Relatório da Unidade Curricular de Laboratórios de Informática 3 Projecto em C 2019/2020

Luís Manuel Pereira 77667
Pedro Miguel Duarte Araújo 70699
Simão Freitas Monteiro 85489

Abril, 2020

Índice

1 Introdução

Tendo em vista a implementação prática de conceitos relativos a modularidade e encapsulamento de dados, com código robusto e reutilizável, o presente trabalho explora o desenvolvimento de um sistema de gestão de vendas de hipermercado (SGV). Trata-se de um problema de contexto real que reflete a gestão de grandes volumes de dados, pelo que se procura implementar soluções otimizadas que num contexto prático possam ser escaláveis.

Este *use case* requer um sistema capaz de ler e processar as linhas de texto (após *load* de ficheiros de texto - .txt) que contêm códigos de produtos e clientes, bem como o registo de todas as compras e vendas efetuadas, por produtos, cliente e filial (incluindo detalhes, como por exemplo se preço encontra em promoção e qual o seu stock).

O programa, que visa cumprir os objetivos mencionados, foi implementado em C, recorrendo-se à biblioteca GLib (2.62.0) para estruturação de dados. Para avaliar a eficiência do sistema na execução e resolução de tarefas foram efetuados testes de performance sobre o mesmo.

2 Módulos

Esta secção visa detalhar os módulos principais do trabalho, que serão expostos individualmente nas Secções 2.1 a 2.4.

2.1 Objectos

A criação das estruturas de dados foi de encontro ao tipo de dados a analisar, de modo a permitir uma manipulação mais eficiente dos mesmos. Os objetos criados possuem estruturas de dados privadas.

```
struct cliente {
        char codigo;
        int digit;
    };
```

```
struct venda {
        char codProd[7];
        char codCli[6];
        double precoUnit;
        int quantidade;
        char tipo;
        int mes;
        int filial;
    };
```

```
struct produto {
        char codigo[3];
        int digit;
    };
```

As funções *newX*[1] foram desenvolvidas para permitir criar um objeto a partir de uma string, já as funções *XValida/o*, e similares, fazem a validação das strings. As funções *get* permitem aceder às variáveis.

[1] Neste caso X representa venda/cliente/produto

```c
typedef struct venda* Venda;

Venda newVenda(String linhaVenda);
Boolean vendaValida(String venda, Cat_Prods cp, Cat_Clientes cc);
void freeVenda(Venda v);

// getters

String getCodProd(Venda v);
String getCodCli(Venda v);
double getPrecoUnit(Venda v);
int getQuantidade(Venda v);
char getTipo(Venda v);
int getMes(Venda v);
int getFilial(Venda v);
```

```c
typedef struct cliente* Cliente;

Cliente newCliente(String linhaCliente);
String toStringCliente(Cliente);
Boolean clienteValido(String linhaCliente);

// getters
char getCodigoCliente(Cliente);
int getDigitCliente(Cliente);
```

```c
typedef struct produto* Produto;

Produto newProduto(String linhaProduto);
String toStringProduto(Produto);
Boolean produtoValido(String linhaProduto);

// getters e setters

int getDigitProduto(Produto);
String getCodigoProduto(Produto);
```

2.2 Catálogos

O armazenamento de grandes quantidades de objetos (clientes e produtos) é feito através de módulos. Em vez de guardar os códigos de todos os produtos e clientes em arrays, os módulos construídos tratam-se de balanced binary trees, recorrendo a *GTree's* da biblioteca GLib.

O catálogo de clientes é constituído por um array de 26 *GTree's* uma vez que um código de cliente é constituído por uma letra seguida de quatro dígitos (ex. A5000), e cada *GTree* corresponde a uma letra do alfabeto, isto é, a *GTree[0]* corresponde à lista de clientes que começam com a letra "A". Esta escolha permite lidar apenas com os quatro dígitos dos códigos dos clientes, sendo que em cada *GTree* só introduzimos um inteiro constituído desses quatro dígitos, permitindo assim agilizar a procura.

No que aos produtos toca o procedimento para lidar com o seu catálogo é similar, no entanto, uma vez que os códigos dos produtos têm duas letras e quatro dígitos (ex.AA9999), foi implementada uma matriz de 26x26 *GTree's* em vez de um array de 26.

Ambos os catálogos descritos incluem as funções de seguida apresentadas:

- *new_ catClis/catProds* - permite criar um catálogo vazio;

- *free_ catClis/catProds* - permite destruir um catálogo e os objectos presentes no mesmo (para libertar memória);
- *insert_ catClis/catProds* - permite inserir um objeto (cliente ou produto) no catálogo;
- *insert_ withdata_ catClis/catProds* - funciona da mesma forma que a função *insert*, onde o objeto é o *key*, com a particularidade de conter um apontador *data* que vai ser o *value* do funcionamento da *GTree*;
- *get_ data_ catClis/catProds* - procede ao inverso da função anterior, ou seja, dado um cliente a função devolve o *value* que está armazenado no cliente;
- *existe_ catClis/catProds* - verifica a exitência de um cliente ou produto no catálogo;
- *tamanho_ catClis/catProds* - contabiliza os elementos contidos no catálogo;
- *lista_ completa_ catClis/catProds* - devolve uma lista de strings de todos os clientes/produtos contidos no catálogo.

Adicionalmente, o catálogo de clientes inclui as funções:

- *new_ catClisList* - cria um catálogo de clientes e insere os códigos dos mesmos contidos numa lista de strings.
- *add_ to_ lista_ catClis* - insere no catálogo todos os códigos de clientes contidos numa lista de strings;

Por sua vez, o catálogo de produtos inclui ainda:

- *lista_ por_ letra_ catProds* - devolve a lista de strings dos produtos que começam por uma determinada letra;
- *foreach_ catProds* - tem como objectivo aplicar a função *foreach* do GLib a todos os AVLs do catálogo.

```
typedef struct catClientes* Cat_Clientes;

Cat_Clientes new_catClis();
Cat_Clientes new_catClisList(List_String);
void free_catClis(Cat_Clientes);

void insert_catClis(Cat_Clientes, Cliente);
void insert_withdata_catClis(Cat_Clientes, Cliente, Data);
Data get_data_catClis(Cat_Clientes, Cliente);

Boolean existe_catClis(Cat_Clientes, Cliente);
int tamanho_catClis(Cat_Clientes);

List_String lista_completa_catClis(Cat_Clientes);
void add_to_lista_catClis(Cat_Clientes c, List_String l);
```

```
typedef struct catProds* Cat_Prods;

Cat_Prods new_catProds();
void free_catProds(Cat_Prods);

void insert_catProds(Cat_Prods, Produto);
void insert_withdata_catProds(Cat_Prods, Produto, Data);
Data get_data_catProds(Cat_Prods, Produto);

Boolean existe_catProds(Cat_Prods, Produto);
int tamanho_catProds(Cat_Prods);

List_String lista_por_letra_catProds(Cat_Prods, char);
List_String lista_completa_catProds(Cat_Prods);

void foreach_catProds(Cat_Prods, GFunc, Data);
```

2.3 Faturação

Este módulo é responsável pela informação relativa aos produtos e respetivas vendas (mensais), tendo em consideração o custo associado, se o produto foi adquirido em promoção ou pelo preço regular (modo de venda) e a filial onde a venda foi efetuada. A figura 1 ilustra os componentes, funcionamento e potenciais outputs deste módulo. Definimos faturação como sendo um catálogo de produtos especial, onde cada produto tem em *value* uma struct - *FaturaTotal*, que contém todas as vendas de um determinado produto. Esta struct é composta por um array tridimensional de *GSList*, que permite separar as listas de vendas em função do mês, filial e modo de venda. Assim, cada *GSList* contém todas faturas relativas a um determinado mês, filial e modo de venda. Definiu-se como fatura a informação sobre o preço unitário e a quantidade de produtos vendidos. Estas encontram-se ordenadas nas listas em função do preço total.

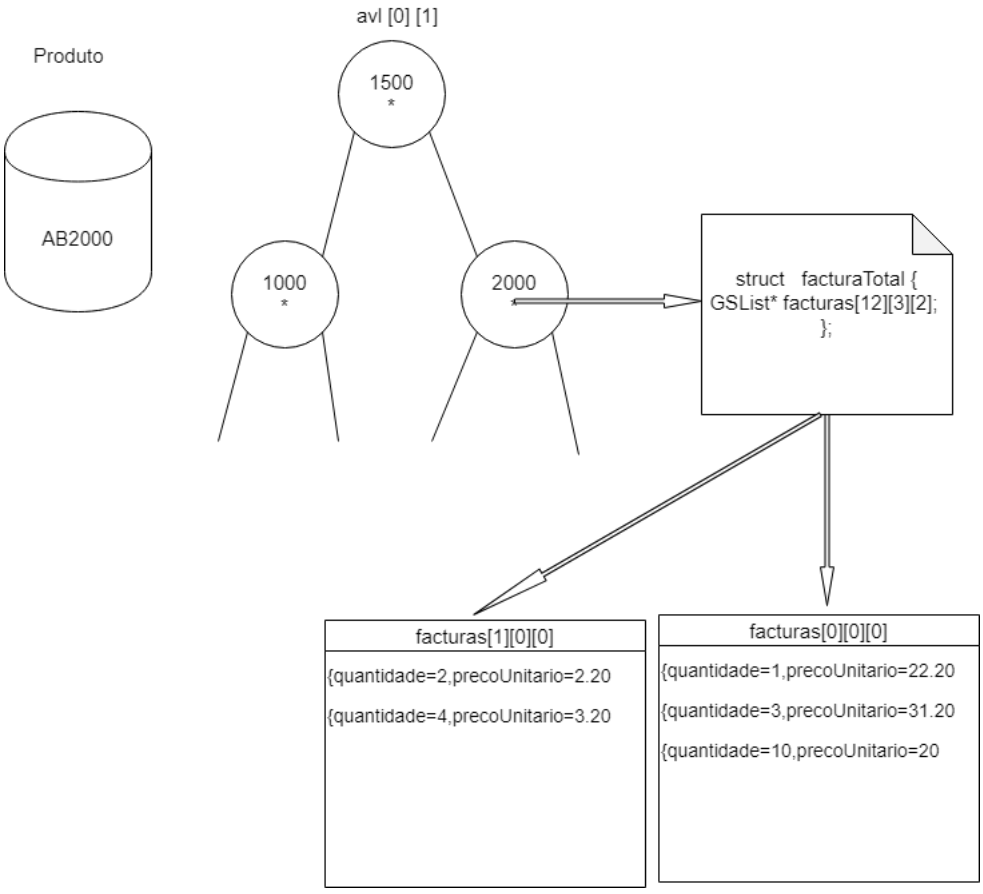


Figure 1: Diagrama do módulo de faturação e da funcionalidade do array tridimensional

O módulo relativo à faturação contém as seguintes funções:

- *new_facturacao* - permite criar um módulo de faturação vazio;
- *free_facturacao* - permite destruir um módulo de faturação (para libertar memória);
- *add_venda_facturacao* - permite inserir as informações de uma venda no módulo faturação;
- *total_vendas_full* - devolve o número de vendas de um produto durante um determinado mês, numa determinada filial e modo de venda (adquirido em promoção ou preço regular);
- *total_facturado_full* - devolve o total faturado por um produto durante um determinado mês, numa determinada filial e modo de venda;
- *total_vendas_facturado* - calcula o número de vendas e o total faturado por todos os produtos durante um determinado mês, numa determinada filial e modo de venda;
- *produtos_nao_comprado* - devolve a lista dos códigos dos produtos que nunca foram comprados.

```
typedef struct facturacao* Facturacao;


Facturacao new_facturacao();
void free_facturacao (Facturacao);
Boolean add_venda_facturacao (Facturacao f, Venda v);

int total_vendas_full (Facturacao f, Produto p, int mes, int filial, char modo);
double total_facturado_full (Facturacao f, Produto p, int mes, int filial, char modo);
void total_vendas_facturado (Facturacao f, int mes, int* vendas, double* facturado);

List_String produtos_nao_comprados(Facturacao f, int filial);

Cat_Prods getCatProd(Facturacao f);
```

2.4 Filial

Este modulo contém a informação da relação entre os produtos e os clientes, mas em referência às filiais. Definiu-se um array bidimensional de catálogo de clientes, onde cada cliente tem em *value* uma lista que contém todas as compras

efetuadas pelo mesmo, ordenadas em função do produto adquirido. O array bidimensional de catálogo de clientes foi usado para poder separar as compras em função do mês e do modo de venda. Assim, cada catálogo de clientes contém todas as compras efetuadas num certo mês e num certo modo.

- *new_filiais* - permite criar um array de três módulos filiais vazios;
- *destroy_filiais* - permite destruir um array de três filiais (para libertar memória);
- *add_venda_filial* - permite inserir as informações de uma venda no módulo filial;
- *compras_clientes* - devolve o número de compras que efetuadas por um cliente durante um certo mês e modo de venda, numa certa filial;
- *clientes_em_todas_filiais* - devolve a lista dos códigos dos clientes que efetuaram compras em todas as filiais;
- *clientes_sem_compras* - devolve a lista dos códigos dos clientes que não fizeram compras em nenhuma filial.

```
typedef struct filiais *Filiais;

Filiais new_filiais(int nbFiliais);
void destroy_filiais(Filiais);
Boolean add_venda_filial(Filiais, Venda);

int compras_cliente(Filiais fs, String codigoCliente, int filial, int mes, char modo);

List_String clientes_en_todas_filiais(Filiais fs);
List_String clientes_sem_compras(Filiais fs, Cat_Clientes allClientes);
```

3 Arquitetura da Aplicação

Neste projeto foi implementada de uma arquitetura MVC (*model-view-controller*), cuja Figura 2 visa exemplificar o funcionamento para o caso da execução da *query* 2, onde o módulo `navegador` funciona como *view*, o módulo `interface` age como *controller*, e os restantes módulos fazem parte de *model*.

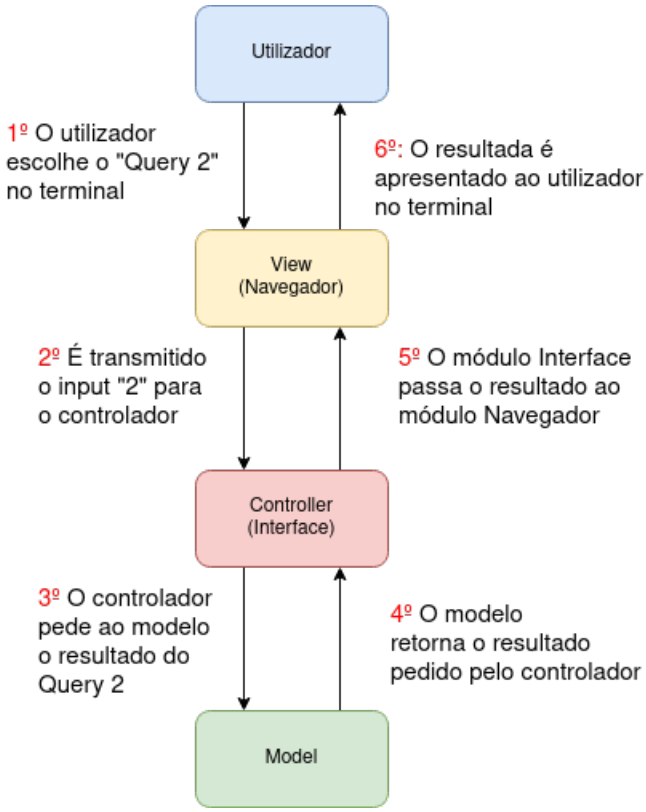


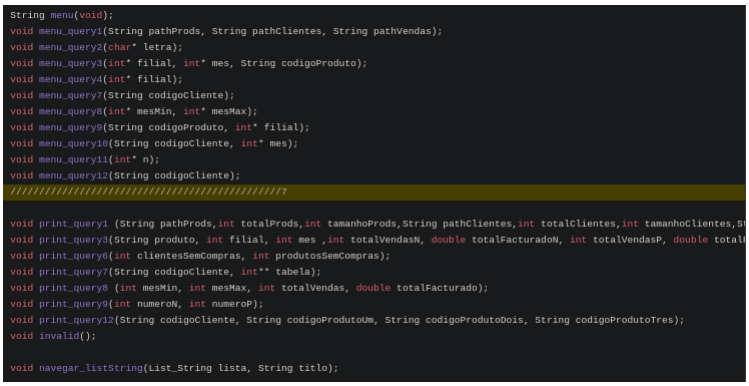
Figure 2: Exemplificação do funcionamento da aplicação (aquando do funcionamento da *Query* 2)

3.1 *View*

O módulo *navegador* contém as funções responsáveis pela interação entre o utilizador e a aplicação. Neste módulo estão presentes as seguintes funções[1]:

[1] X representa o número da *query*

- *menu()* - apresenta o menu da aplicação e recupera o input do utilizador;
- *menu_ queryX()* - permite à aplicação apresentar no terminal o local onde o utilizador irá colocar os argumentos para a sua *query*;
- *print_ queryX()* - permite à aplicação apresentar no terminal os resultado das *query*.
- *navegar_ listString()* - permite ao utilizador navegar nos resultados das *queries* quando estes são de grande volume;



```
String menu(void);
void menu_query1(String pathProds, String pathClientes, String pathVendas);
void menu_query2(char* letra);
void menu_query3(int* filial, int* mes, String codigoProduto);
void menu_query4(int* filial);
void menu_query7(String codigoCliente);
void menu_query8(int* mesMin, int* mesMax);
void menu_query9(String codigoProduto, int* filial);
void menu_query10(String codigoCliente, int* mes);
void menu_query11(int* n);
void menu_query12(String codigoCliente);
////////////////////////////////////////////////

void print_query1 (String pathProds,int totalProds,int tamanhoProds,String pathClientes,int totalClientes,int tamanhoClientes,S
void print_query3(String produto, int filial, int mes ,int totalVendasN, double totalFacturadoN, int totalVendasP, double totalF
void print_query6(int clientesSemCompras, int produtosSemCompras);
void print_query7(String codigoCliente, int** tabela);
void print_query8 (int mesMin, int mesMax, int totalVendas, double totalFacturado);
void print_query9(int numeroN, int numeroP);
void print_query12(String codigoCliente, String codigoProdutoUm, String codigoProdutoDois, String codigoProdutoTres);
void invalid();

void navegar_listString(List_String lista, String titlo);
```

3.2 *Controller*

O módulo *interface* é responsável pela correta ordem de execução, por parte da aplicação, dos pedidos realizados pelo utilizador. Foram então seguidas as recomendações presentes no enunciado do projeto, no qual o módulo *interface* deveria conter a *struct* SGV e onde cada *query* é definida por uma função com um "nome autoexplicativo" do que esta deve fazer para resolver a *query* em questão. Na imagem que se segue encontram-se então:

- *start(SGV sgv)* - age como a função principal da aplicação, sendo nela declarados os módulos que memorizam os dados dos ficheiros de texto;
- *getProductsStartedByLetter()* - responsável pela ordem de execução do código que resolve, neste caso em especifico, a query 2.

```c
void start(SGV sgv){

        String buffInput;
        do {
                buffInput = menu();
                if(strequals(buffInput, "1") ){
                        getCurrentFilesInfo(sgv);
                } else if(strequals(buffInput, "2") ){
                        getProductsStartedByLetter(sgv->catP);
                } else if(strequals(buffInput, "3") ){
                        getProductSalesAndProfit(sgv->faC);
                } else if(strequals(buffInput, "4") ){
                        getProductsNeverBought(sgv->faC);
                } else if(strequals(buffInput, "5") ){
                        getClientsOfAllBranches(sgv->fiL);
                } else if(strequals(buffInput, "6") ){
                        getClientsAndProcuctsNeverBoughtCount(sgv->fiL, sgv->faC, sgv->catC);
                } else if(strequals(buffInput, "7") ){
                        getProductsBoughtByClient(sgv->fiL);
                } else if(strequals(buffInput, "8") ){
                        getSalesAndProfit(sgv->faC);
                } else if(strequals(buffInput, "9") ){
                        getProductBuyers(sgv->fiL,sgv->faC);
                } else if(strequals(buffInput, "10") ){
                        getClientFavoriteProducts(sgv->fiL);
                } else if(strequals(buffInput, "11") ){
                        getTopSelledProducts(sgv->faC, sgv->fiL);
                } else if(strequals(buffInput, "12") ){
                        getClientTopProfitProducts(sgv->fiL);
                }else if(strequals(buffInput, "13") ){
                        infoFiles(sgv);
                } else if(strequals(buffInput, "0") ){
                        destroySGV(sgv);
                }
        }while(!strequals(buffInput, "0"));
}
```

```c
static void getProductsStartedByLetter(Cat_Prods catProds){
        char c;
        menu_query2(&c);

        List_String lista = query2(catProds, c);
        char titulo[] = "Produtos que comecam por a";
        titulo[25] = c;
        navegar_listString(lista, titulo);
        destroy_listString(lista);
}
```

3.3 *Model*

O *model* da arquitetura MVC implementada é constituído por vários ficheiros, pelos módulos apresentados na Secção 2, bem como os três módulos seguintes:

- *Utils*: contém várias funções e structs que são uteis em toda a aplicação;

```c
typedef char* String;
typedef struct lstString* List_String;
typedef void* Data;
typedef enum {false=0, true=1} Boolean;

// funções sobre strings
String mystrdup(const String);
Boolean strequals(const String s1, const String s2);
Boolean strempty(const String);

// função para recuperar um input do utilisador
String userInput(void);

// funções sobre lista de strings
List_String new_listString();
void insert_listString(List_String l, String p);
void destroy_listString(List_String l);
int tamanho_listString(List_String l);
String get_listString(List_String l, int n);
void remove_listString(List_String l, int n);
```

- *Leitura*: contém as funções que permitem ler e tratar os dados dos três ficheiros;

```c
int fileToCatalogoProdutos(String path, Cat_Prods c);
int fileToCatalogoClientes(String path, Cat_Clientes c);
int fileToListStringVendas(String path, List_String vendas, Cat_Prods catProds, Cat_Clientes catClientes);
```

- *Queries*: contém as funções que resolvem as *queries*.

```c
List_String query2(Cat_Prods catProds, char letra);
void query3(Facturacao f, String codigoProduto, int filial, int mes, char modo, int* tVendas, double* tFacturado);
List_String query4(Facturacao f, int filial);
List_String query5(Filiais f);
void query6(Facturacao ff, Filiais f, int* clientesSemCompras, int* produtosSemCompras,  Cat_Clientes allClientes);
void query7(Filiais f, String codigoCliente, int** tabela);
void query8(Facturacao f, int mesMin, int mesMax, int* totalVendas, int* totalFacturado);
void query9(Filiais f,Facturacao fa, int filial, String codigoProduto, List_String listaN, List_String listaP,int* comprasN,int*
List_String query10(Filiais f, int mes, String codigoCliente);
List_String query11(Facturacao ff, Filiais f, int n);
void query12(Filiais f, String codigoCliente, String codigoProdutoUm, String codigoProdutoDois, String codigoProdutoTres);
```

4 Testes de Performance

Foram efetuados testes de performances, usando a métrica *Elapsed Time*, i.e., tempo real de espera por parte do utilizador, para as *Query 1*; *Query 6*, *Query 7* e *Query 9*. Os testes foram realizados para os ficheiros de tamanho 1M(Figura 3), 3M(Figura 4) e 5M(Figura 5), como demonstrado pelas imagens seguintes.

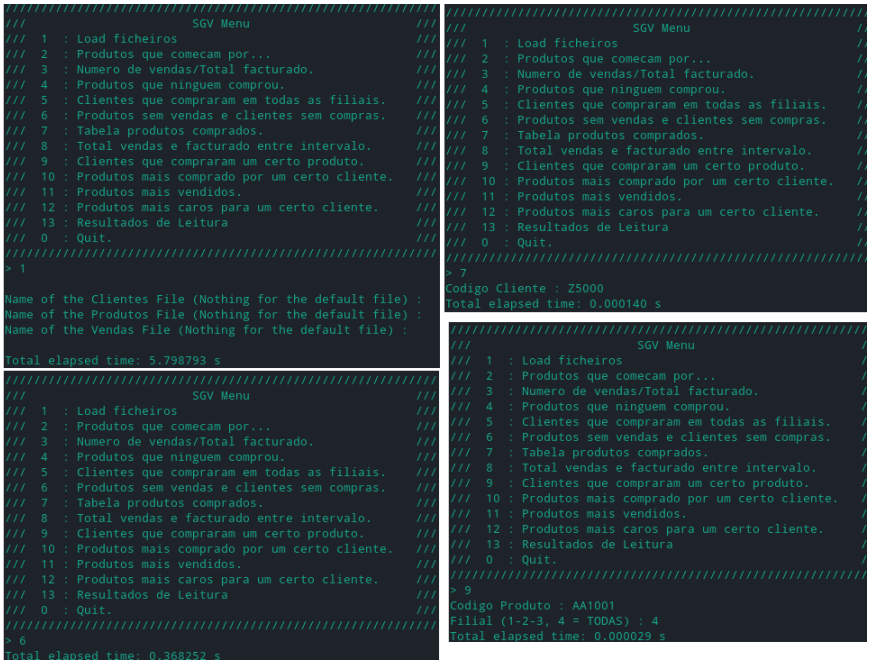


Figure 3: Teste de Performance, ficheiro 1M

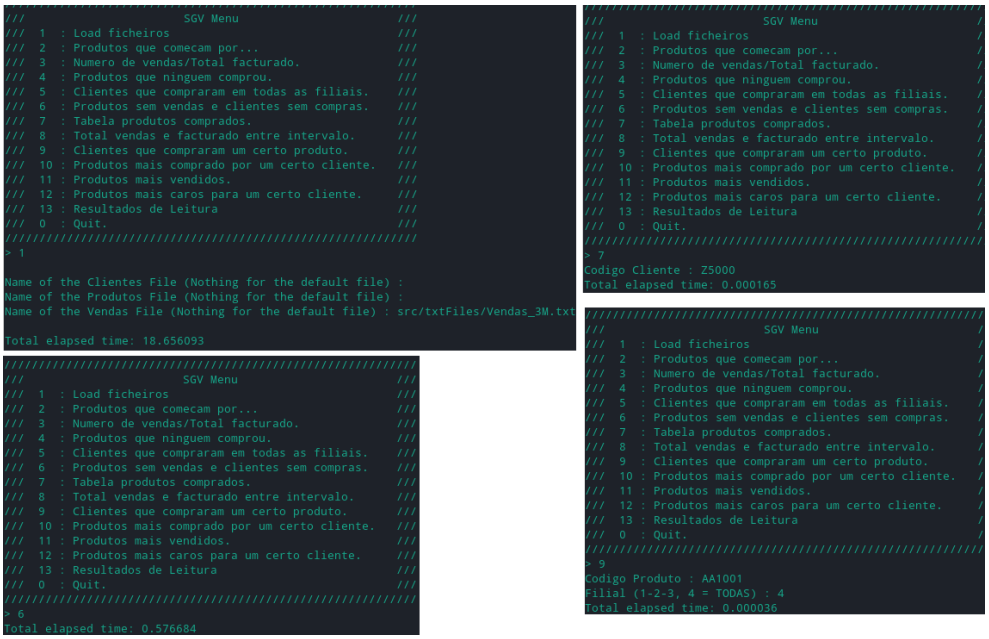


Figure 4: Teste de Performance, ficheiro 3M

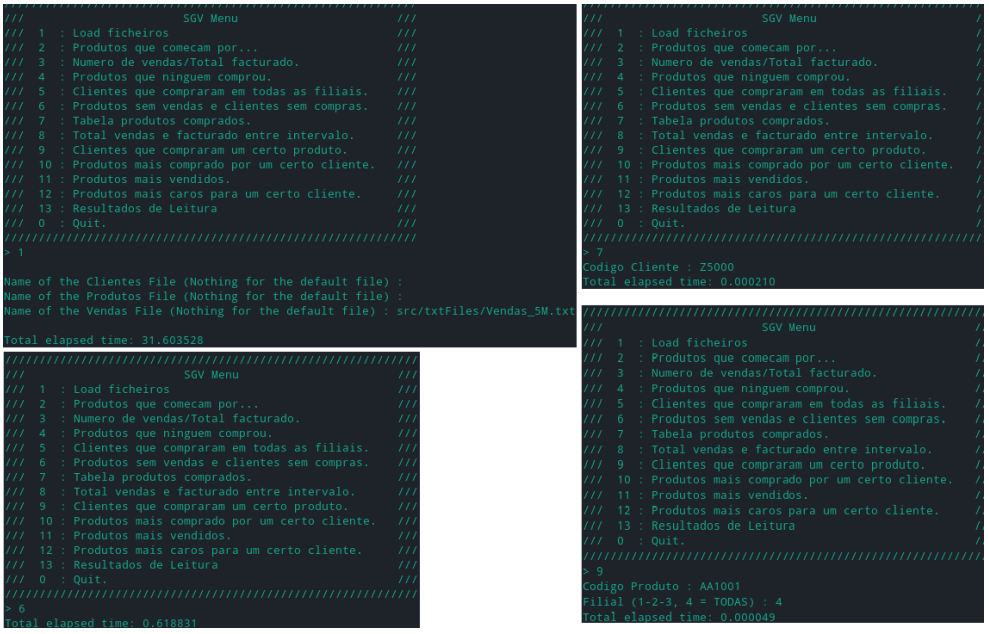


Figure 5: Teste de Performance, ficheiro 5M

5 Conclusões

Neste trabalho foi implementado um sistema de gestão de vendas de hipermercados, de onde os dados foram recolhidos de ficheiros texto que continham informação relativa a clientes, produtos e vendas dos mesmos. O objetivo do trabalho passou pelo desenvolvimento de um sistema que conseguisse incorporar modularidade e encapsulamento de dados, sendo possível obter resultados através de um conjunto de *queries*. Os desafios deste projeto passaram pela gestão de grandes volumes de dados, com vista à implementação de uma solução otimizada e escalável, atendendo à ainda maior quantidade de dados obtidos num contexto real de hipermercados.

As dificuldades encontradas ao longo deste projeto passaram por armazenar, aceder, gerir e retirar eficientemente a enorme quantidade de dados fornecida nos ficheiros de texto fornecidos, o que incentivou à leitura e compreensão da documentação sobre o manuseamento da biblioteca GLib e das suas potencialidades, bem como a várias experiências para corretamente usufruir das mesmas. Adicionalmente, o trabalho desenvolvido consolidou conhecimentos adquiridos ao longo do curso, nomeadamente nas unidades curriculares de Programação Imperativa e Algoritmos e Complexidades. Apesar da solução apresentada não resolver todas as tarefas pedidas, o desenvolvimento deste trabalho fomentou o interesse relativo a estrutura e organização de dados, dando uma noção de contexto prático a engenharia informática.

Bibliografia

[1] B. Kerninghan and R. Pike, *The Practice of Programming.* Addison-Wesley, 1999.

[2] R. Kruse and C. Tondo, *Data Structures and Program Design in C.* Pearson Education, 2007.