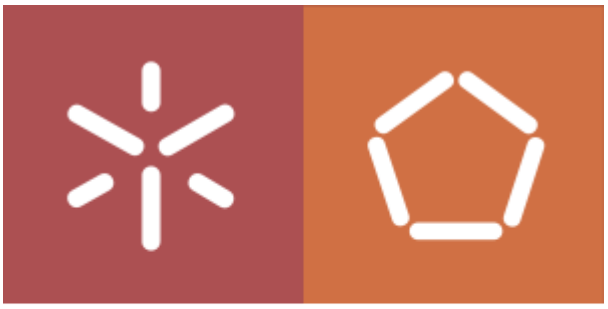

Universidade do Minho
Escola de Engenharia

Laboratório de Informática III

Fase 1

A104433, Francisco Jorge Salgado de Castro
ciscozin

A107361, Miguel Rocha Diegues
MiguelDiegues24

A99432, Nuno Marco Pacheco da Silva
Kouraan

Índice

Conteúdo

1. Introdução

No âmbito da disciplina de LI3, desenvolvemos o projeto que nos foi apresentado, onde tivemos em conta os conceitos de encapsulamento e modularidade. Usamos diferentes estruturas de dados e estratégias de encapsulamento e modularidade.

2. Estruturas de Dados

Para alocar toda a informação relativa aos artists, users, musics decidimos usar structs.

2.1 Artists

Nos artistas usamos uma struct Artists sendo que os artistas são identificados pelo seu id, nome, descrição, país e tipo que guardamos em strings. Já para a receita por stream usamos um float e por fim para o id constituinte usamos uma matriz.

Para validar os artitsts as funções de validação são chamadas nos respetivos setters que por sua vez irão ser chamdos na função create_artist_from_line.

O módulo artistsmanager é chamada a função para criar a linha de um artista guardando a sua informação (já validada) na hashtable artists_table. O módulo database manager irá receber este módulo do artistsmanager. O database manager irá ser chamdo para realizar queries relacionadas com os artists.

2.2 Musics

Para as músicas, usamos a struct Musics para guardar a informação. Usamos um id, nome, duração, género musical e a letra da música, guardados em strings. Quanto ao ano da música guardamos essa informação num inteiro. Relativamente ao id do artista esta informação é guardada numa matriz. Para esta validação, novamente, criamos funções auxiliares que irão ser chamadas nos setters respetivos a cada campo, fazendo a validação das músicas. Setters esses que irão ser chamados na funçã o create_musics_from_line. Relativamente ao módulo musicsmanager, este segue a mesma lógica que o artistsmanager. Mas com os campos relativamente às musicas, onde será criada a hashtable musics_table.

O database manager irá receber as musicsmanager e será chamado para a respetva querie.

2.3 Users

Já no caso dos utilizadores, usamos uma struct Users para guardar o correspondente username, email, primeiro nome, último nome, país e o tipo de subscrição em strings. Para a data de nascimento usamos uma struct Date que guarda o dia, mês e ano de nascimento em diferentes inteiros. Já para o campo do id das músicas liked pelo utilizador guardamos estes dados numa matriz.

Para a validação dos Users, é criado um módulo users_validation em que são criadas funções de validação que são depois chamadas nos setters no módulo users. Setters esses que são chamados na função create_users_from_line onde é feito um parse individual para cada um dos campos que caracterizam um utilizador e criando assim uma linha de um utilizador. No módulo usersmanager é guardado a linha de um utilizador numa hashtable users_table. Esta tabela irá ser guardada juntamente com as outras no módulo database manager. Que irá ser chamado na necessidade de fazer queries. Para destruir esta hashtable é usada uma função da glib para destruir a tabela.

3. Parsing

3.1 Parsing dos dados

Para o parsing dos dados utilizamos um parser genérico para interpretar e organizar os dados de ficheiros CSV. O parser genérico contribui para a manutenção da eficiência do parsing de ficheiros.

Utilizamos também um “parser” específico para cada estrutura de dados – users, artists e musics para garantir que cada tipo de informação seja interpretada e validada de acordo com as suas particularidades.

O parser específico tem como objetivo dividir cada linha do CSV por campos específicos de cada estrutura de dados, para fazer também a validação dos dados pois neste “parse” específico são chamados setters que fazem a validação de cada um dos campos, se a linha não for válida é feita a copia dessa linha para um ficheiro de erros. É também responsável por construir a estrutura e preencher os respetivos campos.

4. Queries

Antes de irmos à implementação das querys, decidimos usar um módulo responsável por ler o ficheiro de comandos e executar a respetiva query com os devidos argumentos. O ficheiro “interpreter.c” verifica a query que está a ser pedida de forma que seja retornado o output que é pedido pela query.

4.1 Query 1

Esta query requer informações sobre os utilizadores, consoante o identificador que é passado. Para auxiliar esta query é criada uma função para calcular a idade, definida módulo users.

Criamos a função *void query1* para implementar o que nos é pedido, recorrendo ao user_manager e a hashtable do users para que seja possível realizar esta query. É feito um lookup na hashtable recorrendo a uma função da glib que irá procurar o id fornecido pelo input. Será assim impresso o email, o primeiro e último nome, a idade e o país do utilizador.

4.2 Query 2

Para a query 2, o objetivo é encontrar os N artistas com as discografias mais longas em termos de duração total das suas músicas, em caso de empate baseia-se no ID dos artistas.

Criamos a função *void query2* que chama o artists manager e o musics manager com as suas hashtables, chama uma função que ordena os artistas em caso de empate.

A hashtable é convertida numa lista ligada para ordenar os artistas pela duração da discografia. Será impresso o output com os N artistas com maior discografia.

4.3 Query 3

Criamos a função *void query3* que recebe os managers das musics e dos users e dois inteiros idade mínima e máxima. Aos respetivos managers irá buscar as respetivas hashtables e cria uma hashtable com os géneros com mais likes. Calcula a idade do utilizador recebido e vê se está dentro do intervalo de idades dado. De seguida, irá buscar o array de liked musics do user, tornando-o numa lista ligada. Itera através da

lista ligada e vê se o id da música pertence à hashtable. Irá buscar o género das músicas e ver os seus respetivos likes. Por fim, ordena, usando a função compara genres a lista, sendo impresso essa mesma lista ordenada.

5. Modularidade e Encapsulamento

Este projeto evidencia uma arquitetura modular e envolvente para assegurar uma estruturação eficiente. A modularidade consiste na separação do programa em diferentes módulos, cada um responsável por uma parte específica da funcionalidade geral. Por exemplo, o módulo usersmanager irá gerir as estruturas de dados referentes aos users, e por sua vez o módulo database manager irá gerir os 3 managers do programa.
O encapsulamento é implementado ao restringir o acesso direto às estruturas de dados dos módulos. Em vez de aceder diretamente aos dados de users, artists, ou musics, o código interage com essas entidades através de funções bem definidas e expostas pelos módulos. Esse encapsulamento garante que a lógica interna de cada módulo fique isolada, permitindo modificações nos detalhes de implementação sem afetar outras partes do sistema. Por exemplo, funções como get_user_username ou get_artist_id fornecem um acesso aos dados dos utilizadores e artistas, preservando a integridade das informações.

6. A implementar na fase 2 e reflexões

Com o objetivo de atacar a abordagem planeada para o projeto, apresentamos algumas considerações adicionais sobre a estrutura e desempenho desejados para a mesma. Pretendemos diminuir o tempo de execução do programa e a memória utilizada para que este possa correr de forma fluída em qualquer tipo de computador e seja mais eficiente a correr o nosso programa.

Relativamente à fase 1 do projeto, a parte onde tivemos grandes dificuldades foi na parte da gestão de memória e na remoção de memory leaks. Porém conseguimos atingir o que era pretendido para esta primeira fase do projeto e esperamos continuar este trabalho para a segunda fase do projeto.