

Laboratórios de Informática III (LI3)

Mestrado Integrado em Engenharia Informática
2ºano - 2ºSemestre
Universidade do Minho

PROJECTO DE C: SISTEMA DE GESTÃO DE VENDAS DE UMA DISTRIBUIDORA COM 3 FILIAIS

Grupo 37



Índice

1.Introdução.....	3
2.Sistema de Gestão de Vendas (SGV.c).....	3
3.Estruturas de Dados.....	4
3.1. Catálogos (<i>avl_base.c</i>).....	4
3.1.1 Estrutura	
3.1.2 Funções presentes	
3.2. Gestão Filial (<i>avl_filial.c</i>).....	4
3.2.1 Estrutura	
3.2.2 Funções presentes	
3.3. Faturação Global(<i>avl_facturacao.c</i>)....	6
3.3.1 Estrutura	
3.3.2 Funções presentes	
3.4. Interface.....	7
3.5. Leitura.....	8
3.6. Navega.....	8
3.7. <i>Queries</i>	9
3.8. Teste.....	11
4.Resultados.....	12
5.Conclusão.....	12

1.Introdução

No 2º semestre do 2º ano do Curso de Engenharia Informática da Universidade do Minho, existe uma Unidade Curricular denominada por "Laboratórios de Informática III", que tem como objetivo ajudar os estudantes a consolidar experimentalmente os conhecimentos teóricos e práticos adquiridos em Unidades Curriculares anteriores e a apresentar aos alunos os desafios que se colocam a quem concebe e programa aplicações com grandes volumes de dados e com elevada complexidade algorítmica e estrutural. O presente projeto pretende implementar uma aplicação, designada SGV, desenvolvida na linguagem C, para gerir as vendas de uma distribuidora com 3 Filiais. Trata-se de um projeto em larga escala, onde a quantidade de dados a tratar é elevada. Este projeto foi desenvolvido tendo em conta os conceitos de modularidade e encapsulamento, criação de código reutilizável, uma escolha otimizada das estruturas de dados e testes de performance.

Os principais objetivos que a aplicação deve satisfazer são os seguintes:

1. Ler 3 ficheiros (Produtos, Clientes e Vendas), cujos nomes poderão ser introduzidos pelo utilizador ou, opcionalmente, assumidos por omissão. O resultado desta leitura deverá ser a apresentação imediata ao utilizador do nome do ficheiro lido e que vai ser usado, o número total de linhas lidas e validadas;
2. Determinar a lista e o nº total de produtos cujo código se inicia por uma dada letra;
3. Dado um mês e um código de produto, ambos válidos, determinar e apresentar o número total de vendas e o total faturado com esse produto em tal mês, distinguindo os totais em modo N e os totais em modo P. O utilizador pode decidir se pretende o resultado global ou o resultado filial a filial;
4. Determinar a lista ordenada dos códigos dos produtos que ninguém comprou;
5. Determinar a lista ordenada de códigos de clientes que realizaram compras em todas as filiais;
6. Determinar o número de clientes registados que não realizaram compras bem como o número de produtos que ninguém comprou;
7. Dado um código de cliente, criar uma tabela com o número total de produtos comprados mês a mês organizada por filiais;
8. Dado um intervalo fechado de meses, determinar o total de vendas registadas nesse intervalo e o total faturado;
9. Dado um código de produto e uma filial, determinar os códigos dos clientes que o compraram, distinguindo entre compra N e compra P;
10. Dado um código de cliente e um mês, determinar a lista de códigos de produtos que mais comprou por quantidade por ordem descendente;
11. Criar uma lista dos N produtos mais vendidos em todo o ano, indicando o número total de clientes e o número de unidades vendidas, filial a filial;
12. Dado um código de cliente determinar quais os códigos dos 3 produtos em que mais gastou dinheiro durante o ano.

2.Sistema de Gestão de Vendas (SGV.c)

No nosso ficheiro SGV.c é onde temos definido a "main" também neste ficheiro temos definido as seguintes estruturas de dados:

```
BASICAVL catClientes[26];
BASICAVL catProdutos[26];
AVLFACTURACAO facturacao[26];
AVLFILIAL filial[26];
```

A razão de termos escolhido um array com 26 posições para guardarmos a informação foi como o código de clientes e códigos de produtos comecem por uma letra podemos colocar todos os códigos começados pela mesma letra na mesma posição, códigos começados por 'A' na posição 0, os códigos começados por 'B' na posição 1 e assim sucessivamente. Deste modo fazemos com que todas

as funções que envolvem procurar um código sejam mais eficientes pois em vez de procurarem numa grande AVL procuram numa mais pequena.

3. Estrutura de Dados

Iremos de seguida explicar os nossos módulos de dados explicando para cada um deles a nossa estrutura de dados, o desenho da estrutura e explicar algumas das funções presentes em cada módulo.

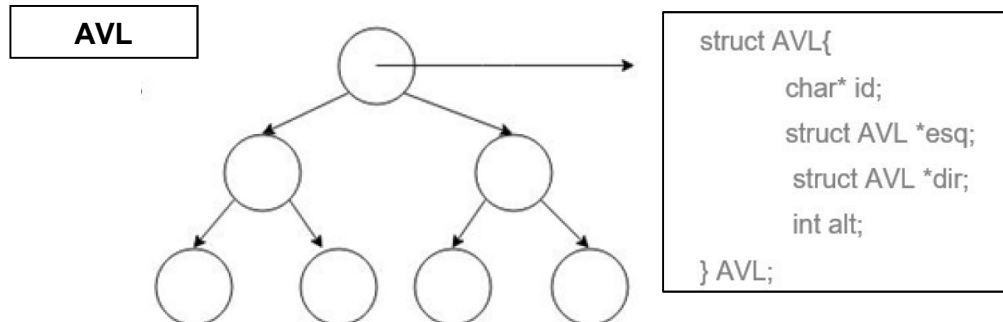
3.1 Catálogos (*avl base.c*)

3.1.1 Estrutura

No ficheiro *avl_base.h* definimos o seguinte tipo de dados:

```
typedef struct AVL* BASICAVL;
```

No ficheiro *avl_base.c* definimos o seguinte tipo de dados:



Os nossos catálogos (de clientes e de produtos) estão definidos como uma BASICAVL em que uma BASICAVL é uma árvore balanceada (AVL) em que em cada nodo tem um identificador (*char* id*), dois apontadores para AVL (*esq* e *dir*) e a altura (*int alt*).

3.1.2 Funções Presentes

Neste módulo para além das funções habituais presentes nas AVL's temos a seguinte a função:

```
char** giveMe (BASICAVL ba,int tot,int* size);
```

Esta função devolve uma lista dos identificadores da BASICAVL recebida como parâmetro, onde *tot* é o número total de vendas válidas e *size* é o tamanho da lista retornada. Função utilizada para responder ao objetivo nº2.

3.2 Gestão Filial (*avl filial.c*)

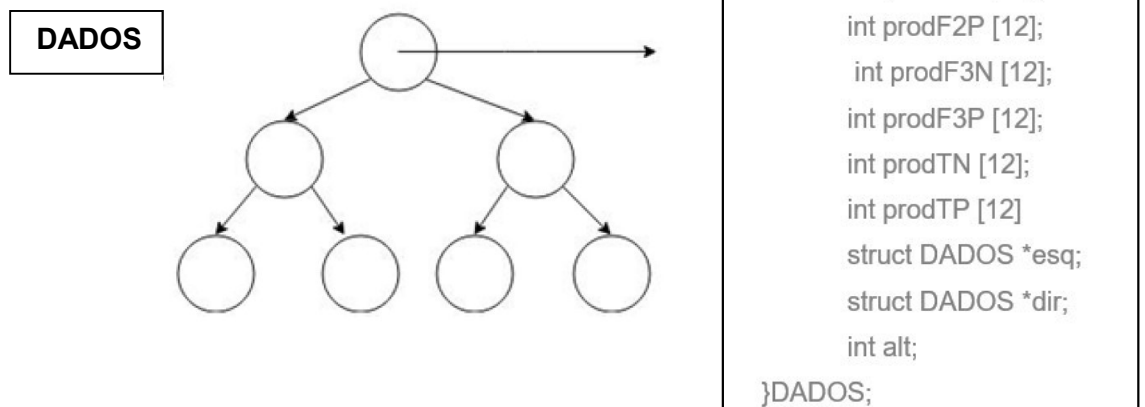
3.2.1 Estrutura

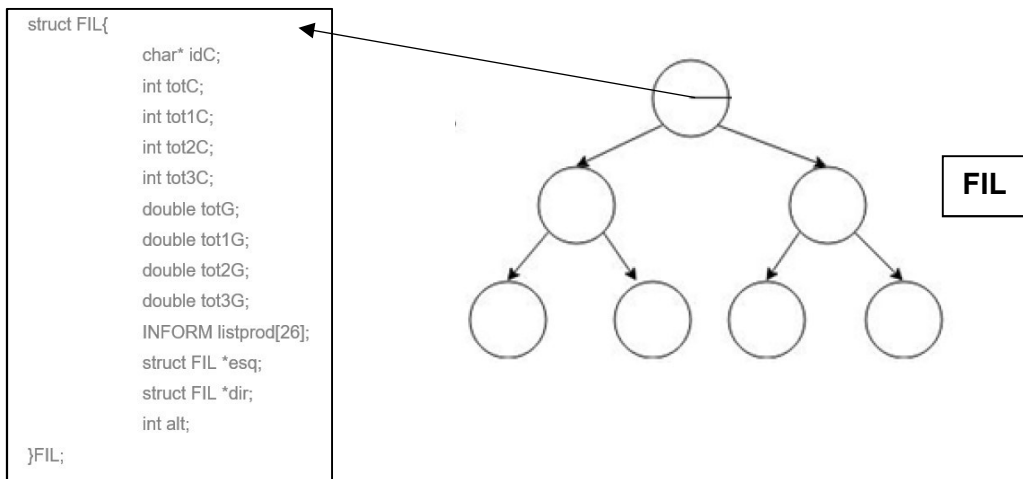
No ficheiro *avl_filial.h* definimos o seguinte tipo de dados

```
typedef struct DADOS* INFORM;
```

```
typedef struct FIL* AVLFILIAL;
```

No ficheiro *avl_filial.c* definimos o seguinte tipo de dados:





A nossa gestão da filial está definida como uma AVLFILIAL em que uma AVLFILIAL é uma árvore balanceada (FIL) em que em cada nodo tem um identificador do cliente (char* idC), quantidade total de unidades que um dado cliente comprou (int totC), quantidade total de unidades que um dado cliente comprou pela diferentes filiais (int tot1C, int tot2C, int tot3C), a quantidade total gasta por um cliente (double totG), quantidade total gasta que um dado cliente comprou pelas diferentes filiais (double tot1G, double tot2G, double tot3G), um array de 26 posições (em que cada posição corresponde a uma letra do alfabeto) de INFORM que são todos os produtos que foram comprados por um dado cliente, dois apontadores para FIL (esq e dir) e a altura (int alt). Um INFORM é uma árvore balanceada, em que cada nodo tem um identificador do produto que foi comprado (char* idP), 8 arrays de inteiros de 12 posições (em que cada posição corresponde a um mês), o normal e o promoção a nível geral e de cada filial preenchidos em cada posição pelo número de unidades compradas desse produto no mês respetivo, contém também a quantidade total comprada deste produto e a quantidade gasta do produto na totalidade (int quantT e double quantG, respetivamente), dois apontadores para DADOS (esq e dir) e a altura (int alt).

3.2.2 Funções presentes

Neste módulo para além das funções habituais presentes nas AVL's temos as seguintes funções:

```
char** criaArrayFi(AVLFILIAL aFi[], int to, int* ta);
```

Esta função devolve uma lista dos clientes que compraram em todas as filiais de uma dada lista de AVLFILIAL recebida como parâmetro, onde tot é o número total de clientes registados e ta é o tamanho da lista retornada. Função utilizada para responder ao objetivo nº5.

```
int ngmCompra (AVLFILIAL aFi);
```

Esta função devolve o número de clientes que não realizaram nenhuma compra de uma dada AVLFILIAL recebida como parâmetro. Função utilizada para responder ao objetivo nº6.

```
void getCompras(AVLFILIAL fi, char cli[], int totCF1[], int totCF2[], int totCF3[]);
```

Esta função preenche os arrays recebidos totCF1, totCF2, totCF3 com o total de unidades mensal compradas por um dado cliente recebido como parâmetro comprou pelas diferentes filiais numa determinada AVLFILIAL. Função utilizada para responder ao objetivo nº8.

```
char** inicArrayCliN(AVLFILIAL fi[], int max, int* tam, char* produto, int f);
```

Esta função devolve a lista dos clientes que compraram um determinado produto recebido como parâmetro em modo normal numa determinada filial também esta recebida como parâmetro, onde max é o número total de produtos válidos e tam é o tamanho da lista retornadas. Função utilizada para responder ao objetivo nº9.

```
char** inicArrayCliP(AVLFILIAL fi[], int max, int* tam, char* produto, int f);
```

Esta função devolve a lista dos clientes que compraram um determinado produto recebido como parâmetro em modo promoção numa determinada filial também esta recebida como parâmetro, onde max é o número total de produtos válidos e tam é o tamanho da lista retornada. Função utilizada para responder ao objetivo nº9.

```
char** inspector(AVLFILIAL aFi, char c[], int m, int* lTot);
```

Esta função devolve a lista de produtos ordenados pela quantidade que um dado cliente comprou num determinado mês. Função utilizada para responder ao objetivo nº10. Por motivos que não conseguimos explicar, as listas das quantidades respetivas não são retornadas apesar de no decorrer desta função conter os valores correspondentes e ajudar na ordenação dos produtos para resolver este problema decidimos criar uma outra função, abaixo descrita, para complementar e atingir o objetivo nº10 na totalidade.

```
int* encontra(AVLFILIAL afi,char c[],int m,char** lProd);
```

Esta função devolve a lista de quantidades respetivas à lista de produtos recebida como parâmetro que um dado cliente comprou num determinado mês. Função utilizada para atingir o objetivo nº10 na totalidade.

```
int percorreCli(AVLFILIAL afi, char* codigo, int ind, int fil);
```

Esta função retorna o número de clientes que compraram um determinado produto numa dada filial. Função utilizada para responder parcialmente ao objetivo nº11 pois esta função retorna para certos códigos o valor 0, valor este sem sentido e incorreto consequentemente não conseguimos solucionar este problema e explicá-lo.

```
void top3C(AVLFILIAL fi,char* cl,char* aPr[],int aTPr[]);
```

Esta função escreve em aPr a lista dos 3 produtos mais comprados por um dado cliente recebido como parâmetro e também escreve em aTPr a quantidade desses 3 produtos mais comprados. Função utilizada para responder ao objetivo nº12.

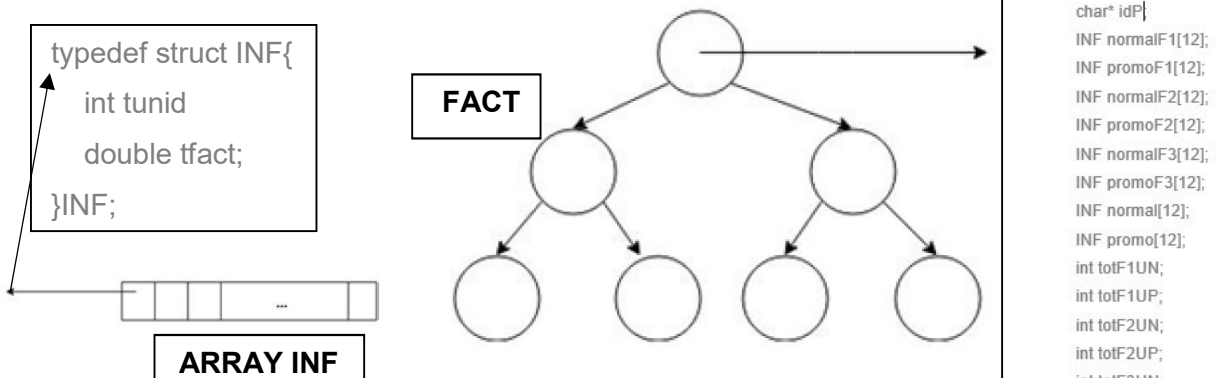
3.3 Faturação global (avl facturacao.c)

3.3.1 Estrutura

No ficheiro avl_facturacao.h definimos o seguinte tipo de dados:

```
typedef struct FACT* AVLFACTURACAO
```

No ficheiro avl_facturacao.c definimos o seguinte tipo de dados:



A nossa faturação global está definida como uma AVLFACTURACAO em que uma AVLFACTURACAO é uma árvore balanceada (FACT) em que em cada nodo tem um identificador do produto (char* idP), 8 inteiros que são os total de unidades anuais para este produto que foram vendidas nos dois tipos de vendas (int totUN e int totUP) os restantes 6 inteiros são os total de unidades anuais nos dois tipos de vendas para cada filial (int totF1UN, totF1UP, totF2UN, totF2UP, totF3UN, totF3UP), 8 doubles que são o total facturado anual de um dado produto nos dois tipos de vendas no global (double totFN e double totFP) e no caso de cada filial (double totF1FN, totF1FP, totF2FN, totF2FP, totF3FN, totF3FP), 8 arrays com 12 posições (meses do ano) de INF's, dois apontadores para FACT(esq e dir) e uma altura (int alt). Os arrays de INF's contém dois números que serão o total de vendas e a faturação para este produto por mês.

3.3.2 Funções Presentes

Neste módulo para além das funções habituais presentes nas AVL's temos as seguintes funções:

```
void conta_Fact1(AVLFACTURACAO af,char* i, int m,double res[])
```

Esta função que retorna através de res quantas unidades e quanto foi faturado nos dois tipos de vendas um determinado produto num determinado mês numa AVLFACTURACAO. Função utilizada para atingir uma parte do objetivo nº3.

```
void conta_Fact2(AVLFACTURACAO af,char* i, int m,double res[])
```

Esta função que retorna através de res quantas unidades e quanto foi faturado nos dois tipos de vendas nas diferentes filiais um determinado produto num determinado mês numa AVLFACTURACAO. Função utilizada para atingir na totalidade o objetivo nº3.

```
char** criaArrayF(AVLFACTURACAO af[],int total,int* tam)
```

Esta função devolve uma lista com todos os produtos que não foram vendidos, ou seja, o total faturado anual destes produtos é 0 numa determinada lista de AVLFACTURACAO, onde tam é o tamanho da lista devolvida e total é total de produtos registados válidos . Função utilizada para atingir uma parte do objetivo nº4.

```
int comprasEntre(AVLFACTURACAO af, int ini, int fim)
```

Esta função retorna o número de unidades vendidas entre um mês e outro (int ini e fim) numa determinada AVLFACTURACAO. Função utilizada para atingir parcialmente o objetivo nº8.

```
int facturacaoEntre(AVLFACTURACAO af, int ini, int fim)
```

Esta função retorna o total faturado entre um mês e outro (int ini e fim) numa determinada AVLFACTURACAO. Função utilizada para atingir na totalidade o objetivo nº8.

```
int percorreProd3(AVLFACTURACAO a,char** li, int un[], int ind, int n)
```

Esta função insere e ordena todos os produtos mais vendidos na filial 3 de uma dada AVLFACTURACAO num array de n elementos com quantidade total de unidades vendidas (int un[]) e num outro array de n elementos com os códigos dos produtos (int li[]), a posição dos dois arrays estão relacionados. Função utilizada para uma parte do objetivo nº11.

```
int percorreProd2(AVLFACTURACAO a, char** li, int un[], int ind, int n)
```

Esta função insere e ordena todos os produtos mais vendidos na filial 2 de uma dada AVLFACTURACAO num array de n elementos com quantidade total de unidades vendidas (int un[]) e num outro array de n elementos com os códigos dos produtos (int li[]), a posição dos dois arrays estão relacionados. Função utilizada para complementar a função utilizada anteriormente para atingir objetivo nº11.

```
int percorreProd1(AVLFACTURACAO a, char** li, int un[], int ind, int n)
```

Esta função insere e ordena todos os produtos mais vendidos na filial 1 de uma dada AVLFACTURACAO num array de n elementos com quantidade total de unidades vendidas (int un[]) e num outro array de n elementos com os códigos dos produtos (int li[]), a posição dos dois arrays estão relacionados. Função utilizada para atingir a totalidade o objetivo nº11.

3.4 Interface

```
void menuInicial()
```

Esta função permite que o utilizador faça a interação com o programa podendo escolher entre ficheiros predefinidos (são utilizados os ficheiros fornecidos pelos docentes) ou novos ficheiros consequentemente é lido a sua opção e de seguida armazenamos a informação contida nos ficheiros indicados nas estruturas de dados acima referidas e utilizadas.

```
void menuQueries()
```

Esta função apresenta ao utilizador as diferentes queries (os objetivos que apresentados na introdução) podendo este escolher qualquer uma delas, apresentando de imediato o resultado ou pedindo mais informação ao utilizador do que pretende visualizar para depois ser apresentado. No fim dos resultados retoma a esta função em qualquer uma das queries.

```
int menu()
```

Esta função apresenta o menu inicial ao utilizador caso o utilizador pretenda terminar o programa retorna 0 e acaba o programa. Esta função liga as duas funções acima apresentadas.

3.5 Leitura

No ficheiro leitura.h definimos a seguinte função `typedef struct ler*` `Leitura`

```
struct ler{
    char* nome;
    int linhas;
    int lval;
    int lmal;
    double tempo;
};
```

Esta estrutura foi criada para auxiliar a leitura de um ficheiro (objetivo nº1) onde é guardado nesta o nome do ficheiro que foi lido, o número de linhas lidas, o número de linhas válidas, o número de linhas inválidas e o tempo que demorou a ler todas as linhas do ficheiro. Como são lidos 3 ficheiros (Produtos, Clientes, Vendas) cada um tem a sua estrutura de leitura que se mantém ao longo do decorrer do programa, em variáveis globais. No momento que o utilizador pretende ler novos ficheiros os dados armazenados nestas estruturas são libertados. Este ficheiro contém as funções típicas para aceder aos dados da estrutura, libertar espaço como criá-lo.

Além destas funções o ficheiro leitura.h também contém as seguintes funções:

```
void lerFpers()
```

Esta função interage com o utilizador perguntando qual o nome dos 3 ficheiros que pretende ler e preencher as estruturas. Os nomes dos ficheiros indicados são guardados nas respetivas estruturas de leitura acima descritas.

```
void lerFpred()
```

Esta função indica que o utilizador pretende usar os ficheiros disponibilizados pelos docentes(Produtos.txt, Clientes.txt, Vendas_1M.txt) para preencher as estruturas de dados. O nome destes ficheiros é armazenado na estrutura reservada para o tipo em causa.

```
void lerClientes(Leitura cli)
```

Esta função recebe uma estrutura Leitura que contém o nome do ficheiro dos Clientes e insere na estrutura de clientes (AVLBASE) e na gestão da filial (AVLFILIAL) a informação obtida na leitura de uma linha deste ficheiro após ser validada, também guarda na estrutura recebida o número de linhas lidas, válidas e incorretas. Guardamos a informação na AVLFILIAL para saber se o cliente posteriormente realizou alguma compra e numa AVLBASE para saber quais os clientes registados que poderão fazer alguma compra no futuro.

```
void lerProdutos(Leitura prod)
```

Esta função recebe uma estrutura Leitura que contém o nome do ficheiro dos Produtos e insere na estrutura de clientes (AVLBASE) e na faturação global (AVLFACTURACAO) a informação obtida na leitura de uma linha deste ficheiro após ser validada, também guarda na estrutura recebida o número de linhas lidas, válidas e incorretas. Guardamos a informação na AVLFACTURACAO para saber se o produto posteriormente terá alguma venda e numa AVLBASE para saber quais os produtos registados que podem ser vendidos num futuro próximo.

```
void lerVendas(Leitura vend)
```

Esta função recebe uma estrutura Leitura que contém o nome do ficheiro das Vendas e insere na gestão da filial (AVLFILIAL) e na faturação global (AVLFACTURACAO) a informação obtida na leitura de uma linha deste ficheiro após ser validada, também guarda na estrutura recebida o número de linhas lidas, válidas e incorretas. Esta função vai atualizar a estrutura de faturação e gestão, previamente inicializadas.

2.6 Navega

No ficheiro navega.h definimos a seguinte função:

```
void navegador(char** array, int tam, int z)
```

Esta função recebe uma lista com os identificadores com tamanho *tam* e uma flag *z*, esta função permite ao utilizador ver de 13 em 13 os identificadores

que se encontram no array como pode avançar na leitura como também pode retroceder além destas opções pode sair se não pretender ver mais identificadores. Esta função no início imprime o total de identificadores que possui em array. A “flag” z se for 1 limpa o terminal antes de apresentar os primeiros 13 indentificadores. Esta função é utilizada para auxiliar a visualização dos resultados das queries(objetivos) nº2,4,5 e 9.

3.7 Queries

Neste ficheiro os números das queries corresponde aos objetivos pretendidos. No ficheiro queries.h definimos o seguinte tipo de dados:

```
typedef struct cp* ConjProds;
```

Esta estrutura possui uma lista com indentificadores para produtos, uma lista de unidades, uma lista de identificadores dedicado para os clientes como estas listas possuem o mesmo tamanho também é guardado esta variável. Esta estrutura foi utilizada para atingir o objetivo nº11.

Neste ficheiro também definimos as funções:

```
void querie0Cli()
```

Esta função testa se já ocorreu alguma leitura, ou seja, as estruturas anteriormente referidas encontram-se com informação caso este teste seja correto liberta a memória dedicada a estas estruturas. Esta libertação não se encontra a funcionar por motivos que não conseguimos explicar logo a releitura de ficheiros de Clientes fica comprometida e consequentemente o objetivo nº1 não é atingido na totalidade. Contudo caso o teste anteriormente referido ser negativo esta função lê o ficheiro dos clientes sem libertar qualquer estrutura. Apresentando quando clientes foram registados.

```
void querie0Pro()
```

Esta função testa se já ocorreu alguma leitura, ou seja, as estruturas anteriormente referidas encontram-se com informação caso este teste seja correto liberta a memória dedicada a estas estruturas. Esta libertação não se encontra a funcionar por motivos que não conseguimos explicar logo a releitura de ficheiros de Produtos fica comprometida e consequentemente o objetivo nº1 não é atingido na totalidade. Contudo caso o teste anteriormente referido ser negativo esta função lê o ficheiro dos produtos sem libertar qualquer estrutura. Apresentando quando produtos foram registados.

```
void querie0Ven()
```

Esta simplesmente lê o ficheiro de Vendas que se encontra na estrutura de Leitura designada a ela. Apresentando quando vendas válidas foram registadas.

```
void querie1Cli()
```

Esta função apresenta ao utilizador mais detalhes sobre a leitura do ficheiro de Clientes.

```
void querie1Pro()
```

Esta função apresenta ao utilizador mais detalhes sobre a leitura do ficheiro de Produtos.

```
void querie1Ven()
```

Esta função apresenta ao utilizador mais detalhes sobre a leitura do ficheiro de Vendas.

```
void querie2()
```

Função que após receber uma letra do utilizador, apresenta uma lista com todos os códigos de produtos iniciados por esta letra. Para apresentar esta lista acedemos à posição corresponde no array de AVLBASE's, percorremos a AVLBASE correspondente criando uma lista com o nome dos produtos e por fim utilizamos a função navegador referida acima para apresentar a lista criada.

```
void querie3()
```

Função que após receber um mês e um código de produto válido do utilizador dá a escolher a este se pretende ver os resultados globais ou filial a filial desse produto e posteriormente apresenta o número total de vendas e o total faturado nesse mês mediante a escolha do utilizador apresenta mais ou menos detalhes. Para a resolução deste problema tirámos vantagem de guardarmos os aglomerados mensais dos vários produtos na nossa estrutura de facturacao. Sendo

assim apenas vamos buscar o aglomerado de compras e faturação à estrutura para o produto pedido no mês indicado.

`void querie4()`

Função apresenta todos os produtos que ninguém comprou. Para isso percorremos o nosso array de AVLFACTURACAO's e criamos uma lista com todos os produtos que não tiveram nenhuma unidade vendida e por fim utilizamos o nosso navegador para apresentar a lista criada.

`void querie5()`

Função apresenta todos os clientes que realizaram compras em todas as filiais, ou seja, o seu total gasto em qualquer filial por estes clientes é diferente de zero. Para isso percorremos o nosso array de AVLFILIAL's e criamos uma lista com todos esses clientes e por fim utilizamos o nosso navegador para apresentar essa lista ao utilizador.

`void querie6_0()`

Função apresenta o número total de clientes que não realizaram qualquer compra, ou seja, não compraram nenhum produto. Para isso percorremos o nosso array de AVLFILIAL's para verificar quais são os clientes que não têm nenhum produto registado (todas as posições da sua lista de produtos a NULL).

`void querie6_1()`

Função apresenta o número total de produtos que não foram vendidos, ou seja, total de unidades vendidas em qualquer modo de venda é zero. Para isso percorremos o nosso array de AVLFACTURACAO's para verificar quais são os produtos que não têm nenhuma venda realizada.

`void querie7()`

Função que após receber um código do cliente apresenta uma tabela dividida por filial com total de compras mensal por este cliente. Para isto utilizamos a estrutura de AVLFILIAL para aceder ao histórico de compras deste cliente, percorrendo a sua lista de compras somando todos os valores mensais (em modo normal e em promoção) guardando estes valores em 3 arrays (um para cada filial) de 12 posições o valor respetivo de compras mensalmente que efetuou.

`void querie8()`

Função que determina o total de faturação e o total de compras num intervalo de meses fechado dado pelo utilizador. Para isto utilizamos duas funções uma que devolve o total faturado e outra o total de compras entre intervalo indicado. Funções essas que vão percorrer o nosso array de AVLFACTURACAO's acendendo a todos os produtos somando os totais de compras mensais e os totais de faturas mensais entre os meses dados pelo utilizador.

`void querie9()`

Função que após receber um código de um produto e uma filial devolve duas listas uma com a lista de códigos dos clientes que compraram esse produto em modo Normal e outra com o código dos clientes que compraram esse mesmo produto em modo Promoção na filial indicada. Neste caso usámos novamente duas funções, uma para criar a lista de clientes que compraram em modo Normal e outra para a lista de clientes em modo Promoção. Para isto procuramos por o código de produto em todos os clientes registados no nosso array de AVLFILIAL's, caso não encontremos o produto significa que o cliente não comprou este produto e podemos avançar para o próximo cliente. No caso de encontrarmos o produto percorremos o array de compras mensal da filial correspondente deste produto procurando um valor diferente de zero, indicativo de uma compra nesse modo do produto em questão. Para apresentar as duas listas usamos novamente o navegador.

`void querie10()`

Dado um código de cliente e um mês esta função determina os produtos mais comprados por esse cliente nesse mês, sendo o código de cliente e o mês dado pelo utilizador. Após aceder ao cliente na AVLFILIAL, corremos todos os produtos comprados pelo cliente, verificando se o total comprado no mês em questão é maior que o valor guardado na posição (Total de Compras do cliente - 1) num array auxiliar, onde está guardada o número de unidades dos produtos mais comprados. Caso seja maior colocámos esse valor no local certo do array removendo o menor, executámos a mesma operação no array de códigos de produtos.

Caso seja menor avançamos para o próximo produto. Por motivos desconhecidos o número de unidades dos produtos não estava a ser retornado logo criamos uma outra função na lógica da anterior descrita, mas onde recebe a lista dos produtos já ordenados simplesmente vai buscar a quantidade do mês e poem no novo array.

```
void querie11()
```

Dado um dado número, número este inferior ao número de produtos registados, apresenta uma lista com n produtos mais vendidos com informação sobre o número de unidades vendidas e o número de clientes que compraram divididos por filial. Para isso criamos 9 listas de n elementos, 3 para nome dos produtos para cada filial, 3 para o número de unidades vendidas para cada filial e 3 para o número de clientes para cada filial. Numa primeira fase percorremos o nosso array de AVLFACTURACAO's para cada filial onde vamos preencher a lista de produtos de cada filial com os produtos que tiverem mais unidades vendidas juntamente preenchemos a lista com as quantidades respetivas para isso preenchemos até ter a listas cheia de elementos, ordenando à medida que se insere, no momento que temos a lista cheia comparamos com o ultimo elemento da lista e assim sucessivamente até não ter nenhum produto para inserir. Após isso já temos uma lista com n produtos mais comprados assim pegando em cada posição dessa lista vamos percorrer o nosso array de AVLFILIAL's e testando se o produto em causa para cada posição encontra-se na lista de compras de um dado cliente e vamos somando até não existir mais nesse momento é vez de analisar o segundo produto mais vendido da mesma forma e assim sucessivamente. No final, todas essas listas são inseridas na estrutura de dados ConjProds para facilitar sua apresentação ao utilizador, apresentação esta que permite ao cliente ver de 13 em 13 os produtos um navegador um pouco mais simples e com mais informação. Esta querie não se encontra totalmente certa pois existem valores de clientes a 0 o que não tem sentido, contudo não conseguimos perceber onde está o erro ou o porquê de tal estar a acontecer.

```
void querie12()
```

Esta função após receber um código do cliente indica os 3 produtos que o cliente mais comprou durante um ano. Para isso utilizamos dois arrays de 3 elementos um dedicado para os produtos e outra para as quantidades respetivas, seguindo o raciocínio da querie10() que é percorrer toda a lista de produtos que o dado cliente tem comparando a sua quantidade gasta global de um dado produto com a existente na última posição do array dedicado a quantidade caso se verifica que a nova quantidade é superior a ultima posição dos dois arrays é trocada pelo nome do novo produto encontrado e pela quantidade gasta respetiva. Posteriormente ordena-se o array de forma a garantir que na última posição se encontra a menor quantidade gasta global de 3 possíveis.

3.8 Teste

No ficheiro teste.h definimos as seguintes funções:

```
int valida_filial(int filial)
```

Esta função testa se uma dada filial recebida é válida retornando 1 caso seja esse o caso ou 0 caso contrário.

```
int valida_mes(int mes)
```

Esta função testa se um dado mês recebido é válido retornando 1 caso seja esse o caso ou 0 caso contrário.

```
int valida_cliente(char* cliente)
```

Esta função testa se um dado cliente recebido é válido retornando 1 caso seja esse o caso ou 0 caso contrário. O filtro utilizado para validar um cliente foi fornecido pelos docentes anteriormente.

```
int valida_tipo(char tipo)
```

Esta função testa se um dado tipo de compra recebido é válido retornando 1 caso seja esse o caso ou 0 caso contrário.

```
int valida_unidades(int unidades)
```

Esta função testa se uma dada quantidade recebida é válida retornando 1 caso seja esse o caso ou 0 caso contrário. Existe um limite máximo de unidades.

```
int valida_preco(double preco)
```

Esta função testa se um dado preço recebido é válido retornando 1 caso seja esse o caso ou 0 caso contrário. O preço não pode não ultrassapar um certo valor.

```
int valida_produto(char* produto)
```

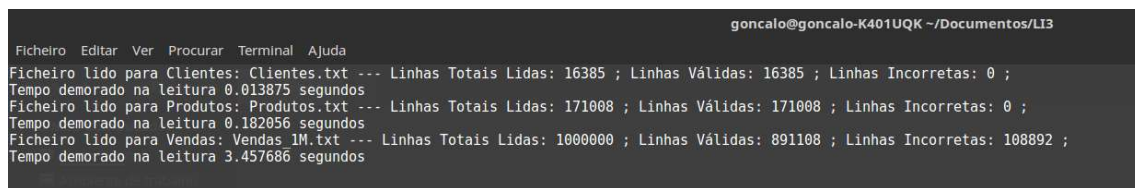
Esta função testa se um dado produto recebido é válido retornando 1 caso seja esse o caso ou 0 caso contrário. O filtro utilizado para validar um produto foi fornecido pelos docentes anteriormente.

```
int testeVenda(char* produto, double preco, int unidades, char tipo, char* cliente, int mes, int filial)
```

Esta função testa se um conjunto de valores cumpre uma venda válida utilizando as funções anteriormente referidas retornando 1 caso seja uma venda válida e 0 caso contrário.

4.Resultados

Utilizando os ficheiros disponibilizados pelos docentes, o nossas estruturas demoram em média a preencher-se o tempo que se vê na imagem a seguir:



```
goncalo@goncalo-K401UQK ~/Documentos/LI3
Ficheiro  Editar  Ver  Procurar  Terminal  Ajuda
Ficheiro lido para Clientes: Clientes.txt --- Linhas Totais Lidas: 16385 ; Linhas Válidas: 16385 ; Linhas Incorretas: 0 ;
Tempo demorado na leitura 0.013875 segundos
Ficheiro lido para Produtos: Produtos.txt --- Linhas Totais Lidas: 171008 ; Linhas Válidas: 171008 ; Linhas Incorretas: 0 ;
Tempo demorado na leitura 0.182056 segundos
Ficheiro lido para Vendas: Vendas 1M.txt --- Linhas Totais Lidas: 1000000 ; Linhas Válidas: 891108 ; Linhas Incorretas: 108892 ;
Tempo demorado na leitura 3.457686 segundos
```

5.Conclusão

O presente relatório descreveu, de forma sucinta, as principais componentes da aplicação desenvolvida para a avaliação do projeto C da unidade curricular de LI3.

Considera-se que os principais objetivos foram cumpridos com a exceção da releitura de ficheiros e a querie11 apresentar problemas anteriormente descritos.

Alguns aspetos a nível da interface poderiam ter sido melhorados bem como a otimização de algumas queries de modo a acederem apenas uma vez as estruturas em vez de duas, por exemplo.

Sentimos também que a realização deste projeto consolidou os nossos conhecimentos da linguagem C, nomeadamente na modularidade e encapsulamento de dados.