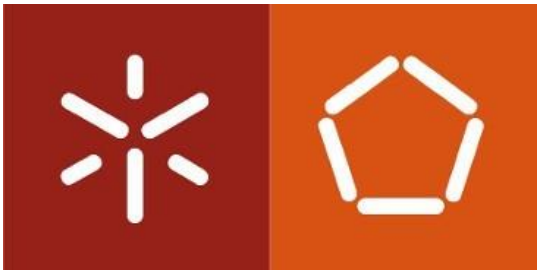


Universidade do Minho

Departamento de Informática

Novembro 2024

Laboratórios de Informática III

Relatório Fase 1

Grupo 77

Hélder Tiago Peixoto da Cruz (A104174)

André Miguel Rego Trindade Pinto (A104267)

Bruno Miguel Vieira Ribeiro (A94662)

Conteúdo

Introdução e Objetivos

Este trabalho prático tem como objetivo desenvolver um sistema de gestão de dados que envolve várias entidades, tais como artistas, utilizadores e músicas, que culminam num sistema de streaming de música que demonstrou ser um tema bastante cativante e que todos nós utilizamos no nosso dia a dia.

O foco principal desta primeira fase é aplicar princípios de programação modular, encapsulamento, e tratamento adequado de dados em C, utilizando diversas estruturas (como tabelas de hash e arrays dinâmicos), incluindo a utilização de instrumentos de compilação, remoção de erros, entre outros. É-nos também pedido a implementação de algumas queries que são, num âmbito geral, o objetivo final do projeto.

No decorrer deste relatório vamos tentar descrever as etapas de resolução dos problemas que encontramos ao longo da realização do projeto, bem como a justificação do raciocínio que nos levou à resolução desses mesmos problemas. Pretendemos também destacar desafios enfrentados e identificar áreas que podem ser melhorados na segunda fase do projeto.

O nosso maior desafio, numa fase inicial, foi encontrar uma forma eficaz de armazenar os dados dos ficheiros, de modo a poder responder às queries de forma mais eficiente, tendo chegado à conclusão de que, embora não fosse necessário para já, a utilização de Hash Tables na gestão das 3 entidades seria a opção mais confiável, tendo já em mente a segunda fase do projeto.

Outro desafio que enfrentámos foi o de ter de lidar com listas com centenas ou milhares de elementos, sendo que a solução encontrada foi, com a ajuda dos guiões e do professor das aulas práticas, criar gestores para cada uma das entidades, para que sempre que uma querie necessitasse dos dados de uma entidade específica tivesse apenas de invocar o gestor da mesma, ao contrário da ideia inicial que seria ter um gestor global que trataria de tudo simultaneamente, o que resultaria num excessivo uso de memória e um tempo de processamento absurdo.

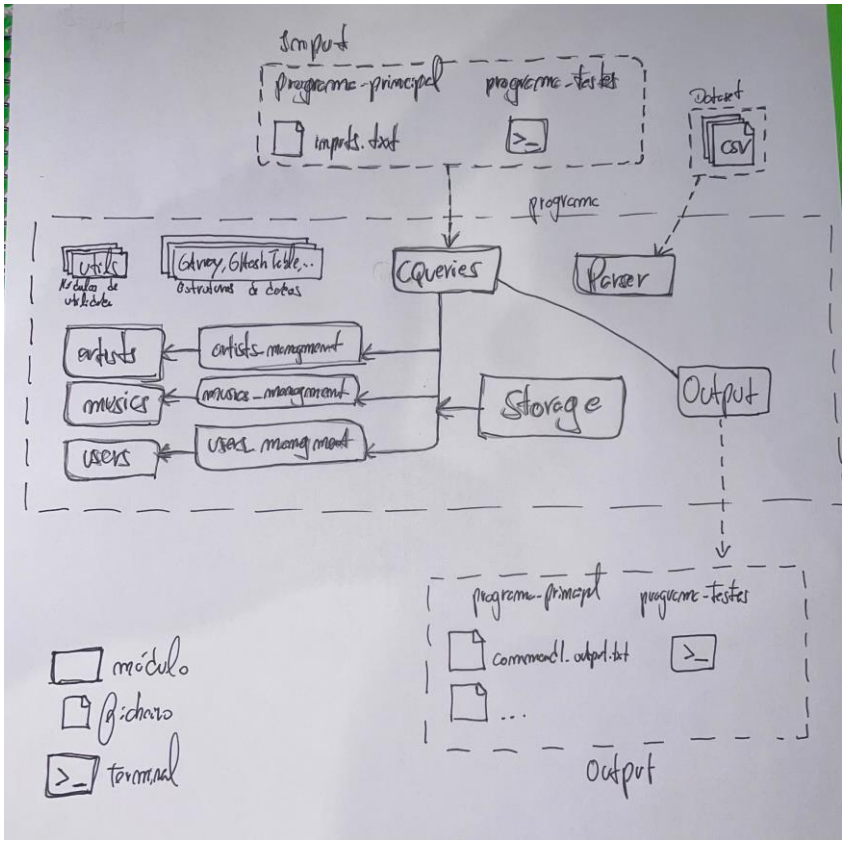
Sistema

• Cada entidade (como *users*, *artists*, *musics)* possui um ficheiro *.c* correspondente para implementar a lógica específica de cada entidade. Além disso, cada entidade tem também funções para gestão, como referenciado na introdução, definidas em *users_management.c*, *artists_management.c*, e *musics_management.c.* Cada ficheiro *.h* define a interface pública do módulo correspondente, enquanto os ficheiros *.c* contêm a implementação interna.

• A parte de Parsing de Dados está implementada no ficheiro *parser.c*, que lê os dados dos ficheiros CSV de entrada e os converte para uma estrutura que possa ser utilizada pelas restantes partes do sistema.

• Cada query possui o seu próprio ficheiro *.c* e *.h*. Esta organização permite que cada consulta seja independente e facilite a extensão ou modificação das consultas, caso necessário.

• Foi desenvolvida também uma abordagem para o programa de testes, que está implementado dentro do *main.c*.



Makefile

A Makefile é um arquivo fundamental para automatizar o processo de compilação do programa principal do projeto, garantindo que todas as dependências sejam resolvidas e os binários sejam gerados corretamente. As principais funcionalidades implementadas nesta Makefile incluem:

• ***Compilação do Programa Principal:*** A Makefile está configurada para compilar todos os arquivos *.c* localizados na pasta src, gerando assim arquivos objeto (*.o*) na pasta obj. Em seguida, todos esses arquivos objeto são vinculados para criar o executável programa-principal. Esta abordagem facilita a organização do código, pois a compilação incremental é mais rápida, uma vez que apenas os arquivos modificados são recompilados.

• ***Limpeza de Ficheiros Gerados:*** A Makefile define uma regra *clean* que remove todos os arquivos objeto gerados na pasta obj e o executável final. Esta regra é útil para garantir um ambiente de compilação limpo e evitar problemas de compilações anteriores que possam interferir numa nova compilação. Uma das vantagens desta utilização é o facto de não haver a necessidade de estar a utilizar o comando *Make clean* repetidamente.

• ***Modos de Compilação: Depuração e Otimização:*** A Makefile suporta dois modos de compilação: debug e release. O modo debug é indicado para identificar e corrigir erros, incluindo flags como *-g3* e *-DDEBUG* que ajudam na depuração do código. Por outro lado, o modo release é otimizado para desempenho, utilizando a flag *-O3* para garantir que o código seja compilado de forma a executar mais rapidamente. O modo release é definido como padrão, permitindo que o comando *make* seja executado diretamente para obter a versão otimizada.

• ***Bibliotecas e Includes Externos:*** São utilizados comandos do *pkg-config* para determinar os includes e as bibliotecas necessárias para compilar o programa, como *glib-2.0* e *ncurses.* Isto facilita a gestão de dependências e torna o código mais portável, uma vez que a localização correta dos cabeçalhos e bibliotecas é feita automaticamente.

• ***Valgrind para Verificação de Memória:*** A Makefile inclui um alvo memcheck, que é útil para executar o programa com o valgrind, permitindo a deteção de memory leaks e problemas relacionados ao uso indevido de memória. Este alvo compila o programa em modo debug para facilitar a análise detalhada.

• ***Compatibilidade com macOS:*** Um dos elementos do grupo utiliza um sistema MacOS, daí a necessidade de terem sido adicionadas definições específicas para INCLUDE e LIBS que ajustam o comando *pkg-config* para compatibilidade com este sistema, o que, de certa forma, melhora e aumenta a portabilidade do código para diferentes plataformas.

Parsing de Dados

O projeto consiste na análise e processamento de três ficheiros principais: artists.csv, musics.csv, e users.csv. Cada um desses ficheiros contém informações relevantes para o sistema em desenvolvimento, e o objetivo é organizar e validar esses dados utilizando as estruturas específicas do projeto como os arrays e as Hash Tables.

• ***Ficheiro artists.csv***

O parsing dos dados presentes no ficheiro artists.csv é realizado por uma função dedicada que lê cada linha e divide-a em tokens. Cada conjunto de dados é armazenado na estrutura Artist. Durante o processo, são utilizados setters para definir os atributos do artista e realizar validações específicas, como a presença de valores nulos ou inválidos. Depois de processados e validados, os artistas são inseridos num GArray, o que permite um armazenamento mais dinâmico.

• ***Ficheiro musics.csv***

O ficheiro musics.csv é analisado de forma semelhante ao artists.csv. Cada linha do ficheiro representa uma música e é lida e dividida em diferentes tokens que são atribuídos aos atributos de uma estrutura Music. Durante o parsing, várias verificações são feitas para garantir que os IDs dos artistas associados são válidos e presentes na tabela *artists_table.* Após a validação, cada música é armazenada num GArray. Além disso, os erros de parsing são documentados num ficheiro de erros, permitindo uma análise posterior de inconsistências.

• ***Ficheiro users.csv***

Para o ficheiro users.csv, o processo de parsing segue uma lógica semelhante, com a leitura das linhas, separação em tokens e validações aplicadas a atributos como email, nome, país, e lista de músicas favoritas. As validações garantem que todos os dados são consistentes antes de serem adicionados à tabela de utilizadores.

A abordagem utilizada para o parsing inclui:

Divisão e Tratamento de Tokens: O uso de *g_strsplit* da GLib permite dividir as linhas dos ficheiros de acordo com o delimitador (;), facilitando a extração de cada campo específico de dados.

Registo de Erros: Ao longo do parsing, erros de dados são registados em ficheiros de erro específicos, como *artists_errors.csv* e *musics_errors.csv*, o que permite a posterior revisão e correção dos problemas detetados durante o processamento.

Discussão

• Durante a fase de implementação, priorizámos a escolha de estruturas de dados que permitissem uma busca eficiente e acessos rápidos, pois as queries envolvem a consulta frequente das diferentes entidades. Utilizou-se h**ash tables** para armazenar os dados das diferentes entidades, uma vez que proporcionam um tempo de acesso e inserção aproximadamente constante, garantindo um bom desempenho mesmo em casos de grande volume de dados, o que imaginamos vir a ser util para as queries pedidas na segunda fase do projeto, mesmo sabendo que o uso desta estrutura possa influenciar negativamente no que toca ao nível de consumo da memória .

• A execução do programa foi testada através de um conjunto extenso de comandos que simulam situações reais de utilização. Os tempos de execução foram avaliados para cada querie, assim como o uso de memória, garantindo que o sistema apresentasse um tempo de resposta rápido para a maioria dos cenários simulados. As otimizações aplicadas no código, especialmente nos acessos às estruturas de dados e no processo de parsing, foram essenciais para minimizar os tempos de resposta.

• Optámos por dividir o projeto em módulos bem definidos, o que possibilitou uma clara separação de responsabilidades. Cada entidade principal (artistas, músicas, utilizadores) possui os seus próprios módulos, que contêm tanto a definição da estrutura quanto as funções específicas para manipulação dos dados. Esta abordagem não só facilita a compreensão do código por terceiros, mas também possibilita a reutilização e extensão de funcionalidades sem grandes alterações na estrutura principal.

• No parsing dos dados, optámos pelo uso de GArray para armazenar temporariamente elementos dinâmicos como IDs de músicas associadas a artistas, uma vez que esta estrutura permite um crescimento dinâmico eficiente e uma manipulação direta dos dados, proporcionando maior flexibilidade ao tratar listas de tamanho variável.

Em resumo, as escolhas tomadas visaram garantir um equilíbrio entre desempenho e organização, permitindo um desenvolvimento flexível e uma estrutura de código facilmente adaptável a novas funcionalidades, como já referenciado, a pensar na segunda fase do projeto.

Testes de Desempenho

Para medir o tempo de execução do nosso programa usámos a biblioteca *time.h*. Fizemos várias medições, através dos ficheiros de inputs que nos é pedido no enunciado, sendo que cada ficheiro deveria tratar apenas e só de uma querie específica.

Conclusão e Trabalho Futuro

A nível geral, e tendo em conta o que foi explicado neste relatório, consideramos ter um projeto bem conseguido, embora as queries não tenham sido garantidas a 100%. No entanto, as estruturas de dados implementadas possuem um acesso eficiente à informação guardada, o que permite que os tempos de resposta das queries sejam razoavelmente curtos.

Sentimos que as nossas estruturas e funções poderiam ser otimizadas em termos de eficiência, um ponto que pretendemos melhorar futuramente. Esta situação deveu-se em grande parte ao foco que colocamos no desenvolvimento e implementação das próximas queries, garantindo que a lógica e a funcionalidade fossem corretamente atendidas para cada uma delas. No entanto, acreditamos que o trabalho realizado até aqui demonstra um grande esforço e dedicação na modularização e encapsulamento, além de uma abordagem robusta para alcançar os objetivos propostos, o que nos deixou orgulhosos dos resultados obtidos até ao momento.

Com a realização da segunda fase do projeto pretendemos melhorar o número de leaks, bem como a utilização de memória, sendo que, como é obvio, temos o objetivo de refazer a querie 3 de modo a esta estar 100% funcional e com tempos de execução/gestão de uso de memória melhorados. Vamos também ter em atenção a gestão do tempo e trabalho por parte dos elementos, que foi algo que nos atrasou bastante na realização desta primeira fase.