Universidade do Minho
Escola de Engenharia

Laboratórios de Informática III

Relatório Fase 1

2024 - 2025

Grupo 10
A106837 - Filipa Cosquete Santos
A106915 - Andreia Alves Cardoso
A107382 - Cátia Alexandra Ribeiro da Eira

Conteúdo

1 Introdução

Este relatório aborda o processo de realização da primeira fase do projeto de grupo no âmbito da UC de Laboratórios de Informática III, no ano letivo de 2024/2025.

O objetivo principal desta fase era criar um programa em C para explorar um conjunto de dados relativos a diferentes entidades (músicas, artistas, e utilizadores) de uma plataforma de *streaming* de música, analisar e trabalhar toda essa informação da forma mais adequada, e utilizá-la para responder a um conjunto de *queries*.

Pretendia-se também o desenvolvimento das habilidades do grupo relativamente à utilização de ferramentas essenciais ao desenvolvimento de projetos deste tipo, como ferramentas de compilação (Makefile), depuração de erros (GDB), avaliação de desempenho, consumo de recursos (Valgrind), e gestão de repositórios colaborativos (GitHub).

2 Sistema

2.1 Arquitetura do projeto

Na nossa arquitetura atual, os dados começam por ser lidos dos ficheiros CSV que são fornecidos ao programa, cada linha enviada então para uma série de funções que extraem da linha a informação útil sobre a respetiva entidade, e a analisam de forma a determinar se se trata de uma entidade válida, de acordo com os requisitos do projeto. Caso tal se verifique, é introduzido um novo elemento na tabela *hash* da entidade correspondente. Caso contrário, a linha original é impressa no respetivo ficheiro de erros.

Depois de todos os dados serem avaliados, o programa efetua uma análise semelhante no ficheiro que contém as instruções a correr, discriminando em cada uma o tipo de *query*, e as informações essenciais à sua resposta. Após cada instrução corrida, é criado o respetivo ficheiro de *output*, que apresenta a resposta à *query* em questão. Mais à frente falaremos detalhadamente sobre o processo de resolução de cada tipo de *query*.

Para manter o código organizado, dividimo-lo em diferentes contextos:

- *datatypes*, que consiste nas *structs* das entidades e comandos, os respectivos *getters*, e funções de libertação de memória;
- *managers*, que inclui as definições das estruturas de dados escolhidas (neste caso, tabelas *hash*) e funções que permitem gerir e controlar os dados de forma a respeitar o encapsulamento dos mesmos;
- *input handling*, que consiste nas funções de *parsing*;
- *validation*, que consiste nas funções de validação dos dados;
- *queries*, que consiste nas funções de resolução dos diferentes tipos de *queries* propostas;
- *output handling*, que consiste nas funções de escrita dos *outputs* nos ficheiros devidos.

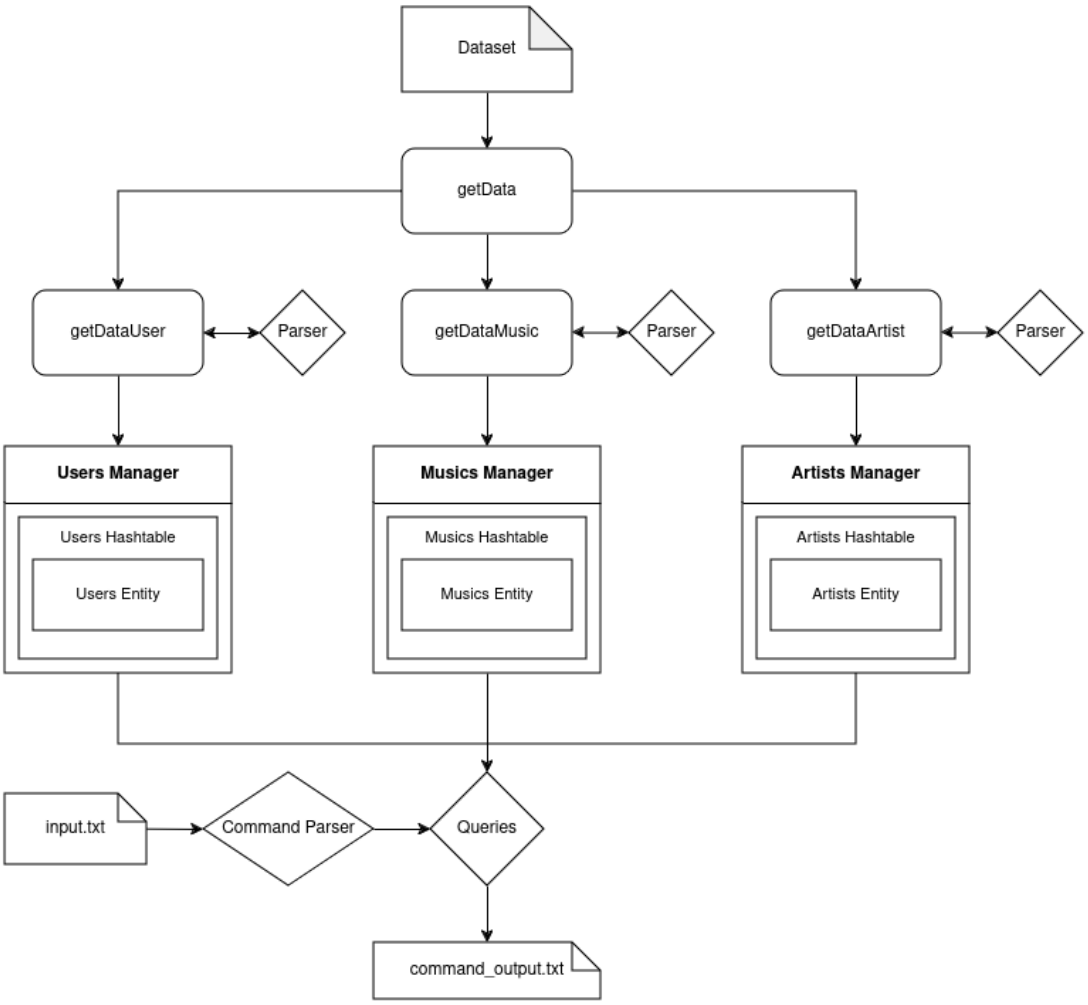


Diagrama simplificado da arquitetura do programa

2.2 Processos de resolução das *queries*

2.2.1 *Query* 1

Na *query* 1, temos como intenção expor informações sobre um determinado utilizador da plataforma. Para isso, o programa recebe o ID do utilizador a pesquisar, verifica se o mesmo se encontra na tabela *hash* dos utilizadores e, caso tal aconteça, adquire os dados sobre esse utilizador (como nome, e-mail, data de nascimento, país, *etc*.) e escreve-os no ficheiro de *output* devido. Caso o ID não corresponda a nenhum utilizador, o ficheiro de *output* fica vazio.

2.2.2 *Query* 2

Na *query* 2, o objetivo é identificar os artistas com maior discografia, com a opção de aplicar um filtro por país. Para isto, usamos uma tabela *hash* auxiliar onde iremos guardar a duração total das músicas, usando o ID dos artistas como chave.

Começamos por percorrer a tabela *hash* das músicas e, para cada uma, somamos a sua duração ao total associado ao ID do artista correspondente, na tabela auxiliar. Se o filtro por país estiver a ser utilizado, o artista apenas é adicionado se se verificar que pertence ao país pretendido.

No fim deste processo, convertemos a tabela *hash* auxiliar num *array* de tuplos, cada um composto por um ID de artista e o valor calculado, que é então a duração total da sua discografia. Ordenamos o *array* por ordem decrescente de durações e, por último, escrevemos o resultado no respetivo ficheiro de *output*, não ultrapassando o número limite de artistas pedido como argumento.

2.2.3 *Query* 3

Na *query* 3 temos como objetivo devolver o número total de *likes* de cada género de músicas numa determinada faixa etária. Para isso o programa recebe através da instrução a idade mínima e a idade máxima da faixa etária pretendida.

O programa itera sobre a tabela *hash* dos utilizadores verificando a sua idade. Se a idade do utilizador estiver na faixa etária pretendida o programa itera sobre a lista de músicas com *like* do utilizador, realizando assim uma contagem dos géneros das músicas. Essa contagem é feita a partir de um *array* de tuplos que contém o género e o número de *likes*. Por cada música com *like*, o programa verifica o seu género e adiciona 1 ao respetivo número de *likes*.

Depois de percorrer todos os utilizadores, o programa ordena o *array* de tuplos de forma decrescente do número de *likes* de cada género.

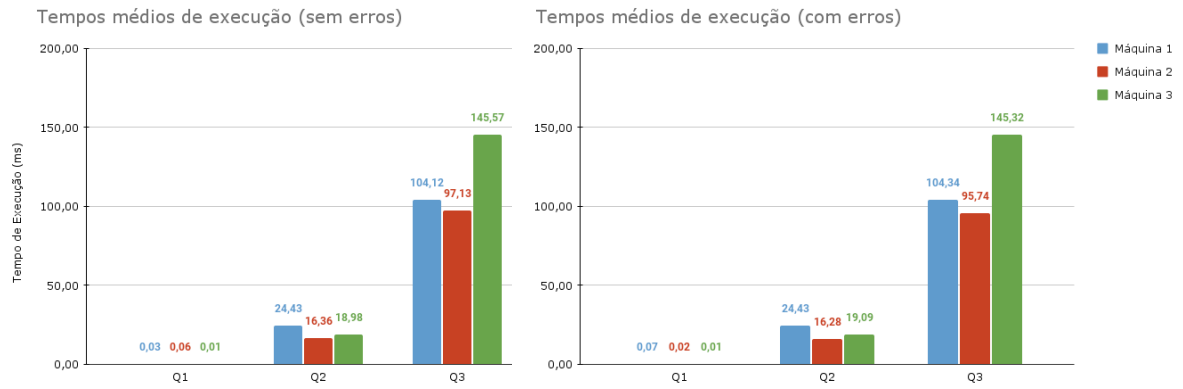
Para utilizadores que não estejam na faixa etária pretendida o programa não faz qualquer operação, seguindo automaticamente para o próximo utilizador. Para faixas etárias que não tenham qualquer *like* o programa devolve um documento vazio.

3 Discussão

Nesta secção apresentamos uma análise do desempenho do programa nas máquinas dos três elementos do grupo, durante a execução do modo de testes. Esta análise é acompanhada por uma tabela que contém os tempos médios de execução, e que indica ainda especificações do *hardware* das três máquinas, de modo a auxiliar na compreensão das diferenças no desempenho. Apresentamos ainda dois gráficos para auxílio visual na diferença de tempos médios de execução por *query* entre máquinas, tanto no *dataset* sem erros como no *dataset* com erros.

Para assegurarmos uma maior confiabilidade dos resultados, foram realizadas 5 execuções de aquecimento, seguidas de 10 medições para cada conjunto de dados. Os valores apresentados na tabela são as médias diretas dos valores obtidos.

MÁQUINA	1	2	3
OS	Ubuntu Linux	Ubuntu Linux	Ubuntu Linux
CPU	Intel® Core™ i7-1065G7 CPU @ 1.30GHz × 8	13th Gen Intel® Core™ i7-1355U × 12	Intel® Core™ i7-9850H CPU @ 2.60GHz × 12
RAM (GiB)	12	16	16
Cores/Threads	4/8	10/12	6/12
Disco (GB)	512.1	512.1	512.1
		Dataset Sem Erros	
Tempo Médio Execução (s)	5.41	4.57	6.35
Tempo Médio Q1 (ms)	0.03	0.06	0.01
Tempo Médio Q2 (ms)	24.43	16.36	18.98
Tempo Médio Q3 (ms)	104.12	97.13	145.57
		Dataset Com Erros	
Tempo Médio Execução (s)	5.71	4.79	6.61
Tempo Médio Q1 (ms)	0.07	0.02	0.01
Tempo Médio Q2 (ms)	24.43	16.28	19.09
Tempo Médio Q3 (ms)	104.34	95.74	145.32



Por observação dos gráficos e da tabela, podemos concluir que não existem diferenças significativas no desempenho entre *datasets*. Entre máquinas, é considerável a discrepância entre a máquina 3 e as restantes, nomeadamente no que diz respeito à *query* 3.

Esta desigualdade pode ser atribuída às características de *hardware* das máquinas, como, por exemplo, o facto de a máquina 3 apresentar um processador ligeiramente mais antigo do que os restantes. Se compararmos os nossos tempos médios de execução com o tempo médio apresentado na plataforma de testes disponibilizada (cerca de 20 segundos), verificamos que todas as máquinas do grupo são substancialmente mais rápidas, diferença que poderá ser justificada pelo aumento de cores/threads.

Visto o claro aumento de tempo de execução da *query* 3 relativamente às outras, pensamos que seria interessante tentar arranjar uma forma de não ter de percorrer toda a tabela *hash* a confirmar a idade dos utilizadores. Uma solução seria criar uma tabela *hash* auxiliar, onde as chaves seriam as idades dos utilizadores, e cada chave levaria a um *array* de inteiros com os IDs dos utilizadores com a respetiva idade. Dessa forma não seria necessário percorrer a tabela *hash* dos utilizadores na sua totalidade, e iríamos aceder apenas e diretamente aos utilizadores das idades que pretendemos. No entanto, não chegamos a implementar esta ideia, por um lado por receio de misturar módulos e, por outro, porque queríamos evitar realizar operações específicas de certas *queries* durante o tratamento dos dados.

4 Conclusão

Nesta primeira fase evoluímos no sentido de melhor trabalhar com ficheiros, I/O, e estruturas de dados, como as *hashtables*. A maior dificuldade sentida foi na implementação da modularidade e do encapsulamento, visto que foi a primeira vez que encarámos os conceitos, o que os tornou mais complexos de entender ao início. Uma das maiores preocupações foi manter uma boa gestão da memória utilizada e do tempo dispensado, com visão a possíveis desafios da segunda fase.