Universidade do Minho
Escola de Engenharia

Laboratórios de Informática III

– Relatório fase I –

Grupo: 7

Projeto desenvolvido por:

Filipa Cangueiro Gonçalves(A107329)

Carolina Silva Martins (A107285)

Diogo José Ribeiro e Ribeiro (A106906)

1. Introdução

Na unidade curricular de Laboratórios de Informática III, foi proposta a realização de um projeto em C, com vista a dar a conhecer aos alunos os princípios fundamentais de Engenharia de Software – modularidade, reutilização, encapsulamento, estruturas dinâmicas de dados, validação funcional e medição de desempenho.

Em complementaridade, foram também trabalhadas ferramentas essenciais para o desenvolvimento de projetos em C, mais concretamente compilação, *linkagem*, apuração e resolução de erros e gestão de repositórios colaborativos.

O projeto encontra-se dividido em duas fases. Nesta primeira fase, foi necessário implementar o *parsing*, a validação de dados, o programa principal, o programa testes e responder às três *queries* divulgadas.

2. Desenvolvimento

No nosso projeto, foi implementado um *parser* geral, uma estrutura que possui estado e que nos permitiu trabalhar pormenorizadamente os *tokens* por esta gerada. Implementamos, assim como era pedido, o programa principal, sendo possível executar as *queries* sobre os dados sequencialmente, encontrando-se esses pedidos num ficheiro que é recebido como argumento. Isto é algo que requer não só a leitura do *dataset*, como o seu armazenamento em estruturas de dados eficientes. Implementamos também o programa testes que avalia os nossos resultados comparando-os com os resultados esperados e efetua a medição do tempo de cada *query* e também verifica se as *queries* e a validação foram executadas devidamente.

3. Estruturas de dados

Nesta fase inicial, uma vez que apenas temos o conhecimento de três *queries*, a nossa implementação passou por maioritariamente respeitar o tipo atribuído aos dados no *CSV´s* e desta forma manter as definições dos nossos parâmetros de acordo com a norma inicial. Dessa forma, a maioria da representação e especificação dos dados possui o tipo *string (*char)*.

Optamos por guardar os dados em três tabelas de Hash(*ArtistTable*, *MusicTable* e *UsersTable*). Consideramos que esta é a melhor estrutura para aquilo que nos era e pode ainda ser pedido uma vez que a maioria das operações que necessitamos de efetuar (operações de pesquisa) são constantes (*O(1)*) em comparação com outras estruturas cujas operações têm velocidades mais versáteis. Para além das três tabelas de *Hash* sentimos também necessidade de implementar uma estrutura extra quando trabalhamos os *Users*, algo que será abordado depois.

O primeiro ficheiro *CSV* que trabalhamos foi o *"/artist.csv"*, onde numa primeira fase optamos por alocar todos os dados com a definição (**char)*. O mesmo aconteceu para os restantes ficheiros, *"/music.csv"* e *"/users.cvs"*.

À medida que fomos progredindo no trabalho sentimos necessidade de modificar alguns desses dados de forma que fosse mais fácil trabalhá-los na implementação das *queries.*

Assim, apesar de termos começado por trabalhar o *"/artist.csv"*, o primeiro ficheiro em que fizemos alguma alteração substancial foi o *"/user.csv"*. A nossa primeira alteração foi ao guardar o *liked_songs_id* do *"/users.cvs"*, onde alteramos o tipo de *string (*char)* para um *array (**char)*. Uma mudança que também efetuamos nos parâmetros artist_id do *"/musics.cvs"* e *id_constituint* do *"/artists.cvs"*. Deste modo, todas as nossas estruturas passaram a ter mais um parâmetro que efetua a contagem dos campos anteriormente falados. Isto permitiu que os problemas anteriormente causados pelo difícil acesso a informações não preparadas fosse resolvido. Em adição a isso, alteramos algumas definições de string (*char) para números, (int ou float) com o mesmo intuito.

Para além disso, e com um intuito de alocar menos memoria decidimos não guardar o parâmetro *lyrics* do *"/musics.cvs"*. Uma decisão que sabemos que poderá ser revertida na segunda fase, mas que mesmo assim consideramos acertada uma vez que estes parâmetros em específicos não são necessários para o trabalho das *queries* divulgadas.

4. Estrutura do trabalho

Após a leitura atenta do enunciado e de entendermos que tínhamos necessidade de ter uma arquitetura pela qual nos pudéssemos guiar decidimos que o nosso trabalho estaria principalmente dividido em: *Main*, *Controllers*, *Entities*, *IOManager e Utils*.

4.1. MainController e Controllers

Os *Controllers*, são geridos por um *MainController* que tem acesso ao *controller* de cada entidade. Assim, enquanto os *controller* das entidades são aqueles \que têm um acesso direto às estruturas de dados principais e realizam operações de *feed*, *parsing e* validação. O *MainController* é aquilo que permite fornecer os dados necessários no restante do nosso programa.

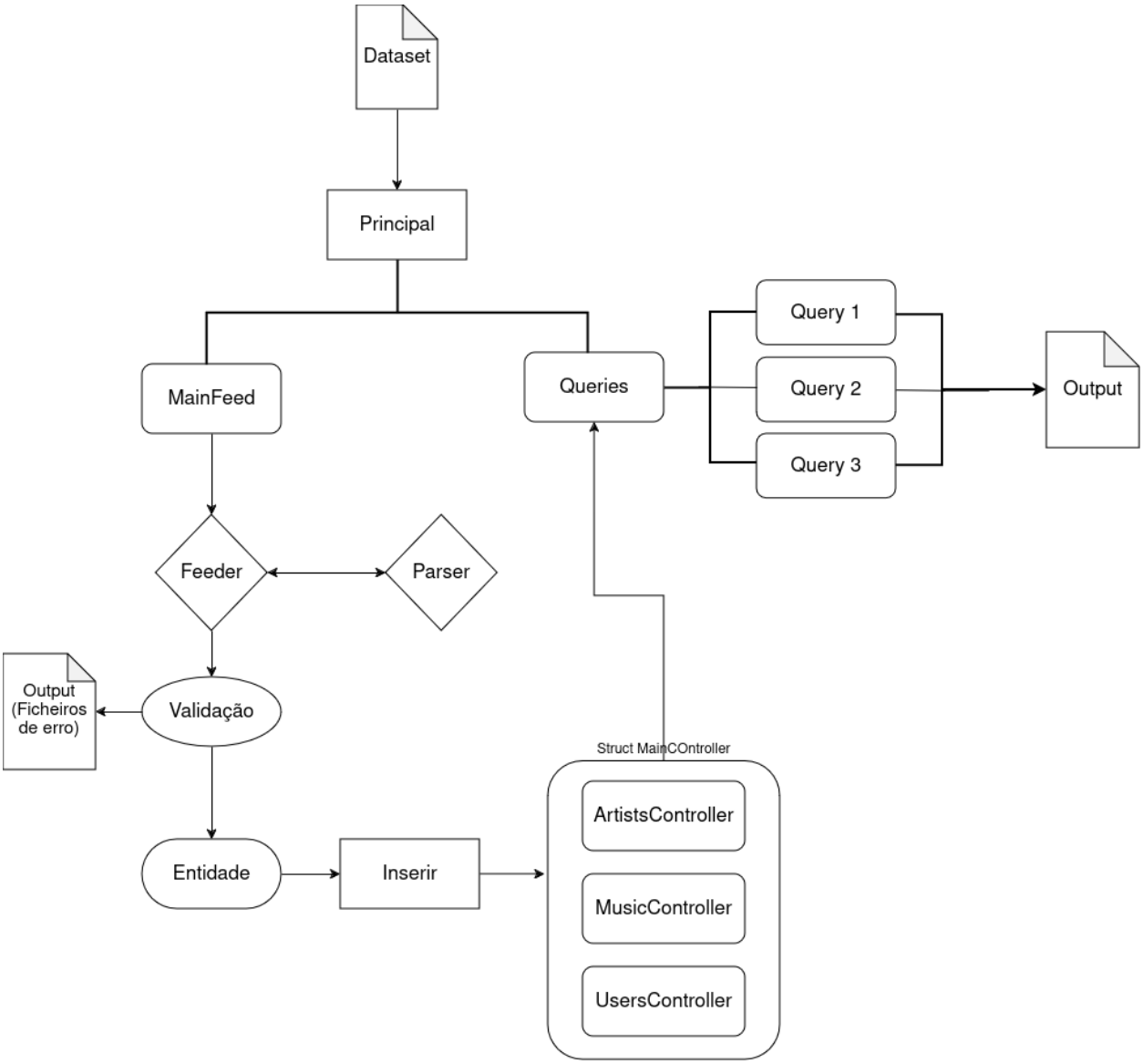
4.2. Entities

As *Entities,* são todas as funções que permitem enviar copias de informações das estruturas (Ex: Tabelas de *Hash* ou *Arrays)*. Estas incluem funções de *getters* e *setter* para além de funções que nos permitem criar mais constituintes das respetivas *entities,* assim como, imprimir os seus constituintes.

4.3. IOManager e Utils

Ao longo do trabalho, existiu a necessidade de separar funções que estão ligadas a *IO* (i.e. *Input/Output*) das restantes funções. Com esse fim criamos o *IOManager* onde estão implementadas funções que abrem e fecham ficheiros, alguns *getters* e *setters* gerais e o nosso P*arser* geral. Optamos por implementar um *parser* geral e não um p*arser* para cada uma das *Entities* porque, para além de preservar o encapsulamento e a modularidade, consideramos que seja algo vantajoso e que no fim nos permite trabalhar com mais detalhe os *tokens* gerados por este.

Nas *Utils*, guardamos funções que são utilizadas em vários módulos ao longo do nosso programa quer com uma funcionalidade especifica quer como apoio a outras funções implementadas.



5. Estratégias seguidas nas Queries

5.1. Query 1

Na *Query1*, o objetivo é um conjunto de dados referentes a um utilizador ao receber o identificador único do utilizador. Escrevemos nos ficheiros de resultados o email, primeiro nome, último nome, idade e país do utilizador dado. Caso esse utilizador não exista, nada é escrito.

Consideramos que esta *Query* não era muito desafiadora e para lhe responder apenas verificamos se o utilizador existe na estrutura onde guardamos os utilizadores, no nosso caso, na

tabela *Hash* dos *users*. Caso esse utilizador exista escrevemos nos ficheiros de resultados os dados pedidos, caso contrário apenas escrevemos uma nova linha.

Concluindo, a *Query1* pretende de forma eficaz fornecer informações referentes a um determinado *user*. A resolução apresentada pretende solucionar este problema de maneira equilibrada, isto é, no melhor rácio entre memoria usada e velocidade de resposta.

5.2. Query 2

Na *Query2* queremos saber quais são os Top N artistas com maior discografia e temos ainda um fator não constante que é o país. Devemos escrever no ficheiro de resultados o nome, tipo de artista (individual ou grupo), a respetiva discografia e país.

Ao começarmos a trabalhar nesta *Query* tivemos algumas dúvidas sobre qual seria melhor abordagem para responder ao que nos foi pedido de forma a equilibrar o uso de memoria e de tempo. Ao analisar tudo e depois de algumas tentativas decidimos que a nossa ideia inicial de implementar outra estrutura com os dados necessários para responder a esta *query* não era a melhor, uma vez que passaríamos a ter informações repetidas que apenas seriam usadas numa parte do programa. Então, consideramos que a forma mais eficaz seria adicionar um novo parâmetro na descrição dos artistas.

Assim, à medida que vamos recebendo os dados de *"/musics.cvs"*, vamos calculando a discografia e colocando-a no respetivo artista. Posteriormente, já dentro da *query*, utilizamos funções da Gl*ib* para conseguir transmutar a tabela de *Hash* que contem os dados dos artistas num *array*, mais especificamente, um *g_array*, já ordenado por ordem decrescente de discografia.

Como o país pode ser ou não uma condicional para esta *query*, isto é, podem pedir os Top N artistas de um determinado país ou apenas os Top N artistas. Com intenções de melhorar o nosso desempenho, caso seja dado um país apenas serão processados para o *array* os dados que respeitarem essa regra.

Em suma, apesar de esta *Query* ter sido mais desafiadora e ter causado alguns problemas consideramos a resolução apresentada satisfatória e eficaz.

5.3. Query 3

Na *Query3*, o objetivo é requerer quais os géneros mais populares de uma determinada faixa etária. Devemos escrever no ficheiro de resultados o género por ordem de popularidade e os respetivos número de *likes*.

Esta foi, sem dúvida, a *Query* mais trabalhosa e que consideramos mais difícil. Inicialmente, consideramos que o melhor seria implementar outra estrutura para ajudar a trabalhar o *Users*. Assim, implementamos o *usersByAge* um *array* que em cada posição continha informações sobre a respetiva idade, isto é, na posição 0 continha as *liked_songs_id* de todos os *users* com 0 anos. Apesar de esta ser uma solução que depois nos ajudava a satisfazer o problema, tinha um gasto de memoria e tempo bastante elevado.

Com o intuito de otimizar o nosso projeto, começamos a procura por diferentes soluções de forma que conseguíssemos responder ao problema, mas o gasto de memoria e de tempo não fosse tão elevado.

Assim, optamos por continuar a usar a *struct userbyAge*, mas mudamos aquilo que a constituía. Passamos agora a ter uma *struct* composta por um *array* de *string (char**)* dos géneros de músicas, um *array* com a respetiva quantidade de *likes* de cada género de música e quantidade de géneros que temos em cada idade. Mantivemos na mesma a nossa logica de na posição 0 estão informações referentes aos *Users* com 0 anos. Depois, basta-nos apenas trabalhar os valores que já temos nos intervalos pedidos.

Consideramos a segunda implementação desta *Query* mais aceitável que a nossa execução inicial. Esta apresenta agora uma melhoria notável no tempo de execução e no uso de memoria. Consideramos assim, que temos uma resolução correta e eficiente.

6. Discussão de resultados

Para testar o desempenho do nosso programa, utilizamos a versão mais recente nos seguintes sistemas. O programa foi executado 10 vezes, com o exemplo de *input* dado no *dataset com* erros. Seguidamente foi calculada a media dos resultados e obtivemos os seguintes resultados:

Computador	Processador	RAM	Tempo Q1	Tempo Q2	Tempo Q3	Tempo Total
Asus VivoBook 14 Pro	Ryzen™ 5 5600H	8 GB DDR4	0,593910ms	2.409593ms	0.820501ms	3,910s
Acer Predator Helios 300 PH315-55-79YR	Core™ i7-12700H 3,5 GHz	16 GB	0,194632ms	1.375381ms	0.380307ms	2,413s
Asus Vivobook 15	Core™ i7-1165G7 3,5 GHz	16GB	0.459646ms	1.537301ms	0.218392ms	3,792s

Como podemos observar os tempos apresentados diferem bastante. Consideramos que isto se deve não só à RAM, o computador com 8GB de RAM apesenta um maior tempo do que os restantes com 16GB. Assim como na RAM, o computador com o processador mais recente (Core™ i7-12700H) é mais rápido do que os restantes.

7. Otimizações

No que diz respeito às possíveis melhorias do projeto reconhecemos que existem diversas áreas que poderiam ser aprimoradas tendo em vista uma melhor eficiência e qualidade do código.

Uma destas áreas consiste na otimização da *Query2*, esta é no momento aquela que demora mais tempo revelando-se menos eficientes. Tal otimização consistiria numa restruturação e analise da logica subjacente a esta *query* e das estruturas de dados usadas.

Para além disso, reconhecemos que o nosso trabalho contém falhas em campos como o encapsulamento e a modularidade. Estes são temos em que queremos aprofundar o nosso conhecimento de forma a garantir uma maior segurança e coesão do código.

Por último, apesar de termos tentado minimizar o uso de memoria ao máximo, reconhecemos que poderíamos ter trabalhado mais esse aspeto e propor-nos a fazê-lo para a próxima fase onde iremos estudar e analisar tanto as nossas estruturas de dados como outras.

8. Dificuldade sentida

Durante o desenvolvimento desta primeira fase do projeto deparamo-nos com várias dificuldades que exigiram muito esforço e trabalho da nossa parte. Apesar de todos os obstáculos podemos destacar que a maior pedra no nosso caminho foi a necessidade de implementar o encapsulamento.

Assim sendo, depois de abordarmos o tema em sala de aula começamos a alterar o nosso projeto para que este respeitasse todas as exigências apresentadas. Devido à falta de familiaridade do grupo com este contexto, foi um conceito que se tornou bastante trabalhoso e mesmo sendo algo vantajoso para o código, é uma melhoria que acarreta dificuldades.

Deste modo, tornou-se necessário procurar por formas de melhorar o uso de memoria e tempo. Isto tornou-se uma necessidade não só devido ao encapsulamento, mas também para melhorar a execução das *queries* o que foi algo que também se apresentou como um desafio.

Como tal, tentamos procurar estratégias que melhorassem o nosso desempenho em cada uma das *queries* apresentadas algo crucial para a melhoria do projeto.

Um desafio constante ao longo do projeto foi o uso correto de memoria. Apesar de já termos trabalhado com alocadores de memoria no passado, a importação da sua alocação e respetiva libertação correta nunca esteve tão presente. Dessa forma e com as ferramentas que nos foram disponibilizadas tentamos analisar minuciosamente o código e implementar alterações que nos permitissem aprimorar o nosso uso de memoria.

Consideramos assim que todas as dificuldades foram encaradas com uma postura de aprendizagem. Algo que proporcionou uma maior oportunidade para aprimorar as nossas habilidades técnicas e enfatizou a importância ao detalhe e à procura continua por melhorias de forma a desenvolver um programa cada vez melhor.

9. Conclusão

Em conclusão, consideramos que após o desenvolvimento da primeira fase do projeto existe um melhor entendimento em relação aos conceitos trabalhados, nomeadamente os princípios de engenharia de software e a modularidade.

Enfrentamos diversas dificuldades que serviram de aprendizagem e nos ajudaram a melhorar o nosso entendimento principalmente no que toca ao encapsulamento e ao bom uso da memoria.

A procura por melhorias na q*ueries*, em especial na *Querie3* mostrou-se desafiadora e tornou evidente a constante necessidade de procurar obter um código melhor e mais eficiente.

Assim sendo, pensamos que realizamos todas as tarefas obrigatórias a esta primeira fase na melhor forma que nos foi possível. Acreditamos ainda que o nosso desenvolvimento estabeleceu estruturas solidas para a seguinte fase.

Apesar disso, reconhecemos também que é sempre possível melhorar sendo esse um dos nossos objetivos para a 2º fase.