Projeto de Laboratórios de Informática 3

Grupo 52

Carlos Miguel Luzia de Carvalho A89605

Francisco Correia Franco A89458

José Pedro Carvalho Costa A89519

A89605

A89458

A89519

Índice

Introdução

Nesta unidade curricular foi-nos proposta a implementação de um Sistema de Gestão de Vendas (SGV) com base na leitura de ficheiros disponibilizados pelos docentes.

A primeira fase deste projeto consistiu no desafio de implementar este sistema na linguagem C. Apesar do intuito de executar as funcionalidades do programa de forma rápida, a modularidade e o encapsulamento das estruturas por nós utilizadas eram prioridade.

Estas preocupações vão de encontro à aprendizagem realizada neste semestre no âmbito da programação orientada aos objetos, bem como o objetivo da segunda fase do projeto, a implementação do mesmo sistema em Java.

Para nós o maior desafio foram a implementação das estruturas para uma melhor organização da informação.

Descrição dos Módulos

A arquitetura do software é definida por 4 módulos principais: Catálogo de Clientes, Catálogo de Produtos, Faturação global e Vendas por filial, cujas fontes de dados são os 3 ficheiros de texto Produtos.txt, Clientes.txt e Vendas_1M.txt e uma interface que permita a comunicação com o cliente.

Cada linha do ficheiro Produtos.txt representa um código de produto e é formado por 2 letras maiúsculas seguidas de 4 dígitos.

Cada linha do ficheiro Clientes.txt representa um código de cliente e é formado por 1 letra maiúscula seguida de 4 dígitos.

Cada linha do ficheiro Vendas_1M.txt representa uma venda efetuada e é formada por: código de produto, preço unitário decimal, número inteiro de unidades compradas, tipo de compra, código de cliente, mês da compra e filial.

Catálogo de Produtos

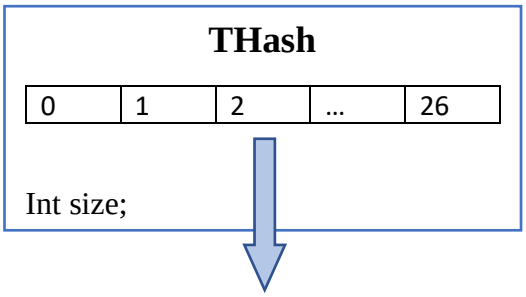
Módulo de dados onde são guardados todos os produtos do ficheiro Produtos.txt.

Os dados são guardados numa Tabela de Hash com 26 posições referentes às 26 letras do alfabeto. Cada índice contém um apontador para um array com os códigos de produtos de uma dada letra associada a esse índice. Deste modo procuramos tornar a procura de um certo produto mais eficiente.

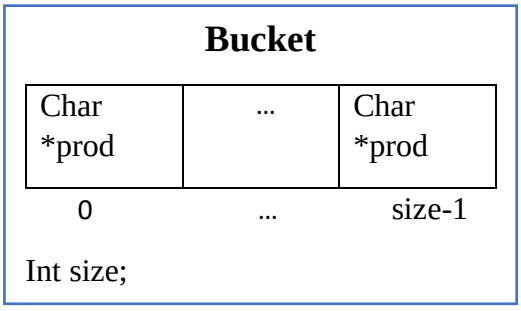
- produtos.h

Tipos de dados criados

```
typedef struct thash{
        int size;
        Bucket *tbl;
} THash;
```




```
typedef struct bucket{
        int size;
        char **arr;
} Bucket ;
```



Api

- **int hash_p (char *cont):** Função que recebe uma string e subtrai ao primeiro elemento da string a letra A.
- **THash* initTab_p ():** Função que inicia uma estrutura THash alocando espaço para todas as estrututas adjacentes.
- **void destroiTab_p (THash *p):** Função que destrói uma estrutura THash libertando o espaço ocupado por esta.
- **void acrescentaTab_p (THash *h, char*cont):** Função que acrescenta a uma THash uma determinada string.
- **swapp (char**arg1, char**arg2):** Função que troca duas strings de posição.
- **quicksortp (char **args, unsigned int len):** Função que ordena um array de strings.
- **int validaproduto (char *produto):** Função que recebe uma string de um produto e verifica se este é válido.
- **int ler_prod (THash *prod, char*filespath, int *p):** Função que lê de um ficheiro para uma Thash.
- **char* getProduto (THash *p, int key, int i):** Função que cria um clone de uma string produto.
- **char** getArrayProd (THash *p, int key):** Função que cria um clone de um array de strings produto.
- **int getArrayProdSize (THash *p, int key):** Função que retorna o tamanho de um determinado array da tabela.

Catálogo de Clientes

Módulo de dados onde são guardados todos os de produtos do ficheiro Clientes.txt.

Os dados são guardados numa Tabela de Hash com 26 posições referentes às 26 letras do alfabeto. Cada índice contém um apontador para um array com os códigos de clientes de uma dada letra associada a esse índice. Deste modo procuramos tornar a procura de um certo cliente mais eficiente.

A estrutura deste módulo é quase igual á do catálogo de produtos, só que em vez de guardar strings de produtos guarda strings de clientes, não achamos necessário criar uma estrutura diferente para estes dois módulos devido á sua semelhança.

Api

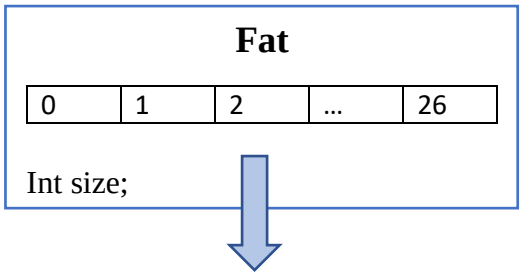
- **int hash_p (char *cont):** Função que recebe uma string e subtrai ao primeiro elemento da string a letra A.
- **THash* initTab_p ():** Função que inicia uma estrutura THash alocando espaço para todas as estrututas adjacentes.
- **void destroiTab_p (THash *p):** Função que destrói uma estrutura THash libertando o espaço ocupado por esta.
- **void acrescentaTab_p (THash *h, char*cont):** Função que acrescenta a uma THash uma determinada string.
- **swapc (char**arg1, char**arg2):** Função que troca duas strings de posição.
- **quicksortc (char **args, unsigned int len):** Função que ordena um array de strings.
- **int validacliente (char *cliente):** Função que recebe uma string de um cliente e verifica se este é válido.
- **int ler_cliente (THash *cliente, char*filespath, int *p):** Função que lê de um ficheiro para uma Thash.
- **char* getCliente (THash *c, int key, int i):** Função que cria um clone de uma string cliente.
- **char** getArrayCl (THash *c, int key):** Função que cria um clone de um array de strings cliente.
- **int getArrayClSize (THash *c, int key):** Função que retorna o tamanho de um determinado array da tabela.

Faturação global

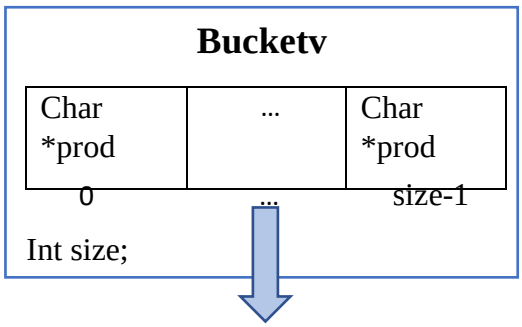
Módulo de dados que irá conter as estruturas de dados responsáveis pela resposta eficiente a questões quantitativas que relaciona os produtos às suas vendas mensais ou globais, em modo Normal (N) ou em Promoção (P). Este módulo deve referenciar todos os produtos, mesmo os que nunca foram vendidos. Tal como nos produtos iniciamos a estrutura Faturação com uma tabela de hash (Fat) que contém 26 posições correspondentes a uma letra do alfabeto. Para cada uma dessas posições definimos um array ordenado de apontadores para uma estrutura Prd que contém uma string de produto e um array de filiais. Este array de apontadores tem tamanho 3 e cada umas das posições corresponde a uma filial, que incorpora um array de 12 posições que correspondem aos meses e um inteiro que verifica se está ocupado. Dentro de cada mês temos então a informação necessária para efetuar a faturação de um produto.

Com esta organização conseguimos detetar certas vantagens tais como a eficaz procura de informação referente a um produto, graças ao fato do array de produtos se encontrar ordenado, porém certas desvantagens devido ao grande número de estruturas usadas.

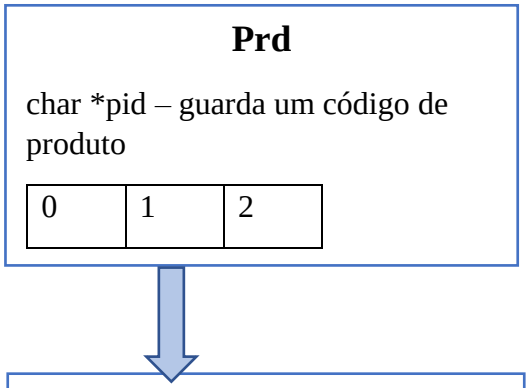
```
typedef struct fat{
        Bucketv *tbl;
} Fat;
```



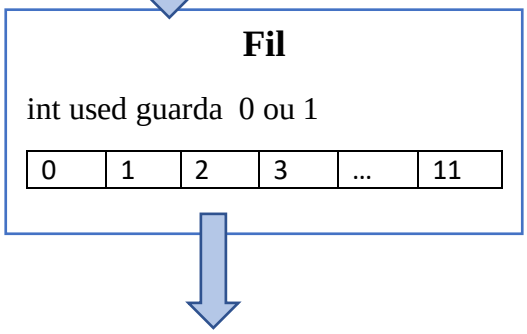

```
typedef struct bucketv{
        int size;
        Prd *arr;
} Bucketv ;
```



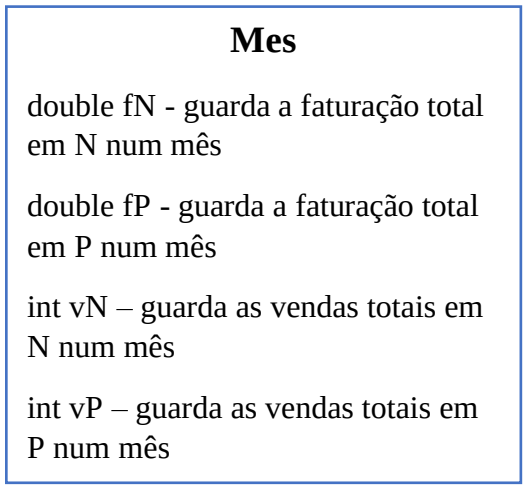

```
typedef struct prd{
        char *pid;
        Fil *fil;
} Prd;
```




```
/* Mes size = 12 */
typedef struct fil{
        int used ;
        Mes *mes;
} Fil;
```




```
typedef struct mes{
        double fN;
        double fP;
        int vN;
        int vP;
} Mes ;
```

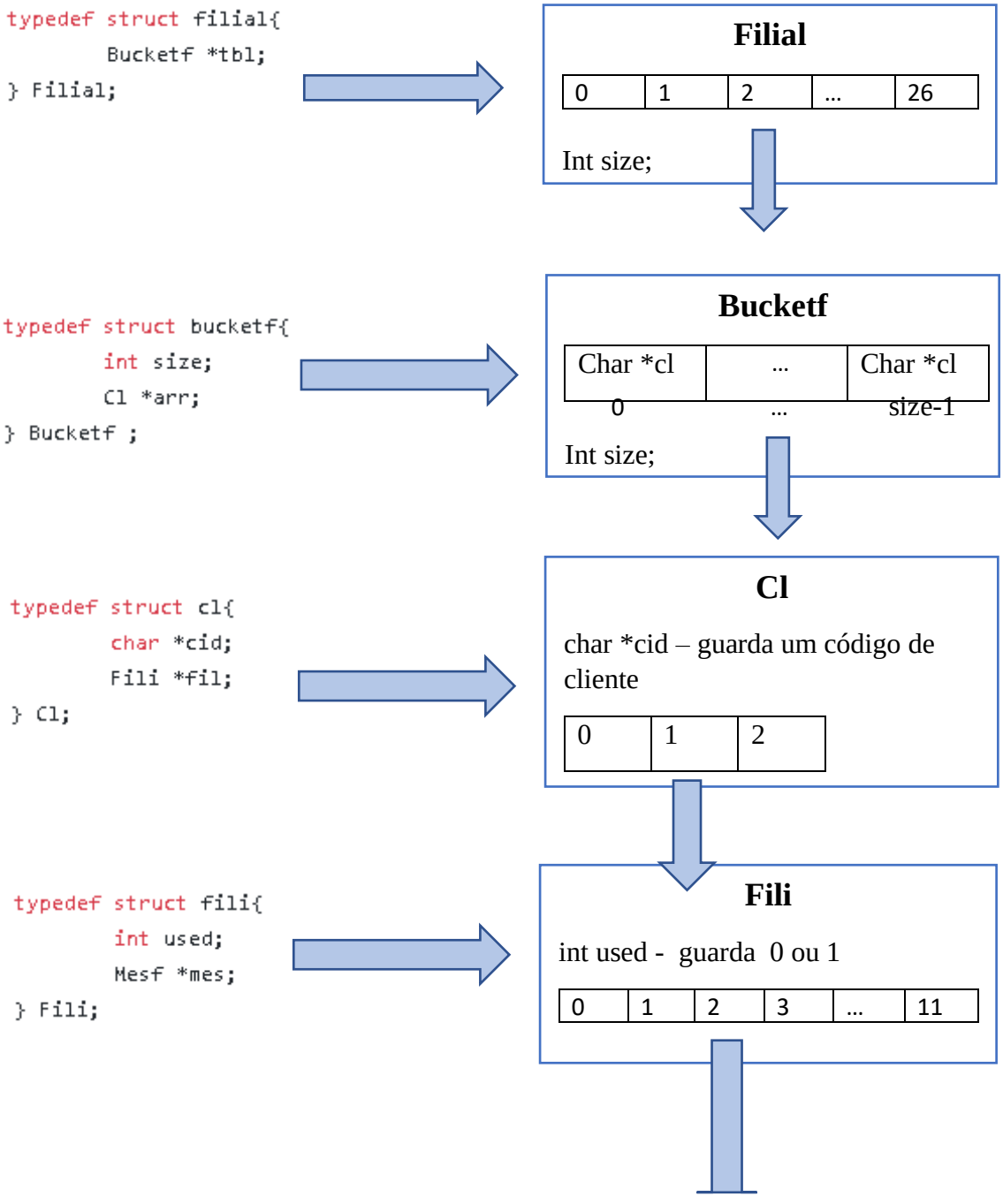


Api

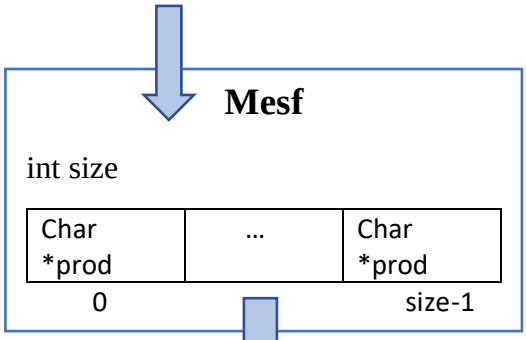
- **Int getPosicaoProd (Fat *fat, char*productID):** Função que retorna o índice de um determinado produto numa estrutura Fat.
- **int getVendasN (Fat *fat, int h, int pos, int m, int f):** Função que retorna o número de venda em N de um determinado produto.
- **int getVendasP (Fat *fat, int h, int pos, int m, int f)**: Função que retorna o número de venda em P de um determinado produto.
- **double getFaturacaoN (Fat *fat, int h, int pos, int m, int f):** Função que retorna a faturação de N de um determinado produto.
- **double getFaturacaoP (Fat *fat, int h, int pos, int m, int f):** Função que retorna a faturação de P de um determinado produto.
- **int getSizeArrayP (Fat *f, int key):** Função que retorna o tamanho de um array de um determinado índice da tabela.
- **int getFilialUsed(Fat *f, int key, int ip, int fil):** Função que retorna se a filiaĺ foi "usada" ou não.
- **char* getProdFat(Fat *f, int key, int ip):** Função que retorna a string correspodente ao produto nos indices dados.
- **int hashfat(char *cont):** Função que recebe uma string e subtrai ao primeiro elemento da string a letra A.
- **Fat* initFat():** Função que inicia uma estrutura Fat alocando espaço para todas as estruturas adjacentes.
- **void destroiFat(Fat *f):** Função que destrói uma estrutura Fat libertando o espaço ocupado por esta.
- **void acrescenta_prod(Fat *f, char *p):** Função que acrescenta a uma Fat um produto.
- **int acrescenta_prods (Fat *f, char **p, int tam):** Função que acrescenta a uma Fat vários produtos lidos de um array a strings.
- **int existe_fat(Prd *arr, char *procurado, int Tam):** Função que retorna o índice de um determinado produto da estrutura Fat caso ele exista -1 caso não.
- **void acrescentaFat(Fat *h, char*p, double pr, int q, char e, char *c, int m, int f):** Função que acrescenta a uma Fat informação correspondente a um produto.
- **char** neverBoughtFil(Fat *f, int fil, int *tamp):** Função que retorna um array de strings com os produtos que nunca foram comprados numa filial em específico.
- **int ProdutosNaoComprados (Fat *f):** Função que retorna o número dos produtos que nunca foram comprados.
- **void FaturacaoeVendasIntervalo (Fat *f, int m1, int m2, int *result, double *result2):** Função que retorna a faturacao(número de vendas e faturado) num intervalo de meses

Filial

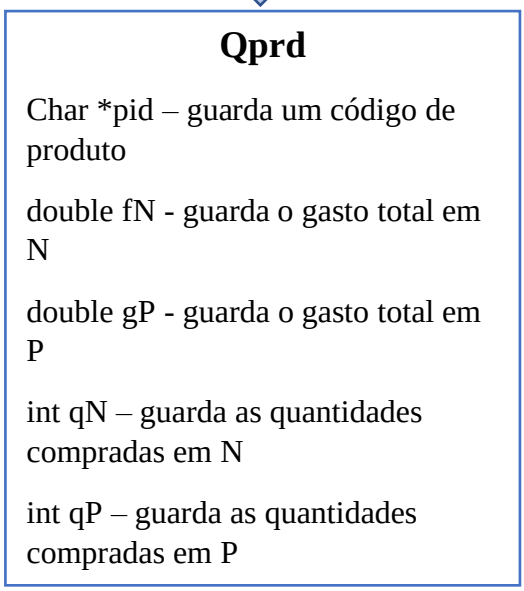
Módulo de dados que, a partir das leituras, conterá estruturas de dados adequadas á representação das relações entre produtos e clientes, ou seja, para cada cliente, saber quais os produtos que compraram, indicando quantidades, e tipo de compras em cada mês e filial. Esta estrutura assemelhasse bastante á estrutura faturação com a única diferença de em vez de receber produtos recebe clientes e é lhe acrescentada mais uma estrutura dentro de cada mês, sendo esta um array não ordenado de estruturas de informação de compras realizadas por um certo cliente.



```
typedef struct mesf{
        int size;
        Qprd *prs;
} Mesf ;
```




```
typedef struct qprd{
        char *pid;
        int qN;
        int qP;
        double gN;
        double gP;
} Qprd ;
```



Api

- **int getFilUsed(Filial *f, int k, int ip, int fil):** Função que verifica se uma filial é usada.
- **char* getCLiente(Filial *f, int k, int ip):** Função que devolve um cliente.
- **int getSizeArrClient(Filial *f, int k):** Função que devolve o tamanho do array num determinado índice da tabela.
- **Cl* getArrByLetter(Filial *f, int key):** Função que devolve um array de clientes começados por uma determinada letra.
- **int getSizeQprd(Filial *f, int k, int id, int fil, int m):** Função que devolve o tamanho do array de produtos num determinado mês.
- **int getQuantN(Filial *f, int k, int id, int fil, int m, int pid):** Função que devolve q quantidade de compras em N para um determinado produto num mês.
- **getQuantP(Filial *f, int k, int id, int fil, int m, int pid):** Função que devolve a quantidade de compras em P para um determinado produto num mês.
- **int getGastoP(Filial *f, int k, int id, int fil, int m, int pid):** Função que devolve gasto total em P de um determinado produto.
- **int getGastoN(Filial *f, int k, int id, int fil, int m, int pid):** Função que devolve o gasto total em N de um determinado produto.
- **char* getOneProd(Filial *f, int k, int id, int fil, int m, int p):** Função que devolve um produto.

- **int hashfil(char *cont):** Função que recebe uma string e subtrai ao primeiro elemento da string a letra A.
- **Filial* initFilial():** Função que inicia uma estrutura Filial alocando espaço para todas as estruturas adjacentes.
- **void destroiFilial(Filial *f):** Função que destrói uma estrutura Filial libertando o espaço ocupado por esta.
- **void acrescenta_cl(Filial *f, char *p):** Função que acrescenta a uma Filial um cliente.
- **int acrescenta_cls (Filial *f, char **p, int tam):** Função que acrescenta a uma Filial vários clientes lidos de um array a strings.
- **int existe_fil(Cl *arr, char *procurado, int Tam):** Função que retorna o índice de um determinado cliente da estrutura Filial caso ele exista, -1 caso não.
- **int existe_prod(Qprd *arr, char *procurado, int Tam):** Função que retorna o índice de um determinado produto de um cliente caso ele exista, -1 caso não.
- **void acrescentaPtoFil(Filial *h, char *p, int it, int a int f, int m, char e, int qnt, int preco):** Função que acrescenta um produto a um cliente na Filial.
- **void acrescentaFil(Filial *h, char*p, double pr, int q, char e, char *c, int m, int f):** Função que acrescenta um produto a um cliente na Filial.
- **char** ClientsOfAllBranches (Filial *f, int *tam):** Função que retorna um array de strings com os clientes que compraram em todas as filiais.
- **int ClientesSemCompras (Filial *f):** Função que retorna um array de strings com os clientes que nunca fizeram compras.
- **int QuantidadesUmClientePorMes(Filial *f, char *clienteID ,int fil, int mes):** Função que retorna à quantidade total de produtos comprados num mês.

Modelo, Fluxo e Apresentação

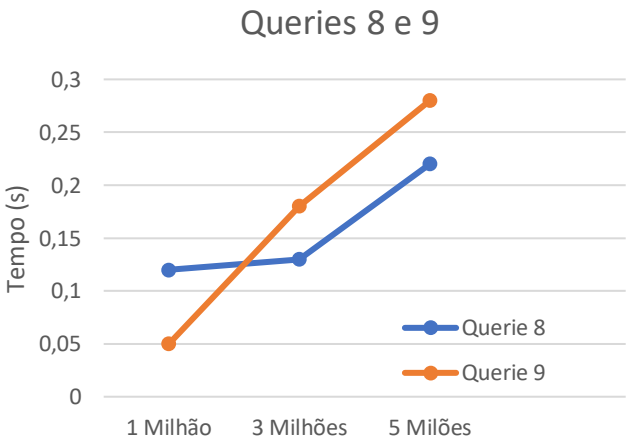
O programa é controlado pela função main, a qual chama o interpretador (controlador), onde são implementadas as funções respetivas ao modelo, ou seja, todas as funções que são usadas na queries incluindo as próprias. São estas funções que leem, executam e armazenam toda a informação necessária para uso da api.

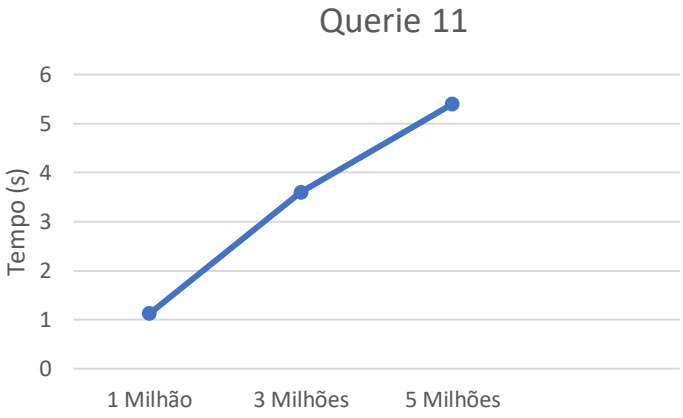
Relativamente á apresentação esta é moderada pelo controlador (o qual recebe informação necessária para dar resposta aos pedidos) e dá display no ecrã o output da informação pedida pelo utilizador.

De forma a exemplificar o contato entre o modelo o fluxo e a apresentação temos então, numa fase inicial a interface começa por imprimir no ecrã todas as opções de consulta dos módulos de acordo com as queries, sendo que não é possível escolher nenhuma sem antes ler os ficheiros (Querie 1). Ao selecionar uma opção é pedido ao utilizador que insira os dados necessários para a consulta dos módulos e então são chamadas as funções de consulta que recebem os dados inseridos como parâmetros. Estas funções guardam os resultados em estruturas especialmente criadas para tal e depois enviam cópias dos resultados de volta para a interface para que esta imprima os resultados da procura no ecrã.

Testes de performance

Posteriormente ao desenvolvimento e codificação de todo o projeto, foi-nos proposto realizar alguns testes de performance. Nós depois de efetuar estes testes para todas as queries decidimos por bem apresentar as queries que têm uma alteração significativa relativamente aos ficheiros de Vendas_1M.txt, Vendas_3M.txt, Vendass_5M.txt.





Em primeiro lugar é de notar que estes tempos foram obtidos numa máquina com um processador de 1.8GHz e 8GB de RAM, pelo que em computadores com especificações diferentes destas, é de se esperar resultados diferentes.

Com a análise destes três gráficos concluímos que o tempo que demoramos a ler os ficheiros e a executar estas 3 queries tem tendência a aumentar, o que consideramos aceitável uma vez que a quantidade de vendas vai aumentando de ficheiro para ficheiro. Tendo em conta apenas o gráfico das queries 8 e 9, nota-se que o tempo de execução não varia significativamente, uma vez que são registadas variações na ordem dos milissegundos.

Makefile

```
TARGET = projeto

SRC= src
INC= include
LIB= libs
BIN= bin
DOC= docs

SOURCE = $(wildcard $(SRC)/*.c)
OBJECT = $(patsubst %,$(BIN)/%, $(notdir $(SOURCE:.c=.o)))

CC= gcc
CFLAGS= -Wall -g -I$(INC) -ansi -D_GNU_SOURCE


## Link
$(BIN)/$(TARGET): $(OBJECT)
	$(CC) -o $@ $^

## Compile
$(BIN)/%.o: $(SRC)/%.c
	$(CC) $(CFLAGS) -c $< -o $@

program: $(BIN)/$(TARGET)



clean:
	rm $(OBJECT) $(BIN)/$(TARGET)




help:
	@echo "src: $(SOURCE)"
	@echo "obj: $(OBJECT)"
```

Variáveis

Todos os ficheiros .c

Todos os ficheiros .o

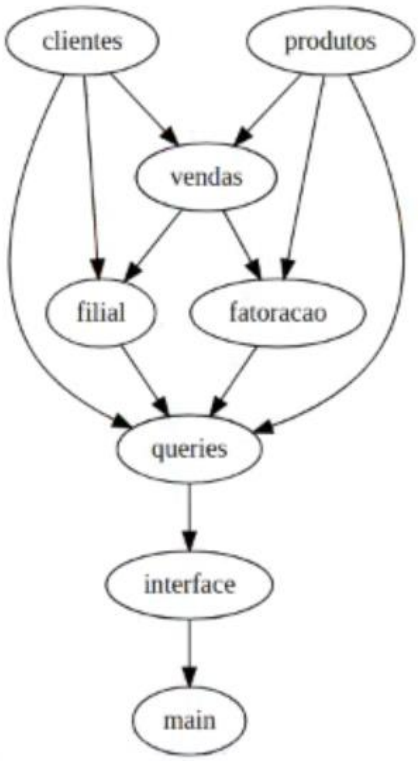
Compilador a

Flags de compilação

Regras da Make

Limpa os .o geradores e o executável

Grafo de dependências



Conclusão

A realização deste projeto provou-se desafiante no sentido em que não estávamos habituados a trabalhar com uma perspetiva MVC, não familiarizados com a questão do encapsulamento nem habituados a trabalhar com grandes quantidades de informação, o que nos trouxe dificuldades na implementação e escolha das estruturas para leitura e armazenamento dos módulos. Relativamente às estruturas usadas para a implementação dos módulos de leitura e armazenamento, apesar do uso frequente de hashtables e arrays damo-nos satisfeitos com os resultados embora reconheçamos que poderíamos ter tirado proveito de outros tipos de estruturas.

Posto isto, ganhamos mais ferramentas para trabalhar com programas de maior escala como o Valgrind que nos ajudou na verificação de memory leaks , o doxygen para gerar documentação em html e familiarizarmo-nos bastante com o gdb e aprofundamos o conhecimento no uso desta linguagem.