Laboratórios de Informática III (LI3)
Projeto de C 2015/2016

GEREVENDAS

Gestão das Vendas de uma Cadeia de Distribuição com 3 Filiais

MIEI – 2º ano – 2º semestre

GRUPO 38

José António Dantas Silva
A74576

Adriana Eliana Fernandes Pereira
A67662

André Almeida Gonçalves
A75625

Índice

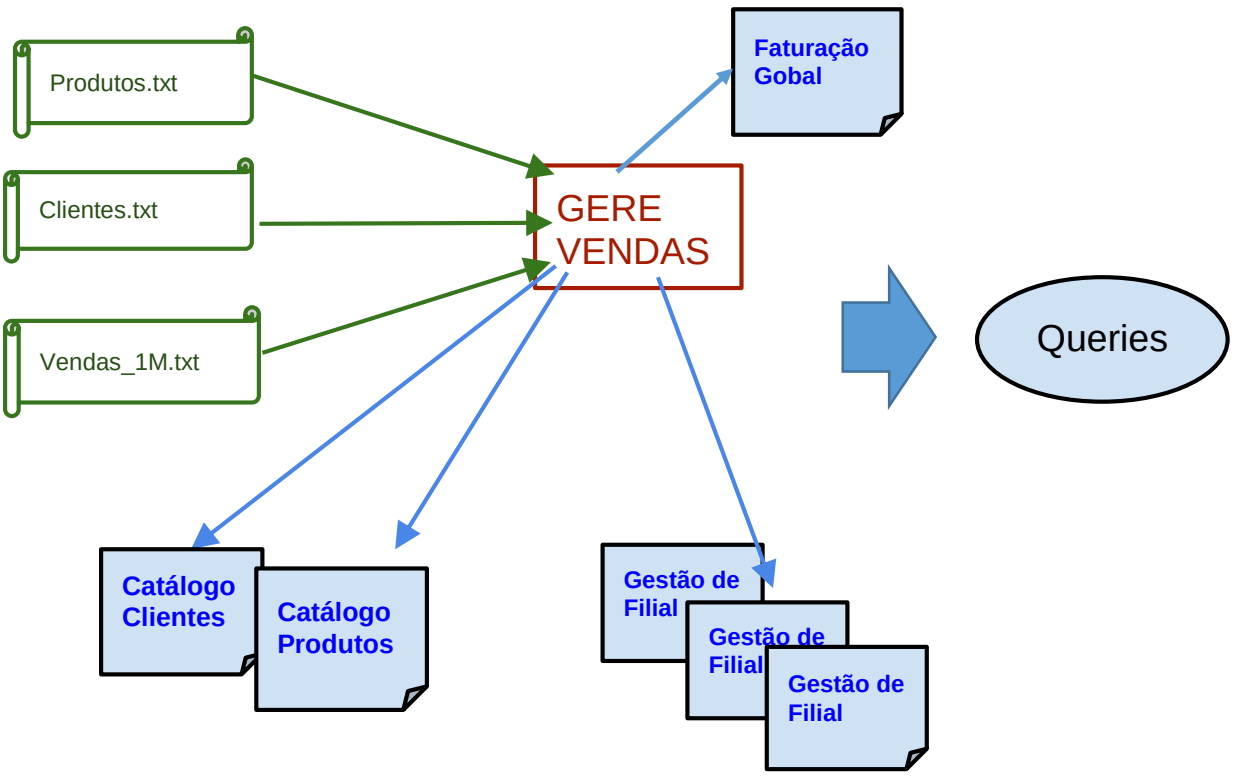
Introdução

No âmbito da unidade curricular de Laboratórios de Informática III do 2º Ano do Mestrado Integrado em Engenharia Informática foi proposto o desenvolvimento de um Projeto em linguagem C sobre Gestão das Vendas de uma Cadeia de Distribuição com 3 Filiais que tem por objetivo ajudar à consolidação dos conteúdos teóricos e práticos e enriquecer os conhecimentos adquiridos nas UCs de Programação Imperativa e de Algoritmos e Complexidade.

Este projeto considera-se um grande desafio pelo facto de passarmos a realizar programação em grande escala, uma vez, que são aplicações com grandes volumes de dados e com mais elevada complexidade algorítmica e estrutural. Nesse sentido, o desenvolvimento deste programa será realizado a partir dos princípios da modularidade (divisão do código fonte em unidades separadas coerentes), do encapsulamento (garantia de proteção e acessos controlados aos dados), da conceção de código reutilizável, escolha otimizada das estruturas de dados.

Descrição dos módulos

A arquitetura do software é definida por quatro módulos principais: Catálogo de produtos, Catálogo de clientes, Faturação global e Vendas por filial cujas fontes de dados são os três ficheiros de texto Produtos.txt, Clientes.txt e Vendas_1M.txt, e uma interface que permita a comunicação com o cliente.



O ficheiro Produtos.txt contém 200.000 códigos de produtos. Cada linha representa o código de um produto do hipermercado onde cada código é formado por duas letras maiúsculas seguidas de quatro dígitos.

O ficheiro Clientes.txt contem 20.000 códigos de clientes. Em cada linha de código existe um cliente identificado do hipermercado. Cada código é formado por em letra maiúscula e quatro dígitos.

O ficheiro Vendas_1M.txt contem 1.000.000 de registos de compras e é a nossa maior fonte de dados. Cada linha representa o registo de uma compra efetuada no hipermercado. Cada código é formado por: código de produto, preço unitário do produto em decimal, o número de unidades compradas, a letra N ou a letra P, o código do cliente que lhe corresponde e o mês em que foi efetuada a compra e em que filial ocorreu a compra.

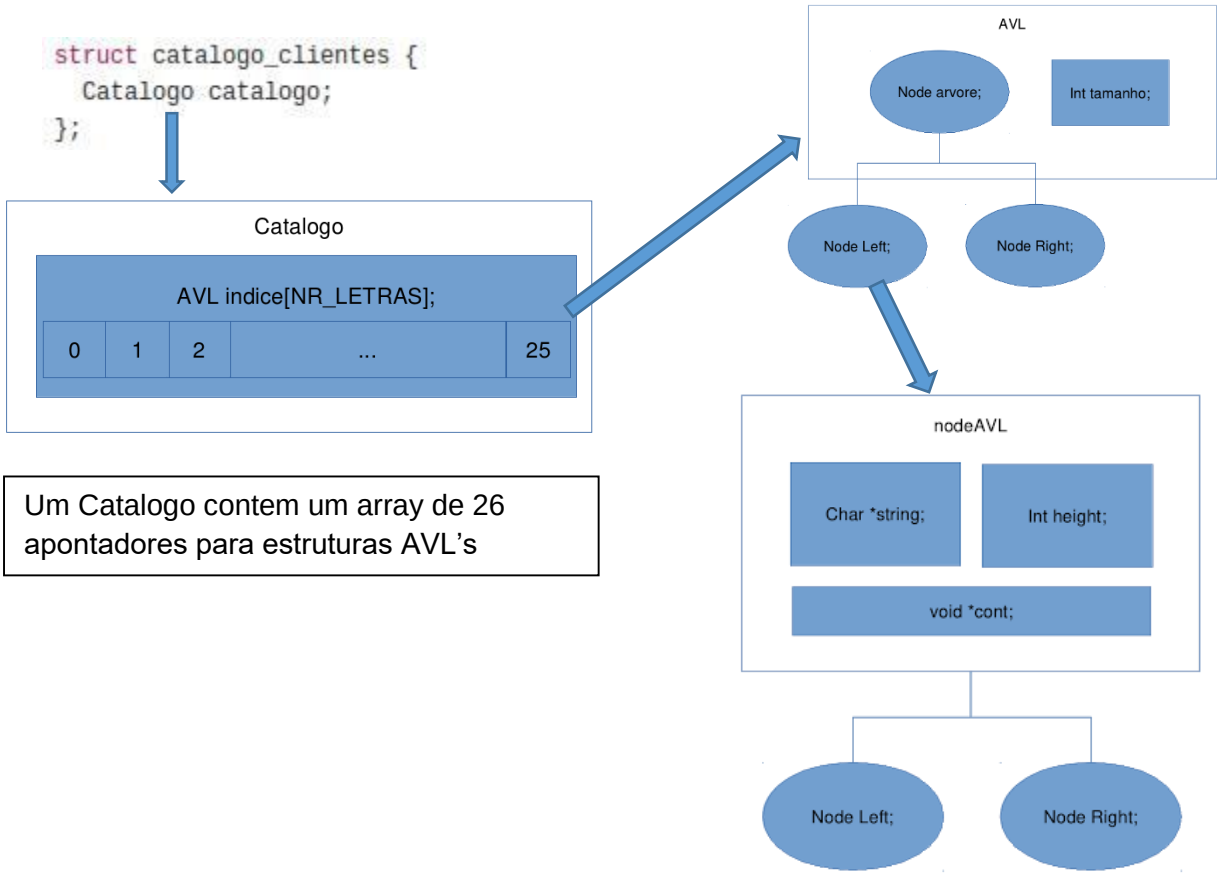
Catálogo de Clientes

Módulo de dados onde são guardados os códigos de todos os clientes do ficheiro Clientes.txt. Os dados são guardados num array de AVL's com 26 posições referentes às 26 letras do alfabeto. Cada índice contem um apontador para uma AVL com os dados referentes aos códigos dos clientes de uma dada letra associada a esse índice. Este módulo tem também a estrutura de um Cliente a qual contem a identificação de um cliente ou seja o seu código.

- ## clientes.h

Tipos de Dados Criados

typedef struct catalogo_clientes *Cat_Clientes;



Um Catalogo contem um array de 26 apontadores para estruturas AVL's

typedef struct cliente *Cliente;

```
struct cliente {
  char *name;
};
```

Um Cliente é uma estrutura que contem apenas um char* onde será colocado o código de um Cliente.

API

- **Cat_Clientes init_cat_clientes ():** Inicia a estrutura de um novo catálogo de clientes.
- **Cat_Clientes insere_Cliente (Cat_Clientes clients, Cliente client):** Função que insere num dado catálogo de clientes um novo cliente.
- **Catalogo get_Catalogo_Clientes (Cat_Clientes clientes, Catalogo novo):** Devolve um clone do catálogo inserido na estrutura do Catálogo de Clientes, sendo o clone colocado no Catalogo "novo", garantindo assim que ninguém pode aceder ao Catalogo original, mantendo o encapsulamento total da estrutura.
- **Boolean existe_Cliente (Cat_Clientes clients, Cliente client):** Função que devolve um Boolean referente a um dado cliente se encontrar no Catalogo de Clientes.
- **int total_Clientes (Cat_Clientes clients):** Função que devolve o número total de clientes que um catálogo de clientes contém.
- **int total_Clientes_letra (Cat_Clientes clients, char letra):** Função que determina o número de clientes presentes num catálogo de clientes contem cujo nome inicia por uma dada letra.
- **void remove_Catalogo_Clientes (Cat_Clientes clients):** Função com o objetivo de limpar da memória um Catálogo de clientes.
- **Cliente criaCliente ():** Função que inicia a estrutura de um cliente.
- **void alteraCliente (Cliente client, char *info):** Função que altera o código de um dado cliente.
- **char* getNomeCliente(Cliente cliente, char* novo):** Função que devolve o código de um cliente. Para ser garantido o encapsulamento é devolvido um clone desse código em vez de um apontador para o código original do cliente.
- **void free_cliente (Cliente client):** Função que limpa da memória um dado Cliente.

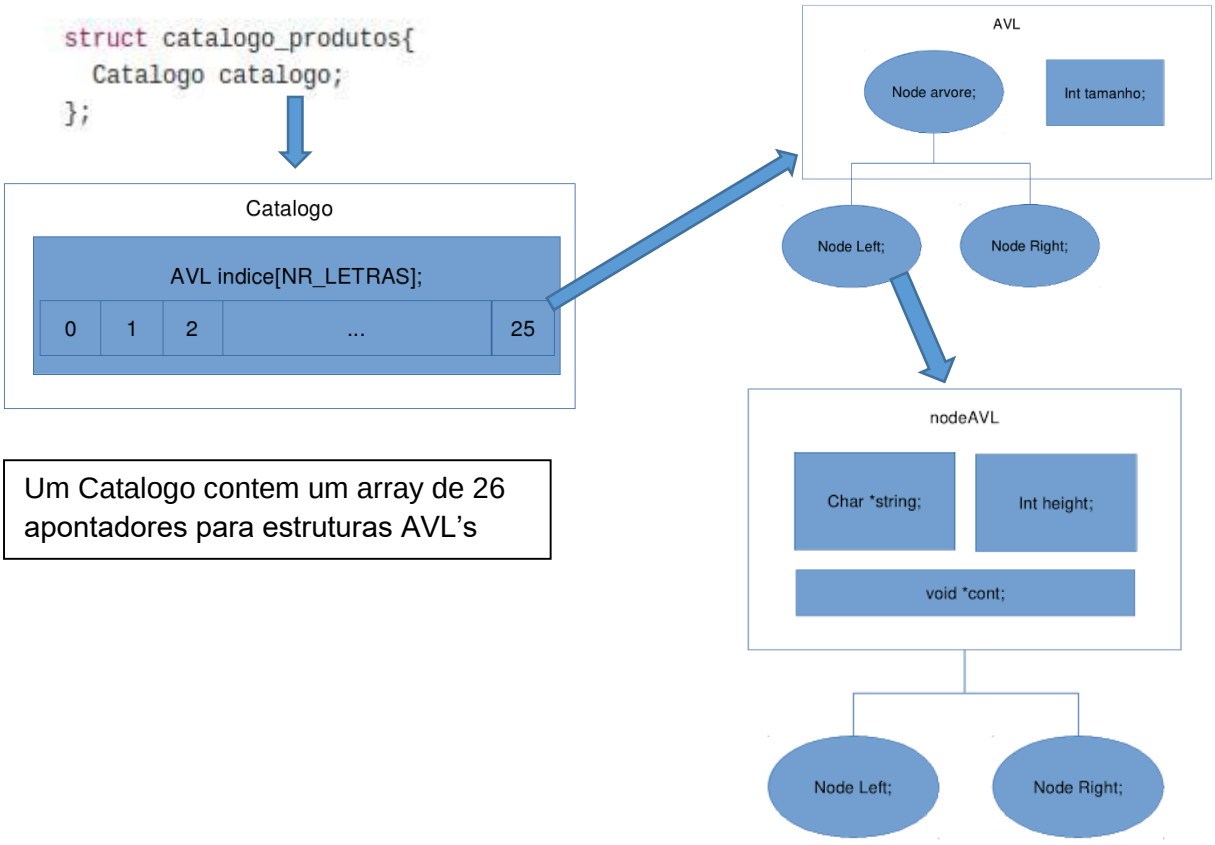
Catálogo de Produtos

Módulo de dados onde são guardados os códigos de todos os produtos do ficheiro Produtos.txt. Os dados são guardados num array de AVL's com 26 posições referentes às 26 letras do alfabeto. Cada índice contem um apontador para uma AVL com os dados referentes aos códigos dos produtos de uma dada letra associada a esse índice. Este módulo tem ainda a estrutura de um Produto a qual contem a identificação de um produto dado o ficheiro ou seja o seu código, bem como um Conj_Produtos, estrutura que contem uma lista onde podem ser armazenados códigos de produtos.

- **produtos.h**

Tipos de Dados Criados

typedef struct catalogo_produtos *Cat_Produtos;



Um Catalogo contem um array de 26 apontadores para estruturas AVL's

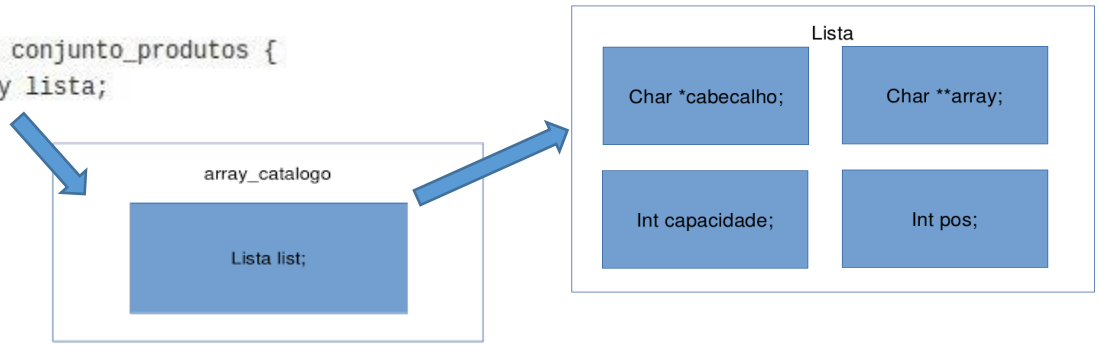
typedef struct produto *Produto;

```
struct produto {
  char* prod;
};
```

Um Produto é uma estrutura que contem apenas um char* onde será colocado o código de um Produto.

typedef struct conjunto_produtos *Conj_Produtos;

```
struct conjunto_produtos {
  Array lista;
};
```



API

- **Cat_Produtos init_cat_produtos ():** Inicializa a estrutura de um Catálogo de produtos.
- **Cat_Produtos insere_produto (Cat_Produtos products, Produto prod):** Função que insere um Produto no Catálogo de produtos.
- **Catalogo get_Catalogo_Produtos (Cat_Produtos products):** Função que retorna um clone do catálogo de produtos inserido na estrutura garantindo o encapsulamento não deixando que terceiros tenham acesso ao Catalogo original.
- **Boolean existe_Produto (Cat_Produtos products, Produto product):** Função que verifica se um dado produto existe no Catalogo de Produtos.
- **int total_Produtos (Cat_Produtos products):** Devolve o número de produtos de um catálogo de produtos.
- **int total_Produtos_letra (Cat_Produtos products, char letra):** Devolve o número de produtos começados com a letra indicada.
- **void remove_Catalogo_Produtos (Cat_Produtos products):** Função que liberta o espaço alocado em memória pelo catálogo.
- **Produto criaProduto ():** Função que inicializa a estrutura de um produto.

- **Produto alteraProduto (Produto product, char *info):** Função que altera a informação de um produto.

- **char* getNomeProduto (Produto product):** Devolve o código de um Produto. Para que o encapsulamento não seja quebrado é devolvido um clone do mesmo ao invés do código original impossibilitando assim a sua alteração por terceiros.

- **void free_produto (Produto product):** Função que liberta o espaço alocado em memória por um produto.

- **Conj_Produtos converte_Produtos (Conj_Produtos lista, Cat_Produtos products, char letra):** Função que converte de um catálogo de produtos para um conjunto de produtos.

- **Conj_Produtos init_Conjunto (int capacidade):** Função inicializa um conjunto de produtos.

- **Lista get_Lista_Produtos (Conj_Produtos conjunto):** Retorna a lista com o conteúdo de um Conj_Produtos.

- **Conj_Produtos converte_Produtos (Conj_Produtos lista, Cat_Produtos products, char letra):** Converte os elementos de um Cat_Produtos para um Conj_Produtos começados por uma dada letra.

- **void free_Conj_Produtos (Conj_Produtos c):** Remove da memória um Conj_Produtos.

Faturação Global

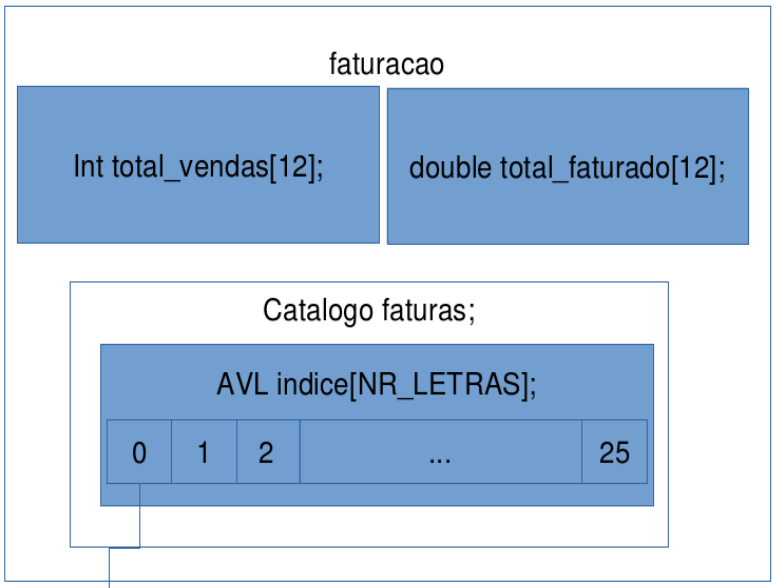
Módulo de dados que irá conter as estruturas de dados responsáveis pela resposta eficiente a questões quantitativas que relacionam os produtos às suas vendas mensais ou globais, em modo Normal (N) ou em Promoção (P). Este módulo deve referencia todos os produtos, mesmo os que nunca foram vendidos.

- **Faturacao.h**

Tipos de Dados Criados

typedef struct faturacao

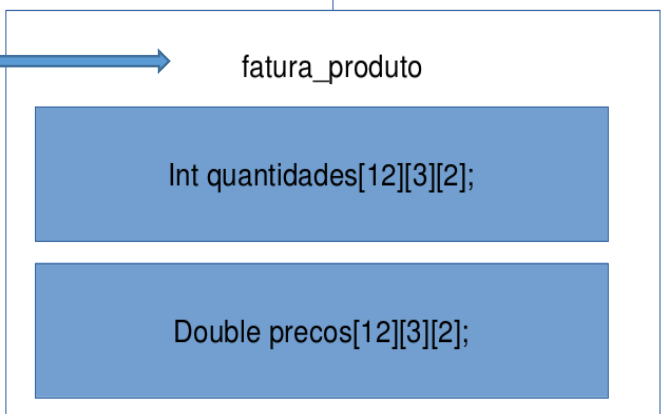
```
struct faturacao {
        Catalogo faturas;
        int total_vendas[12];
        double total_faturado[12];
};
```



total_vendas guarda o número de vendas total em cada mês de todos os produtos.
total_faturado guarda a faturação total em cada mês de todos os produtos.

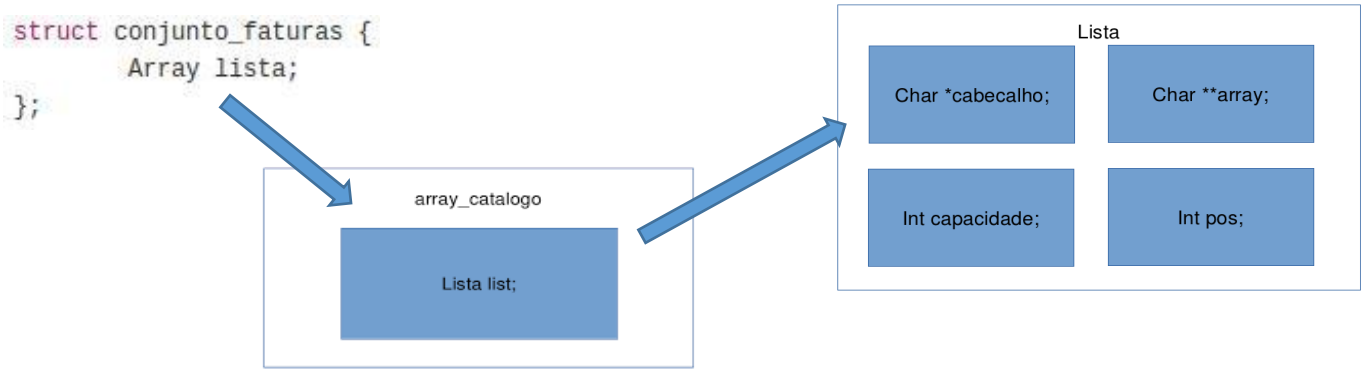
O Catalogo faturas é um clone do catalogo dos produtos permitindo assim ter à partida registo sobre todos os produtos existentes.

```
typedef struct fatura_produto {
        int quantidades[12][3][2];
        double precos[12][3][2];
}*Fatura_Produto;
```



Dentro de cada produto há uma estrutura fatura_produto a qual contem informações sobre as vendas e faturação tendo como principal índice o mês e separando ainda por filiais e vendas em modo normal ou promoção, sendo toda esta informação armazenada em duas matrizes cúbicas.

typedef struct conjunto_faturas *Conj_Faturas;



API

- **Faturacao init_Faturacao ():** Função que inicia uma nova estrutura Faturação.

- **Faturacao cria_Dados_Faturacao (Faturacao fat, Cat_Produtos produtos):** Função responsável por clonar o catálogo de produtos por forma a termos a informação previamente ordenada dos mesmos na faturação. Como a Faturação recebe um clone de um Catalogo de Produtos o encapsulamento é mantido.

- **Faturacao adiciona_Fatura (Faturacao contas, Venda venda):** Função que tem a responsabilidade da inserção dos dados provenientes de uma Venda na estrutura Faturacao.

- **void free_Faturacao (Faturacao faturacao):** Função com o objetivo de limpar da memória uma estrutura Faturação.

- **double get_total_precos_mes_produto_filial (Faturacao fatura, char* produto, int mes, char modo, int filial):** Devolve a faturação total de um produto num dado mês e numa dada filial em promoção ou normal.

- **int get_total_vendas_mes_produto_filial(Faturacao fatura, char* produto, int mes, char modo, int filial):** Devolve o número total de vendas de um produto num dado mês e numa dada filial em promoção ou normal.

- **double get_total_precos_mes_produto (Faturacao fatura, char* produto, int mes, char modo):** Devolve a faturação total de um produto num dado mês em promoção ou normal.

- **int get_total_vendas_mes_produto (Faturacao fatura, char* produto, int mes, char modo):** Devolve o número de vendas total de um produto num dado mês em promoção ou normal.

- **double get_total_faturado_intervalo (Faturacao fatura, int mes1, int mes2):** Devolve o total faturado num intervalo de meses.

- **int get_total_vendas_intervalo (Faturacao fatura, int mes1, int mes2):** Devolve o número total de vendas num intervalo de meses.

- **Faturas faturas_nao_comprado_filial (Conj_Faturas conjunto, Conj_Faturas nao_comprados,**

- **Faturacao faturas, int filial):** Cria um Conj_Faturas com os produtos não comprados numa dada filial.

- **Conj_Faturas init_Lista_Faturacao (int capacidade):** Inicializa uma nova estrutura Conj_Faturas.

- **Conj_Faturas adiciona_Conjunto (Conj_Faturas conjunto, char* info):** Adiciona ao conjuto de faturas um produto.

- **void free_Conj_Faturas (Conj_Faturas c):** Remove da memória um dado Conj_Faturas.

- **Conj_Faturas faturas_produtos_nao_comprados_totais(Conj_Faturas conjunto, Faturacao faturacao):** Devolve um Conj_Faturas com os produtos nunca comprados.

- **Conj_Faturas cria_lista_total (Conj_Faturas conjunto, Faturacao faturacao):** Cria uma lista com todos os produtos da estrutura.

- **Lista apresenta_faturas(Conj_Faturas conjunto):** Devolve uma lista com os elementos de um Conj_Faturas.

- **char* get_elemento_lista(Conj_Faturas conjunto, int pos):** Devolve um elemento de um Conj_Faturas, dada uma posição.

- **int faturacao_getPos(Conj_Faturas conjunto):** Devolve o tamanho de um Conj_Faturas.

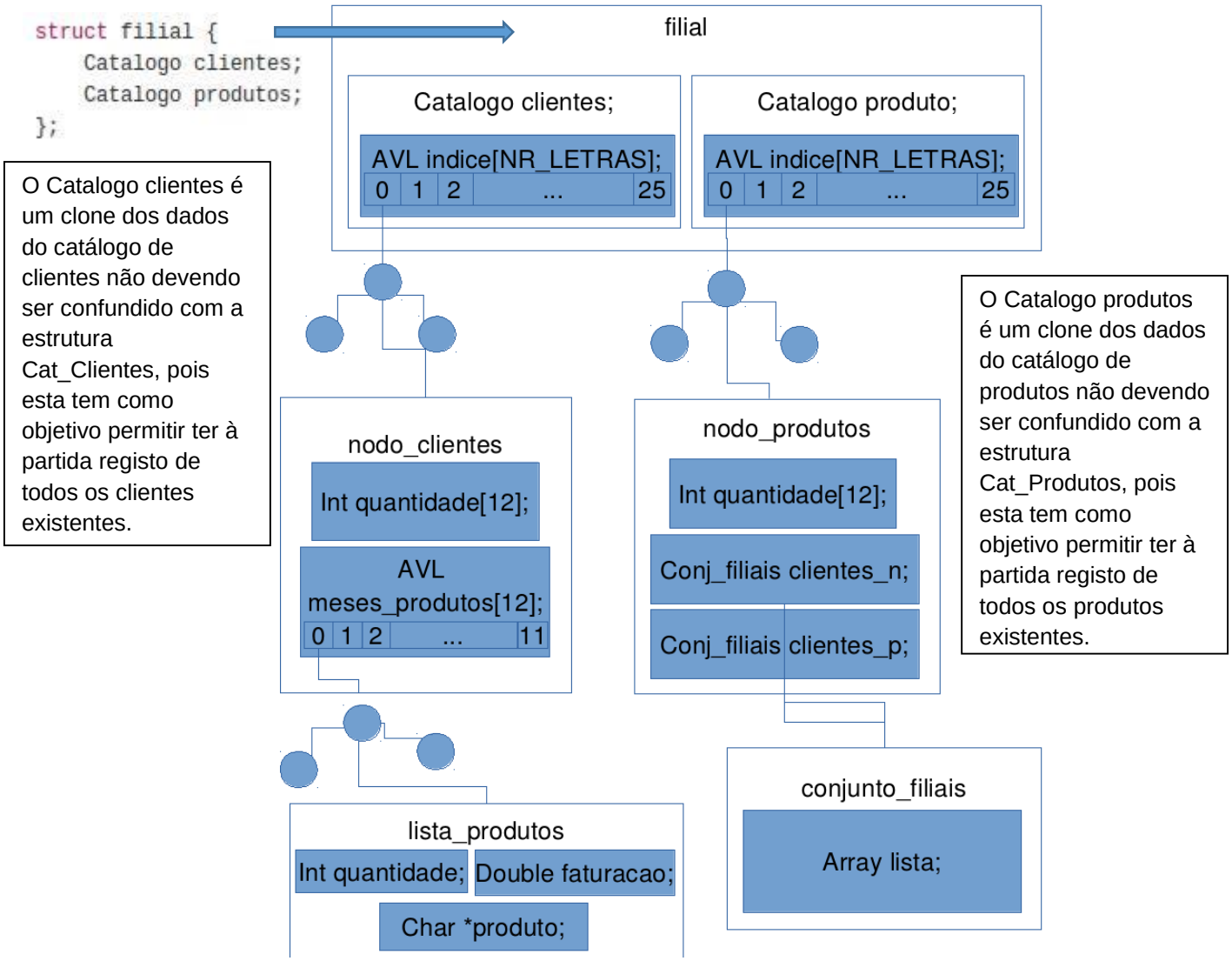
Filial

Módulo de dados que, a partir das leituras, conterá as estruturas de dados adequadas à representação das relações 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 e vice-versa. Por uma questão de otimização foi necessário moldar a estrutura usada de acordo com as queries pedidas.

- **filial.h**

Tipos de Dados Criados **typedef struct filial *Filial**

```
struct filial {
    Catalogo clientes;
    Catalogo produtos;
};
```



O Catalogo clientes é um clone dos dados do catálogo de clientes não devendo ser confundido com a estrutura Cat_Clientes, pois esta tem como objetivo permitir ter à partida registo de todos os clientes existentes.

O Catalogo produtos é um clone dos dados do catálogo de produtos não devendo ser confundido com a estrutura Cat_Produtos, pois esta tem como objetivo permitir ter à partida registo de todos os produtos existentes.

```
typedef struct nodo_clientes {
    AVL meses_produtos[12]; /*
    int total_quantidades[12];
} *Nodo_Clientes;
```

A estrutura nodo_clientes contem um array de 12 AVL's referentes a meses, na qual em cada nodo se encontra um produto com a estrutura lista_produtos associada á mesma.
Esta estrutura contem também um array de inteiros com o tamanho dos meses do ano onde são guardados os totais de quantidades de um dado cliente num dado mês.

```
typedef struct lista_produtos {
        int quantidade;
        double faturacao;
        char* produto;
} *Lista_Produtos;
```

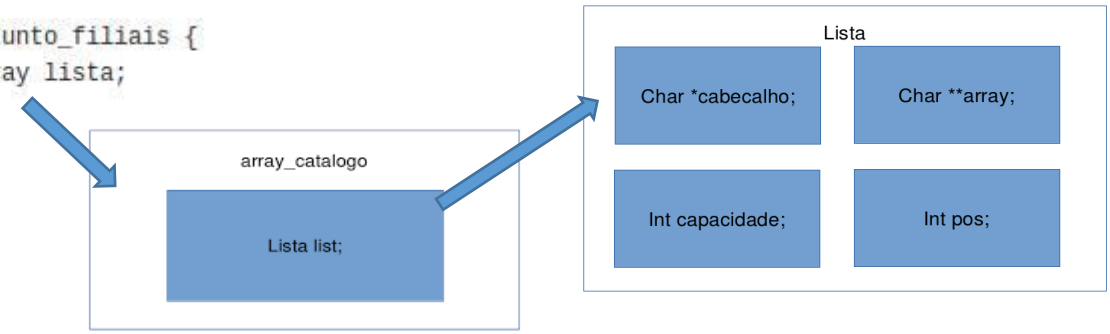
A estrutura lista_produtos está associada a cada produto comprado por um cliente num dado mês, e contem a quantidade que foi vendida desse produto, a faturação com esse mesmo produto e o código do produto em questão.

```
typedef struct nodo_produtos{
        int quantidade;
        Conj_Filiais clientes_N;
        Conj_Filiais clientes_P;
} *Nodo_Produtos;
```

A estrutura nodo_produtos, inserida em cada produto do Catalogo com os produtos de uma filial contem a quantidade vendida desse mesmo produto nessa filial, bem como dois Conj_Filiais, um com os clientes que compraram em modo Normal e um para os clientes que compraram em Promoção.

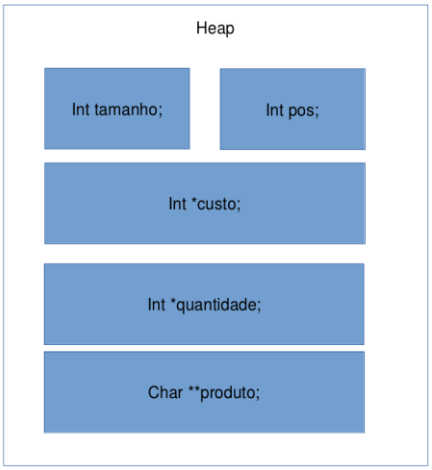
typedef struct conjunto_filiais *Conj_Filiais;

```
struct conjunto_filiais {
        Array lista;
};
```



typedef struct heap_filial *HEAP;

```
struct heap_filial {
        Heap heap;
};
```



A estrutura heap, como o próprio nome indica é uma estrutura de dados que representa uma max heap, tendo neste caso a capacidade "tamanho", o número de elementos que contem "pos" e armazena em 3 arrays dinâmicos distintos os códigos de produtos, quantidades e custos dos mesmos, podendo ser ordenada pela quantidade ou custo.

API

- **Filial init_Filial ():** Função que inicializa a estrutura de uma Filial.

- **Filial cria_Dados_Filial (Filial filial, Cat_Produtos produtos, Cat_Clientes clientes):** Função que permite á estrutura Filial obter os dados relativos aos clientes e aos produtos. Para que tal ação mantenha o encapsulamento dos dados são criados clones dos Catalogos com os produtos e dos Catalogos com os clientes não permitindo assim a alteração por terceiros de tais dados.

- **Filial adiciona_Venda_Filial (Filial f, Venda v):** Função que a uma estrutura de uma Filial dados de uma dada Venda.

- **void free_Filial (Filial f):** Limpa da memória uma dada estrutura de uma Filial.

- **Conj_Filiais init_Conj_Filiais (int n):** Função que inicia uma estrutura de um Conj_Filiais.

- **Conj_Filiais adiciona_Nome (Conj_Filiais c, char* nome**): Função que adiciona um elemento a um dado Conj_Filiais.

- **void free_Conj_Filiais (Conj_Filiais c):** Limpa da memória um dado Conj_Filiais.

- **int nr_total_unidades_compradas (Filial f, char* cliente, int mes):** Retorna o numero de unidades compradas de um dado um cliente num mês.

- **Boolean filial_existe_Cliente (Filial f, char* cliente):** Verifica se um Cliente existe numa dada Filial.

- **Boolean filial_existe_Produto (Filial f,char* produto):** Verifica se um Produto existe numa dada Filail.

- **Boolean verifica_cliente_comprado (Filial f, char* c):** Verifica se um Cliente comprou algum produto.

- **int getQuantidadeProduto (Filial f, char* produto):** Função que retorna o número de unidades vendidas de um dado produto.

- **int nr_clientes_de_um_produto (Filial f, char* produto):** Função que retorna o número de clientes de um dado produto numa filial.

- **Lista get_Lista_Filial (Conj_Filiais c):** Função que devolve uma lista com os elementos de um Conj_Filiais.

- **int filial_getPos (Conj_Filiais conjunto):** Função que retorna o tamanho de um Conj_Filiais.

- **char* filial_get_elemento_lista (Conj_Filiais conjunto, int pos):** Função que retorna um elemento de um Conj_Filiais dada uma posição.

- **int filiais_nr_elementos_diferentes (Conj_Filiais a, Conj_Filiais b):** Função que retorna o número de elementos diferentes entre dois Conj_Filiais.

- **HEAP init_HEAP ():** Função que inicia uma nova estrutura HEAP.

- **void free_HEAP (HEAP h):** Limpa da memória uma dada HEAP.

- **int heap_tamanho (HEAP h)**: Função que retorna o tamanho de uma dada HEAP.

- **HEAP lista_codigos_de_clientes (Filial f, HEAP h, char* cliente, int mes, char ordenacao):** Função que coloca numa HEAP as informações dos produtos de um cliente numa filial, dado um mês.

- **HEAP top3_clientes (Filial f, HEAP h, char* cliente, char ordenacao):** Função que coloca numa HEAP os dados sobre os produtos comprados por um dado cliente num ano, ordenando os mesmos por Faturação.

- **Conj_Filiais lista_top3 (Conj_Filiais c, HEAP h, char ordenacao):** Função que retorna um Conj_Filiais com os 3 produtos mais comprados de um cliente num ano, dada uma HEAP com os dados.

- **Conj_Filiais convert_Heap_Lista (Conj_Filiais c, HEAP h, char ordenacao):** Função que retorna um Conj_Filiais com códigos armazenados numa dada HEAP ordenados por Quantidades.

- **HEAP heap_produtos_mais_vendidos (Filial f, HEAP h):** Função que converte numa HEAP os dados sobre os produtos organizando-os por quantidade comprada.

- **Conj_Filiais retira_N_Produtos (Conj_Filiais c, HEAP h, int n):** Função que retorna um Conj_Filiais com os n produtos mais comprados ordenadamente, dada uma HEAP com os dados.

- **Conj_Filiais lista_clientes_compraram_filial (Conj_Filiais c, Filial f):** Função que retorna um Conj_Filiais com os clientes que efetuaram compras numa dada filial.

- **Conj_Filiais lista_clientes_de_produto (Filial f, char* produto, char promo):** Função que retorna um Conj_Filiais com os clientes de um dado produto em modo Noraml (N) ou modo Promoção (P).

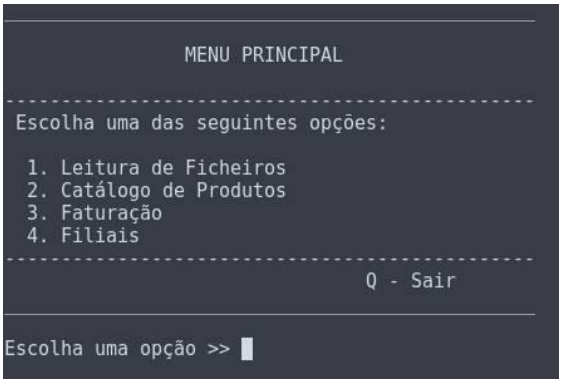
- **Conj_Filiais converte_total_clientes (Conj_Filiais c, Filial f):** Função que converte num Conj_Filiais os clientes de uma dada filial.

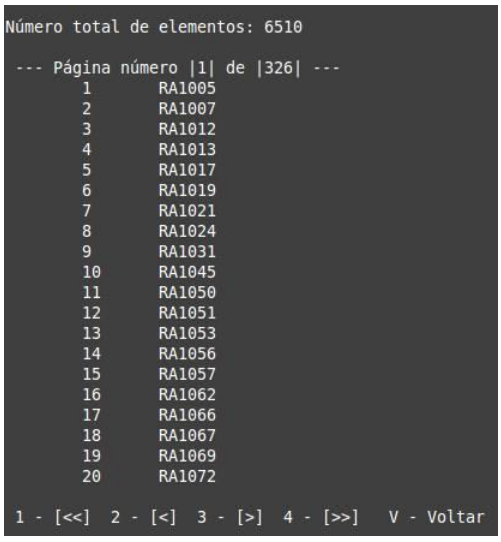
Main.c

O programa é controlado pela função main.c, a qual define as estruturas de um Catálogo de Clientes, um Catálogo de Produtos, uma Faturacao e 3 estruturas Filial, de seguida entrando num ciclo que tem um “estado”, o mesmo define se o programa executa ou termina. Entrando neste ciclo as estruturas são inicializadas e é invocada a função “menu_principal”. Esse menu principal devolve 2 valores ao ciclo, um indicando a saída do programa sendo nesse caso as estruturas limpas de memória antes de o programa terminar, ou então de releitura dos ficheiros sendo as estruturas igualmente limpas, mas não termina o ciclo voltando a ser inicializadas e iniciando o menu novamente.

Interface do utilizador

Quando o utilizador executa o programa é-lhe apresentado um menu principal em que tem de escolher entre as opções: 1. Leitura de Ficheiros, 2. Catálogo de Produtos, 3. Faturação e 4. Filiais, mas nenhuma das outras opções é permitida sem antes ser efetuada a leitura de ficheiros para que o programa contenha dados e as queries possam ser executadas.

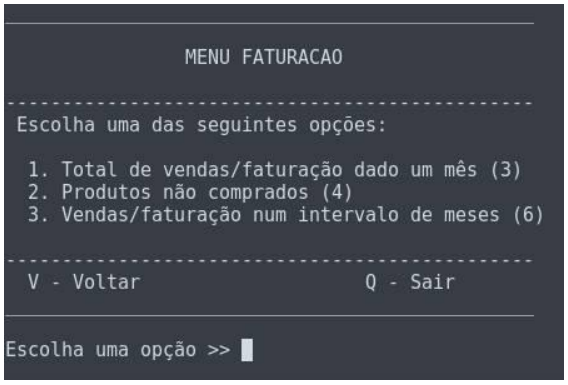




Após a leitura dos ficheiros o utilizador tem a possibilidade de escolher 1 dos 4 menus disponíveis inicialmente. Caso opte pelo primeiro uma releitura dos ficheiros será efetuada.

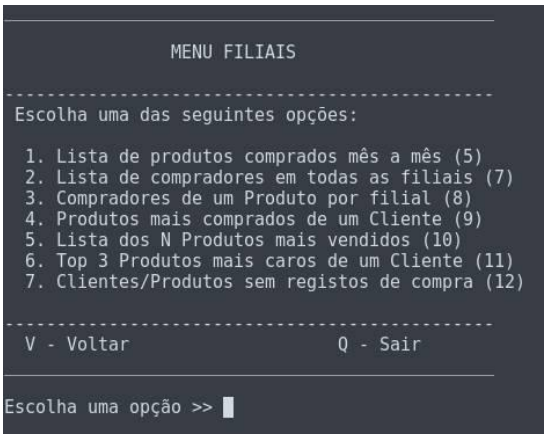
Caso a escolha recaia sobre a opção “Catálogo de Produtos” o utilizador tem disponível a funcionalidade de visualizar os produtos de uma letra à sua escolha ordenadamente, sendo os mesmos apresentados em Páginas de no máximo 20 elementos cada, podendo navegar livremente nas mesmas.

Tal decisão foi necessária devido à complexidade do tamanho dos resultados a apresentar. Este modo de apresentação de valores listados ao utilizador foi aplicado a todas as queries que exigissem um possível tamanho considerável de elementos a apresentar, e como o número de queries com esta característica era grande foi criada uma estrutura Página dedicada à apresentação de dados ao utilizador.



Selecionada a opção Faturação no menu principal, o utilizador tem a possibilidade de executar cada uma das funcionalidades referentes as queries pedidas que têm por base a estrutura da faturação, estando o numero da opção na esquerda e a query relativa na direita.

Por último existe a opção Filiais que tal como a opção Faturação permite a escolha de execução das funcionalidades pedidas pelas queries sendo que estas têm por base a estrutura da Filial existindo de novo a escolha do lado esquerdo e a query respetiva indicada no lado direito após uma breve descrição da mesma.

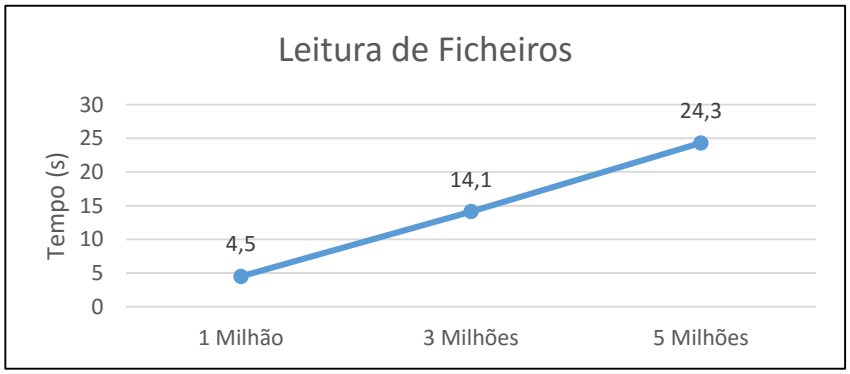


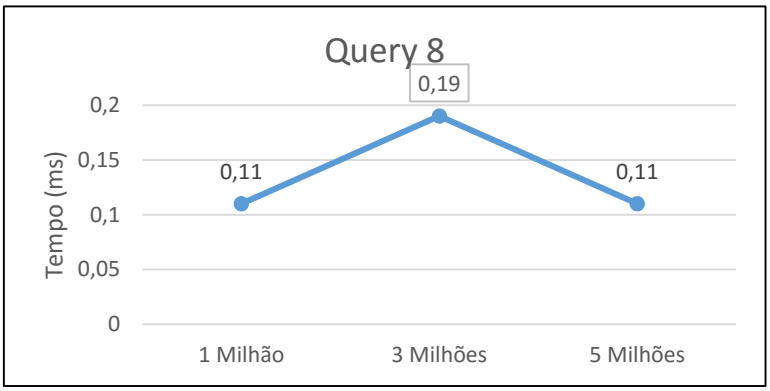
Para uma navegação mais facilitada todos os submenus têm a hipótese de Voltar ao menu principal, bem como a hipótese de Sair do programa.

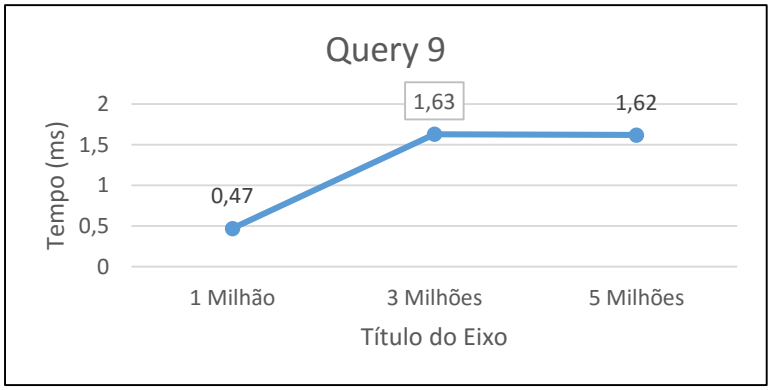
Resultados e comentários sobre os testes de performance

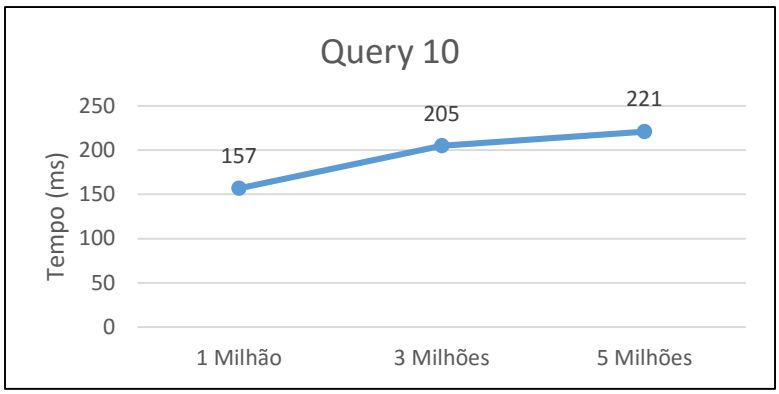
Posteriormente ao desenvolvimento e codificação de todo o projeto foi-nos proposto realizar alguns testes de performance que consistem na obtenção dos tempos de execução das querias 8, 9, 10, 11 e 12 usando os ficheiros Vendas_1M.txt, Vendas_3M.txt (3 milhões de vendas) e Vendas_5M.txt (5 milhões de vendas) usando a biblioteca standard de C time.h.

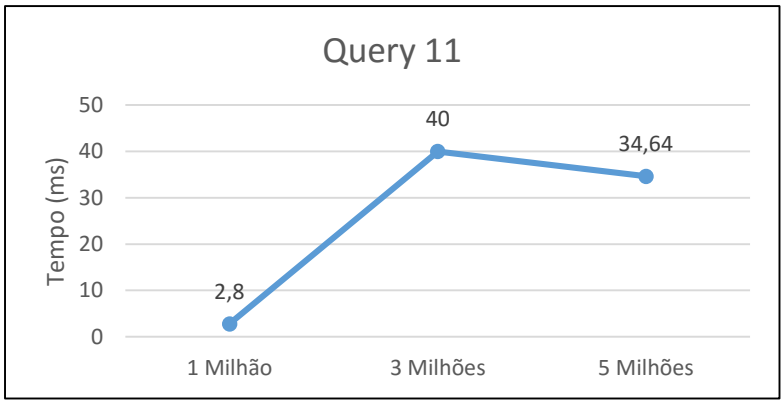
Em seguida são demonstrados os resultados dos mesmos:

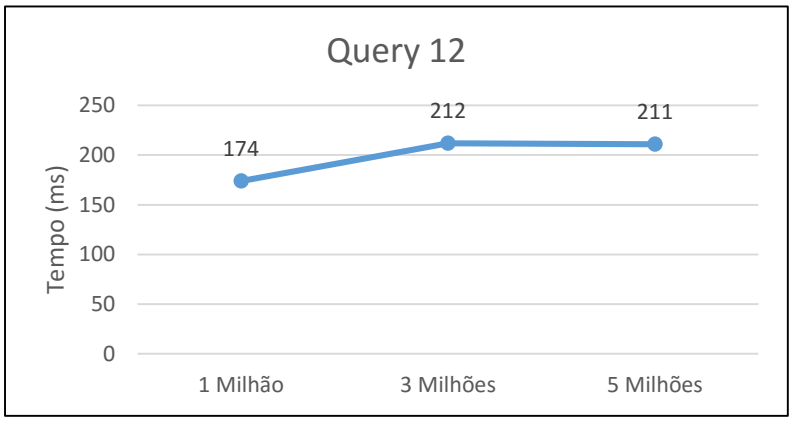












Uma vez que a quantidade de vendas vai aumentando de ficheiro para ficheiro é aceitável que os tempos de execução para os carregar, ou seja, a fase de leitura e inserção de dados também aumente significativamente. Comparando os valores de execução das queries pretendidas, como podemos observar nos respetivos gráficos apresentados, o tempo de execução não varia significativamente, uma vez que são registadas variações na ordem dos milissegundos, devido a uma estruturação dos dados independente da quantidade de dados acumulada.

Estes testes foram realizados numa máquina com um processador de 2,4 GHz, com uma memória Ram de 8GB.

Makefile

```
CC=gcc
```
Compilador a usar

```
OBJS := $(patsubst src/%.c, obj/%.o,$(wildcard src/*.c))
```
Ficheiros a compilar

```
CFLAGS= -Wno-unused-result -O2 -ansi
```
Flags de compilação

```
compile:$(OBJS)
	$(CC) $(CFLAGS) -o gereVendas $(OBJS)
```
Compila o programa executando "make"

```
debug: CFLAGS := -g
debug: compile
```
Compila o programa para fazer debug

```
obj/%.o: src/%.c
	@mkdir -p obj
	$(CC) $(CFLAGS) -o $@ -c $<
```
Gera os .o numa pasta obj

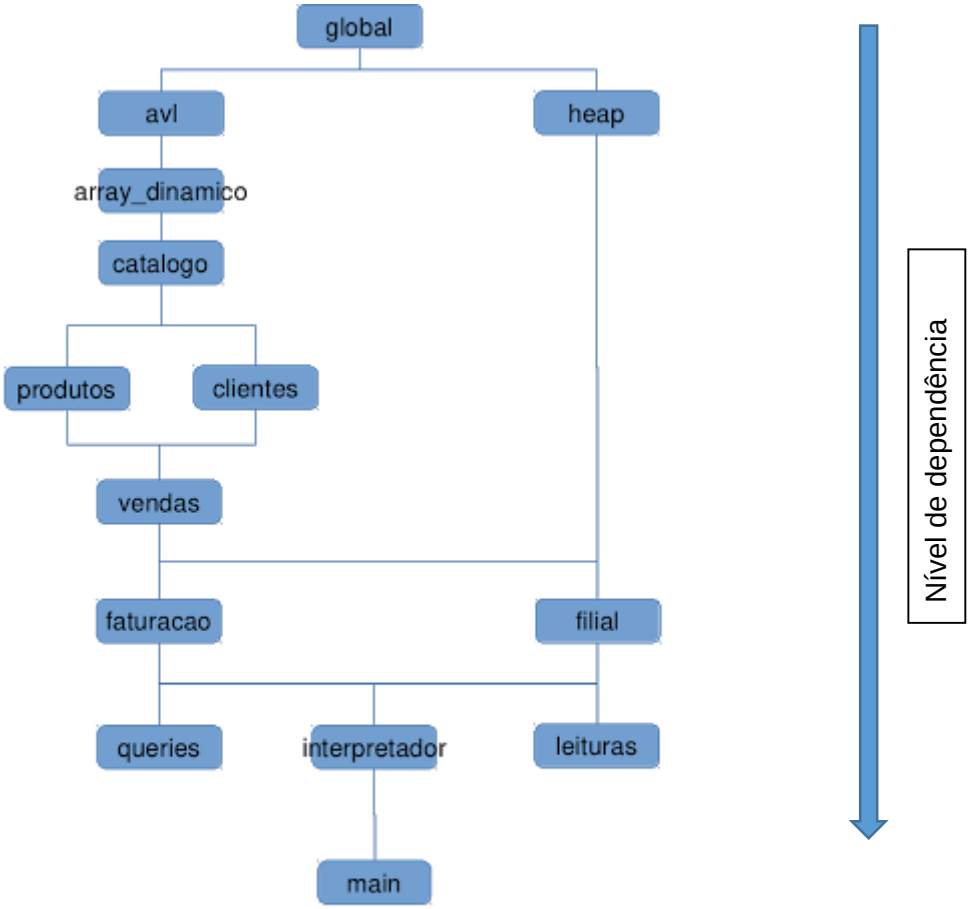
```
run:$(OBJS)
	$(CC) $(CFLAGS) -o gereVendas $(OBJS)
	./gereVendas
```
"make run" compila o programa e executa-o automaticamente de seguida

```
clean:
	-@rm gereVendas
	-@rm -rf obj
```
Limpa os .o gerados e o executável

```
cleanAll: clean
	-@rm -rf doc
	-@rm -rf html
```
Limpa os .o gerados, o executável e a documentação

```
.PHONY: doc
doc:$(OBJS)
	doxygen doxygen.conf
```
Gera a documentação do código

Grafo de dependências



Grafo de dependências

```
obj/main.o: src/headers/global.h src/headers/produtos.h src/headers/clientes.h src/headers/filial.h
            src/headers/interpretador.h src/headers/faturacao.h
obj/avl.o: src/headers/avl.h src/headers/global.h
obj/leituras.o: src/headers/leituras.h src/headers/clientes.h src/headers/catalogo.h src/headers/produtos.h
                src/headers/venda.h src/headers/filial.h src/headers/faturacao.h src/headers/global.h
obj/array_dinamico.o: src/headers/array_dinamico.h src/headers/avl.h src/headers/global.h
obj/catalogo.o: src/headers/catalogo.h src/headers/avl.h src/headers/array_dinamico.h src/headers/global.h
obj/clientes.o: src/headers/clientes.h src/headers/catalogo.h src/headers/global.h
obj/produtos.o: src/headers/produtos.h src/headers/catalogo.h src/headers/global.h
obj/venda.o: src/headers/venda.h src/headers/clientes.h src/headers/produtos.h src/headers/global.h
obj/interpretador.o: src/headers/interpretador.h src/headers/clientes.h src/headers/produtos.h src/headers/faturacao.h
                     src/headers/filial.h src/headers/global.h
obj/heap.o: src/headers/heap.h src/headers/global.h
obj/global.o: src/headers/global.h
obj/faturacao.o: src/headers/faturacao.h src/headers/catalogo.h src/headers/venda.h src/headers/produtos.h
                 src/headers/global.h
obj/filial.o: src/headers/filial.h src/headers/avl.h src/headers/array_dinamico.h src/headers/heap.h
              src/headers/catalogo.h src/headers/venda.h src/headers/produtos.h src/headers/clientes.h
              src/headers/global.h
obj/queries.o: src/headers/queries.h src/headers/faturacao.h src/headers/filial.h src/headers/avl.h
               src/headers/array_dinamico.h src/headers/heap.h src/headers/catalogo.h src/headers/venda.h
               src/headers/produtos.h src/headers/clientes.h src/headers/global.h
```

Conclusão

A realização deste projeto foi muito enriquecedora no sentido em que ao trabalharmos com blocos de dados de tamanho muito maior ao que acontecia até então, tudo se tornou mais complexo, o que obrigou à utilização de técnicas particulares e tendo sempre como objetivo que este trabalho fosse concebido de modo a que seja facilmente modificável, e seja, apesar da complexidade, o mais otimizado possível a todos os níveis. As maiores dificuldades existiram na prática de estruturas opacas com o objetivo de delimitar a vista do utilizador, pois era uma prática não conhecida até este trabalho, e na limpeza das estruturas devido ao facto de não haver conhecimento entre os módulos, na gestão das estruturas de forma eficiente, por forma a obter uma boa performance.

Este trabalho foi muito importante para o aprofundar do conhecimento no que toca ao uso desta linguagem e deste “modo de pensar”, e apesar de várias dificuldades conseguimos cumprir os objetivos.