerar:

- Datas:
  - O formato deverá ser sempre do tipo *nnnn/nn/nn*, onde n é um número entre 0 e 9 (inclusivo). Exemplo de possíveis erros: `2023/1234`, `09/10`, `20230901`, `09/01/2023`, `2023|09|01`, `abcd/09/01`, `...`;
  - O mês deverá estar entre 1 e 12 (inclusivo) e o dia entre 1 e 31 (inclusivo). Ignorar a validação dos dias consoante o mês (e.g., datas como 2023/02/31 não surgirão). Exemplos de erros: `2023/01/52`, `2023/14/03`, ...

[6]Comandos são sempre válidos em termos de formato. Contudo, podem receber argumentos que não retornem nenhum resultado (e.g., identificador não existente na Q1).

- Datas com tempo:
  - O formato deverá ser sempre do tipo *nnnn/nn/nn nn:nn:nn*, onde n é um número entre 0 e 9 (inclusivo);
  - As mesmas regras das datas aplicam-se aqui;
  - A hora deverá estar entre 0 e 23 (inclusivo), os minutos entre 0 e 59 (inclusivo), e os segundos entre 0 e 59 (inclusivo). Exemplos de erro: `2023/10/01 32:23:05`, `2023/10/01 12:95:05`, `2023/10/01 12:23:75`, ...;
- As datas/datas com tempo de fim não poderão ser superiores às datas de início. Nos utilizadores, o *birth_date* tem que vir antes de *account_creation*; Nos voos, o *schedule_departure_date* tem que vir antes de *schedule_arrival_date*, e o *real_departure_date* tem que vir antes de *real_arrival_date*; E nas reservas, o *begin_date* tem que vir antes do *end_date*;
- O *email* de um utilizador tem que ter o seguinte formato: "*<username>@<domain>.<TLD>*". O *<username>* e o *<domain>* têm que ter pelo menos tamanho 1; O *<TLD>* tem que ter pelo menos tamanho 2. Exemplo de erros: `@email.com`, `john@.pt`, `john@email.a`, `john@email,pt`, `john.email.pt`, ...;
- O *country_code* de um utilizador deverá ser formado por duas letras;
- O *account_status* de um utilizador deverá ter o valor "*active*" ou "*inactive*", sendo que diferentes combinações de maiúsculas e minúsculas também são válidas (e.g., "*Active*", "*aCtive*", e "*INACTIVE*" também são válidos);
- O número de lugares de um voo (*total_seats*) não poderá ser inferior ao número de passageiros nesse voo;
- Os aeroportos de origem e destino de um voo tem que ser constituídos por 3 letras. Considera-se que dois aeroportos são iguais quando são formados pela mesma sequência de letras, mesmo que existam diferenças entre maiúsculas e minúsculas (e.g., *OPO*, *opo*, e *opO* são considerados o mesmo aeroporto);
- O número de estrelas de um hotel (*hotel_stars*) tem que ser um valor inteiro entre 1 e 5, inclusive. Exemplos de erros: `0`, `-3`, `1.4`, ...;
- A percentagem de imposto da cidade de uma reserva (*city_tax*) tem que ser um valor inteiro maior ou igual a zero; Exemplos de erros: `-3`, `1.4`, ...;
- O preço por noite de uma reserva (*price_per_night*) tem que ser um valor inteiro maior que 0. Exemplo de erros: `0`, `-3`, `1.4`, ...;
- O campo *includes_breafast* de uma reserva pode ser especificado das seguintes formas (com diferentes combinações de maiúsculas e minúsculas): Para valores falsos, "*f*", "*false*", "*0*", e "" (string vazia); Para valores verdadeiros, "*t*", "*true*", e *"1"*.

- As classificações de uma reserva (*rating*) têm que ter um valor inteiro entre 1 e 5, inclusive. Exemplos de erros: `0`, `-3`, `1.4`, ... Opcionalmente, podem estar vazias caso o utilizador não tenha classificado o hotel;
- Os seguintes restantes campos têm que ter tamanho superior a zero:
  - Utilizador: *id*, *name*, *phone_number*, *sex*, *passport*, *address*, *pay_method*;
  - Voo: *id*, *airline*, *plane_model*, *pilot*, *copilot*;
  - Reserva: *id*, *user_id*, *hotel_id*, *hotel_name*, *address*.

6 Testes funcionais e de desempenho

Nesta componente do trabalho, pretende-se que sejam desenvolvidos testes que validem e avaliem o funcionamento do programa desenvolvido. Desta forma, deverão ser desenvolvidos testes que avaliem o funcionamento de cada *query* descrita na Secção 4.

O "`programa-testes`" deverá receber **três argumentos**: O caminho para o dataset com os CSVs, o ficheiro com os comandos a executar, e uma pasta com os ficheiros de *output* esperado (com o formato *commandN_output.txt*). O programa deverá **comparar** cada resultado esperado com o resultado real da *query* executada, indicando se o teste passou ou não. Caso o resultado obtido seja diferente do esperado, deverão indicar a linha onde a primeira incongruência foi encontrada. Para além da validação, o "`programa-testes`" deverá indicar o **tempo de execução** de cada *query*, bem como o tempo de execução geral. Para além disso, deverão apresentar a **memória usada** pelo programa. O formato do *output* do "`programa-testes`" fica ao critério de cada grupo.

Para o relatório da Fase 2 do projeto, será ainda necessário realizar uma análise do desempenho. O grupo deverá fazer testes para cada *query* nas diferentes máquinas dos seus elementos e apresentar os resultados, bem como o ambiente de testes usado (hardware, número de repetições, ...). Naturalmente, os resultados deverão ser acompanhados de uma discussão que vise justificar os resultados obtidos, de acordo com os algoritmos e estruturas de dados usadas. São valorizadas análises extensivas e apresentadas com auxílio a gráficos/tabelas.

Tal como a restante aplicação, os testes deverão seguir os princípios de modularidade e encapsulamento.

Algumas ferramentas úteis:

- Medir o tempo de CPU através do `time.h`:

```c
#include <time.h>
#include <stdio.h>
// ...
    struct timespec start, end;
    double elapsed;
    // Start time
    clock_gettime(CLOCK_REALTIME, &start);
    // Execute work
```

```
    // ...
    // End time
    clock_gettime(CLOCK_REALTIME, &end);
    // Elapsed time
    elapsed = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;
    printf("Elapsed time: %.6f seconds\nx=%d\n", elapsed, x);
```

- Medir a memória máxima do programa com o `sys/resource.h`:

```
#include <stdio.h>
#include <sys/resource.h>
// ...
    struct rusage r_usage;
    getrusage(RUSAGE_SELF, &r_usage);
    printf("Memory usage: %ld KB\n", r_usage.ru_maxrss);
```

A Simulação do ambiente de testes

- Através de uma máquina virtual:

```
# Descarregar Ubuntu Server 22.05 LTS
# https://ubuntu.com/download/server

# Instalar usando um software de virtualização (ex: https://www.virtualbox.org/)
# No processo de instalação, escolher a opção para instalar o OpenSSH server

# Se o VirtualBox for usado, ir às configurações da máquina e alterar o adaptador para
↪  bridged:
# Settings > Network > Adapter 1 > Attached to: Bridged Adapter > Ok

# Fazer login na máquina usando o GUI do software de virtualização

# Determinar o IP
sudo apt install net-tools -y
ifconfig
# O ip deverá começar por 192.168... ou 10.... (ex: 192.168.1.2)
# Sair da máquina

# Entrar na máquina usando ssh (substituir o user e host pelos valores corretos; 'exit' para
↪  sair)
ssh user@host
# ex: ssh ubuntu@192.168.1.2

# Instalar dependências
sudo apt update
sudo apt install gcc make libglib2.0-dev libgtk2.0-dev \
    valgrind libncurses-dev libncurses6 libncursesw6 libreadline8 \
    libreadline-dev -y

# Entrar na máquina usando vscode (para desenvolvimento direto na máquina)
# https://code.visualstudio.com/docs/remote/ssh

# Copiar ficheiros para a máquina
scp ficheiro user@host:
```

- Através de um *container* Docker:

```
# Instalar docker
# https://docs.docker.com/get-docker/

# Criar um ficheiro 'Dockerfile', com o seguinte conteúdo:
FROM ubuntu:22.04
RUN apt update
RUN apt install gcc make libglib2.0-dev libgtk2.0-dev valgrind \
    libncurses-dev libncurses6 libncursesw6 libreadline8 \
    libreadline-dev -y

# Criar a imagem, executando o seguinte comando na diretoria do ficheiro 'Dockerfile'
docker build . -t li3-img
```

```
# Criar o container
docker run --name li3 -d -t li3-img

# Entrar no container pelo terminal ('exit' para sair)
docker exec -it li3 bash

# Entrar no container usando o vscode (para desenvolvimento direto no container)
# https://code.visualstudio.com/docs/remote/attach-container

# Copiar ficheiros para o container
docker cp ficheiro li3:/

# Parar o container
docker stop li3
# Inciar o container
docker start li3
```