Introdução

O trabalho prático desenvolvido no âmbito desta UC tem como objetivo principal o nosso desenvolvimento como programadores através da implementação de um projeto estruturado em C, com especial destaque na modularização e encapsulamento. Para além disso, proporciona-nos uma experiência prática no desenvolvimento colaborativo de projetos, com recurso a ferramentas como o valgrind e o gdb.

Nesta segunda fase do projeto, a complexidade aumentou com a introdução de novos dados, relativos a álbuns e histórico, permitindo a exploração de um sistema de streaming de música mais completo e robusto. O trabalho envolveu a implementação de queries adicionais, um modo de execução interativa e a validação dos dados de entrada, elementos que acrescentaram novas camadas de complexidade ao sistema.

Entre as características mais relevantes do programa desenvolvido destacam-se:

- Eficiência no tratamento de dados: Foram utilizadas estruturas de dados adequadas para lidar com o volume crescente de informações e permitir respostas relativamente rápidas às queries.
- Interface interativa: Um menu intuitivo foi concebido para facilitar a interação com o sistema, promovendo uma experiência de uso amigável.
- Validação rigorosa de dados: Implementamos mecanismos para assegurar a integridade dos dados, separando de forma clara as entradas válidas e inválidas.
- Foco em qualidade de código: Foi dada atenção à documentação e à clareza do código, de forma a garantir a manutenibilidade e extensibilidade do sistema.

Com estes desenvolvimentos, procuramos não apenas responder aos requisitos estabelecidos, mas também otimizar o desempenho e a usabilidade do sistema, criando uma solução prática e eficaz para o processamento e análise de dados de streaming de música.

Sistema

O sistema desenvolvido neste trabalho prático foi concebido para responder aos desafios colocados pela implementação de um sistema de streaming de música eficiente e modular. Esta secção apresenta a arquitetura do sistema, descrevendo os principais módulos e justificando as escolhas realizadas. O objetivo foi garantir a clareza, a eficiência e a escalabilidade do programa, assegurando que cada módulo desempenha uma função específica de forma independente.

Arquitetura do Sistema

A arquitetura do sistema é composta por vários módulos interligados, cada um com responsabilidades bem definidas. A divisão modular permitiu uma organização clara do código, facilitando a implementação, a depuração e a manutenção. A seguir, descrevemos os principais módulos e a sua função:

1. **Módulo de Entrada de Dados (Input)**
    - **Descrição**: Este módulo é responsável por ler os dados provenientes dos ficheiros CSV fornecidos (como o musics.csv, users.csv, ...).
    - **Justificação**: A separação do input em um módulo próprio permite centralizar a validação sintática e lógica dos dados, promovendo uma reutilização eficiente em diferentes partes do sistema.
2. **Módulo de Validação**
    - **Descrição**: Encapsula todas as operações de validação, tanto sintáticas quanto lógicas, assegurando a integridade dos dados utilizados.
    - **Justificação**: Centralizar a validação simplifica a deteção de erros e permite isolar esta funcionalidade de outras partes do sistema.
3. **Módulo de Estruturas de Dados**
    - **Descrição**: Implementa as estruturas de dados necessárias para armazenar e manipular as informações, como hashtables, listas ligadas e arrays dinâmicos.
    - **Justificação**: Escolher estruturas adequadas ao tipo de queries e ao volume de dados contribui para otimizar o desempenho do sistema.
4. **Módulo de Gestores**
    - **Descrição**: Este módulo é responsável por gerir os dados validados e armazená-los nas estruturas adequadas, garantindo um acesso rápido e eficiente.
    - **Justificação**: Separar a gestão dos dados permite uma organização clara, mantendo as operações de armazenamento e manipulação independentes das validações e do input.

5. **Módulo de Execução de Queries**
    - **Descrição**: Contém as funções responsáveis por executar as diferentes queries solicitadas pelo utilizador ou pela plataforma de testes.
    - **Justificação**: Centralizar as operações das queries neste módulo garante que a lógica associada a cada uma está isolada e fácil de ajustar.
6. **Módulo de Interface Interativa**
    - **Descrição**: Implementa o menu interativo que permite ao utilizador explorar os dados do sistema de forma amigável.
    - **Justificação**: Uma interface interativa melhora a usabilidade do sistema e oferece uma alternativa prática à execução de queries pré-definidas.
7. **Módulo de Output**
    - **Descrição**: Responsável pela formatação e gravação dos resultados das queries em ficheiros de saída específicos.
    - **Justificação**: Este módulo permite manter a lógica de output separada das operações de processamento, garantindo uma apresentação consistente dos resultados.
8. **Módulo do Programa de Testes**
    - **Descrição**: Desenvolve testes automatizados para validar a funcionalidade e o desempenho das queries e do sistema como um todo.
    - **Justificação**: Automatizar os testes facilita a identificação de erros e possibilita a análise sistemática de desempenho e robustez do programa.
9. **Módulo Principal**
    - **Descrição**: Controla o fluxo geral do programa e integra os módulos anteriores para executar as operações conforme especificado.
    - **Justificação**: Este módulo funciona como um ponto central que coordena as ações, assegurando que os diferentes componentes do sistema trabalham de forma coesa.

Funcionamento e Implementação

A execução do programa começa com o módulo principal, que inicia o carregamento dos dados através do Módulo de Entrada de Dados e encaminha os mesmos para o Módulo de Validação. Após a validação, os dados são geridos pelo Módulo de Gestores, que os organiza e armazena nas estruturas apropriadas – em hashtables (estruturas estas que permitem uma inserção e procura muito rápidas).

Quando uma query é solicitada, seja no modo interativo ou através de um ficheiro, o gestor das Queries chama a devida query e esta processa os dados e retorna os resultados formatados através do Módulo de Output.

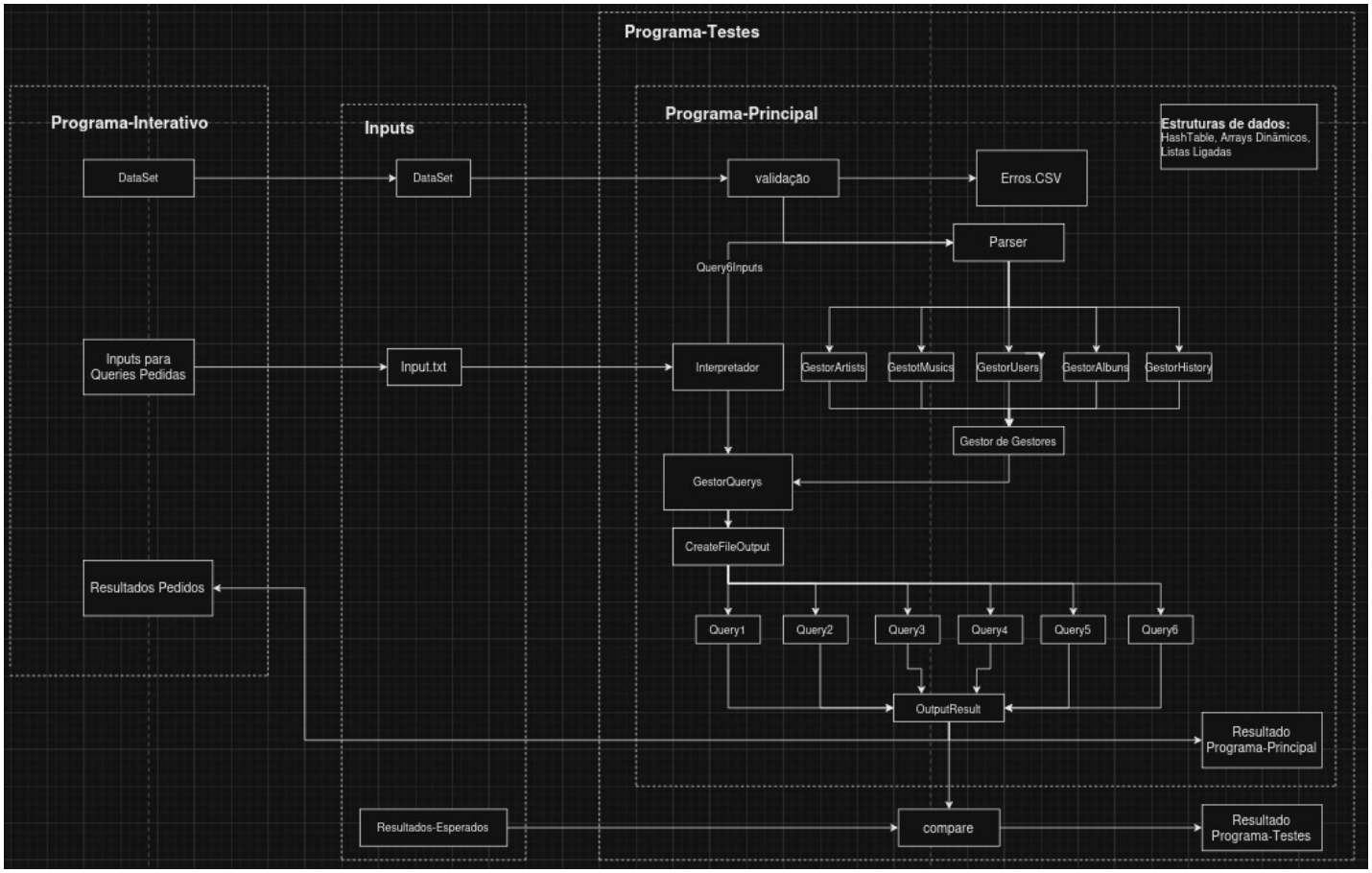
O Módulo do Programa de Testes desempenha um papel crucial na validação contínua do sistema. Este avalia o funcionamento correto das queries, mede o desempenho em termos de

tempo e memória e identifica discrepâncias entre os resultados esperados e os obtidos. Este módulo garante que o sistema mantém um comportamento previsível mesmo em cenários de grande volume de dados ou complexidade.

Adicionalmente, o sistema foi projetado para detetar e lidar com entradas inválidas, armazenando estas em ficheiros de erro apropriados. Este mecanismo contribui para a robustez e permite uma análise posterior.

Diagrama de Arquitetura

O diagrama a seguir ilustra a organização modular do sistema, destacando a interação entre os componentes:



Esta arquitetura foi fundamental para assegurar que o sistema fosse escalável, eficiente e fácil de manter. Cada módulo foi implementado com uma preocupação específica em termos de desempenho e clareza, alinhando-se aos objetivos do projeto.

Discussão

Análise Crítica das Escolhas Realizadas

No desenvolvimento deste projeto, a utilização de hashtables como estrutura de dados principal foi uma escolha fundamental para garantir a eficiência do sistema. Esta decisão permitiu um acesso rápido aos dados armazenados, com operações de inserção e pesquisa em tempo quase constante, independentemente do tamanho do dataset. A implementação de arrays dinâmicos e listas ligadas para suportar a execução das queries complementou a funcionalidade do sistema, equilibrando simplicidade e desempenho nas tarefas específicas exigidas.

Justificativa para o Uso de Hashtables

- **Eficiência**: A escolha das hashtables minimizou o tempo de procura pelos identificadores únicos (username para os users, id para os artists, ...), permitindo respostas rápidas em queries que exigem acesso direto.
- **Escalabilidade**: Com o crescimento do dataset na Fase 2, a capacidade de lidar com grandes volumes de dados foi essencial.
- **Colisões**: Não há colisões devido aos identificadores serem únicos.

Uso de Arrays Dinâmicos e Listas Ligadas para Queries

- **Arrays Dinâmicos**: Utilizados na query 3, 4 e 6. Útil porque facilmente se acede ao índice e gasta menos memória que uma hashtable. Utilizamos, por exemplo, na query 6 para aceder ao ano do resumo do user que queríamos saber, sendo o índice o ano atual – ano resumo. A facilidade de redimensionamento dinâmico permitiu com que o código suportasse os dados que forem precisos (fazendo com que não se tornasse “hard code”).
- **Listas Ligadas**: Implementadas nas queries 2, 3 e 6. Útil para iterar sobre conjuntos de dados de tamanho variável, como os géneros musicais mais populares no caso da query 3. A sua flexibilidade eliminou a necessidade de redimensionamento e otimizou operações de inserção e remoção.

Técnicas de Modularização

A modularização foi um pilar essencial na organização do projeto. Cada módulo foi projetado para desempenhar uma função específica. Módulos principais:

- **Módulo de Gestores**: Centralizou o armazenamento e gestão de dados usando hashtables. O seu encapsulamento garantiu que outras partes do sistema não precisassem manipular diretamente as estruturas internas.
- **Módulo de Validação**: Foi isolado para verificar a integridade dos dados antes de armazená-los. Isso simplificou o fluxo do programa e evitou inconsistências.

- **Módulo de Queries**: Permitiu que as operações de processamento de dados fossem separadas das restantes, garantindo clareza e manutenção.

Importância do Encapsulamento

O encapsulamento não apenas protegeu os dados internos de acessos indevidos, mas também melhorou significativamente a manutenção do código. Por exemplo:

- **Interação entre Módulos**: Os módulos comunicaram-se através de funções bem definidas, reduzindo dependências e minimizando o risco de introduzir erros ao modificar um componente.
- **Facilidade de Testes**: O encapsulamento permitiu testar cada módulo isoladamente, garantindo que erros pudessem ser identificados e corrigidos rapidamente.
- **Flexibilidade**: Caso fosse necessário alterar a implementação interna de uma estrutura, as modificações foram feitas sem impacto nas outras partes do sistema.

Análise de Desempenho

Os testes foram conduzidos para avaliar o desempenho do sistema em termos de tempo de execução e uso de memória. As métricas foram coletadas usando as nossas 3 máquinas com hardwares variados:

- **Configurações**:
    - **Máquina A**: Asus M515UA, 16 GB RAM, proc. AMD Ryzen 7;
    - **Máquina B**: Asus Vivobook, 16 GB RAM, proc. AMD Ryzen 7;
    - **Máquina C**: Asus M515DA, 8 GB RAM, proc. AMD Ryzen 5;
- **Método**: Cada query foi executada 5 vezes, e o tempo médio para cada máquina foi registado.

Resultados dos Testes

- Dataset pequeno/normal

Query	Máquina A (tempo médio em s)	Máquina B (tempo médio em s)	Máquina C (tempo médio em s)
Q1	0,002013	0,001539	0,002215
Q2	0,001531	0,001061	0,001989
Q3	0,000301	0,000259	0,000349
Q4	0,004213	0,003024	0,004901
Q5	0,701314	0,659795	0,781143
Q6	0,000706	0,000485	0,000802
Total	19,03421	15,487396	21,84732

- **Dataset Grande**

Query	Máquina A (tempo médio em s)	Máquina B (tempo médio em s)	Máquina C (tempo médio em s)
Q1	0,007100	0,006225	0,007511
Q2	0,015209	0,011734	0,016506
Q3	0,003892	0,003668	0,004201
Q4	0,031501	0,026827	0,032500
Q5	5,938023	4,912369	6,689124
Q6	0,003472	0,002947	0,003810
Total	171,108197	148,270038	182,753200

(Máquina A – Pedro, Máquina B – Francisco, Máquina C - Simão)

Discussão dos Resultados

- **Eficiência do Sistema**: Os tempos de execução confirmaram a eficiência das hashtables para operações de procura e inserção. Algumas queries que demorariam mais tempo, foram tornadas eficientes porque todos os dados necessários foram determinados na fase do parsing, sendo desta forma calculados apenas uma vez e, por isso, diminuindo o tempo de execução. Apesar deste método diminuir muito o tempo de execução pode provocar um grande aumento na memória utilizada e foi o que aconteceu na query 6 - o programa estava a gastar 10 GB. No entanto, contornamos isto passando antes da execução da query pelo input e vendo quais os users e os anos para os quais o resumo teria de ser feito. Desta forma, a memória gasta total passou para menos de 2 GB e a query 6 passou a ser a mais rápida. A query 5 é a mais demorada porque, ao contrário das outras, não tira proveito da fase da parsing, fase esta que não está contabilizada no tempo de cada query. O programa está sem nenhum memory leak, porque todos os que tivemos foram corrigidos com a ajuda do Valgrind. Tendo tudo em conta, consideramos o nosso projeto bastante eficiente, pois encontramos um balanço entre o tempo de execução e a memória gasta.

Desafios e lições aprendidas

Um dos principais desafios desta fase foi a garantia de uma gestão eficaz da memória, uma vez que lidamos com grandes volumes de dados. A utilização do gdb e do valgrind foi essencial para identificar e corrigir leaks de memória, assegurando que o sistema mantivesse a estabilidade durante as execuções. Além disso, a necessidade da validação e tratamento de dados antes de os guardar revelou-se fundamental para a precisão das queries, uma vez que dados inválidos poderiam comprometer os resultados.

Além do problema e da resolução do mesmo mencionados acima para a query 6, o facto de não guardar alguns dos dados (os que não seriam necessários) contribuiu também para uma melhor gestão da memória.

A implementação modular permitiu um desenvolvimento colaborativo e mais eficiente, além de facilitar a identificação e resolução de problemas em cada componente específico.

Em resumo, o sistema desenvolvido para esta segunda e última fase do projeto alcançou os objetivos propostos, apresentando uma estrutura sólida e um bom desempenho.

Conclusão

O desenvolvimento deste projeto permitiu consolidar conceitos importantes de programação, sobretudo modularização e encapsulamento. Durante a sua implementação, enfrentamos desafios relacionados à manipulação de grandes volumes de dados, ao mesmo tempo em que tentamos garantir a eficiência e a manutenção do sistema.

Entre os aspetos mais importantes, destaca-se o uso de hashtables como estrutura de dados principal. Esta escolha foi crucial para garantir acesso rápido e eficiente às informações, permitindo um desempenho consistente mesmo com o aumento do volume de dados. Complementarmente, usamos arrays dinâmicos e listas ligadas para atender às necessidades específicas das queries.

A aplicação de princípios de encapsulamento mostrou-se fundamental para a organização do sistema. Cada módulo foi projetado com responsabilidades bem definidas, o que facilitou tanto o desenvolvimento quanto os testes permitindo identificar e corrigir erros de forma eficiente.

Os resultados obtidos com o programa de testes demonstraram a eficiência (considerando o grande volume de dados) das escolhas feitas. O desempenho do sistema atendeu às expectativas, com tempos de execução aceitáveis para todas as queries e sem memory leaks detetados.

Este projeto não apenas reforçou os nossos conhecimentos, mas também destacou a importância de boas práticas, como o uso de ferramentas como o valgrind e o gdb, testes automatizados e documentação clara. As lições aprendidas serão valiosas no futuro desafios, particularmente no que diz respeito ao planeamento e à implementação de sistemas escaláveis e eficientes.