Universidade do Minho
Escola de Engenharia

Trabalho Prático (Fase 2)

Laboratórios de Informática III

2025/2026

Síntese das alterações introduzidas pela Fase 2

- **Secção 2:** clarificação relativa ao caráter indicativo dos fatores de ponderação dos critérios de avaliação de cada uma das fases; e explicitação dos critérios de avaliação da segunda fase do projeto.
- **Secção 4.1:** indicações relativas à documentação do código-fonte.
- **Secção 5.1:** introdução do modo de execução interativo.
- **Secção 5.2:** introdução de três novas queries e atualização da Q1.
- **Secção 5.4:** exemplo de utilização do programa interativo.

1 Introdução

Este trabalho prático tem por objetivo o desenvolvimento de competências essenciais à implementação de programas de forma estruturada na linguagem C. Pretende-se que os alunos desenvolvam capacidades de utilização prática da linguagem, bem como de utilização das suas ferramentas auxiliares como debuggers (gdb), ou ferramentas de análise de uso de memória (valgrind). Será dada especial atenção ao exercício de estruturação do programa em vários módulos, separando-os nas suas componentes de interface (.h) e de implementação (.c), tendo em consideração também os mecanismos que garantem o isolamento entre estes. Por outras palavras, o foco estará na apreensão e na aplicação dos conceitos e técnicas de modularidade e encapsulamento ao desenvolvimento de projetos em C.

O tema do trabalho prático centra-se num sistema baseado em aeroportos e voos. Os alunos terão que explorar um conjunto de dados relativo a aeroportos, voos, passageiros e dados relativos a reservas, carregar os dados para estruturas de dados adequadas em memória e utilizar essa informação para responder a um conjunto de queries (i.e., perguntas ou interrogações).

2 Avaliação do trabalho prático

O trabalho prático será avaliado em duas fases. Cada fase terá diferentes objetivos e terá como conclusão um momento de avaliação presencial. Os grupos serão compostos por **três** elementos, e o desenvolvimento do projeto será realizado colaborativamente com o auxílio de *Git* e *GitHub*. **O contributo individual de cada aluno será avaliado através não só do seu desempenho nas sessões presenciais de discussão do projeto, mas também do seu histórico de commits**, não sendo aceites justificações para a inexistência desse registo, tais como: a programação em conjunto utilizando o mesmo computador, falta de commits por avaria de equipamento, ou co-autoria de commits. Da mesma forma, **não será tolerada qualquer situação de plágio**.

Fases de avaliação

Fase 1

A Fase 1 do trabalho terá em conta o código presente no repositório de cada grupo às **23:59** do dia **14/11/2025**. A apresentação presencial da mesma deverá decorrer entre 24/11/2025 e 28/11/2025. Esta primeira fase terá um peso de 40% da nota final. Nesta fase os seguintes elementos serão avaliados:

- Makefile (Peso: 5%);
- Modularidade (Peso: 25%);
- Percentagem de queries da Fase 1 corretamente implementadas (Peso: 25%);
- Validação do dataset (Peso: 15%);
- Programa de testes (Peso: 5%);
- Qualidade do código (Peso: 5%);

- Ausência de memory leaks (Peso: 15%);
- Relatório (Peso: 5%).

A avaliação da correção das queries (interrogações) da Fase 1 bem como a ausência de memory leaks serão avaliados através dos resultados apresentados pela plataforma de testes (ver Secção 3). Quanto à validação do dataset, a mesma deverá seguir as regras descritas na Secção 5.3.

Fase 2

A Fase 2 do trabalho irá ter um peso de 60% da nota final. Nesta fase o dataset irá sofrer um aumento no volume total de dados. Adicionalmente, o número de queries que será executado na plataforma de testes também irá aumentar. A Fase 2 do trabalho terá em conta o código presente no repositório de cada grupo às **23:59** do dia **05/01/2025**. Nesta fase será avaliada **a totalidade do sistema**:

- Modularidade (Peso: 20%);
- Encapsulamento (Peso: 25%);
- Percentagem de queries corretamente implementadas (Peso: 10%);
- Validação do dataset (Peso: 5%);
- Programa de testes (Peso: 5%);
- Modo interativo (Peso: 10%);
- Ausência de leaks e de erros de memória no Valgrind (Peso: 5%);
- Qualidade do código (Peso: 10%);
- Relatório (Peso: 10%).

Avaliações superiores a 18 valores (Fase 2)

Os grupos que pretendam alcançar uma nota superior a 18 valores deverão utilizar e eventualmente implementar estruturas (coleções) de dados que apresentem, justificadamente, uma boa relação entre consumo de memória e de desempenho

para o tipo de interrogações a que o projeto deve dar resposta. Por outras palavras, os grupos terão de utilizar estruturas de dados mais "avançadas" – desejavelmente genéricas – como *hash tables* ou árvores, em módulos onde o seu uso seja justificado. Otimizações ao nível de consumo de memória serão também valorizadas. Para além das decisões tomadas ao nível do código, os grupos deverão efetuar uma análise do impacto da utilização dessas estruturas no desempenho das queries. Esta análise **deverá constar do relatório final**.

Cálculo da avaliação final

Assim, o cálculo final da nota será efetuado da seguinte forma:

$$Nota = \frac{(Fase\,1 * 0.4 + Fase\,2 * 0.6) * 18 + Estruturas\ e\ Desempenho * 2}{20}$$

Muito importante: as componentes de modularidade e encapsulamento constituem componentes **obrigatórios** do trabalho prático. Caso – no final das duas sessões de apresentação – não fique demonstrada a apreensão destes conceitos e da sua aplicação no desenvolvimento do projeto, considera-se que não foram atingidos os objetivos mínimos de aprendizagem de Laboratórios de Informática III e, por conseguinte, o aluno não poderá ser aprovado a esta unidade curricular.

3 Repositórios e plataforma de testes

O desenvolvimento deste projeto deve ser realizado colaborativamente com o auxílio de *Git* e *GitHub*, sendo a entrega do trabalho realizada também através desta 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 original 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. O repositório deverá seguir a seguinte estrutura:

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

Deverão ter em conta que **os ficheiros dos datasets usados localmente nos computadores em que desenvolvem o projeto não devem nuncar ser adicionados ao vosso repositório Github!** A correção das queries implementadas por cada grupo será avaliada periodicamente através de uma plataforma de testes automática, que pode ser encontrada em `https://li3.di.uminho.pt`. Para além de avaliar o resultado das queries, a plataforma irá também avaliar a existência de leaks de memória. Datasets com e sem erros de validação serão periodicamente alternados, para que os grupos possam avaliar separadamente a correção da implementação das suas queries bem como dos seus mecanismos de validação. Mais informações podem ser encontradas na página de FAQ[1] da plataforma de testes. **Note-se que queries não implementadas, quando invocadas, deverão ser capazes de retornar sem causar um *crash* do programa, de modo a que as restantes queries possam ser corretamente avaliadas**

[1] `https://li3.di.uminho.pt/faq`

pela plataforma. A execução da implementação de cada grupo na plataforma encontra-se limitada quer em tempo de execução quer em memória, pelo que deverão ter especial atenção a estes aspetos aquando do desenvolvimento do projeto (valores indicados na FAQ).

4 Aplicação de gestão de voos

Como referido anteriormente, a aplicação a desenvolver contempla a análise dos dados relativos a um sistema de gestão de voos. O sistema considera dados relativos a voos, aeroportos, aeronaves, passageiros e reservas. Em seguida são apresentados os seus campos de dados e respetivas descrições.

- **Voos** (*flights.csv*):
  - *flight id* – identificador único do voo;
  - *departure* – data e hora de partida estimada do voo;
  - *actual departure* – data e hora de partida real do voo;
  - *arrival* – data e hora de chegada estimada do voo;
  - *actual arrival* – data e hora de chegada real do voo;
  - *gate* – porta de embarque do voo;
  - *status* – estado do voo, i.e., *On Time*, *Delayed* ou *Cancelled*;
  - *origin* – código IATA do aeroporto de origem;
  - *destination* – código IATA do aeroporto de destino;
  - *aircraft* – identificador único da aeronave utilizada no voo, i.e., *tail number*;
  - *airline* – nome da companhia aérea responsável pelo voo;
  - *tracking url* – *URL* para o rastreamento do voo;
- **Aeroportos** (*airports.csv*):
  - *code* – código IATA do aeroporto;
  - *name* – nome do aeroporto;
  - *city* – cidade onde o aeroporto se encontra localizado;
  - *country* – país onde o aeroporto se encontra localizado;
  - *latitude* – latitude do aeroporto em graus decimais;
  - *longitude* – longitude do aeroporto em graus decimais;
  - *icao* – código ICAO do aeroporto;
  - *type* – tipo do aeroporto;

- **Aeronaves** (*aircrafts.csv*):
  - *identifier* – identificador único da aeronave, i.e., *tail number*;
  - *manufacturer* – fabricante da aeronave;
  - *model* – modelo da aeronave;
  - *year* – ano de fabricação da aeronave;
  - *capacity* – capacidade máxima de passageiros da aeronave;
  - *range* – alcance máximo da aeronave em km;
- **Passageiros** (*passengers.csv*):
  - *document number* – número do documento de identificação do passageiro;
  - *first name* – primeiro nome do passageiro;
  - *last name* – último nome do passageiro;
  - *dob* – data de nascimento do passageiro;
  - *nationality* – nacionalidade do passageiro;
  - *gender* – género do passageiro;
  - *email* – email do passageiro;
  - *phone* – número de telefone do passageiro;
  - *address* – morada do passageiro;
  - *photo* – fotografia do passageiro;
- **Reservas** (*reservations.csv*):
  - *reservation id* – número da reserva;
  - *flight ids* – identificadores dos voos associados à reserva;
  - *document number* – número do documento de identificação do passageiro associado à reserva;
  - *seat* – número do lugar reservado (e.g., 12A);
  - *price* – preço da reserva;
  - *extra luggage* – indica se a reserva inclui bagagem extra (*true* ou *false*);
  - *priority boarding* – indica se a reserva inclui embarque prioritário (*true* ou *false*);
  - *qr code* – código QR associado à reserva;

Novo na fase 2!

4.1 Nota sobre documentação

O código desenvolvido para o trabalho prático deverá estar devidamente documentado, estando esta componente integrada no critério de qualidade do código. Assim, considera-se que o código está corretamente documentado se seguir o estilo de documentação da ferramenta doxygen nos ficheiros de interface (.h). Cada função deverá incluir, no mínimo, uma descrição breve (*@brief*), uma descrição mais detalhada, a descrição de cada um dos parâmetros (*@param*) e a descrição do valor de retorno (*@return*)[a]. Para além disso, deverão ainda ser incluídos comentários nos ficheiros de implementação (.c) para esclarecer algum passo que considerem ser de difícil compreensão.

[a]Para mais exemplos consultar `https://www.doxygen.nl/manual/docblocks.html`

5 Trabalho prático

Neste trabalho prático deverão desenvolver um programa capaz de responder a um conjunto de queries (i.e., interrogações) relativas aos dados anteriormente descritos. Para isso, deverão carregar o conteúdo dos diferentes ficheiros para estruturas em memória que julguem adequadas, como por exemplo: listas ligadas, arrays, stacks, entre outros. Para além das queries, o vosso programa deverá ainda ser capaz de: suportar o uso de três modos de execução – **principal**, **interativo** e **testes**; e de validar as entradas de dados. **Relembramos que a correta utilização dos conceitos de modularidade e encapsulamento é fundamental para a obtenção de aproveitamento neste trabalho prático**. Para além destes, também a ausência de *leaks* de memória (i.e., memória não libertada), a qualidade do código, uma correta formulação da *Makefile* e a escolha de estruturas de dados adequadas às interrogações serão considerados como elementos de avaliação, tal como explicitado na Secção 2.

A equipa docente irá facultar os seguintes recursos para a execução do trabalho:

- Conjunto de dados sem erros: *flights.csv*, *airports.csv*, ...
- Conjunto de dados com erros: *flights.csv*, *airports.csv*, ...

- Conjunto de queries de exemplo para o modo principal: *input.txt*
- Conjunto de resultados para as queries de exemplo: *command1_output.txt*, *command2_output.txt*, ...

Cada grupo deverá recorrer ao uso de *Makefile* para gerar um conjunto de executáveis, nomeadamente o **programa-principal**, **programa-interativo** e **programa-testes**.

São permitidas as seguintes bibliotecas para o desenvolvimento da aplicação: biblioteca padrão de C (i.e., libc); bilioteca glib2 para manipulação de estruturas de dados; e as bibliotecas ncurses, termcap, terminfo, GNU readline e GNU history para implementação do modo interativo. A utilização das bibliotecas que não a padrão (libc) é opcional, não sendo essenciais à realização do projeto.

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., *principal* e *interativo*), interpretação de comandos, escrita de dados, execução de *queries*, gestores, entidades, estruturas de dados, e módulos de utilidade. **É importante destacar que o diagrama apresenta uma abstração geral do sistema que pode ser desenvolvido, pelo que não estão limitados ao que se encontra representado.**

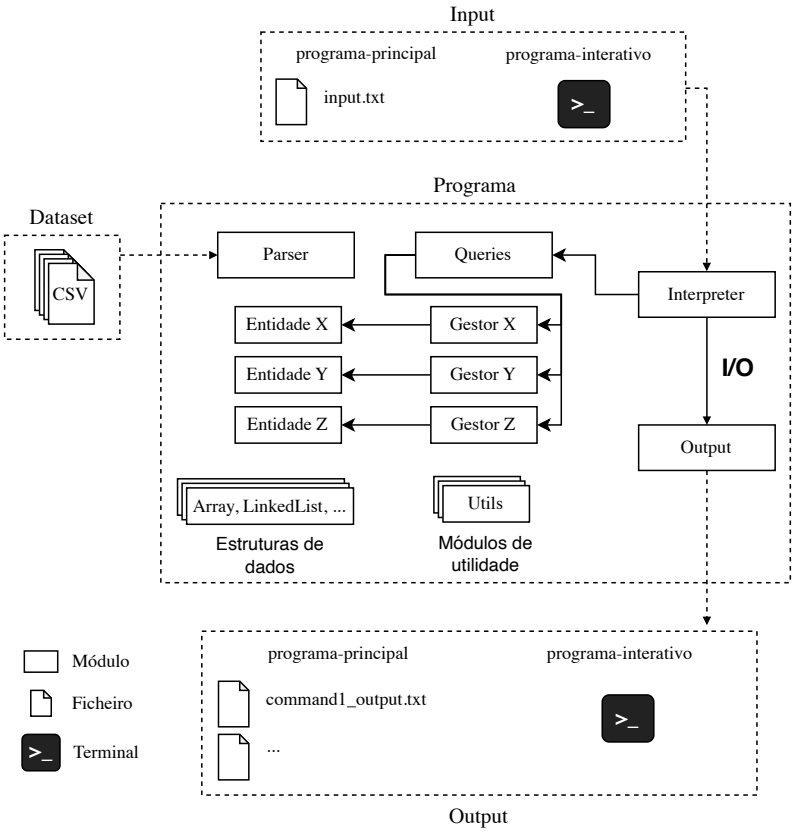

Figura 1: Arquitetura de referência para a aplicação a desenvolver.

5.1 Executáveis - principal, interativo e testes

A aplicação deverá assumir três modos de execução (cada um com o seu executável), diferenciados pela forma como o utilizador interage com o programa e pelo resultado esperado. Os grupos deverão validar no seu código que cada executável é invocado com os argumentos corretos. Note-se que o comando *make* por omissão deverá produzir os executáveis *programa-principal* e *programa-testes*, ficando ao critério do grupo o comando que irá dar origem ao executável *programa-interativo*.

programa-principal

O *programa-principal* é o executável invocado pela plataforma `https://li3.di.uminho.pt`. Este executável deve receber **dois argumentos**. O primeiro é o caminho para a pasta onde estão os ficheiros CSV de entrada (e.g., flights.csv, airports.csv, etc). Já o segundo corresponde ao caminho para um ficheiro de texto que contém uma lista de comandos (*queries*) a serem executados (cf. secção 5.2). O resultado da execução de cada comando deverá ser escrito num ficheiro individual localizado na pasta "`resultados`" da raiz da pasta "`trabalho-pratico`", e.g, *resultados/command1_output.txt*. Os ficheiros de comandos e de resultados seguem o formato abaixo. Estão também disponíveis alguns exemplos de utilização na secção 5.4.

Exemplo ficheiro de comandos "input.txt":

```
1 OPO
2 10 Boeing
...
```

Exemplo ficheiro de output "command1_output.txt":

```
OPO;Francisco de Sá Carneiro Airport;Porto;PT;large_airport
```

Novo na fase 2!

programa-interativo

Este executável **não deve receber qualquer argumento**. O grupo deve disponibilizar um menu interativo via terminal que permita executar comandos (i.e., queries) sobre os dados. O programa deverá inicialmente perguntar ao utilizador qual o caminho do dataset a processar. [a] De seguida deverá pedir ao utilizador para introduzir qual a *query* a executar e quais os argumentos. Os alunos deverão ter o cuidado de validar os dados introduzidos pelos utilizadores, de forma a que o programa não sofra um *crash* nem se comporte de forma errática quando forem introduzidos comandos ou argumentos inválidos. É dada liberdade aos alunos para conceberem a interface interativa da forma que desejarem, no entanto, devem considerar o peso relativo da mesma na avaliação face aos restantes componentes do trabalho prático. Está disponível um exemplo de utilização na secção 5.4.

[a] 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.

programa-testes

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 5.2.

O *programa-testes* deverá receber **três argumentos**: o caminho para a pasta com os CSVs de entrada, 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 do programa com o esperado, indicando o número de ocorrências corretas para cada tipo de *query*. 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 médio** para cada tipo de *query*, bem como o tempo de execução geral do programa. Finalmente, deverá ainda apresentar a **memória usada** pelo programa. A secção 5.4 apresenta um exemplo do output esperado, sendo que poderão incluir outras informações que julguem serem relevantes. O formato do *output* do *programa-testes* fica ao critério de cada grupo.

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\n", elapsed);
```

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

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

5.2 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 argumentos obrigatórios e o formato *[x]* argumentos opcionais. **Para casos em que se apliquem intervalos de valores, considere o intervalo incluindo os seus limites**, e.g., para o intervalo $10 - 18$ os valores 10 e 18 também devem ser considerados. Finalmente, sempre que for necessário executar um cálculo com referência temporal, e.g., calcular a idade de uma aeronave, **devem assumir a data *2025/09/30* como sendo a data atual**.

Novo na fase 2!

Em termos de formatação, as queries são representadas por um número, seguido dos seus argumentos. O formato de output das queries segue uma de duas variantes consoante o número da *query* é acrescido (ou não) do carater 'S'. Por omissão, os outputs deverão ser separados por ';', enquanto que o formato alternativo deverá utilizar o separador '='. Veja o exemplo abaixo:

Comando	
1 <code>	*1S <code>*
Output	
code;name;city;country;type	*code=name=city=country=type*

Para simplificação do enunciado, os exemplos abaixo assumem o formato por omissão.

Alterado na fase 2!

Q1: Listar o resumo de um aeroporto, consoante o identificador recebido por argumento.

A *query* recebe como argumento o identificador único do aeroporto. Deverá escrever no ficheiro de resultado o código, nome, cidade, país, tipo do aeroporto, **número de passageiros** que **aterraram** no aeroporto, **número de passageiros** que **partiram** do aeroporto. É garantido que não existem identificadores repetidos entre as diferentes entidades. Deverá ser retornada uma linha vazia caso o id não conste da lista de aeroportos.

Comando
1 <code>
Output
code;name;city;country;type;arrival_count;departure_count

Q2: Top *N* aeronaves com mais voos realizados

A *query* recebe como argumento o número de aeronaves que devem constar do output, podendo ainda receber (ou não) um filtro opcional, o filtro de fabricante, sendo que quando presente só devem ser considerados aeronaves desse fabricante. Deverá escrever para o ficheiro de resultado os campos que constam do exemplo abaixo. Para a contabilização de voos realizados, considere apenas os voos que não foram cancelados, isto é, voos com o estado diferente de cancelado. Adicionalmente, deverá ser possível a utilização de um filtro opcional para fabricantes de aeronaves, onde apenas deverão ser consideradas voos de aeronaves do fabricante indicado.

Em caso de empate, as aeronaves devem ser ordenadas por ordem crescente do seu identificador. A ordenção deve ser feita posição a posição (i.e., "XB-NIQ0"vem antes de "XB-OIQ0").

Comando
2 <N> [manufacturer]
Output
aircraft 1;manufacturer 1;model 1;flight_count 1
aircraft 2;manufacturer 2;model 2;flight_count 2
...

Q3: Listar o aeroporto com mais partidas entre 2 datas

A *query* recebe como argumento as datas inicial e final (inclusive). Deve produzir como output o aeroporto com mais partidas e o número total de *partidas* associados. Para a contabilização de partidas de um aeroporto deverão ser considerados voos em que o origem é igual ao aeroporto em questão, e voos com estado diferente de cancelado. Em caso de empate, considere o aeroporto como menor identificador, isto é, o código que seja menor em ordem lexicográfica. Caso não existam voos no intervalo de datas indicado, deverá ser retornada uma linha vazia. As datas a ser consideradas deverão ser as datas reais de partida (*actual departure*).

Comando
3 <data inicial> <data final>
Output
code;name;city;country;departure_count

Novo na fase 2!

Q4: Qual o passageiro que esteve mais tempo no top 10 de passageiros que mais gastaram em viagens durante um período?

Opcionalmente, a *query* pode receber um filtro que imponha o intervalo de datas a considerar. Nesse caso apenas deverão ser tidos em conta os top 10 compreendidos entre esse intervalo de tempo. Considere que o top 10 é calculado semanalmente e é determinado pelo maior valor gasto em viagens, ou seja, a soma dos preços de todas as reservas referentes ao passageiro. Considere ainda que uma semana tem início no domingo e termina no sábado seguinte. Se o intervalo de tempo captar uma semana de forma parcial, seja no início ou no fim, o top 10 dessa(s) semana(s) também deve ser considerado. Em caso de empate, prevalece o passageiro com id mais baixo.

Comando
4 [begin_date end_date]
Output
document_number;first_name;last_name;dob;nationality;count_top_10

Q5: Top N companhias aéreas com mais tempo de atraso médio por voo

A *query* recebe como argumento o número de companhias aéreas que devem constar do output. Deverá escrever para o ficheiro de resultado os campos que constam do exemplo abaixo. Para a contabilização do tempo de atraso, considere a diferença entre o tempo de partida e chegada efetivos, isto é, *actual_departure* e *actual_arrival*. Adicionalmente, deverão apenas ser considerados voos que tenham o estado *Delayed*. O tempo de atraso médio deverá ser calculado em minutos. Em caso de empate, as companhias aéreas devem ser ordenadas por ordem alfabética.

Comando
5 <N>
Output
airline 1;delayed_flights_count 1;average_delay 1
airline 2;delayed_flights_count 2;average_delay 2
...

Q6: Listar o aerporto de destino mais comum para passageiros de uma determinada nacionalidade

A *query* recebe como argumento a nacionalidade dos passageiros. Deverá escrever para o ficheiro de resultado o código IATA do aeroporto de destino mais comum entre os passageiros dessa nacionalidade, bem como o número de passageiros que viajaram para esse aeroporto. Em caso de empate, deverá ser considerado o aeroporto com o menor código IATA em ordem lexicográfica. Caso não existam passageiros com a nacionalidade indicada, deverá ser retornada uma linha vazia.

Comando
6 <nationality>
Output
code;passenger_count

5.3 Validação dos ficheiros de dados (.csv)

Para a implementação da aplicação, deverão ainda considerar um conjunto de validações a executar sobre os dados recebidos com formato *CSV*. As validações a executar são de dois tipos: **sintáticas** e **lógicas**.

Validação sintática

Certos campos podem conter erros, e.g., uma data com valor 12/193456/12. Nesses casos, a respetiva entrada no ficheiro *CSV* não deve ser considerada para efeitos de reposta a queries. Estes erros aplicam-se apenas aos valores dos campos, sendo que as linhas terão sempre o número correto de colunas. Para além disso, as entradas inválidas deverão ser registadas num ficheiro, tal como aparecem nos *CSVs* originais. Os ficheiros deverão ser guardados na diretoria $resultados$, com o nome $file\_errors.csv$, onde $file$ corresponde ao nome da respetiva entidade (e.g., $flights\_errors.csv$), juntamente com o *header* original na primeira linha. A ordem das linhas inválidas não é relevante.

De seguida, são apresentadas as validações que devem considerar:

- **Datas:**
  - O formato deverá ser sempre do tipo $aaaa-mm-dd$, onde a, m e d são números entre 0 e 9 (inclusive). Alguns possíveis erros incluem: $2023/12/04$, $09/10$, 20230901, $09/01/2023$, $2023|09|01$, $abcd/09/01$, $\ldots$;
  - O mês deverá estar entre 1 e 12 (inclusive) e o dia entre 1 e 31 (inclusive). Para efeitos de simplificação, devem ignorar a validação dos dias consoante o mês (e.g., datas como 2023/02/31 não surgirão). Alguns exemplos de erros incluem: $2023/01/52$, $2023/14/03$, $\ldots$;
  - As datas não poderão ser mais recentes do que a data atual;
  - No caso de campos do tipo *datetime*, a componente da data seguirá as mesmas regras, apenas sendo acrescentada a componente do tempo;
  - O formato de campos com data e hora segue o esquema $aaaa-mm-dd\ hh:mm$, onde h e m são números entre 0 e 9 (inclusive). As horas deverão estar entre 0 e 23 (inclusive), enquanto que os minutos deverão estar compreendidos entre 0 e 59 (inclusive). Um exemplo de erro é o caso $2023/10/01\ 12:23$;

  - No caso de campos onde apenas se aplique o ano $aaaa$, as regras anteriores aplicam-se de igual forma. Alguns exemplos de erros incluem: $20a3$, 203, 20234, ...

- **Email:**
  - O formato deverá ser sempre do tipo *username@domínio*, onde *username* corresponde a um conjunto de carateres do intervalo [a-z0-9], podendo incluir o caráter ., e o domínio corresponde a um domínio válido.
  - Um domínio válido segue o formato *<lstring>.<rstring>*, onde *lstring* corresponde a um conjunto de 1 ou mais caracteres, e onde *rstring* corresponde a um conjunto de 2 ou 3 caracteres.
  - Todos os carateres de *lstring* e *rstring* deverão pertencer ao intervalo [a-z].
  - Exemplos de endereços inválidos incluem: *user@domain*, *user&email.p*, *@email.pt*, ...

- **Tipo Aeroporto:**
  - O campo *type* apenas poderá tomar os valores *small_airport*, *medium_airport*, *large_airport*, *heliport*, ou *seaplane_base*.

- **Latitude/Longitude:**
  - O campo *latitude* deverá estar compreendido entre -90 e 90 (inclusive).
  - O campo *longitude* deverá estar compreendido entre -180 e 180 (inclusive).
  - O formato destes campos deverá ser do tipo *[-]dd.ddddd*, onde *d* corresponde a um número entre 0 e 9 (inclusive) e o caráter - é opcional. No máximo, poderão existir 8 casas decimais. Alguns exemplos de erros incluem: *-18.AB123*, *12-12345*, *12123456789*, ...

- **Código Aeroporto:**
  - Os campos referentes a códigos identificadores de aeroportos deverão ter exatamente 3 carateres, todos eles do intervalo [A-Z] (formato IATA). Alguns exemplos de erros incluem: *ABCDEF*, *AB1*, ...

- **Identificador Voo:**
  - O campo *flight id* segue a estrutura `ccddddd`, onde c corresponde a um caráter do intervalo [A-Z] e d a um número entre 0 e 9 (inclusive). Alguns exemplos de erros incluem: `AB1234567`, `12345678`, ...;
- **Identificador Reserva:**
  - O campo *reservation id* é composto pelo prefixo R seguido de nove dígitos compreendidos entre 0 e 9 (inclusive). Alguns exemplos de erros incluem: `R12345`, `ABC12345`, ...;
- **Número Documento:**
  - Os campos referentes a *document number* são compostos por nove dígitos compreendidos entre 0 e 9 (inclusive). Alguns exemplos de erros incluem: `1234567A`, `123456789876`, ...;
- **Género Passageiro:**
  - Os campos referentes a *gender* apenas devem conter os valores *M*, *F*, ou *O*. Qualquer outro valor deve ser considerado inválido.
- **Listas de CSVs:**
  - Campos dos ficheiros CSV do tipo lista devem começar com os caracteres "["e terminar com os caracteres "]". Na ausência destes delimitadores, a entrada deve ser considerada inválida.

Validação lógica

Para além de erros sintáticos, podem ainda existir certas incongruências lógicas que devem ser tidas em conta na utilização do programa.

- **Voos:**
  - O campo *destination* de um voo, deverá ser diferente do campo *origin*.
  - Os campos *arrival* e *actual arrival* não poderão ser anteriores aos respetivos campos *departure* e *actual departure*.
  - O campo *aircraft* de um voo, deverá corresponder a uma aeronave existente.

  - Caso o campo *status* de um voo tome o valor *Cancelled*, os campos *actual departure* e *actual arrival* deverão conter o valor *"N/A"*.
  - Caso o campo *status* de um voo tome o valor *Delayed*, os campos *actual departure* e *actual arrival* não poderão ser anteriores aos respetivos campos *departure* e *arrival*.

- **Reservas:**
  - O campo *flight ids* de uma reserva, deverá corresponder a uma lista de um ou dois voos existentes.
  - O campo *document number* de uma reserva, deverá corresponder a um passageiro existente.
  - O campo *flight ids* de uma reserva, quando possua dois voos, deverá conter identificadores de voos em que o campo *destination* do primeiro voo seja igual ao campo *origin* do segundo voo.

Note-se que estas validações não se aplicam ao ficheiro de queries a executar, *input.txt*, cujos comandos são sempre válidos. Contudo, algumas queries podem receber argumentos que não retornem nenhum resultado (e.g., identificador não existente na Q1).

5.4 Exemplos de utilização

Abaixo apresentamos alguns exemplos de como os diferentes executáveis deverão ser invocados, e do resultado esperado. Os exemplos são dados da perspetiva de comandos a invocar na linha de comandos, e dos resultados produzidos na diretoria do projeto ou no terminal.

programa-principal

```
make
./programa-principal dataset-errors/ input.txt
```

```
ANTES DA EXECUÇÃO:                APÓS A EXECUÇÃO:
trabalho-pratico                  trabalho-pratico
|- include                        |- include
|  |- ...                        |  |- ...
|- src                            |- src
|  |- ...                        |  |- ...
|- resultados                     |- resultados
```

```
|   |- *vazio*                          |- command1_output.txt
|                                       |- command2_output.txt
|                                       |- ...
|                                       |- flights_errors.csv
|                                       |- airports_errors.csv
|                                       |- ...
|- dataset                          |- dataset
|   |- flights.csv                      |- flights.csv
|   |- airports.csv                     |- airports.csv
|   |- ...                              |- ...
|- dataset-errors                   |- dataset-errors
|   |- flights.csv                      |- flights.csv
|   |- airports.csv                     |- airports.csv
|   |- ...                              |- ...
|                                   |- programa-principal
|                                   |- programa-testes
|- input.txt                        |- input.txt
|- resultados-esperados             |- resultados-esperados
|   |- command1_output.txt              |- command1_output.txt
|   |- ...                              |- ...
```

Novo na fase 2!

programa-interativo

```
make
./programa-interativo
```

Output do terminal:

```
Introduza o caminho dos ficheiros de dados: <input do utilizador>
Dataset carregado...
Que query deseja executar? <input do utilizador>
Código do aeroporto: <input do utilizador>
Output: ...
```

Note-se o campo `<input do utilizador>` que simboliza a introdução de texto por parte de um utilizador, na linha de comandos.

programa-testes

```
make
./programa-testes dataset-erros/ input.txt resultados-esperados/
```

Output do terminal:

```
Q1: 100 de 100 testes ok!
Q2: 90 de 100 testes ok
    Discrepância na query 84: linha 10 de "resultados/command84_output.txt"
    Discrepância na query 87: linha 2 de "resultados/command87_output.txt"
    ...
Q3: ...
Memória utilizada: 312MB
Tempos de execução:
    Q1: 100.0 ms
    Q2: 235.6 ms
    Q3: ...
Tempo total: 113s
```

6 Relatório

O relatório deve ter um máximo de 10 páginas de conteúdo, com eventuais páginas extra para capas/anexos, em formato PDF. Este 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 relatório deverá conter pelo menos as seguintes secções: **introdução**, **sistema** (com imagem de arquitetura), **discussão** e **conclusão**.

A introdução deverá contextualizar os objetivos do trabalho prático e introduzir brevemente os pontos mais relevantes do trabalho do grupo, por exemplo, destacando algumas caraterísticas do programa que julguem serem particularmente interessantes, mas sem ainda dar detalhe sobre como funcionam.

A secção de sistema deve apresentar um diagrama da arquitetura do sistema [2], explicando os módulos em que este se divide e justificando essa escolha. Podem ser incluídas outras informações relativas ao funcionamento e implementação do programa.

A secção de discussão visa efetuar uma análise do trabalho feito, argumentando de forma crítica as escolhas que foram feitas. Será necessário realizar uma análise de desempenho (e descrever a respetiva metodologia), descrever técnicas de modularidade e encapsulamento aplicadas e justicar a escolha de estruturas de dados. O grupo deverá fazer testes para cada *query* nas diferentes máquinas dos seus elementos e apresentar os resultados (e.g., do programa-testes), bem como

[2] Nota: O diagrama deve representar a arquitetura efetivamente implementada. **Não deve ser utilizada a imagem presente neste enunciado.**

o ambiente de testes usado (hardware, número de repetições, ...), de acordo com os requisitos do executável *programa-testes*. 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.

A secção de conclusão deverá resumir o que foi aprendido com o trabalho, voltar a destacar os aspetos que pensem terem sido mais importantes (e.g., lições aprendidas ou componentes particularmente bem/mal desenvolvidos) e ainda apontar possíveis melhorias futuras.

Note-se que a descrição de métodos de encapsulamento e da escolha de estruturas de dados só é esperada para o relatório da Fase 2.

A Simulação do ambiente de testes

Em casos muito raros, diferenças entre o ambiente de programação dos alunos e o ambiente da plataforma de testes podem resultar em problemas de compilação ou diferenças na execução do programa. Assim, será possível executarem o vosso programa num ambiente idêntico ao da plataforma de testes recorrendo às ferramentas descritas abaixo.

- 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
# Iniciar o container
docker start li3
```