Universidade do Minho
Escola de Engenharia

Trabalho Prático (Fases 1 e 2)

Laboratórios de Informática III

2024

Súmula 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: introdução de novos dados a considerar no projeto (álbuns e histórico).
- Secção 4.1: indicações relativas à documentação do código-fonte do projeto.
- Secção 5.1: introdução do modo de execução interativa (`programa-interativo`).
- Secção 5.2: introdução de três novas queries (Q4-6) e atualização da query 1.
- Secção 5.4: exemplo de utilização do programa de execução interativa.

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 modularização e encapsulamento ao desenvolvimento de projetos em C.

O tema do trabalho prático centra-se num sistema de streaming de música. Os alunos terão que explorar um conjunto de dados relativo a músicas, artistas, utilizadores e estatísticas de uso do sistema, 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 do seu histórico de commits**, não sendo aceites justificações para a inexistência desse registo explicações 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 **09/11/2024**. A apresentação presencial da mesma deverá decorrer entre 11/11/2024 e 15/11/2024. 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 e de erros de memória no Valgrind (Peso: 15%);
- Relatório (Peso: 5%).

A avaliação da correção das queries (interrogações) da Fase 1 bem como a ausência de problemas de memória 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.

Novo na fase 2!

Fase 2

A fase 2 irá ter um peso de 60% da nota final. Nesta fase o dataset irá ser incrementado através de novos módulos e novos campos nos módulos já existentes. Para além disso, o volume total de dados será também incrementado, bem como o número de queries que será executado na plataforma de testes. 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 de 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", 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 também efetuar uma análise do impacto da utilização dessas estruturas na performance das queries. Esta análise deverá constar do relatório final. Finalmente, para os grupos que decidam implementar o seu próprio recomendador para a query 5 (ver Secção 5.2), esta componente irá também contar para estes 2 valores, devendo a explicação desse módulo estar também presente no relatório.

Cálculo da avaliação final

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

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

Novo na fase 2!

Muito importante: os componentes de modularização 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. Note-se ainda que os valores para os critérios de avaliação são de caráter indicativo, ficando ao critério da equipa docente eventuais ajustes nos momentos de avaliação.

3 Repositórios e plataforma de testes

O 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 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-testes (depois do make)
```

Embora possam e devam ter os datasets de dados localmente, estes **ficheiros de dados não devem 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, e de maior ou menor dimensão, 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 da plataforma de testes `https://li3.di.uminho.pt/faq`. **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 pela plataforma.**

4 Aplicação de streaming de música

Como referido anteriormente, a aplicação a desenvolver contempla a análise dos dados relativos a um sistema de streaming de música. O sistema considera dados relativos a músicas, utilizadores, artistas, álbuns e histórico de utilização. Em seguida são apresentados os seus campos de dados e respetivas descrições.

- **Músicas** (*musics.csv*):
  - *id* – identificador único da música;
  - *title* – nome da música;
  - *artist_id* – lista de identificadores dos autores da música;

    Novo na fase 2!

    - *album_id* – identificador único do álbum ao qual a música pertence;
  - *duration* – tempo de duração;
  - *genre* – género da música;
  - *year* – ano de lançamento;
  - *lyrics* – letra da música.
- **Utilizadores** (*users.csv*):
  - *username* – identificador único do utilizador;
  - *email* – email de registo do utilizador;
  - *first_name* – primeiro nome do utilizador;
  - *last_name* – apelido do utilizador;
  - *birth_date* – data de nascimento;
  - *country* – país onde a conta do utilizador foi registada;
  - *subscription_type* – tipo de subscrição, i.e., normal ou premium;
  - *liked_musics_id* – lista de identificadores únicos das músicas gostadas pelo utilizador.
- **Artistas** (*artists.csv*):
  - *id* – identificador único do artista;
  - *name* – nome do artista;
  - *description* – detalhes do artista;
  - *recipe_per_stream* – dinheiro auferido de cada vez que uma das músicas do artista é reproduzida;
  - *id_constituent* – lista de identificadores únicos dos seus constituintes, no caso de se tratar de um artista coletivo. Este campo pode ser uma lista vazia.
  - *country* – nacionalidade do artista.
  - *type* – tipo de artista, i.e., individual ou grupo musical.

Novo na fase 2!

- **Álbuns** (*albums.csv*):
  - *id* – identificador único do álbum;
  - *title* – título do álbum;
  - *artists_id* – lista de identificadores únicos dos artistas que lançaram o álbum;
  - *year* – ano de lançamento;
  - *producers* – lista de produtores.
- **Histórico** (*history.csv*) – histórico de músicas ouvidas no sistema:
  - *id* – identificador único do registo;
  - *user_id* – identificador único do utilizador a que o registo se refere;
  - *music_id* – identificador único da música a que o registo se refere;
  - *timestamp* – data e hora em que a música foi ouvida pelo utilizador;
  - *duration* – tempo de duração da audição da música. E.g., um utilizador pode ter ouvido apenas 30 segundos de uma música;
  - *platform* – plataforma em que a música foi reproduzida. I.e., computador ou dispositivo móvel.

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 de 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 um exemplo consultar `https://fnch.users.sourceforge.net/doxygen_c.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 apresentados. Para isso, deverão carregá-los 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 modularização e encapsulamento é fundamental para a obtenção de aproveitamento neste trabalho prático**. Para além destes, também a ausência de leaks de memória, a qualidade de 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:

- Dados sem erros: musics.csv, users.csv, ...
- Dados com erros: musics.csv, users.csv, ...
- Queries de exemplo: input.txt
- Resultados corretos 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. Considere, 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.

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 convocado com os argumentos corretos. Note-se que o comando *make* por defeito deverá produzir o executável *programa-principal*, ficando ao critério do grupo o(s) comando(s) que irão dar origem aos restantes executáveis.

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., musics.csv, users.csv, etc). 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`", 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 U0071877
2 10 Finland
...
```

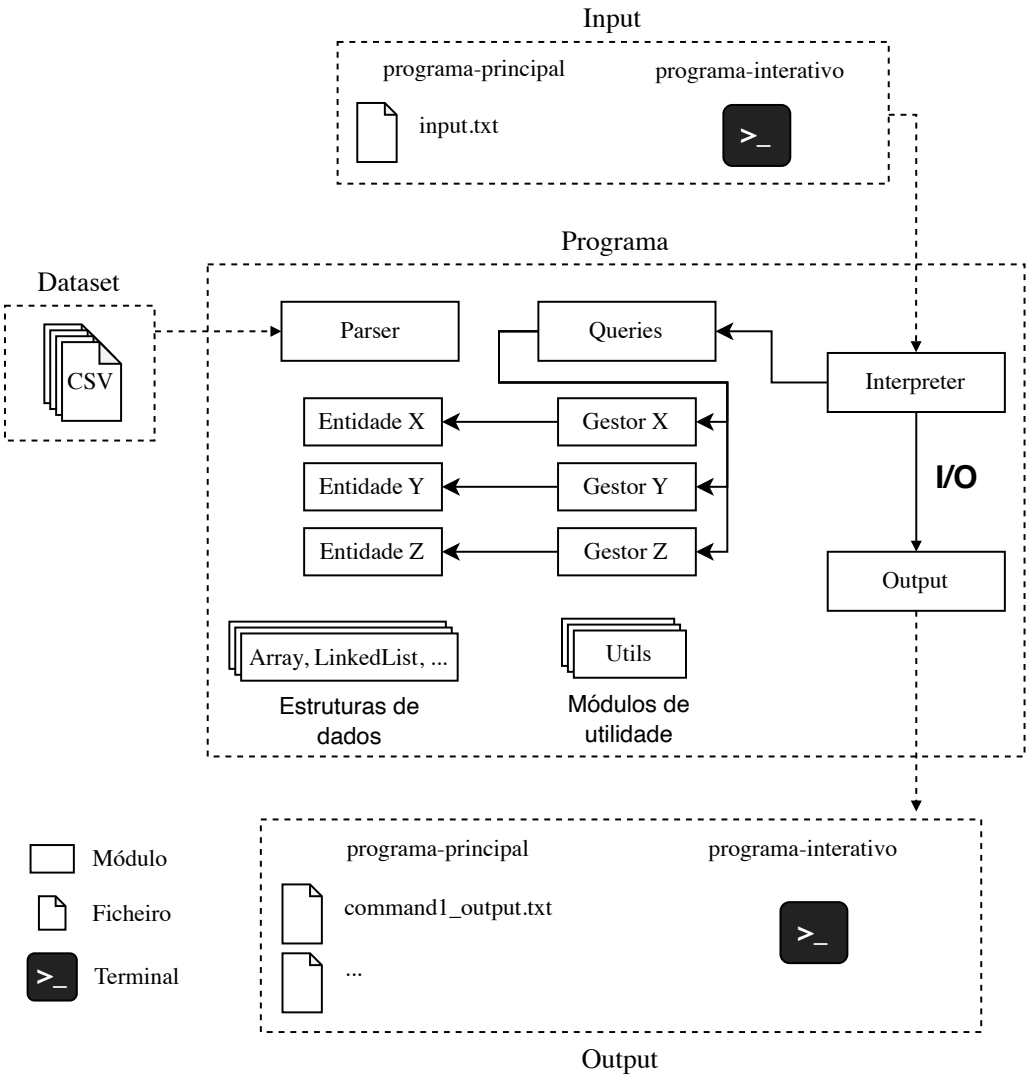

Figura 1: Arquitetura de referência para a aplicação a desenvolver.

Exemplo ficheiro de output "command1_output.txt":

```
martinezcraig@example.net;Barbara;Wilson;52;Belgium
```

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 ocurrências 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, deverão 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`:

```
#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 um utilizador, **devem assumir a data *2024/09/09* 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 defeito, os outputs deverão ser separados por ';', enquanto que o formato alternativo deverá utilizar o separador '='. Veja o exemplo abaixo:

Comando	
1 <ID>	*1S <ID>*
Output	
name;type;country;total_recipe	*name=type=country=total_recipe*

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

Alterado na fase 2!

Q1: Listar o resumo de um utilizador ou artista, consoante o identificador recebido por argumento.

A query recebe como argumento o identificador único de um utilizador ou artista, sendo garantido que não existem identificadores repetidos entre as diferentes entidades. Deverá ser retornada uma linha vazia caso o id não conste do sistema. Nos casos em que conste, a query deverá retornar as seguintes informações:

- Utilizador

  email;first_name;last_name;age;country

- Artista

  name;type;country;num_albums_individual;total_recipe

Comando
1 <ID>
Output
(ver acima)

O campo *num_albums_individual* de um artista corresponde ao número de álbuns em que o artista surge como autor enquanto artista individual.
A receita total de um artista depende da sua receita por reprodução (*recipe_per_stream*), auferindo esse valor de cada vez que uma das suas músicas é ouvida. No caso de artistas coletivos, a receita do artista pode ser calculada exclusivamente a partir do número de vezes que as suas músicas foram reproduzidas. No caso de artistas individuais, a sua receita irá também depender da receita de artistas coletivos em que esteja envolvido. O valor deverá ser **representado com duas casas decimais**. Considerem-se as seguinte fórmulas:

Funções auxiliares

$$\text{receita}_{\text{Artista}} = \text{reproducoes} \times \text{rate\_per\_stream}_{\text{artista}} \tag{1}$$

$$\text{receita}_{\text{Participacao}} = \sum_{i \in C} \frac{\text{reproducoes}_i \times \text{rate\_per\_stream}_i}{|\text{constituintes}_i|}, \tag{2}$$

$$C = \{\text{artistaColetivo} : \text{artistaIndividual} \in \text{artistaColetivo}\}$$

Cálculo de receitas

$$\text{receita}_{\text{artistaColetivo}} = \text{receita}_{\text{Artista}} \tag{3}$$

$$\text{receita}_{\text{artistaIndividual}} = \text{receita}_{\text{Artista}} + \text{receita}_{\text{Participacao}} \tag{4}$$

Q2: Quais são os top *N* artistas com maior discografia?

A query recebe como argumento o número de artistas que devem constar do output, podendo ainda receber (ou não) um filtro opcional, o filtro de país, sendo que quando presente só devem ser considerados artistas desse país. Deverá escrever para o ficheiro de resultado os campos que constam do exemplo abaixo. O tamanho da discografia dos artistas é calculado pela soma da duração das suas músicas. Artistas coletivos e individuais devem ser tratados de forma separada, ignorando relações que existam entre eles.

Em caso de empate, os artistas devem ser ordenados por ordem crescente de id (i.e., ids mais pequenos devem surgir primeiro).

Comando
2 <N> [country]
Output
name 1;type 1;discography duration 1;country 1
name 2;type 2;discography duration 2;country 2
...

Q3: Quais são os géneros de música mais populares numa determinada faixa etária?

A query recebe como argumento as idades mínima e máxima da faixa etária dos utilizadores a considerar. Deve produzir como output uma lista ordenada de géneros por ordem decrescente de popularidade e o número total de *likes* associados. Considere que a popularidade de um género é determinada a partir dos géneros das *liked musics* dos utilizadores que se encontram nessa faixa etária. Por exemplo, se músicas do género *A* tiverem 10 likes e músicas do género *B* apenas tiverem 8, considera-se o género *A* mais popular. Em caso de empate prevalece a ordem alfabética.

Comando
3 <min age> <max age>
Output
genre 1;total likes
genre 2;total likes
...

Novo na fase 2!

Q4: Qual é o artista que esteve no top 10 mais vezes?

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 tempo de reprodução efetivo, ou seja, o tempo de reprodução registado no histórico de atividade, e não o tempo de duração de uma música. 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 top10 dessa(s) semana(s) também deve ser considerado. Artistas coletivos e individuais devem ser tratados de forma separada, ignorando relações que existam entre eles. Em caso de empate, prevalece o artista com id mais baixo.

Comando
4 [begin_date end_date]
Output
name;type;count_top_10

Q5: Recomendação de utilizadores com gostos parecidos

Construa uma matriz em que cada linha corresponde a um utilizador e cada coluna corresponde a um género de música. O valor de cada célula na matriz deverá corresponder ao número de vezes que o utilizador ouviu músicas com esse género.
De seguida faça uso da biblioteca fornecida pela equipa docente para gerar um conjunto de utilizadores com gostos parecidos ao de um dado utilizador, utilizando a função que se apresenta abaixo. Consulte a interface da biblioteca (.h) para obter mais detalhes sobre a função.

```c
//recomendador.h
char **recomendaUtilizadores(char *idUtilizadorAlvo,
                             int **matrizClassificacaoMusicas,
                             char **idsUtilizadores, char ** nomesGeneros,
                             int numUtilizadores, int numGeneros,
                             int numRecomendacoes);
```

A query inclui dois argumentos: o id do utilizador para o qual deve ser gerada a lista de utilizadores semelhantes e o número de utilizadores a retornar. Para valorização extra, os alunos poderão tentar implementar a sua própria biblioteca de recomendações, com a mesma interface da biblioteca fornecida pelos docentes. Esta componente terá uma valorização de até 0.5 valores, e fará parte dos dois valores destinados ao componente "Estruturas/Performance/Recomendador", tal como descrito na Secção 2.

Comando
5 <username> <nr_utilizadores>
Output
username_1
username_2
username_3
...

Q6: Resumo anual para um utilizador

A query recebe dois argumentos: o id do utilizador para o qual queremos obter um resumo de estatísticas anuais; e o ano ao qual essas estatísticas se referem. Pode ainda ser passado um argumento opcional numérico, N, sendo que em tal caso devem-se imprimir em linhas consecutivas os N artistas mais ouvidos pelo utilizador nesse ano, em função do tempo de reprodução.
A análise deve incluir o tempo total de audição do utilizador (no formato *hh:mm:ss*), o número de músicas ouvidas (cada música deve ser contabilizada apenas uma vez), o artista mais ouvido (com desempate por id mais baixo), o dia em que reproduziu mais músicas (incluindo músicas repetidas, com desempate por data mais recente, em formato de data), o género de música mais ouvido (com desempate por ordem alfabética), o seu álbum favorito (com desempate por ordem alfabética) e a hora do dia em que costuma ouvir mais música (representado por dois dígitos, no intervalo [00-23], e com desempate por hora mais cedo no dia). À exceção do número de músicas ouvidas e do dia em que o utilizador reproduziu mais músicas, todos os campos deverão ser determinados com base em tempo acumulado de reprodução.
Para cada um dos N artistas deve apresentar o nome do artista, o número de músicas distintas ouvidas desse artista e o tempo total de audição. Caso o utilizador tenha ouvido menos do que N artistas, só deverão ser imprimidos os artistas que efetivamente ouviu.
Caso o utilizador não possua nenhum registo de atividade nesse ano, o *output* deverá ser vazio, ou seja, $\backslash n$.

Comando
6 <user_id> <year> [N]
Output
listening time;#musics;artist;day;genre;favorite_album;hour
[artista_preferido_1;#musicas;listening_time]
[artista_preferido_2;#musicas;listening_time]
[...]

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., $users\_errors.csv$), 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: $2023/1234$, $09/10$, 20230901, $09/01/2023$, $2023|09|01$, $abcd/09/01$, ...;
  - 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: $2023/01/52$, $2023/14/03$, ...;
  - As datas não poderão ser mais recentes do que a data atual;
- **Duração:**
  - O formato da duração ou tempo de reprodução de uma música segue a estrutura $hh:mm:ss$, onde h, m e s são números entre 0 e 9 (inclusive). Alguns exemplos de erros incluem: 000341, $3:31$, $00.03.41$, ...;
  - As horas deverão estar contidas entre 0 e 99 (inclusive), os minutos e segundos devem encontrar-se entre 0 e 59 (inclusive);
- **Email:**
  - O formato deverá ser sempre do tipo *username@domínio*, onde *username* corresponde a um conjunto de carateres do intervalo [a-z0-9], 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*, ...
- **Subscription Type:**
  - O campo *subscription_type* apenas poderá tomar os valores *normal* ou *premium*.

- **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.

Novo na fase 2!

- ***Platform*:**
  - O campo *platform* apenas poderá tomar os valores *mobile* ou *desktop*. Deverão ser permitidas diferentes combinações, tais como *MOBILE*, *DESKtop* ou *moBiLe*.
- ***Type*:**
  - O campo *type* do artista apenas poderá tomar os valores *individual* ou *group*. Deverão ser permitidas diferentes combinações, tais como *INdividual*, *GROUP* ou *INDIVIDUAL*.

Validação lógica

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

- **Utilizadores:**
  - O campo *liked_musics_id* de um utilizador, deverá conter apenas músicas existentes e válidas.
- **Músicas:**
  - O campo *artist_id* de uma música, deverá corresponder a um artista existente e válido.

Novo na fase 2!

- O campo *album_id* de uma música, deverá corresponder a um álbum existente e válido.

- **Artistas:**
  - O campo *id_constituent* de um artista individual não poderá ter elementos.

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
|                                       |- ...
|                                       |- users_errors.csv
|                                       |- musics_errors.csv
|                                       |- ...
|- dataset                          |- dataset
|   |- users.csv                        |- users.csv
|   |- musics.csv                       |- musics.csv
|   |- ...                              |- ...
|- dataset-errors                   |- dataset-errors
|   |- users.csv                        |- users.csv
|   |- musics.csv                       |- musics.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:
Introduza o caminho dos ficheiros de dados: <input do utilizador>
Dataset carregado...
Que query deseja executar?
Que query deseja executar? <input do utilizador>
Nome de utilizador:
Nome de utilizador: <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
    Descrepância na query 84: linha 10 de "resultados/command84_output.txt"
    Descrepância 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 [1], 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, descrever técnicas de modularização 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 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 apreendido 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.

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

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
```