Universidade do Minho
Escola de Engenharia

Laboratórios de Informática III
Relatório de Projeto: FASE 1

Afonso Xavier Cardoso Marques 94940
Inês Daniela Alves Ferreira 97372
Luís Tiago Pereira Borges 96357

2022/2023

Contents

1 Introdução

Este projeto consistiu no desenvolvimento de um Sistema de Consulta de Ficheiros CSV na linguagem de programação C. Para desenvolver a aplicação foi necessário recorrer a técnicas de encapsulamento e modularidade havendo a necessidade de implementar estruturas de dados eficientes para armazenar e consultar grandes quantidades de informação.

Todas as estruturas de dados que usámos neste trabalho ou foram criadas por nós ou pertencem à biblioteca GLib da GNOME, nomeadamente as Hash Tables.

Iremos expor neste relatório as dificuldades sentidas relativamente à primeira fase do projeto e quais as soluções e desisões tomadas para as ultrapassar.

2 Estrutura do projeto

O trabalho que realizamos está dividido em diferentes módulos onde cada um serve um determinado propósito mantendo assim o critério de modularidade que é pedido no enunciado.

Alguns dos módulos que criamos utilizam estruturas de dados que eleaboramos especificamente para guardar informação retirada dos ficheiros CSV.

Estão listadas de seguida as estruturas que criamos:

- **User:**
  A estrutura é composta por sete campos, todos string (char*).
  Os campos são: username, name, gender, birth_date, account_creation, pay_method e account_status.

- **Driver:**
  A estrutura é composta por nove campos, todos string (char*).
  Os campos são: id, name, birth_date, gender, class, plate, city, account_creation e status.

- **Ride:**
  A estrutura é composta por dez campos, todos string (char*).
  Os campos são: id, date, driver, user, city, distance, score_user, score_driver, tip e comment.

Estão listadas de seguida os módulos criados:

- **Users -** Módulo que serve para armazenar todos os utilizadores que são retirados do ficheiro 'users.csv'. Usamos uma hashtable de tamanho dinâmico, que a cada username (uma string) faz corresponder uma estrutura de dados User.
  A API associada a este módulo permite criar um novo utilizador, fazer uma cópia de um utilizador já existente, imprimir a informação relativa a um utilizador tanto no terminal como num ficheiro de texto, obter o username e nome de um utilizador e a função de parse.
  A estrutura associada a este módulo foi criada por nós, neste caso 'User'.
  Este módulo está representado pelo ficheiro 'parse-users.c'.

- **Drivers -** Módulo que serve para armazenar todos os condutores que são retirados do ficheiro 'drivers.csv'. Semelhante aos utilizadores, usamos uma hashtable de tamanho dinâmico, que a cada id (uma string) faz corresponder uma estrutura de dados Driver.
  Neste módulo, a API implementada permite criar um novo condutor, imprimir a informação relativa a um condutor tanto no terminal como num ficheiro de texto, obter o id, nome e classe de um condutor e a função de parse.
  A estrutura 'Driver', criada por nós, está associada a este módulo.
  A API está no ficheiro 'parse-drivers.c'.

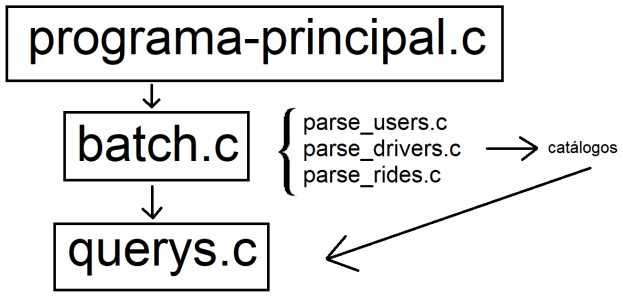
- **Rides -** Neste módulo armazenamos todas as viagens (rides) que são retiradas do ficheiro 'rides.csv'. Semelhante aos utilizadores e condutores, usamos uma hashtable de tamanho dinâmico, que a cada id faz corresponder uma estrutura de dados Ride.
  A API construída para este módulo permite criar uma nova viagem, imprimir a informação relativa a uma viagem tanto no terminal como num ficheiro de texto, obter o id, data, condutor, utilizador, cidade, distância e gorjeta de uma viagem e a função de parse.
  A estrutura 'Ride' é a que está associada a este módulo.
  A API está no ficheiro 'parse-rides.c'.

- **Batch -** Este módulo é relativo ao modo batch da aplicação que estamos a desenvolver.
  Aqui é possível criar catálogos de utilizadores, condutores e viagens. No entanto o mais relevante é a implementação da função 'parse-BATCH' que é responsável pela leitura de um ficheiro de texto que contem comandos para executar queries.
  A API deste módulo está contida no ficheiro 'batch.c'.

- **Queries -** Este módulo contem toda a lógica por detrás da resolução das queries que são feitas à nossa aplicação.
  É composto por nove funções, cada uma relativa a uma query, e funções auxiliares para ajudar na resolução.
  A API deste módulo está presente no ficheiro 'querys.c'.

No que toca ao funcionamento do programa a nível geral, começamos por gerar os catálogos, guardando toda a informação relativa às viagens, aos condutores e aos utilizadores em HashTables correspondentes. Seguidamente, identificamos quais as queries pedidas no input, e quais os seus argumentos. Por fim, utilizamos as funções relativas às queries para obter os resultados pedidos.

Serve a imagem abaixo como demonstração visual simples do funcionamento do programa.



3 Resolução das queries

As queries escolhidas para serem avaliadas nesta fase foram a 4, 5 e 6.

Query 4:

Na query 4, é pedido o preço médio de cada viagem, para uma certa cidade.

Na nossa resolução, incialmente, são criados dois doubles (preco e nr). É também criada uma GList cujos elementos são estruturas do tipo ride com os dados de uma certa viagem. Para cada viagem, verificamos, primeiramente, se se realizou na cidade pedida.

Se sim, criamos uma string "driver_id", para onde copiamos o id do condutor, que será usado para, acendendo à tabela correspondente aos condutores, obter a classe do carro, guardada na string class.

Seguidamente, identificamos o tipo de carro e adicionamos ao preço o montante correspondente. Adicionamos também 1 ao double nr.

No fim, temos o total da soma dos preços e o número total de viagens, pelo que resta apenas dividir estes dois valores, obtendo o preço médio, guardado no double preco_medio, e adicionar esse valor ao ficheiro de output.

Query 5:

Na query 5, é pedido o preço médio de cada viagem feita num certo intervalo de tempo.

Esta resolução é extremamente semelhante à da query 4. Incialmente, são criados dois doubles (preco e nr).

Convertemos os valores das datas em int, através da seguinte fórmula:

dia/mes: AB = A*10 + B
ano: ABCD = A*1000 + B*100 + C*10 + D

(os valores são arbitrários, mas garantem que, numa comparação de duas datas, transformadas de string para int desta forma, a data mais recente corresponde a um número maior).

É também criada uma GList cujos elementos são estruturas do tipo ride com os dados de uma certa viagem. Para cada viagem, convertemos a data em que aconteceu, utilizando a fórmula acima, e verificamos se essa data se encontra dentro do intervalo de tempo dado pelo input.

Se sim, criamos uma string "driver_id", para onde copiamos o id do condutor, que será usado para, acendendo à tabela correspondente aos condutores, obter a classe do carro, guardada na string class.

Seguidamente, identificamos o tipo de carro e adicionamos ao preço o montante correspondente. Adicionamos também 1 ao double nr.

No fim, temos o total da soma dos preços e o número total de viagens, pelo que resta apenas dividir estes dois valores, obtendo o preço médio, guardado no double preco_medio, e adicionar esse valor ao ficheiro de output.

Query 6:

Na query 6, é pedida a distância média percorrida, numa certa cidade, num certo intervalo de tempo.

Inicialmente, criamos dois double: dist, que representa a distância total, e nr, que representa o total de viagens.

Convertemos os valores das datas em int, através da fórmula mencionada na descrição da query 4.

Criamos também uma GList lista_6, onde guardamos elementos correspondentes, cada um, a uma estrutura do tipo ride.

Analisemos essa lista: Para cada um dos seus elementos, ou seja, para cada viagem, verificamos se se dá na cidade pedida. Se sim, transformamos a data dessa viagem pela fórmula acima, e, através da função auxiliar compara3Datas, verificamos se a data da nossa viagem se encontra entre o intervalo de datas referido.

Em caso positivo, adicionamos a distância da viagem ao double dist, adicionamos 1 ao número de viagens.

Por fim, resta apenas dividir esses dois valores, obtendo a distância média, guardada no double r e adicionar esse valor ao ficheiro output.

4 Conclusão

Num panorama geral, e tendo em conta o que foi explicado ao longo deste relatório, achamos que temos um projeto bem conseguido e que cumpre todas as etapas desta primeira fase. As estruturas de dados que implementamos são eficientes, o que permite que as queries sejam executadas com relativa rapidez.

Sentimos algumas dificuldades no inicio da realização do projeto, especialmente na utilização das HashTables da GLib, mas acabamos por conseguir ultrapassar esse obstáculo.

A forma como o projeto estava inicialmente estruturado também foi razão de dificuldade mas graças à advertência e ajuda dos docentes conseguimos reestruturar o projeto, garantindo assim que a nossa aplicação cumpre com os requisitos de modularidade e encapsulamento.

Seguimos assim para a segunda fase mais confiantes das nossas capacidades.