Laboratórios de Informática III

***

Docentes: Fernando Martins
Vitor Fonte

Desenvolvedores:

Daniel Cerveira Furtado Malhadas
A72293

Joel Tomás Morais
A70841

Rui Pedro Mesquita Miranda
A75488

Tabela de Conteúdos

1.Introdução....3

1.1.Contextualização....3

1.2.Motivação e objetivos....3

1.3.Estrutura do relativo....4

2.Estruturas....5

2.1.Catálogo....5

2.2.Faturação....8

2.3.Gestão....11

2.4.Apresentação....15

2.5.Cápsula....16

2.5.Compra....18

3.Interface....20

3.1.Utilização....20

4.Testes de performance e Análise de Resultados....21

4.1.Leitura dos ficheiros....21

4.2.Mapa e Performance de Queries....22

5.Makefile....24

5.1.Código....24

5.2.Grafo de dependências....25

6.Conclusões e sugestões....27

1.Introdução

***

1.1.Contextualização

O presente relatório serve de apoio à compreensão do projeto de Laboratórios de Informática realizado no presente ano letivo 2015/2016.

1.2.Motivação e objetivos

Com este projeto pretende-se desenvolver um sistema capaz de emular o funcionamento de um supermercado devendo ser capaz de:

- Armazenar todos os produtos;
- Armazenar todos os clientes;
- Reconhecer 3 filiais do supermercado em questão;
- Reconhecer dois tipos de regimes de vendas, N de normal e P de promoção;
- Armazenar quantidade global ou por filial de compras em que participa cada produto e cada cliente participam, quantidade vendida ou comprada e faturação;
- Registar ralação de compra entre produto e cliente assim como entre cliente e produto.

Para organizar estes dados criaram-se 3 módulos:

- Catálogo;
- Faturação;
- Gestão.

Tendo em conta esta organização dos dados (sendo que cada módulo será melhor explicado no próximo capítulo) o nosso sistema terá que suportar as seguintes funcionalidades, a que chamaremos de queries:

- ➔ Possibilitar a inserção e reinserção dos ficheiros;
- ➔ Obter ter todos produtos de uma determinada letra;
- ➔ Obter dados das vendas e faturação de um produto globais ou de uma determinada filial sobre um mês específico;
- ➔ Obter a lista global ou por filial dos produtos que nunca foram comprados;
- ➔ Ver uma tabela com o número de produtos por mês comprados por um determinado cliente;
- ➔ Obter o total de vendas e o total faturado entre um determinado intervalo de meses;
- ➔ Obter todos os clientes que realizaram compras em todas as filiais;
- ➔ Obter os clientes que compraram determinado produto em determinada filial;
- ➔ Obter os produtos mais comprados durante um mês por um determinado cliente;
- ➔ Obter a lista do N produtos mais vendidos durante o ano e dados sobre os mesmos;
- ➔ Obter os N produtos em que determinado cliente gastou mais dinheiro;
- ➔ Ver o número de clientes que nunca realizaram nenhuma compra e o número de produtos que nunca foi comprado.

1.3.Estrutura do relatório

No decorrer deste relatório tivemos em consideração duas questões que entendemos ser fundamentais e que explicamos de seguida:

- Estruturação:

Entendemos que não faz sentido começar imediatamente a construir "qualquer coisa" ou rápidamente começar a pensar nas mais complicadas maneiras de chegar ao fim se isso apenas significa ter algo mal pensado e mal estruturado que terá que vir a ser mudado no futuro. Não interessa quão bons sejam os programadores ou quão complexo seja o modelo do projeto que terão criado, se não tiver sido bem estruturado de raiz então apenas irá dar azo a uma incessante necessidade de modificações que poderão vir a pôr em risco a integridade do que já estava feito, criando necessidade de refazer partes significativas do projeto a longo prazo com a falsa esperança de "remediar" problemas que poderiam ter sido facilmente evitados numa fase de análise inicial. Isso é um exemplo de má estruturação e análise dos requisitos que sabemos ter de evitar a qualquer custo, visto que qualquer tempo "perdido" nesta fase será um indubitável ganho exponêncial em tempo na implementação. Por esta razão temos a noção da enorme importância que este primeiro obstáculo tem e tivemos todo o cuidado para o levar a sério e para o não apressar e isso demonstramos nas estruturas que desenvolvemos já explicitadas no próximo capítulo.

- Simplicidade:

É de grande importância para nós que a mensagem que desejamos transimitir com este relatório seja percebida na sua integridade sem que haja possibilidade de interpretações erradas ou alternativas. Sabendo nós que grande parte dos problemas com o software hoje em dia baseia-se na má comunicação entre os desenvolvedores e os seus empregadores quisemos que todos os esquemas e explicações neste relatório fossem sucintos, diretos ao assunto e simples de perceber. Desta forma emulamos a estrutura de um relatório que poderia ser relativo a um trabalho no mundo de tabalho fora da Universidade e onde o nosso empregador, mesmo com poucos conhecimentos na área seria, mesmo assim, capaz de ler, perceber e fazer uma crítica constructiva sobre a direcção do projeto. Isto permite evitar interpretações erradas do objetivo por parte dos programadores e também permite ao empregador compreender melhor se realmente os seus objectivos podem ser todos realizados da forma que idealizou ou se são demasiado optimistas.

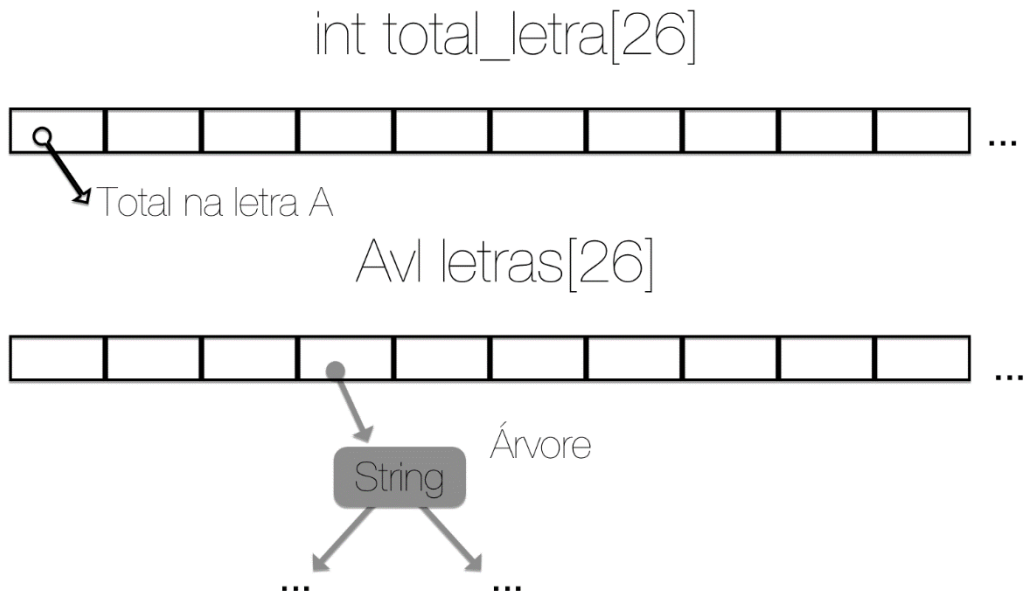
2.Estruturas

***

2.1.Catálogo

Módulo de dados que, a partir dos ficheiros lidos, **contém todos os produtos ou clientes importados**. Os elementos estão organizados em árvores balanceadas, 26 árvores ao certo, sendo que cada árvore irá conter elementos somente de uma letra, existindo por isso tantas árvores quanto letras do abcedário. Estas **àrvores podem ser acedidas através de um array**, sendo que cada índice do array corresponde à árvore com a letra de igual número no abcedário (**sendo que consideramos os números a começar por 0**).

Segue-se uma representação gráfica da estrutura em questão



Segue-se de seguida o respectivo .h e documentação

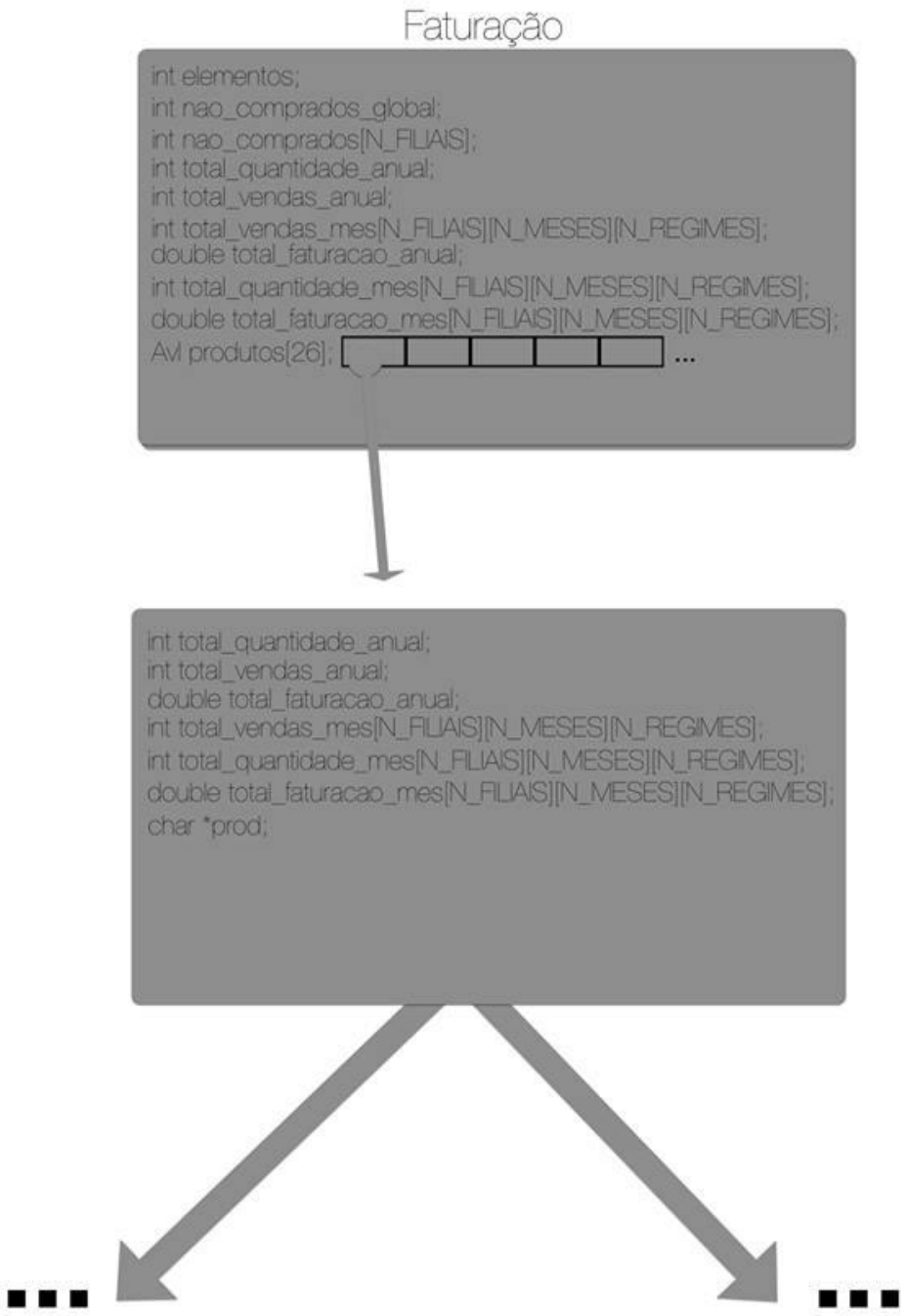
```
#ifndef Catalogo_H
#define Catalogo_H
#include "../modulos_auxiliares/apresentacao/apresentacao.h"
#include "../modulos_auxiliares/avl/avl.h" /*http://adtinfo.org/index.html*/
#include "../../tipos/tipos.h"
typedef struct catalogo *Cat; /*Tipo opaco para catalogo*/
/*Modulo*****************************************************************/
/*Inicializar o catalogo*/
Cat  init_catalogo       ();
```

```
/*Inserir elemento no catalogo. Pode ser uma string com qualquer tamanho*/
Cat  inserir             (Cat c, ELEMENTO str);
/*Remover elemento do catalogo.*/
Cat  remover             (Cat c, ELEMENTO str);
/*Verificar a existência de um elementos no catalogo*/
int  existe              (Cat c, ELEMENTO str);
/*Obter o número de elementos que comecem pela letra argumento*/
int  num_elementos_letra (Cat c, char l );
/*Obter o número de elementos total inserido no catalogo*/
int  num_elementos       (Cat c);
/*eliminar espaço alocado para o catalogo*/
void libertar_catalogo   (Cat c);
/*Queries*****************************************************************/
/* preenche apresentacao com todos os elementos que obdecem ao filtro.
 * Se o filtro for o caracter * então todos os elementos do catalogo
 * são inseridos na apresentação. Caso o filtro seja uma letra então
 * apenas os elementos que comecem por essa mesma letra serão inseridos*/
Apresentacao preenche_apresentacao_catalogo (Cat c, char filtro);
/*Apresentacao************************************************************/
/*Funções para utilização do modulo importado: apresentacao.*/
/* Obter o elemento da apresentação que está na pagina e na linha argumentos.
 * Devolve NULL se esse elemento não esxistir.*/
char*  apresenta_elemento_catalogo   (Apresentacao a, int pagina, int linha);
/*Obter a página argumento da apresentação. Devolve NULL se essa página não
existir*/
char** get_pagina_catalogo           (Apresentacao a, int pagina);
/*Obter o número de elementos por página da apresentacao*/
int    get_elem_por_pagina_catalogo  (Apresentacao a);
/*Obter o numero de paginas da apresentacao*/
int    get_paginas_catalogo          (Apresentacao a);
/*Obter o número de elementos na apresentacao*/
int    get_elem_total_catalogo       (Apresentacao a);
/*Eliminar espaço alocado para a apresentacao*/
void   elimina_apresentacao_catalogo (Apresentacao a);
/*************************************************************************/
#endif
```

2.2.Faturação

Módulo de dados que irá conter as estruturas de dados responsáveis pela resposta eficiente a questões quantitativas que **relacionam os produtos às suas vendas mensais, em modo Normal (N) ou em Promoção (P)**, para cada um dos casos **guardando o número de vendas e o valor total de facturação** de cada um destes tipos. Este módulo deve referenciar todos os produtos, mesmo os que nunca foram vendidos. Este módulo não contém qualquer referência a clientes, mas deve ser capaz de distinguir os valores obtidos **em cada filial.** Este módulo contém um **array** com **árvores balanceadas**, exactamente como no módulo catalogo, com o objetivo de tornar mais rápido e fácil o acesso a cada um dos elementos pretendidos. Estas **árvores irão connter todos os produtos e os dados sobre eles referidos anteriormente**.

Segue-se uma representação gráfica desta estrutura



Segue-se o respetivo .h e documentação

```
#ifndef Faturacao_H

#define Faturacao_H

#include "../modulos_auxiliares/apresentacao/apresentacao.h"

#include "../modulos_auxiliares/capsula/capsula.h"

#include"../modulos_auxiliares/avl/avl.h"/*http://adtinfo.org/index.html*/

#include "../../tipos/tipos.h"

#define N_FILIAIS 3 /*Número máximo de filiais*/

#define N_REGIMES 2 /*Número máximo de regimes*/

#define N_MESES  12 /* Número máximo de meses pode ser aumentado caso se queira guardar dados ao longo de vários anos pode ser diminuído para um espaço de tempo mais pequeno*/

typedef struct faturacao *Faturacao; /*Tipo opaco de faturacao*/

/*Modulo*****************************************************************/

/*Inicializaçao da faturacao*/

Faturacao    init_faturacao            ();

/* Inserir compra na faturacao. Espera-se que os produtos estejam já todos inseridos modificando assim apenas os seus valores*/

Faturacao    inserir_compra_faturacao  (Faturacao f, PRODUTO cod, int mes, int tipo, int filial, int quant, float preco);

/*Inserir produto na faturacao sendo que todos os valores são inicializados a zero.*/

Faturacao    inserir_produto_faturacao (Faturacao f, PRODUTO cod);

/*Obter o número de produtos inseridos na faturacao*/

int          get_num_elementos         (Faturacao f);

/*Libertar o espaço alocado para a faturacao*/

void         libertar_faturacao        (Faturacao f);

/*Queries*****************************************************************/

/* Devolve uma cápsula com dados sobre o produto argumento, no mês argumento.
 * Caso o argumento filial seja NULL então os dados são globais, caso contrário
 * serão relativos á filial de número igual ao que esse argumento aponta.
 * A capsula contêm 4 inteiros e 2 doubles. sendo os inteiros (por ordem)
 * quantidade comprada em modo N, quantidade comprada em modo P, vendas em modo
 * N,vendas em modo P. Da mesma foram os doubles são (por ordem) faturacao em
 * modo N, faturacao em modo P. Consultar a API da capsula para entender como
 * aceder a estes valores.*/
```

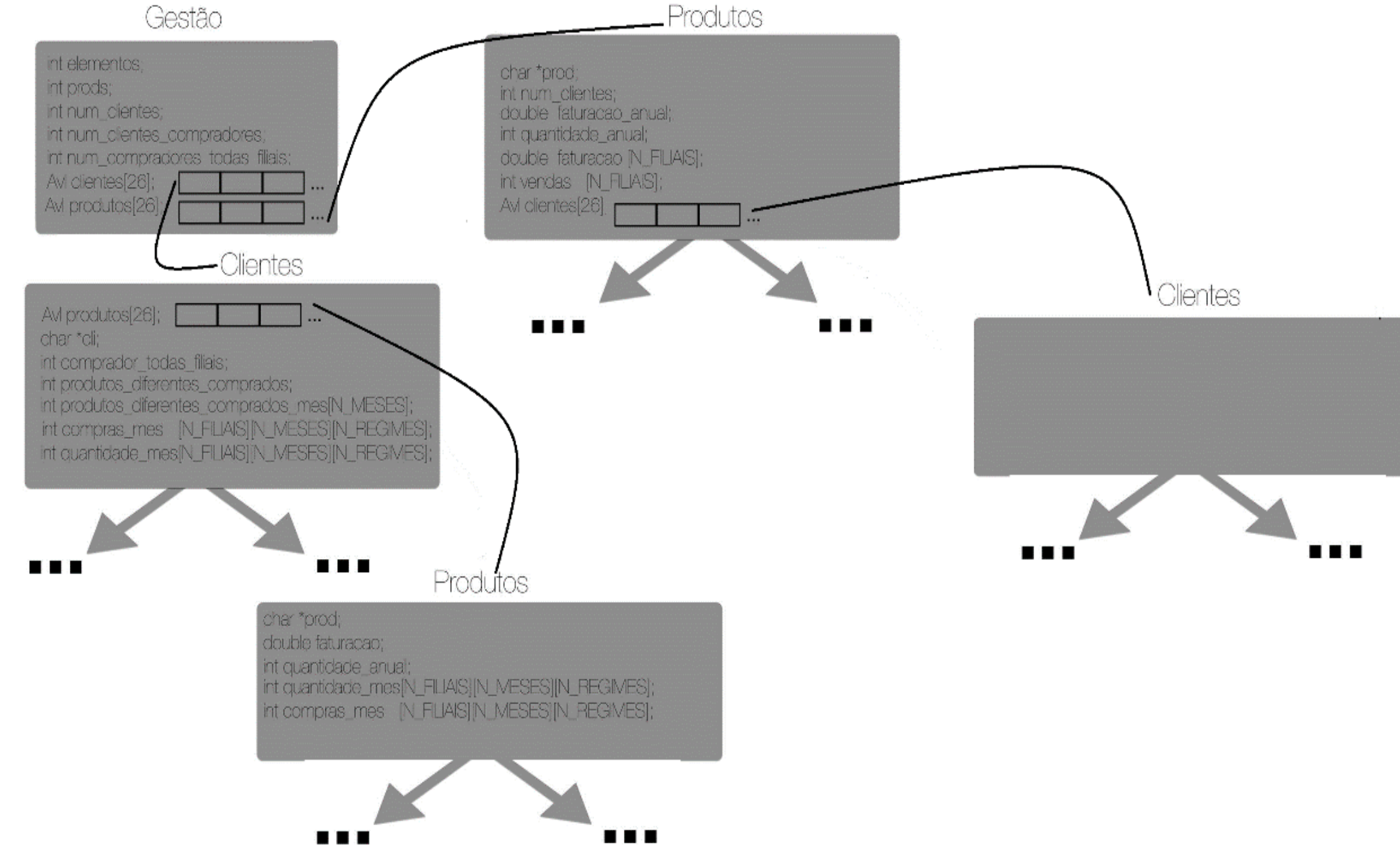
```
Capsula      total_dados_produto          (Faturacao f, int *filial, int mes,
PRODUTO cod);

/* Devolve uma cápsula com dados globais da faturacao entre os meses argumento.
 * Sendo mes1 o primeiro e mes2 o segundo, [mes1..mes2]. A capsula contêm 2
 * inteiros e 1 double. sendo os inteiros (por ordem) a quantidade global vendida
 * e o número de vendas global. Da mesma foram o double é a faturacao global.
 * Consultar a API da capsula para entender como aceder a estes valores.*/
Capsula      dados_globais_intervalo      (Faturacao f, int  mes1  , int mes2);

/*Devolve uma apresentacao que inclui os produtos nunca comprados.
* Caso o argumento filial seja NULL então os dados são globais, caso contrário
* serão relativos á filial de número igual ao que esse argumento aponta.*/
Apresentacao nunca_comprados              (Faturacao f, int *filial);

/*Obter o número de produtos que nunca foram comprados globalmente*/
int          get_num_nao_comprados_global (Faturacao f);

/*Apresentacao*****************************************************************/

/*Funções para utilização do modulo importado: apresentacao.*/

/* Obter o elemento da apresentação que está na pagina e na linha argumentos.
 * Devolve NULL se esse elemento não esxistir.*/
char*  apresenta_elemento_faturacao   (Apresentacao a, int pagina, int linha);

/*Obter a página argumento da apresentação. Devolve NULL se essa página não
existir*/
char** get_pagina_faturacao           (Apresentacao a, int pagina);

/*Obter o número de elementos por página da apresentacao*/
int    get_elem_por_pagina_faturacao  (Apresentacao a);

/*Obter o numero de paginas da apresentacao*/
int    get_paginas_faturacao          (Apresentacao a);

/*Obter o número de elementos na apresentacao*/
int    get_elem_total_faturacao       (Apresentacao a);

/*Eliminar espaço alocado para a apresentacao*/
void   elimina_apresentacao_faturacao (Apresentacao a);

/*Capsula**********************************************************************/

/*Funcoes para utilizacao do modulo importado: capsula*/

/*Obter o inteiro do indice argumento presente na capsula*/
int     get_int_capsula_faturacao    (Capsula c, int indice);

/*Obter o double do indice argumento presente na capsula*/
double  get_double_capsula_faturacao (Capsula c, int indice);

/*Libertar espaço alocado para a capsula*/
```

```
void    libertar_capsula_faturacao   (Capsula c);
/*obter o tipo da última inserção na cápsula*/
int     get_tipo_capsula_faturacao   (Capsula c);
/*obter a quantidade de inteiros presente na capsula*/
int     get_q1_capsula_faturacao     (Capsula c);
/*obter a quantidade de doubles presente na capsula*/
int     get_q2_capsula_faturacao     (Capsula c);
/*****************************************************************************/
#endif
```

2.3.Gestão

Módulo de dados que, a partir dos ficheiros lidos, conterá as estruturas de dados adequadas à representação dos relacionamentos, fundamentais para a aplicação, entre **produtos e clientes**, ou seja, **para cada produto, saber quais os clientes que o compraram, quantas unidades cada um comprou, em que mês e em que filial**. Para a estruturação optimizada dos dados deste módulo de dados será crucial analisar as queries que a aplicação deverá implementar, tendo sempre em atenção que pretendemos ter o histórico de vendas organizado por filiais para uma melhor análise, não esquecendo que **existem 3 filiais nesta cadeia**. Este módulo irá conter, na sua estrutura principal que funciona como hall de entrada para as outras estruturas, dois **arrays de árvores balanceadas** que irão funcionar exatamente da mesma forma que os arrays de árvores presentes nas estruturas dos módulos anteriores. Um destes arrays será **relativo a produtos** e o **outro a clientes**. As árvores do array relativo a produtos contido no nosso hall de entrada irá conter **todos os produtos** sem exceção, sendo que cada um deles conterá outro array de árvores balanceadas somente com os **clientes que o compraram**. As árvores do array relativo a clientes contido no nosso hall de entrada irá conter **todos os clientes** sem exceção, sendo que cada um deles conterá outro array de árvores balanceadas somente com os **produtos que comprou**. Estas estruturas, embora redundantes, com alguns dados que poderão consederar-se repetidos possibilitam uma performance a nível das queries acima do que seria esperado de qualquer outra forma, como se pode ver no próximo capítulo (capítulo 3).

Segue-se uma representação gráfica desta estrutura


Gestão
int elementos;
int prods;
int num_clientes;
int num_clientes_compradores;
int num_compradores_todas_filiais;
Avl clientes[26];
Avl produtos[26];
Produtos
char *prod;
int num_clientes;
double faturacao_anual;
int quantidade_anual;
double faturacao [N_FILIAIS];
int vendas [N_FILIAIS];
Avl clientes[26];
Clientes
Avl produtos[26];
char *cli;
int comprador_todas_filiais;
int produtos_diferentes_comprados;
int produtos_diferentes_comprados_mes[N_MESES];
int compras_mes [N_FILIAIS][N_MESES][N_REGIMES];
int quantidade_mes[N_FILIAIS][N_MESES][N_REGIMES];
Clientes
Produtos
char *prod;
double faturacao;
int quantidade_anual;
int quantidade_mes[N_FILIAIS][N_MESES][N_REGIMES];
int compras_mes [N_FILIAIS][N_MESES][N_REGIMES];

Segue-se o respetivo .h e documentação

```
#ifndef GESTAO_H

#define GESTAO_H

#include "../modulos_auxiliares/apresentacao/apresentacao.h"

#include "../modulos_auxiliares/capsula/capsula.h"

#include "../modulos_auxiliares/avl/avl.h" /*http://adtinfo.org/index.html*/

#include "../../tipos/tipos.h"

#define N_FILIAIS 3 /*Número máximo de filiais*/

#define N_REGIMES 2 /*Número máximo de regimes*/

#define N_MESES  12 /* Número máximo de meses. Pode ser aumentado caso se queira guardar dados ao longo de vários anos. Pode ser diminuído para um espaço de tempo mais pequeno*/

typedef struct gestao *Gestao; /*Tipo opaco de gestao*/

/*Modulo*****************************************************************/

/*Inicializar a gestao*/

Gestao init_gestao            ();

/*Inserir compra na gestão*/

Gestao inserir_compra_gestao  (Gestao g, CLIENTE cl, PRODUTO pr, int mes, int tipo, int filial, int quant, double preco);

/*Inserir cliente na gestão, o cliente é inserido com os seus valores inicializados a zero*/

Gestao inserir_cliente_gestao (Gestao g, CLIENTE cod);

/*Inserir produto na gestao, o produto é inserido com os seus valores inicializados a zero*/

Gestao inserir_produto_gestao (Gestao g, PRODUTO cod);

/*Libertar espaço alocado para a gestão*/

void   libertar_gestao        (Gestao g);

/*Obter número de produtos e clientes presente na gestão*/

int    num_elementos_gestao   (Gestao g);

/*Queries*****************************************************************/

/*Obter uma apresentação dos N produtos em que o cliente argumento gastou mais dinheiro*/

Apresentacao top_cliente              (Gestao g, CLIENTE cod, int N);

/* Obter uma apresentacao dos produtos que o cliente argumento comprou no mês argumento por ordem descendente de quantidade.*/

Apresentacao top_mes_descendente      (Gestao g, CLIENTE cod, int mes);
```

```
/*Obter uma apresentação dos clientes que compraram o produto argumento*/

Apresentacao clientes_produto          (Gestao g, PRODUTO cod);

/* Devolve uma cápsula com as quantidades compradas em cada mês, para cada filial
pelo cliente argumeento. A capsula contêm 12*N_FILIAIS inteiros sendo os
primeiros 12 a quantidade em cada mês (do 1 ao 12) na filial 1, os segundos 12
o mesmo na filial 2 e os terceiros o mesmo na filial 3. Consultar a API da
capsula para entender como aceder a estes valores.*/

Capsula      dados_cliente             (Gestao g, CLIENTE cod);

/*Obter uma apresentacao dos N produtos que venderam mais em quantidade.*/

Apresentacao top_produtos              (Gestao g, int N);

/*Obter uma apresentacao dos clientes que compraram em todas as filiais*/

Apresentacao compradores_todas_filiais (Gestao g);

/*Obter o número de clientes na gestao que são compradores*/

int          num_clientes_gestao       (Gestao g);

/*Apresentacao*****************************************************************/

/*Funções para utilização do modulo importado: apresentacao.*/

/* Obter o elemento da apresentação que está na pagina e na linha argumentos.
 * Devolve NULL se esse elemento não esxistir.*/

char*  apresenta_elemento_gestao   (Apresentacao a, int pagina, int linha);

/*Obter a página argumento da apresentação. Devolve NULL se essa página não
existir*/

char** get_pagina_gesta            (Apresentacao a, int pagina);

/*Obter o número de elementos por página da apresentacao*/

int    get_elem_por_pagina_gestao  (Apresentacao a);

/*Obter o numero de paginas da apresentacao*/

int    get_paginas_gestao          (Apresentacao a);

/*Obter o número de elementos na apresentacao*/

int    get_elem_total_gestao       (Apresentacao a);

/*Eliminar espaço alocado para a apresentacao*/

void   elimina_apresentacao_gestao (Apresentacao a);

/*Capsula**********************************************************************/

/*Funcoes para utilizacao do modulo importado: capsula*/

/*Obter o inteiro do indice argumento presente na capsula*/

int     get_int_capsula_gestao     (Capsula c, int indice);

/*Obter o double do indice argumento presente na capsula*/

double  get_double_capsula_gestao (Capsula c, int indice);

/*Libertar espaço alocado para a capsula*/

void    libertar_capsula_gestao    (Capsula c);
```

```
/*obter o tipo da última inserção na cápsula*/
int     get_tipo_capsula_gestao   (Capsula c);
/*obter a quantidade de inteiros presente na capsula*/
int     get_q1_capsula_gestao     (Capsula c);
/*obter a quantidade de doubles presente na capsula*/
int     get_q2_capsula_gestao     (Capsula c);
/*****************************************************************************/
#endif
```

2.4.Apresentação

Este módulo tem o propósito de **armazenar strings** (para posterior apresentação ao utilizador), cada uma podendo ter tamanho variável, **e organizá-las numa lista paginada**. É dada **a possibilidade de ordenar** as strings inseridas ou sobre ordem de inteiros inseridos aquando da mesmo ou de doubles inseridos aquando da mesma, podemos também tomar a opção de não ordenar, deixando então a ordem pela qual as strings são inseridas como a ordem que se apresenta ao utilizador. De notar também que pode ser intenção do programador ordenar, por exemplo, 900 strings mas apenas querer apresentar as 100 primeiras depois de ordenadas, isso é também possível com este módulo. Contém também as funções que irão permitir ao programador, **obter uma página específica, ou um elemento específico de uma página específica** da apresentação facilitando assim a apresentação ao utilizador das strings inseridas.

Segue-se o respetivo .h e documentação

```
#ifndef Apresentacao_H
#define      Apresentacao_H
#include "../../../tipos/tipos.h"
#define N_PAG 20 /*Número de elementos(linhas) por cada página da apresentação*/
/*Valores usados para ordenação (ou não ordenação) dos elementos inseridos para
apresentação*/
#define SEM_ORDENACAO_INT    -1
#define SEM_ORDENACAO_DOUBLE -1
#define SEM_ORDEM            -1
#define ASCENDENTE            1
#define DESCENDENTE           2
typedef struct apresentacao* Apresentacao; /*Tipo opaco de apresentação*/
/* Inicializar a apresentação. Deve ser dado em argumento o total de elementos
que apresentação irá incluir assim como um offset que se pretenda incluir para
o número de elementos por página. Este offset irá dividir o valor definido por
N_PAG de modo a reduzir o número de páginas, se assim se pretender. Caso o valor
dado em argumento seja 1 ou menor, então o número de elementos por página não
será modificado.*/
Apresentacao init_apresentacao(int total, int elem_pag_offset);
```

```
/* Inserir elemento(string). Deverá ser fornecido o tamanho da string a inserir
(os elementos não precisam de ter todos os mesmo tamanho). E deverão ser
fornecidos dados sobre a ordenação. caso se pretenda ordenar as strings sobre
uma determinada ordem de inteiros então deve-se inserir os inteiros
correspondentes a cada string aquando da sua inserção pelo argumento
ordenacaoInt. o mesmo para uma ordenação com doubles mas usando o argumento
ordenacaoDouble. Quando não se pretender ordenação deve-se usar os valores
definidos em cima: SEM_ORDENACAO_INT e/ou SEM_ORDENACAO_DOUBLE. Pode ser dada
uma ordem descendente ou ascendente à ordenação com o argumento ordem usando
ASCENDENTE, DESCENDENTE ou SEM_ORDEM quando não se pretende ordenar. O último
argumento existe para os casos em que se pretende ordenar mas apenas apresentar
um certo número de elementos, esse certo número pode ser fornecido por este
argumento, sendo que quando não se pretende ordenar este argumento é ignorado
podendo então ser enviado qualquer valor. É conveniente indicar que a ordenação
é realizada internamente com recurso ao algorítmo mergeSort.*/

Apresentacao recebe_elemento (ELEMENTO elemento, Apresentacao a, int tamanho,
int ordenacaoInt, double ordenacaoDouble, int ordem, int maximoAordenar);

/*Para uma determinada página e uma determinada linha obter o elemento respetivo.
Caso não exista devolve NULL*/

char* pede_elemento       (Apresentacao a, int pagina, int linha);

/*Obter o número de páginas*/

int   get_paginas         (Apresentacao a);

/*Obter o número de elementos por página*/

int   get_elem_por_pagina (Apresentacao a);

/*Obter o número de elementos total inseridos*/

int   get_elem_total      (Apresentacao a);

/*Devolve uma página completa, sendo a página de número dado com o argumento
pagina. Caso a página não exista é devolvido NULL*/

char** get_pag(Apresentacao a, int pagina);

/*Eliminar apresentacao*/

void elimina_apresentacao(Apresentacao a);

/*****************************************************************************/

#endif
```

2.5.Cápsula

Este módulo tem como **objetivo encapsular dados**. Com isto pretende-se que se entenda que em certos momentos **há a necessidade que uma só função devolva não só um tipo de dados**, mas sim vários. Para que não se usem endereços como argumento que serão modificados e posteriormente utilizados fora da função em questão, tomou-se a opção de criar este módulo auxiliar de nome "capsula" onde se **pode guardar um número variável de inteiros e/ou doubles**.

Segue-se o respetivo .h e documentação

```
#ifndef Capsula_H

#define Capsula_H

#define INT     0   /**usado na inserção de um inteiro***************/

#define DOUBLE  1  /**usado na inserção de um double****************/

#define AMBOS   2 /**usado na inserção de um inteiro e de um double*/

typedef struct capsula *Capsula; /*Tipo opaco*/

/*Modulo*******************************************************/

/*Inicialização da capsula.*/

Capsula  init_capsula      ();

/*Eliminar espaço alocado para a capsula*/

void     libertar_capsula  (Capsula c);

/* Inserir elemento(s) na cápsula. O argumento tipo deve ser uma das seguintes 3 possibilidades: INT, DOUBLE, AMBOS. Esse argumento irá indicar o que estamos a inserir, sendo que se for INT, o argumento item2 deverá ser NULL e o item1 não. Se for DOUBLE, o argumento item1 deverá ser NULL e o item2 não. Se for AMBOS, nenhum argumento deverá ser NULL*/

Capsula  insere_capsula    (Capsula c, int tipo, int *item1, double *item2);

/*Devolve o tipo da última inserção*/

int      tipo_capsula      (Capsula c);

/* Devolve um dos inteiros presentes na cápsula. O índice irá indicar qual pretendemos. Sendo que índice zero corresponde ao primeiro inserido, 1 ao segundo e por aí adiante.*/

int      int_capsula       (Capsula c, int indice);

/* Devolve um dos doubles presentes na cápsula. O índice irá indicar qual pretendemos. Sendo que índice zero corresponde ao primeiro inserido, 1 ao segundo e por aí adiante.*/

double   double_capsula    (Capsula c, int indice);

/*Devolve a quantidade de inteiros presentes na cápsula*/

int      q1_capsula        (Capsula c);

/*Devolve a quantidade de doubles presentes na cápsula*/

int      q2_capsula        (Capsula c);

/*************************************************************************/

#endif
```

2.6.Compra

Este módulo tem o objetivo de auxiliar na validação de uma venda (linha do ficheiro de vendas).

Segue-se o seu respetivo .h e documentação

```
#ifndef Compra_H
#define Compra_H
#include "../../modulos/catalogo/catalogo.h"
#include "../../tipos/tipos.h"
#define PRECO_MAX 999.99 /*Maior preço possível de uma compra válida*/
#define PRECO_MIN   0.0 /*Menor peço possível de uma compra válida */
#define QUANT_MAX 200 /*Quantidade máxima de produto comprado por compra válida*/
#define QUANT_MIN   1 /*Quantidade mínima de produto comprado por compra válida*/
#define N_FILIAIS 3 /*Número máximo de filiais*/
#define N_MESES  12 /* Número máximo de meses. Pode ser aumentado caso se queira guardar dados ao longo de vários anos. Ppode ser diminuído para um espaço de tempo mais pequeno*/
#define REGIME_N 0  /*Valor inteiro atribuído ao regime de compra N*/
#define REGIME_P 1 /*Valor inteiro atribuido ao regime de compra P*/
typedef struct compra *Compra; /*Tipo opaco de compra*/
/*Modulo*****************************************************************/
/*Inicializa a compra com todos os campos a zero e o cliente e produto a NULL*/
Compra init_compra    ();
/*Elimina o espaço alocado para a compra*/
void   limpa_compra  (Compra c);
/* Recebe um compra inicializada e valida uma linha argumento do tipo COMPRA. Caso a linha não seja válida a compra é de imediato eliminada (o espaço alocado é libertado) e devolve-se NULL, caso a linha seja válida devolve-se a compra com os dados validados guardados para posterior utilização.*/
Compra valida_compra (Compra c, COMPRA linha, Cat clientes, Cat produtos);
/*Getters*****************************************************************/
/*Obter quantidade da compra*/
int    quantidade_compra (Compra c);
/*Obter tipo da compra, N ou P*/
int    tipo_compra       (Compra c);
/*Obter filial da compra*/
int    filial_compra     (Compra c);
```

```
/*Obter mes da compra*/
int    mes_compra        (Compra c);
/*Obter preco da compra*/
double preco_compra      (Compra c);
/*Obter produto da compra*/
PRODUTO  produto_compra    (Compra c);
/*Obter cliente da compra*/
CLIENTE  cliente_compra    (Compra c);
/*Setters*****************************************************************/
/*inserir quantidade da compra*/
Compra insere_quantidade_compra (Compra c, int    quant);
/*Inserir tipo da compra*/
Compra insere_tipo_compra       (Compra c, int     tipo);
/*Inserir filial da compra*/
Compra insere_filial_compra     (Compra c, int   filial);
/*Inserir mes da compra*/
Compra insere_mes_compra        (Compra c, int      mes);
/*Inserir preco da compra*/
Compra insere_preco_compra      (Compra c, double preco);
/*Inserir produto da compra*/
Compra insere_produto_compra    (Compra c, PRODUTO   prod);
/*Inserir cliente da compra*/
Compra insere_cliente_compra    (Compra c, CLIENTE    cli);
/*************************************************************************/
#endif
```

3.Interface

***

3.1.Utilização

O programa a que se dá o nome de ***GereVendas*** é um **programa não para programadores mas sim para utilizadores**. Por mais banal que esta frase pareça nunca é demais frisar a colossal importância do seu significado. Até porque **um programa não tem propósito de existir se o seu público alvo não o saber usar**. Por esta razão tivemos o consciente esforço de tornar a utilização do nosso programa o mais fácil possível, assim como o menos cansativa possível. São duas características que achamos essencial qualquer projeto deste tipo ter.

Não deve ser cansativo. Exemplos de um programa cansativo seria por exemplo um programa que ao apresentar listas grandes de items não permitisse um rápido acesso a qualquer página ou fácilmente voltar atrás. O nosso programa **permite sempre o cancelamento da visualização de qualquer lista assim como a escolha da página para onde se pretende saltar**. Outro exemplo é um programa que seja demasiado 'banal' visualmente, por exemplo, um programa só com letras, menus mal formatados e opções para nomes de comandos mal pensados. **Os nossos menus são todos envoltos em caixas, sendo cada campo a demonstrar devidamente separado de informação distinta** de modo a possibilitar uma mais fácil compreensão. **Os nomes dos comandos foram também estudados de modo a que fossem intuitivos**, por exemplo, em qualquer momento pode escrever-se 'h' para obter a lista dos comandos possíveis no menu presente. Sendo 'h' a primeira letra de 'help' este comando torna-se bastante intuitivo.

Deve ser fácil. Na nossa interface o **utilizador nunca se sentirá perdido** sendo que em momento algum olha para o ecrã sem que lhe seja indicada pelo menos uma tecla a carregar para prosseguir assim como em momento algum tem a necessidade de comprênder minimamente o funcionamento interior do programa (a sua programação), com isto queremos afirmar, **qualquer pessoa seria capaz de usar o nosso programa**, até mesmo alguém que nunca tenha sequer ouvido falar em programação.

De notar que o nosso programa lê de imediato os ficheiros após ser iniciado, sendo questionado o utilizador sobre os nomes dos mesmos, de seguida qualquer uma das queries pode ser executada sem nenhuma restrição, sendo que **inputs errados para as queries são sempre detetados como tal** e a interface volta a solicitar o input ao utilizador em vez de o programa terminar inesperadamente.

Aquando da terminação do programa toda a **memória alocada para o correto funcionamento do mesmo é libertada sem nenhuma exceção** de modo a evitar potênciais perigos na execução repetida e consecutiva do programa para o computador em que é executado. Como prova deste facto o programa pode ser corrido com a ferramente valgrind, sendo que na pasta do projeto se encontra já um ficheiro que contêm o output dessa ferramenta, depois de ter sido corrido o programa com o ficheiro Vendas_1M.txt e todas as queries realizadas, onde se pode ver que não existe possibilidade de haverem 'memory leaks', fugas de memória.

Sabemos também que muitas vezes interfaces podem se tornar demasiado complexas de ler a nível de programação, algo que não acontece no nosso projeto sendo que tivemos o cuidado de usar "switches" em vez do uso abusivo e menos legível de "if's" assim como nas opções que tomamos de aumentar o número de linhas usadas em certas partes para evitar linhas de tamanhos absurdos e exagerados.

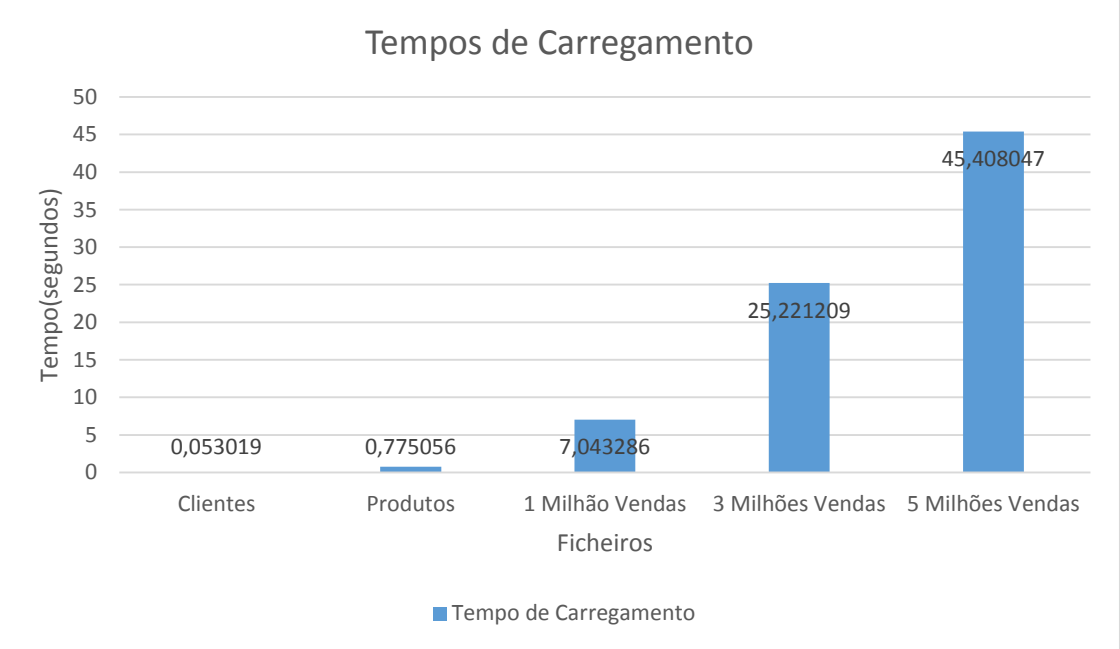
As funcionalidades/queries encontram-se já explicitadas na introdução, no sub-capítulo **Motivação e Objetivos**, caso se queira saber sobre as mesmas.

4.Testes de performance e análise dos resultados

***

4.1.Leitura dos ficheiros

Apresentamos de seguida um gráfico de barras com os tempos de leitura de cada ficheiro. Importa referir que **todos os tempos que se apresentam neste relatório estão em segundos e não são o resultado de uma só medição mas sim da média obtida ao fim de 1000 medições**.



Vemos que **a leitura "menos eficiente" acontece ao ler os ficheiros de vendas**. No entanto tal justifica-se ao estudar a estrutura gestao explicada no capítulo anterior, vemos que a complexidade da estrutura justificaria até um tempo mais elevado de inserção, sendo que estes tempos apenas se conseguem obter graças à estratégia de dividir cada àrvore balanceada por um array de 26 espaços o que aumenta em muito a eficiência para procurar, inserir, ou remover elementos. A prova disso é o tempo que se media antes de implementarmos esta estratégia, rondava os 30 segundos (para Vendas_1M.txt), sendo que agora ronda os 7 segundos como se pode ver no gráfico de barras.

4.2.Mapa e Performance de Queries

Antes de apresentar os tempos de cada query em cada ficheiro apresentamos um mapa que mostra os modulos utilizados por cada uma, assim como o input que necessita e o output que se espera.

Query	Produtos	Clientes	Faturação	Gestão	Resultado
2	Char				Lista dos produtos + total
3			Mês+prod		Totais mês
4			Varrimento		Lista de produtos não vendidos por filial ou global
5				cliente	Tabela de compras por mês em cada filial
6			Intervalo de meses		Total de vendas+faturação
7				Consulta de todos os clientes	Lista de clientes que compraram em todas as filiais
8				Prod	Lista de clientes que compraram determinado produto
9				Cliente+mês	Lista ordenada por quantidade dos produtos mais comprados pelo cliente num determinado mês
10				Consulta de todos os produtos +N	Lista ordenada dos N produtos mais comprados ordenada por quantidade
11				Cliente+TopN	Lista de N produtos em que o cliente mais gastou dinheiro ordenada por faturação
12	Consulta de valor	Consulta de valor	Consulta de valor	Consulta de valor	Total prods + total clientes + total prods não comprados + total clientes não compradores

Com este mapa em mente, olhamos agora para os tempos de cada query

Query	Vendas_1M	Vendas_3M	Vendas_5M	Input do utilizador
Q2	0.042589	0.041700	0.041564	Todas as letras
Q3	0.000012	0.000018	0.000016	Mês 1, Todas as filiais, AF1184
Q4	0.023671	0.046616	0.022716	Todas as filiais
Q5	0.000013	0.000012	0.000012	Cliente Z5000
Q6	0.000008	0.000006	0.000008	Entre mês 1 a 12s
Q7	0.008497	0.007064	0.007060	
Q8	0.000017	0.000030	0.000104	Produto AF1184
Q9	0.000086	0.000092	0.000788	Cliente Z5000, mês1
Q10	0.449996	0.470538	0.478372	Top 171008 (todos)
Q11	0.000176	0.000510	0.000644	Cliente Z5000, Top 10
Q12	0.000003	0.000002	0.000004	

De notar o cuidado tido para que, com o aumento exponêncial do tamanho do ficheiro das vendas, os tempos das queries não aumentassem também. **É aqui que justificamos a complexidade aparentemente exagerada do módulo gestao**, podemos observar que as únicas queries que realmente aumentam um pouco o tempo de espera por parte do utilizador á medida que se aumenta o tamanho do ficheiro vendas são a **8, a 9 e a 11**. Sendo que todas **as outras ou atravessam uma árvore somente do tamanho do número de produtos ou do número de clientes ou então apenas consultam valores**, e isto é algo que permanece constante com o mudar de ficheiro de vendas. Note-se também que há um limite para quão grande o tempo das queries referidas 8, 9 e 11 pode crescer, pois foi tido o cuidado na sua implementação para que no máximo fosse preciso uma travessia de uma árvore do tamanho de todos os clientes ou de todos os produtos. Resumindo, **nenhuma query, por maior que seja o ficheiro de vendas, irá fazer uma travessia de árvore maior que o número de produtos ou o número de clientes** (dependendo da query).

5.Makefile

***

5.1.Código

Apresentamos agora o código da makefile assim como a explicação de cada comando

```
##############DEFINICOES#######################################
#-------------------------------------------------------------#
BINARY        = GereVendas                                    #
CC            = gcc                                           #
CFLAGS        = -Wall -pedantic -g -ansi -O3                  #
VALGRINDFLAGS = --leak-check=full --log-file="Valgrind.out"#
FILES         = $(shell find src -name "*.c")                 #
HEADERS       = $(shell find src -name "*.h")                 #
OBJS          = $(FILES:.c=.o)                                #
#-------------------------------------------------------------#
##############################################################

###############REGRAS############################
#-----------------------------------------#
GereVendas: $(OBJS)
	$(CC) $(CFLAGS) $(OBJS) -o $(BINARY)
	rm $(FILES:.c=.o)
#-----------------------------------------#
run: GereVendas
	./$(BINARY)
#-----------------------------------------#
apenasRun :
	./$(BINARY)
#-----------------------------------------#
```

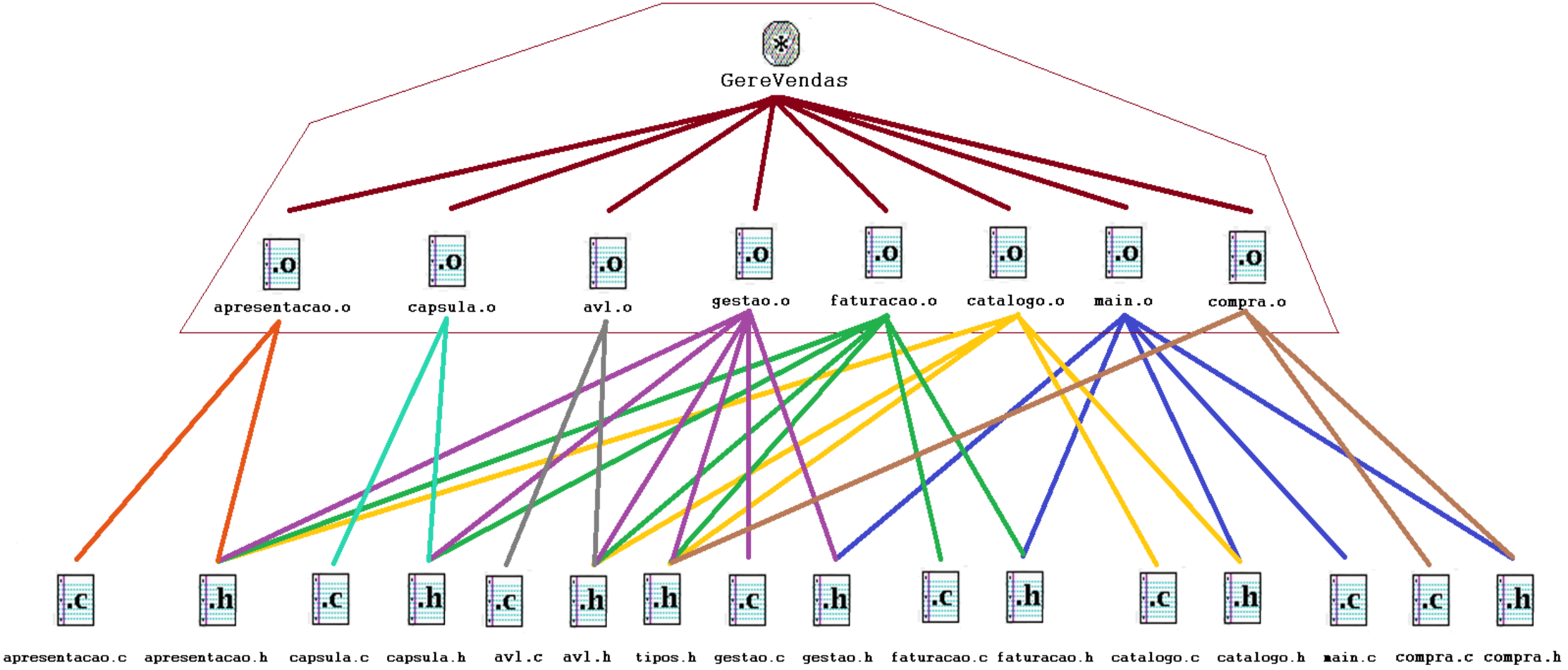
```
ValgrindRun:
	valgrind $(VALGRINDFLAGS) ./$(BINARY)
#--------------------------------------------#
clean:
	rm -f $(BINARY)
#--------------------------------------------#
##############################################
```

Estão disponíveis os seguintes comandos:

- Make GereVendas - Compila o programa e elimina os ficheiros .o;
- Make run - Compila, elimina os ficheiros .o e executa o programa;
- Make ValgrindRun - Executa o programa usando a ferramenta Valgrind com as seguintes flags ativas --leak-check=full --log-file="Valgrind.out";
- Make clean - Elimina o executável;

5.2.Grafo de Depências da Makefile

Apresentamos agora o grafo de dependências da nossa makefile que nos mostra graficamente os ficheiros necessários para compilar cada objeto .o e os .o necessários para compilar o projeto.


GereVendas
apresentacao.o
capsula.o
avl.o
gestao.o
faturacao.o
catalogo.o
main.o
compra.o
apresentacao.c
apresentacao.h
capsula.c
capsula.h
avl.c
avl.h
tipos.h
gestao.c
gestao.h
faturacao.c
faturacao.h
catalogo.c
catalogo.h
main.c
compra.c
compra.h

6.Conclusões e Sugestões

Terminada a primeira fase deste projeto, depois de feita uma minunciosa análise de requisitos e avaliação do problema, chegamos a conclusões (apresentadas ao longo do corpo deste relatório) que nos deixaram satisfeitos. Pensamos ter sido capazes de apresentar explicações detalhadas no sentido em que toda a informação essencial para entender a implementação do nosso sistema e o seu funcionamento se encontra presente sem nenhuma exceção, mas ao mesmo tempo bastante simples na maneira como será possível para qualquer programador, sem conhecimento pévio sobre o projeto ou sobre a programação que lhe deu existência , que pretenda usar os módulos criados para este projeto num outro projeto distinto o consiga fazer com toda a facilidade. Com isto pretendemos afirmar que estamos satisfeitos com as conclusões a que chegamos e com o plano de acção que elaboramos tendo conseguido realizar tudo o que foi proposto de forma simples e eficaz sendo que foram ainda realizados vários testes de performance nunca obtendo resultados não aceitáveis.

Tendo isto em conta temos também a noção que nunca nenhum projeto se pode dar por realmente terminado ou por realmente perfeito, há sempre a possibilidade de melhorar algo e esse facto não deve nunca ser ignorado, por mais experiência que venhamos a ter, existe sempre um limite para quão boas as nossas primeiras impressões poderão ser. Desta forma, estamos preparados para uma eventual necessidade de modificar algo que aqui poderá ter sido afirmado e/ou concluído. Isto é, o nosso projeto foi elaborado com a ideia de criar algo que permita uma fácil expansão tanto de funcionalidades como de novas ideias que poderiamos vir a ter ou que poderiamos necessitar de implementar na continuação do projeto caso lhe fossemos dar continuidade (algo que aconteceria num projeto de trabalho real fora da Universidade). Isto é essencial em qualquer projeto, pois sabemos que no mundo real o empregador nem sempre sabe de raiz tudo aquilo que realmente quer implementar por isso torna-se natural que, num momento mais avançado da implementação, liste novas funcionalidades que quer ver presentes. Um projeto mal pensado para expansões e modificações poderia ter de ser refeito de raiz para não dar problemas ou graus baixos de eficiência, no entanto tivemos a ideia presente de tentar criar algo que podesse minimizar todas essas possíveis perdas que poderiam acontecer se este fosse um projeto no mundo real e não um trabalho académico protegido pela redoma que acaba sempre por ser uma universidade. Esses cuidados estão explicitados no capítulo 4, onde mostramos que nunca em situação alguma, seja qual for o ficheiro de vendas, alguma query faz mais do que percorrer uma árvore do tamanho de todos os produtos ou de todos os clientes. Para além disso no capítulo 3 explicamos ainda a estruturação do main, onde se teve todo o cuidado para que fosse conciso e concreto, pois sabemos que muitas vezes interfaces podem se tornar demasiado complexas de ler a nível de programação, algo que não acontece no nosso projeto sendo que tivemos o cuidado de usar switches em vez do uso abusivo de “if’s” assim como nas opções que tomamos de aumentar o número de linhas usadas em certas partes para evitar linhas de tamanhos absurdos e exagerados.

Ao longo deste projeto refizemo-lo 3 vezes com o intuito de melhorar o equilíbrio tempo de queries/tempo de inserção. A versão que temos agora foi a mais equilibrada que conseguimos desenvolver no tempo fornecido e estamos bastante satisfeitos com ela, no entanto sabemos não haver limites, por vezes existem sim milagres na programação, afirmamos por isso que é sempre possível melhorar mais e mais, nenhum equilíbrio é alguma vez suficiente ou máximo, se dado o tempo de desenvolvimento suficiente, teriamos tempos ainda melhores por mais absurdo que pareça, ou por mais improvável que seja, há sempre maneiras para melhorar, ou novas estratégias para pensar. Com este parágrafo queremos apenas mostrar que, embora achemos que o nosso esforço foi compensado olhando com orgulho para o projeto final, sabemos que de final tem nada, quem afirma que não há espaço para melhorar então é porque realmente não compreende o que é programação. Até porque, mesmo que chegassemos ao impossível e fantástico tempo de 0 em todas as queries, de certeza que havia muito a melhorar a nível de memória alocada. Há sempre algo que pode ser melhorado e por isso este trabalho nunca terá fim, apenas uma pequena pausa.