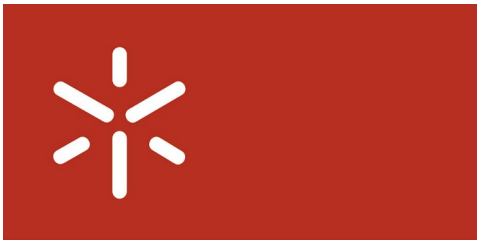


Universidade do Minho

Licenciatura em Engenharia Informática

LABORATÓRIOS DE INFORMÁTICA III

Ano Letivo 2022/2023

Projeto – Fase I

Fábio Jorge Almeida Cunha, 105089

Gabriel Araújo, 102023

Tiago Ferreira Soares, 97381

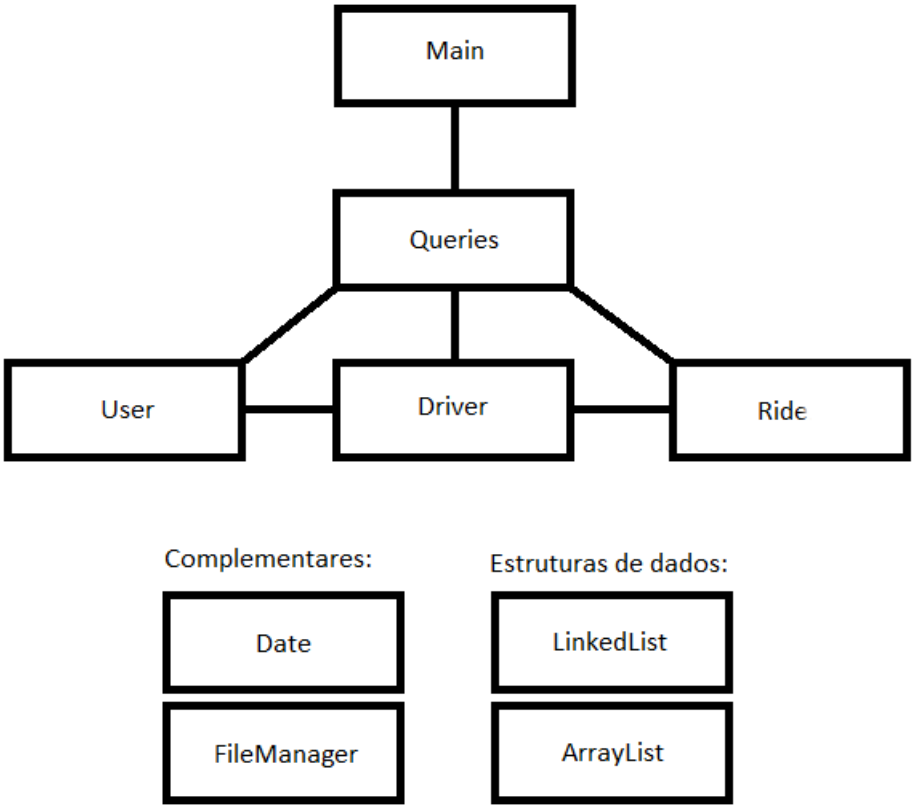
Introdução

Este trabalho foi realizado no âmbito da Unidade Curricalar de Laboratórios de Informática III no segundo ano da Licensiatura em Engenharia Informática na Universidade do Minho. Consiste na criação de um programa em linguagem C capaz de realizar vários calculos de análise de dados relacionados com boleias de automóvel, e os condutores e utilizadores envolvidos.

Para esta primeira fase, foi nos pedido a resolução de pelo menos três queries e de uma utilização do programa com argumentos.

Encapsulamento de dados

De modo a fazer com que os dados fiquem o mais escondidos possível, optamos pela seguinte hierarquia:



Os únicos módulos que vão trabalhar com os dados e que podem trocar dados entre si, são os módulos nível mais baixo da hierarquia. Os outros níveis não terão acesso aos dados, obtendo assim, o encapsulamento desejado.

Os módulos complementares ou de estruturas de dados, apenas vão trabalhar com cópias dos dados, nunca podendo alterar os originais.

Modularização

Para a realização deste projeto, decidimos utilizar os seguintes módulos:

***ArrayList*, *Date*, *Driver*, *FileManager*, *LinkedList*, *Main*, *Queries*, *Ride*, *User*.**

Em seguida, passamos a explicar o que cada um faz:

- ***Driver***, ***Ride*** e ***User*:** Estes três módulos estão no fundo da hierarquia. São responsáveis por carregar os dados do seu respetivo ficheiro para memória.
  Além disso, é aqui que se realizam todas as operações de consulta de informação e cálculo de estatísticas pedidas pelas *queries*. As funções aqui criadas devolvem apenas valores ou cópias dos dados que estão nos ficheiros.

- ***Date***: Módulo onde transformamos a *string* de data contida nos ficheiros, separando-a em inteiros e agrupando-os numa estrutura do tipo Date.
  Também realizamos aqui, algumas operações sobre datas, como, por exemplo, calcular a idade de uma pessoa.

- ***Queries***: Módulo que lê um ficheiro com comandos de consulta de informação de dados e os executa em seguida, chamando as funções que calculam o resultado respetivo a cada um deles.

- ***FileManager***: Módulo que faz o tratamento de ficheiros. Lê os ficheiros de dados para depois serem carregados para memória e escreve os resultados de cada comando no seu respetivo ficheiro.

- ***Main***: Módulo de entrada, que nesta fase apenas trata dos argumentos no modo batch.

- ***ArrayList*** e ***LinkedList***: Módulos de estruturas de dados. O primeiro usa um *array* de tamanho estático que guarda em memória todos os dados contidos nos ficheiros. O segundo são listas ligadas que são maioritariamente usadas juntamente com *Hashtables* do *glib* para categorizar os dados do módulo ***Ride***.

Dificuldades encontradas e metodologia

- Logo de inicio que achamos mais correto utilizar *arrays* para guardar a informação toda em memória, para poder tomar partido da localidade espacial na sua travessia, que era a nossa maior preocupação, devido ao número enorme de informação a guardar. Primeiro, usamos os *array* básicos da linguagem *C*, mas para melhor encapsulamento de dados, mais tarde criamos a *ArrayList*.

- Decidimos usar um *array* estático nas *ArrayLists* pois achamos que em longo termo ia ser mais beneficioso, pois com um array dinámico iamos ter de o aumentar e copiar os dados todos várias vezes, o que também ia resultar em memória alocada não utilizada. Mesmo que a nossa solução de percorrer o ficheiro de *input* para calcular as linhas adicione algum tempo, acaba por valer a pena.

- Antes de entendermos melhor como encapsular e modular o nosso programa, o *FileManager* trabalhava diretamente com os dados. Mais tarde passamos para o carregamento ser feito nos próprios módulos, mas isso tornava o *FileManager* redundante, pois apenas abria o ficheiro a usar. Por último, o *FileManager* ficou a abrir os ficheiros e ler todas as linhas, mas a interpretação destas e a inserção em memória é feita nos respetivos módulos.

- No início um dos nossos membros utilizava o *Windows* no computador, que não tem acesso à função *strsep*. Tentamos utilizar 3 implementações diferentes encontradas na internet e uma criada por nós, mas nenhuma funcionou. Trocamos para o *strtok*, mas quando eventualmente entendemos como utilizar o *strsep*, acabamos por usá-lo pois achamos a sua utilização mais clara.

- Como o número de *rides* é enorme, achamos melhor dividi-las em várias categorias para não termos de correr o *array* todo. Para isso, usamos *HashTables* da biblioteca *glib* que ligam *keys* a *LinkedLists*, que contem as *rides* dessa respetiva *key*. Dividimos as *rides* em cidades, *drivers* e *users*.

- Falhamos no encapsulamento de dados ao realizar as *queries*, pois este módulo mexia diretamente nos dados para realizar os vários cálculos. Quando entendemos o erro, trocamos para que apenas junte e organize os dados, mas os cálculos são realizados nos seus respetivos módulos.

- Tinhamos criado a nossa propria estrutura de *boolean* até isso dar conflitos com o *gboolean* do *glib*, e decidimos apagar a nossa. O erro que nos dava não era nada claro e acabamos por perder algum tempo a tentar encontrar a razão.

- Tentamos realizar a *query* 8, mas optamos por desistir e deixa-la para a segunda fase. Os maiores problemas foram dados a ser corrumpidos num *array* dinâmico de *strings*, a ordenação das *rides* não funcionar e guardar muitas mais *rides* do que era correto. Como já não faltava muito tempo para a data de entrega, decidimos nos focar em *queries* mais simples do que tentar resolver os problemas desta.

- Originalmente, a *query* 4, apesar de estar correta, demorava cerca de 15 minutos a realizar. Passamos muito tempo a pensar que era normal, pois cada cidade tem um número muito grande de *rides*, mas depois entemos que o problema era da maneira como as *LinkedLists* eram percorridas. Sempre que pretendiamos aceder a uma posição na lista, esta era percorrida desde o inicio até essa posição, o que tornava uma operação que devia ser O(N) em O(2^N).