Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Redes de Computadores

Ano Letivo de 2023/2024

Trabalho Prático LI3
Fase 1
Grupo 35

Flávio David Rodrigues Sousa (A100715)
André Miguel Alves de Carvalho (A100818)
Sandro José Rodrigues Coelho (A105672)

22 de novembro de 2023

LI3

Índice

1 Introdução

Este projeto que se encontra em desenvolvimento ao longo do ano letivo 2023/24 na unidade curricular de LI3 (Laboratórios de Informática III). Esta tem como objetivo refinar os conhecimentos adquiridos da linguagem C e de aprimorar o mesmo conhecimento sobre Engenharia de Software, nomeadamente, conhecimento envolvendo modularidade, encapsulamento, validação funcional, estruturas dinâmicas de dados e medição de desempenho de um programa.

O projeto em questão encontra-se dividido em duas fases, sendo este relatório destinado à primeira fase do trabalho prático, referindo-se a *parsing* de dados, o modo batch e o desenvolvimento de no mínimo sessenta porcento das *queries*, ficando no critério dos alunos sobre quais *queries* querem implementar para esta fase.

Estas *queries* serão executadas por comandos pré-definidos pelos docentes da unidade curricular, de modo a poder testar todos os aspetos do código presente, armazenando por fim os resultados de cada comando em ficheiros de texto.

Por fim, estes ficheiros são testados na plataforma online da unidade curricular (Plataforma de Teste), de modo a descobrir se existe algum erro com as *queries*, ou com o modo com que entregam os seus outputs.

2 Desenvolvimento

Para realizar o trabalho de forma mais eficaz, decidimos dividir o código, pois seria mais fácil de controlar o que estaria a ser realizado sobre cada parte do projeto, sendo a realização total do pedaço de código, resolução de *memory leaks* ou até mesmo renovação de código de modo a que o mesmo seja mais eficaz.

Com essa ideia em mente, foi possível dividir o desenvolvimento do *parser* para um aluno em particular enquanto ou outros dois alunos realizavam as *queries* escolhidas.

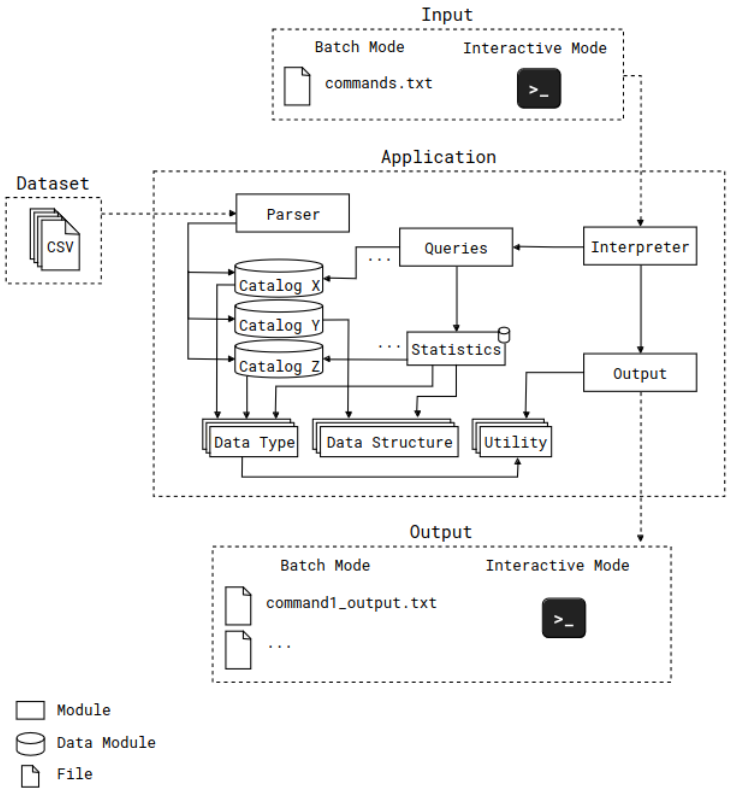

Figura 2.1: Arquitetura de referência entregue pelos docentes

Após abordarmos a fase de planeamento do trabalho, começamos com a base do mesmo, verificando a integridade de todos os dados que seriam manipulados pelo programa, de modo a que seja possível validar cada documento.

Com isso, foi implementado o *parser*, sendo a parte mais importante do programa, permitindo a análise e entendimento dos dados fornecidos. Esta etapa envolveu a criação eficiente de um algoritmo para processar e interpretar todos os dados que o atravessam, garantindo assim uma transição rápida e eficaz.

Em seguida, foi necessário criar as funções responsáveis por manipular as *queries* criadas, de modo a que fosse possível obter os resultados necessários para os comandos fornecidos. Durante este processo, percebemos a necessidade de criar estruturas adicionais para otimizar certas *queries*. Neste caso, foram usadas árvores para certas *queries*.

Por fim, incorporamos um modo de execução *batch*, facilitando a experiência dos utilizadores. Esta é a etapa final, conseguindo trazer todo o potencial existente das *queries* para o programa.

2.1 Estruturas de dados utilizadas

Para a realização deste projeto decidimos utilizar três tipos de estruturas de dados, sendo elas: *HashTables*, Árvores e Listas ligadas.

As *hashtables* armazenam os dados que são *parsed* e são as principais estruturas utilizadas. Estas possuem uma chave que serve para aceder aos valores a que estas estão associadas. Esses valores são *structs* que contêm todas as informações que é necessário armazenar na *hashtable*.

As árvores são utilizadas nas *queries* 2, 4 e 5 uma vez que era necessário comparar valores e imprimi-los de acordo com alguns parâmetros.

As listas ligadas apenas são utilizadas na *query* 2 uma vez que era pedido que, caso nenhum parâmetro que especificasse o tipo fosse passado à função esta teria de apresentar tanto os voos como as reservas de um utilizador em ordem, então para facilitar a comparação entre as datas dos voos e das reservas, as árvores foram convertidas em listas ligadas.

2.2 Queries

A implementação das *queries* foi uma parte do projeto que necessitou de atenção extra, devido à complexidade que a maioria delas tinha, sendo essenciais para o funcionamento do programa, estas representam os pedidos dos utilizadores, obtendo assim as informações pedidas. Com isto, foi possível verificar a importância dos pedidos, assim como a importância da estratégia usada para que o programa pudesse ser resolvida da forma mais eficaz possível, fornecendo resultados coerentes e rigorosos.

2.2.1 Query 1

A primeira *query* consiste em *"Listar o resumo de um utilizador, voo, ou reserva, consoante o identificador recebido por argumento."*, este entrega as informações destinadas ao argumento entregue pelo utilizador, dependendo do identificador recebido.

Se o identificador começar com *"0000"*, a função irá trata-lo como um voo, entregando as informações relacionadas com ele, tal como a companhia, avião, origem, destino, hora de partida estimada, hora de chegada estimada, número de passageiros e o tempo de atraso.

Se o identificador começar com *"Book"*, a função irá trata-lo como uma reserva, entregado as suas informações, tal como *ID* do hotel, nome do hotel, quantidade das estrelas do hotel, data de começo, data de fim da reserva, se a reserva inclui pequeno-almoço, quantidade de noites passadas e preço total da mesma.

Se o identificador não for nenhum dos anteriores, será tratado como um utilizador, entregando assim as informações referentes ao mesmo, tal como, nome, sexo, idade, código do país, passaporte, numero de voos, numero de reservas e total gasto.

Por fim, a função verifica se *type* da função, dependendo do seu valor, as funções serão escritas no formato correto para o ficheiro de saída. Além disso, é alocada memória sempre que é armazenada alguma informação, logo, é necessário libertar essa memória de modo a não haver *memory leaks* no programa.

2.2.2 Query 2

A segunda *query*, consiste em *"Listar os voos ou reservas de um utilizador, se o segundo argumento for flights ou reservations, respetivamente, ordenados por data (da mais recente para a mais antiga). Caso não seja fornecido um segundo argumento, apresentar voos e reservas, juntamente com o tipo."*

Esta função utiliza funções auxiliares para criar e organizar as listas de voos e reservas, imprimindo os seus resultados no formato desejado para o ficheiro de saída.

Com as funções auxiliares criadas, é possível identificar os voos e reservas que se encontram associados ao utilizador, permitindo assim a criação de árvores ordenadas através da data e do *ID*.

Com isto, são usadas outras funções de modo a organizar as árvores por data, realizando a comparação entre as mesmas.

2.2.3 Query 3

A terceira *query*, consiste em *"Apresentar a classificação média de um hotel, a partir do seu identificador"*.

A função percorre as reservas, e verifica se pertencem ao hotel desejado, se assim for, soma as classificações e a quantidade de classificações, criando assim a média da mesma no fim.

Caso não haja nenhuma reserva para o hotel, é escrita uma linha vazia no ficheiro de saída, caso contrário, calcula a média e escreve-a no ficheiro de saída.

2.2.4 Query 4

A quarta *query*, consiste em *"Listar as reservas de um hotel, ordenadas por data de início (da mais recente para a mais antiga). Caso duas reservas tenham a mesma data, deve ser usado o identificador da reserva como critério de desempate (de forma crescente)."*.

A função utiliza uma função auxiliar de modo a construir uma árvore capaz de armazenar as reservas associadas a um certo hotel.

As várias funções auxiliares presentes, desempenham os seus papéis para a funcionalidade da árvore, tal como a criação, manipulação, libertação, contagem e ordenação das árvores criadas.

No fim, são revistas todas as reservas presentes, identificando aquelas associadas ao hotel que se encontra escolhido pelo utilizador, ordenando-as assim por data e *ID*, imprimindo as informações no formato desejado.

2.2.5 Query 5

A quinta *query*, consiste em *"Listar os voos com origem num dado aeroporto, entre duas datas, ordenados por data de partida estimada (da mais antiga para a mais recente)."*.

Para situações em que dois voos possuem a mesma data, a ordenação deve ser feita através do *ID* de voo em ordem crescente.

São usadas funções auxiliares, de modo a construir uma árvore que contém todos os voos que respeitam os critérios estabelecidos pelo utilizador, nomeadamente a origem e as datas, com isso, as funções permitem identificar aqueles voos com a origem desejada e os que se encontram no intervalo de datas especificado pelo utilizador.

Em seguida, a árvore é percorrida pela ordem estabelecida e imprime as informações no formato desejado.

As funções auxiliares presentes, desempenham os seus devidos papéis para a funcionalidade das

árvores, tal como a *query* anterior, resultando na criação, manipulação, libertação, contagem e ordenação das árvores criadas.

2.2.6 Query 9

A nona *query*, consiste em *"Listar todos os utilizadores cujo nome começa com o prefixo passado por argumento, ordenados por nome de forma crescente."*. Caso ambos nomes sejam iguais, é organizado através do *ID* do utilizador.

A função compara ambos nomes de modo a organizar ambos através de ordenação sensível a cada idioma. Durante a execução, a função compara os nomes através do prefixo de cada nome, se ambos prefixos forem iguais, é verificada a letra seguinte, até achar uma letra que seja diferente da que está a ser comparada, ordenando assim os nomes, caso contrário, se os nomes forem iguais, é necessário comparar os *IDs* dos utilizadores e ordenar os nomes através deles.

Por fim, os resultados são ordenados usando a função *qsort* e a função criada para comparação personalizada, formatando e escrevendo os resultados no ficheiro de saída.

3 Dificuldades Encontradas

Durante esta primeira fase do projeto a maior dificuldade encontrada foi na resolução de *memory leaks*, sendo que aproximadamente 5Mb de memória continuam perdidos no parser.

No ficheiro *parser.c* existe um ponteiro denominado *linha_copia* que está a ser separado utilizando a função *strsep* que modifica esse ponteiro e, quando este tenta ser libertado no fim da execução da função, partes do apontador original ficam perdidas e consequentemente, perde-se também memória.

A solução mais a óbvia para este problema seria criar um outro apontador para o apontador que vai ser separado, contudo ao tentar esta solução o programa apresentava outros problemas relacionados à memória.

Assim, decidimos manter o programa a ser executado com esse *memory leak*, mas garantir o seu bom funcionamento e abordar um docente posteriormente.

4 Conclusão

Em suma, apesar de esta fase não ter corrido como esperado devido aos problemas que surgiram de forma tardia no manuseamento da memória, no geral o balanço desta fase foi positivo dado que os objetivos para este fase foram alcançados com sucesso, nomeadamente a criação do *parser*, das *queries* e do *batch*. Para além disso, foi notória a nossa evolução na programação em C uma vez que para resolver algumas *queries*, foi necessário implementar árvores e listas ligadas, sendo estas estruturas de dados que nenhum membro do grupo estava confortável a implementar/utilizar.