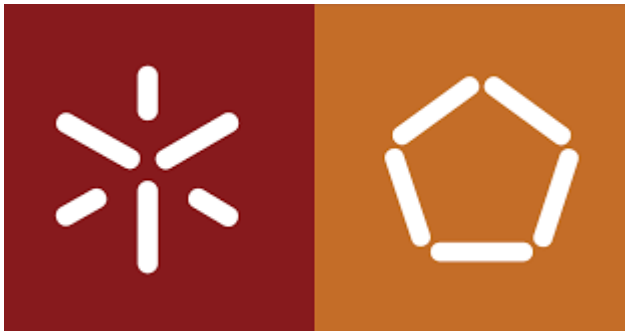

Relatório da 1ª Fase do Projeto

Laboratórios de Informática III

Grupo 23

Licenciatura em Engenharia Informática
Universidade do Minho
Novembro 2023

Grupo de trabalho:
Fábio Rafael Oliveira Magalhães, A104365
Miguel Tomás Antunes Pinto, A100815
Pedro Miguel Costa Azevedo, A100557

Indice

1. Objetivo do Trabalho

Nesta etapa inicial, o desafio deste projeto é consolidar e aprofundar os conhecimentos fundamentais na linguagem de programação C e na disciplina de Laboratórios de Informática III. Estamos focados em explorar conceitos como modularidade, encapsulamento e implementação de estruturas dinâmicas de dados.

Além disso, o trabalho envolve o desenvolvimento de habilidades no uso de ferramentas essenciais para projetos, incluindo a utilização de instrumentos de compilação, linguagem, estabelecimento de metas de projeto com base em dependências, remoção de erros e gestão de repositórios colaborativos.

No decorrer deste relatório, tentamos descrever as etapas de resolução do problema proposto, justificando o raciocínio que nos conduziu à solução adotada. Pretendemos também destacar desafios enfrentados e identificar áreas que podem ser aprimoradas na próxima fase.

2. Batch Mode e Queries

A etapa inaugural de implementação deste projeto envolve a criação do modo batch. Nesse modo, o programa é executado com três argumentos. O primeiro argumento indica o caminho para o diretório que contém os ficheiros de entrada, enquanto o segundo corresponde ao caminho para um arquivo de texto que abriga uma lista de comandos a serem executados.

Esta função em C representa o modo batch, onde ela lê um arquivo de entrada que contém uma lista de comandos a serem executados. Em cada iteração do loop, um comando é lido, executado e a memória alocada para o comando é libertada. Este processo continua até que todos os comandos no arquivo de entrada sejam processados.

É-nos proposta a implementação de pelo menos 60% das queries. Para esta fase escolhemos a resolução das queries 1, 2, 3, 4, 6, 9.

Leitura de Inputs

O código contém uma função chamada `batchMode` que lida com a leitura de inputs de um ficheiro. Essa função abre o arquivo especificado e de seguida lê linha por linha do ficheiro. A cada iteração do loop, é lida uma linha do ficheiro usando a função *fgets*. Essa linha é então passada para a função *execute_command*.

Dentro da função *execute_command*, o conteúdo da linha é processado. Em primeiro lugar, é feita uma interpretação do comando usando a função *command_interpreter*, que extrai informações específicas do comando, como o *ID* da consulta, a *flag* de formato e os argumentos, os quais são manipulados de acordo com o tipo de consulta a ser realizada.

Com base no *ID* da consulta identificado, a função *execute_command* determina qual função de consulta específica deve ser executada. Dependendo do caso, a função apropriada é chamada, executando a consulta e processando os resultados de acordo com a lógica de cada consulta.

A estrutura geral do código garante que cada linha do arquivo de entrada seja lida, interpretada e processada sequencialmente até que todas as linhas tenham sido processadas. Isso permite que os comandos de consulta sejam executados com base nas informações fornecidas no ficheiro de entrada, gerando os resultados correspondentes conforme cada consulta é processada.

3. Parsing de Dados

O projeto consiste na análise e processamento de dados de quatro ficheiros principais: *flights.csv*, *passengers.csv*, *reservations.csv* e *users.csv*. Cada um desses ficheiros contém informações cruciais para o sistema em desenvolvimento.

- **Ficheiro *flights.csv***

O parsing dos dados presentes no ficheiro *flights.csv* é realizado por uma função específica, acedendo e lendo cada linha correspondente a um voo. Cada conjunto de dados é armazenado numa estrutura chamada *FLIGHT*. Esses conjuntos são então inseridos numa *hash table* denominada *flights*, onde cada entrada representa um voo distinto.

Durante o processo de parsing, são efetuados cálculos relevantes aos voos, como avaliação média, número de viagens, custo total, datas mais recentes e distância total percorrida. Também são aplicadas verificações de validação para garantir a precisão e a integridade dos dados processados.

- **Ficheiro *passengers.csv***

No ficheiro *passengers.csv*, os dados são analisados para registrar os passageiros associados a cada voo. Esses dados são armazenados numa *hash table* chamada *passengers*, onde cada entrada corresponde a um voo e à sua lista de passageiros.

Durante o processo de parsing, é verificado se os passageiros são válidos com base nas informações contidas no arquivo *users.csv*. Além disso, são realizadas validações adicionais para garantir a consistência dos dados.

- **Ficheiro *reservations.csv***

O ficheiro *reservations.csv* contém informações sobre reservas em hotéis. Esses dados são processados e armazenados numa *hash table* chamada *reservations*. Durante o *parsing*, verificações são realizadas para validar as datas, o preço das reservas, e outros detalhes relevantes.

- **Ficheiro *users.csv***

O ficheiro *users.csv* é essencial, contendo informações detalhadas sobre os *users*. Os dados são organizados e inseridos numa *hash table* chamada *users*, onde cada entrada representa um *user* distinto. Durante o processo de parsing, verifica-se a validade das informações, incluindo datas de nascimento, e-mails, detalhes de passaporte, entre outros.

Em todos estes ficheiros, após a validação dos dados, são registrados eventuais erros num ficheiro separado para futuras correções e melhorias no sistema.

4. Estrutura de Dados

Para a realização desta primeira fase do trabalho foram utilizadas as seguintes estruturas
de dados:

- Struct
- Estrutura de dados principais – HashTable
- Estrutura de dados auxiliares - GList

Estruturas de dados principais

A estrutura de dados HashTable oferece inúmeras vantagens significativas. A sua eficiência de pesquisa é muito boa e notável, permitindo acesso direto aos elementos por meio de uma chave específica, tornando-se altamente eficiente ao buscar e recuperar dados, especialmente em conjuntos de dados extensos. Além disso, apresenta um desempenho médio constante para operações de busca, inserção e remoção desde que seja implementada com uma função de hash bem distribuída. Essa característica de desempenho estável a torna ideal para manipular grandes volumes de informações, sendo uma escolha frequente por exemplo em bases de dados, onde a adaptabilidade e a eficiência são essenciais para lidar com conjuntos de dados em escala.

Estruturas de dados auxiliares

A estrutura de dados *GList* oferece uma flexibilidade notável na manipulação de dados. A sua estrutura flexível permite a inserção e remoção eficiente em qualquer ponto da lista, tornando-se uma escolha vantajosa quando a manipulação dinâmica dos elementos é necessária. Além disso, a *GList* possui um crescimento dinâmico, expandindo-se conforme os dados são adicionados, economizando memória quando o tamanho total da lista não é previsível. A capacidade de adaptar-se a diferentes volumes de dados a torna uma opção versátil. Adicionalmente, é eficiente na iteração, possibilitando a travessia sequencial de elementos, o que simplifica a manipulação e acesso aos dados na ordem em que estão organizados.

5. Modularidade e Encapsulamento

A modularidade e encapsulamento são importantes no desenvolvimento de um software para que este se faça de forma controlada e reutilizável, permitindo assim uma fácil deteção e correção de erros.

De forma a cumprirmos os princípios da modularidade do encapsulamento neste trabalho, dividimos o código-fonte em unidades separadas e tentámos garantir a proteção e os acessos controlados aos dados.

O módulo *statistics* é um exemplo da modularidade implementada neste trabalho. Este módulo faz cálculos do preço total numa reserva, compara se uma data é maior que outra, etc. Este módulo é reutilizado ao longo do trabalho e funciona de forma independente. Os únicos dados a que acede são preservados e os dados que devolve também.

As queries também irão ser todas realizadas em módulos de forma a possíveis erros e correções não interferirem com o funcionamento do resto do código.

6. Makefile

A Makefile é um arquivo responsável por automatizar o processo de compilação de programas escritos em C. Algumas das funcionalidades implementadas nesta Makefile incluem:

- **Compilação do Programa Principal e de Testes:** A Makefile é capaz de compilar tanto o programa principal como um programa de testes, atribuindo os devidos caminhos aos ficheiros .c em ficheiros .o e, em seguida, unindo-os para formar o executável desejado.

- **Limpeza de Ficheiros Gerados:** A regra *clean* foi implementada para remover todos os ficheiros gerados durante o processo de compilação, incluindo os ficheiros objeto (.o) e os executáveis gerados. Isto é útil para limpar o ambiente e possibilitar uma recompilação sem a interferência de artefatos anteriores ou sem precisar de estar a usar o comando *Make clean* repetidamente.

- **Suporte a Modos de Depuração e Otimização:** A Makefile incorpora um suporte para dois modos distintos: debug e otimização. No modo de depuração (`make debug`), a compilação é feita com flags especiais para possibilitar a depuração usando ferramentas como *gdb* ou *lldb*. No modo de otimização (`make release`), as flags são ajustadas para otimização de desempenho (sendo esta a estabelecida como “default” do comando `make`).

- **Bibliotecas e Includes Externos:** São especificadas as bibliotecas e os includes externos necessários para a compilação dos programas. Isso inclui o uso de bibliotecas como *glib-2.0* e *ncurses*.

- **Gerar, limpar e visualizar a documentação:** Como modo de documentação (não apenas utilizando comentários durante o código), foi optado pela a utilização de uma biblioteca de documentação, Doxygen, muito utilizada para documentação de código clang, C, C#, etc. E para tal são especificados alguns comandos para gerar (`make docs`), visualizar (`make open-docs`) e limpar (`make clean-docs`) a documentação do código. Importante é também relevar que o comando de visualização apenas foi testado em máquinas UNIX e utiliza a diretiva “xdg-open” para a abertura do ficheiro gerado.

7. Conclusão e Trabalho Futuro

Na realização da segunda fase do projeto pretendemos ainda melhorar certos aspetos onde reconhecemos falhas. Ambicionamos ainda a otimização das estrutura de dados utilizadas para algumas das queries. E contamos com a realização de uma melhoria na gestão de memória e tempo de execução.