LABORATÓRIOS DE INFORMÁTICA III

RELATÓRIO DO TRABALHO PRÁTICO FASE 1

Realizado por:
Francisco Ribeiro Martins - A106902 - franciscormartins
Hugo Ferreira Soares - A107293 - yhugosoares
Marco Rafael Vieira Sèvegrand - A106807 - marcosevegrand

ÍNDICE

1. INTRODUÇÃO

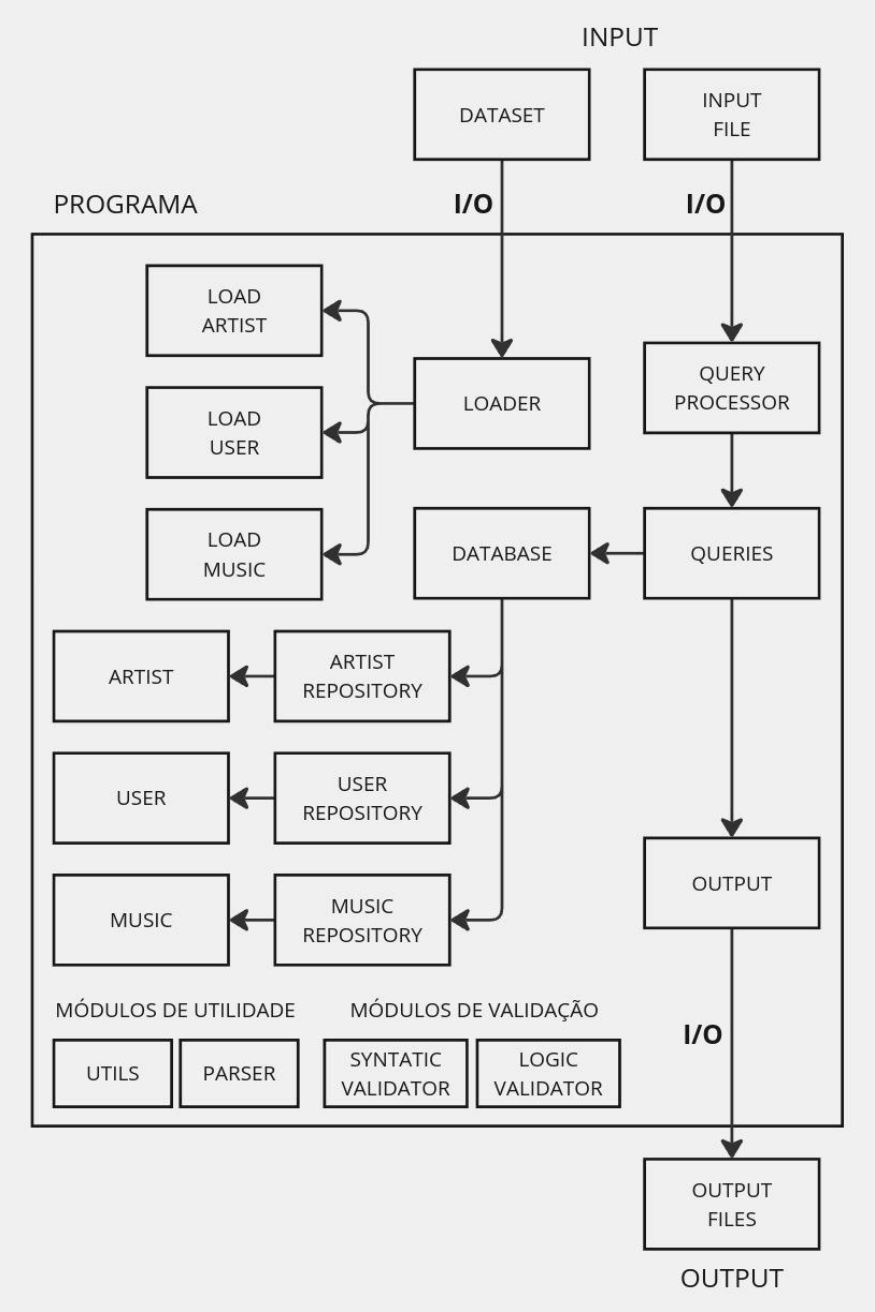
Este relatório visa apresentar o desenvolvimento, a estrutura e a implementação de um sistema de gestão e processamento de dados de artistas, músicas e utilizadores.
Serão detalhados os objetivos do trabalho, o funcionamento do programa, a organização do sistema, as justificativas das escolhas dos módulos e uma análise de desempenho dos testes realizados.

1.1 Objetivos do Trabalho Prático

O trabalho prático tem como objetivo o desenvolvimento de um sistema de streaming de música, utilizando a linguagem de programação C, com particular foco na construção de um código modular, eficiente e encapsulado. O sistema deve ser capaz de carregar, processar e manipular dados relativos a músicas, artistas e utilizadores, permitindo responder a consultas (queries) específicas sobre esses dados. Com a realização deste projeto, pretende-se consolidar competências fundamentais em C, incluindo o uso de conceitos de modularização e encapsulamento, e a aplicação prática de ferramentas de análise de memória e debugging, como o valgrind e o gdb.

2. SISTEMA

2.1 Arquitetura do Sistema (Esquema)



2.2. Módulos

2.2.1 Descrição dos Módulos

O projeto está organizado em módulos lógicos de forma a separar as funcionalidades do programa. Abaixo está uma descrição dos módulos:

Entidades
artist, music, user: Contêm as definições da estrutura de dados Artist, Music e User, respetivamente, e funções relacionadas à criação, cópia, libertação e acesso aos seus campos.

Repositórios
repository_artist, repository_music e repository_user: Módulos de repositórios que fornecem estruturas de dados baseadas em GHashTable para armazenar e gerenciar as estruturas artist, music e user.

Carregamento de Dados
loader, load_artist, load_music e load_user: Responsáveis pelo carregamento dos dados dos ficheiros CSV para os repositórios correspondentes.

Base de Dados
database: Módulo que encapsula as estruturas de repositórios e centraliza a inicialização e libertação de recursos.

Validação de Dados
logic_validation e syntatic_validation: Módulos de validação que verificam a coerência e a sintaxe dos dados fornecidos.

Utilidades
parser: Contém funções para manipulação de strings e fazem parsing de dados para listas.

utils: Inclui macros para a verificação alocação de memória, constantes de tempo e tamanho de buffer e ainda funções auxiliares para conversão de tempo.

Queries
process_queries: Módulo responsável por processar e executar as consultas com base nos dados carregados.

query1 query2, query3: Módulos que implementam diferentes tipos de consultas: a query1 procura um utilizador a partir de um ID e lista alguns dos seus atributos; a query2 determina e lista os artistas com a maior discografia em duração podendo ser especificado o país; a query3 determina e lista os gêneros musicais por popularidade dentro de uma faixa etária.

I/O
output: Módulo para manipulação de saídas, incluindo a criação e escrita de matrizes de output em ficheiros.

2.2.2 Justificação dos Módulos

Modularidade: Cada módulo tem uma responsabilidade bem definida, o que facilita a compreensão e a manutenção. Por exemplo:

O módulo *artist* gerencia apenas a lógica relacionada aos artistas, permitindo alterações na estrutura Artist sem impactar o *music* ou *user*. Isto reduz a complexidade do código e torna mais intuitivo fazer manutenções ou expansões do código.

O módulo *process_queries* coordena a execução das consultas, enquanto *query1*, *query2* e *query3* implementam lógicas específicas para cada tipo de consulta, permitindo que novas funcionalidades de consulta sejam adicionadas sem a necessidade de alterar a lógica de processamento.

Reutilização de Código: Funções genéricas e utilitárias estão localizadas no módulo *utils*, permitindo o seu reaproveitamento em vários módulos. Por exemplo, macros como CHECK_ALLOC e funções de parsing são amplamente usadas nos módulos de validação e carregamento de dados.

Testabilidade: Cada módulo pode ser testado isoladamente para garantir que suas funcionalidades estejam corretas. Por exemplo, o *syntatic_validation* pode ser testado independentemente de outros módulos.

Manutenção: Mudanças num módulo (como melhorias na estrutura de User) não impactam de forma significativa outros módulos. Isto significa que aprimoramentos ou correções num módulo específico não requerem revisões em cascata em outros módulos, minimizando a propagação de erros e aumentando a produtividade na manutenção do projeto.

Escalabilidade: A estrutura modular permite adicionar novos módulos, como novas funções de consulta ou repositórios, com impacto mínimo no código existente. Por exemplo, a adição de uma nova query pode ser feita sem modificar os módulos de repositório ou o módulo *database*. Isto possibilita a expansão do projeto para atender a novos requisitos sem comprometer o código existente.

2.3 Funcionamento do Programa

O programa foi projetado para gerir e consultar dados sobre artistas, músicas e utilizadores de forma eficiente e organizada. Começa com o carregamento de dados a partir de ficheiros CSV que contêm informações detalhadas sobre cada entidade. Estes arquivos são lidos e os dados são validados para garantir que estejam em conformidade com o formato esperado.

O programa verifica se as informações estão no formato correto e se cumprem os critérios lógicos, como formatos apropriados de datas e durações e a existência de IDs válidos. Esta etapa assegura que apenas dados consistentes sejam processados nas consultas. Entradas validadas são armazenadas em repositórios compostos por HashTables, que

facilitam um acesso rápido e estruturado aos dados. Entradas com erros são ignoradas e escritas num ficheiro.
O programa depois lê um ficheiro com consultas (queries) a realizar sobre os dados como obtenção de informações sobre um utilizador específico, a listagem de artistas com as maiores discografias e a classificação de géneros musicais por popularidade numa determinada faixa etária. Os resultados das consultas são organizados e escritos em ficheiros de saída.

A execução do programa é começada por um módulo que inicializa a base de dados, carrega as informações e chama as funções responsáveis pelo processamento das consultas. No final da execução, o programa assegura a libertação adequada dos recursos, garantindo que a memória utilizada seja devolvida ao sistema.

3. ANÁLISE DE DESEMPENHO

3.1 Ambiente de Testes

Os testes foram realizados em 3 ambientes:

Ambiente 1
Processador: AMD Ryzen 7 4800H
Memória: 16GB (DDR4)
Disco: 512GB SDD
Placa Gráfica: NVIDIA GeForce GTX 1650
Sistema Operativo: Kubuntu 24.04 LTS

Ambiente 2
Processador: AMD Ryzen 5 2600
Memória: 32GB (DDR4)
Disco: 256GB SDD M.2
Placa Gráfica: NVIDIA GeForce GTX 1060 (6GB)
Sistema Operativo: Windows - WSL 2.0

Ambiente 3
Processador: Intel Core i7-12700H
Memória: 16 GB (DDR4)
Disco: 1 TB SSD
Placa Gráfica: NVIDIA GeForce RTX 3070 Ti
Sistema Operativo: Ubuntu 22.04.3 LTS

3.2 Casos de Testes e Cobertura

Para analisar a desempenho do programa foram elaborados alguns casos-teste tendo em conta diversas métricas. Foram testados 2 datasets, com e sem erros, e dois ficheiros com queries, ordenadas por tipo e desordenadas, tendo em consideração que deveriam ser testados para cada query o caso de não serem encontrados quaisquer resultados para a procura solicitada e o caso de ser fornecida uma query de tipo não implementado.

Casos:

Caso 1.1: Queries Ordenadas / Dataset com Erros
Caso 1.2: Queries Ordenadas / Dataset sem Erros

Caso 2.1: Queries Desordenadas / Dataset com Erros
Caso 2.2: Queries Desordenadas / Dataset sem Erros

3.3 Resultados dos Testes

Para cada caso foram executados 3 testes sendo a média dos resultados apresentada.
(Consultar tabelas de resultados em 5. Anexos)

3.4 Discussão dos Resultados e Identificação de Limitações

Os resultados demonstraram um desempenho consistente entre os diferentes casos de teste, com algumas variações no tempo de execução das queries. Observou-se que, embora a execução média das queries varie de forma ligeira, o uso de memória se mantém estável em todos os cenários. A análise evidenciou que as queries de tipo não implementado foram corretamente identificadas e que o tempo de resposta para buscas sem resultados foi adequado, garantindo uma execução robusta.

Limitações identificadas incluem a necessidade de melhorias na leitura e tratamento do dataset para obter uma redução no tempo de carregamento de dados. É evidente também a necessidade de diminuir o tempo médio de execução da query 3 que apresenta um impacto significativo na performance do programa. Ajustes na otimização de memória também poderiam ser considerados para garantir a capacidade para lidar com datasets maiores.
(Sugestões neste sentido são apresentadas no capítulo 4.1 Propostas de Melhoria)

5. CONCLUSÃO

5.1 Propostas de Melhoria

Para a fase 2 do projeto, pretendemos implementar diversas melhorias que visam aumentar a eficiência e a modularidade do sistema. Primeiramente, uma das prioridades será aprimorar a modularidade do código, reorganizando módulos existentes com especial foco nos módulos de carregamento de dados. Esta melhoria permitirá uma separação mais clara das responsabilidades, facilitando a manutenção e expansão futura do projeto.

Para além disso, introduziremos uma etapa de pré-processamento dos dados. Esta etapa adicional de processamento prévio tem como objetivo diminuir o tempo médio de execução das queries, particularmente da query 3, que demonstrou um tempo de resposta mais elevado nos testes. Com esta abordagem, espera-se uma otimização significativa na performance geral do sistema.

Outra melhoria proposta é a implementação de uma string pool, que permitirá uma melhor gestão da memória ao evitar a duplicação desnecessária de strings. Esta técnica irá reduzir o uso de RAM e contribuirá para uma execução mais eficiente do programa, especialmente em cenários com grandes volumes de dados.

Por fim, a implementação de uma cache será uma adição importante para armazenar os resultados de queries comuns. Esse mecanismo de armazenamento temporário possibilitará a reutilização de resultados em consultas subsequentes, diminuindo a carga de processamento e melhorando o tempo de resposta do sistema.

Essas melhorias propostas visam tornar o sistema mais robusto, eficiente e escalável, garantindo um desempenho otimizado e uma experiência de uso mais fluida.

5.2 Reflexão sobre o Aprendizado e Aplicação dos Conhecimentos

Este projeto foi uma ótima oportunidade para juntar o que aprendemos sobre programação em C e desenvolvimento de sistemas modulares. Durante a implementação, enfrentamos desafios que nos ajudaram a aplicar conceitos de organização de código, alocação de memória e otimização de performance.

Aprendemos bastante sobre como usar bibliotecas como a GLib para trabalhar com tabelas de hash e outras estruturas de dados, o que reforçou nosso conhecimento em como criar soluções eficientes.

Dividir o projeto em módulos mostrou a importância de ter uma arquitetura bem pensada, facilitando a manutenção e expansão do sistema.

Outro ponto importante foi a experiência com ferramentas de debugging e análise de performance, como o Valgrind, que nos ajudou a encontrar leaks de memória e outros problemas, consolidando boas práticas de programação.

Por fim, este trabalho melhorou a nossa capacidade de trabalhar em equipe, dividir tarefas e colaborar em um ambiente compartilhado, destacando a importância da comunicação e da documentação para o sucesso de um projeto.

Em suma, o projeto foi um aprendizado valioso que nos preparou para desafios maiores e mostrou a aplicação prática do que aprendemos no curso.

5. ANEXOS

5.1 Tabelas de Resultados dos Testes

5.1.1 Tabela de Resultados no Ambiente 1

	Casos			
	1.1	1.2	2.1	2.3
Tempo de carregamento dos Dados	3.12 s	3.25 s	3.71 s	3.51 s
Tempo de Execução Médio Query 1	0.07 ms	0.08 ms	0.05 ms	0.05 ms
Tempo de Execução Médio Query 2	72.63 ms	74.90 ms	72.30 ms	75.21 ms
Tempo de Execução Médio Query 3	769.10 ms	772.86 ms	770.89 ms	758.49 ms
Tempo de Execução Total	24.16 s	24.45 s	24.79 s	24.35 s
Uso de memória	761.04 MB	761.83 MB	760.88 MB	761.12MB

5.1.2 Tabela de Resultados no Ambiente 2

	Casos			
	1.1	1.2	2.1	2.3
Tempo de carregamento dos Dados	4.79 s	4.35 s	4.38 s	4.41 s
Tempo de Execução Médio Query 1	0.06 ms	0.07 ms	0.05 ms	0.08 ms
Tempo de Execução Médio Query 2	96.39 ms	103.87 ms	93.69 ms	104.25 ms
Tempo de Execução Médio Query 3	903.21 ms	964.19 ms	878.74 ms	935.19 ms
Tempo de Execução Total	29.79 s	31.05 s	28.69 s	30.40 s
Uso de memória	760.99 MB	760.99 MB	761.02 MB	760.80 MB

5.1.3 Tabela de Resultados no Ambiente 3

	Casos			
	1.1	1.2	2.1	2.3
Tempo de carregamento dos Dados	2.58 s	1.97 s	2.45 s	1.99 s
Tempo de Execução Médio Query 1	0.02 ms	0.03 ms	0.02 ms	0.01 ms
Tempo de Execução Médio Query 2	35.33 ms	43.39 ms	35.81 ms	35.60 ms
Tempo de Execução Médio Query 3	470.63 ms	466.51 ms	468.35 ms	467.43 ms
Tempo de Execução Total	15.23 s	14.72 s	15.05 s	14.57 s
Uso de memória	798.44 MB	798.44 MB	798.59 MB	798.28 MB