# Universidade do Minho

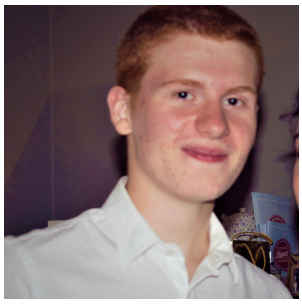
## Mestrado Integrado em Engenharia Informática

# Sistema de Gestão de Vendas

Laboratórios de Informática 3

2º Ano, 2º Semestre, 2018/2019

Grupo 25

*84668 - FranciscoPeixoto*

*86268 - Maria Pires*

*85242 - Maria Regueiras*


8 de abril de 2019

# Conteúdo

# Capítulo 1

# Introdução

Este relatório aborda a resolução do projeto prático em C de LI3. O projeto consiste, resumidamente, em construir um sistema de gestão de vendas, respondendo a queries de forma eficiente, aplicando conhecimentos de algoritmia e programação imperativa.

Ao logo deste relatório são abordadas as decisões tomadas na implementação do projeto, nomeadamente as estruturas utilizadas para criar cada um dos módulos, a razão das escolhas tomadas e as suas APIs.

Inicialmente, foi necessário fazer a validação dos ficheiros e foram criados diferentes módulos. Estes módulos são: a validação e criação dos novos ficheiros, as estruturas de cada tipo de dados e o código auxiliar para executar corretamente as queries fornecidas pelos professores, e finalmente, a interface do sistema.

Depois dos ficheiros serem carregados, somos capazes de executar uma lista de queries disponibilizadas pela equipa docente. Para responder às diferentes queries são utilizadas as funções definidas nas API dos diferentes módulos anteriormente referidos.

# Capítulo 2

# Validação e geração de ficheiros

## 2.1 Validação

O projeto foi inicializado com a validação dos ficheiros de texto fornecidos de modo a garantir que não existiam clientes ou produtos com o código errado, ou vendas com informação errada. A validação dos produtos e dos clientes foi feita de modo semelhantes uma vez que só diferiam de um carácter. Analisando e separando os caracteres dos números para facilitar a organização por ordem alfabética.

Para a validação das vendas recorremos à função *tokens()* de modo a separar a informação de uma venda pelos campos necessários à resolução do trabalho.

## 2.2 Geração de ficheiros validos

Posteriormente, a informação valida foi transferida para novos ficheiros de texto por ordem alfabética, conforme indicado no enunciado, para simplificar a resolução dos seguintes problemas.

# Capítulo 3

# Estruturas Base

Fazendo uma análise às queries, o grupo optou por criar as suas próprias estruturas que servem de base para todo o trabalho.

```
typedef struct array
{
  int inUse;
  int freeSpace;
  int* valor;
}*Array;
```

```
typedef struct strings
{
  int inUse;
  int freeSpace;
  char** string;
}*Strings;
```

```
typedef struct strings_u
{
  int inUse;
  int freeSpace;
  char** string;
  int* unidades;
}*Strings_Unidades;
```

As estruturas são arrays dinâmicos, diferem apenas no tipo. Sendo o primeiro um array dinâmico do tipo int e os restantes de strings.Todas têm um campo "inUse"que indica quantas posições estão a ser usadas no array e um campo "freeSpace"que indica quantas posições estão livres. A terceira tem ainda um campo adicional "unidades"que se tornou importante no módulo vendas por registar as unidades vendidas de um certo produto. As estruturas tornaram a leitura dos ficheiros para a memória e posterior procura de elementos mais rápida e simples de visualizar.

## 3.1 Catálogo de Produtos

### 3.1.1 Estruturas Aplicadas

```
struct produtos{
  Array tabela_produtos[676];
};

typedef struct produtos* Produtos;
```

Neste módulo criamos uma tabela de Produtos com 676 linhas, onde cada linha corresponde a uma combinção de letras, por exemplo, "AA". Escolhemos fazer desta forma pois facilitava tanto a inserção como a procura de um Produto na tabela. Definimos "Produtos"como um apontador para a estrutura em si.

### 3.1.2 Funções

No .h deste módulo incluímos as seguintes funções:

```
int search_P(Produtos p, char id[]);
```

Nesta função passamos uma estrutura Produto p e um id que queremos procurar em p, através de uma procura binária no array dinâmico (devolvendo se encontrou ou não).

```
Produtos init_Produtos(int num[6]);
```

Passando uma lista de ints de 6 posições, a função inicia a estrutura Produtos lendo o ficheiro de Produtos, validando, e escrevendo um novo ficheiro todos os Produtos validados. Escreve nas posições correspondentes no array o número de clientes na estrutura e o número de Clientes escritos no ficheiro.

## 3.2 Catálogo de Clientes

### 3.2.1 Estruturas Aplicadas

```
struct clientes{
  Array tabela_clientes[26];
};

typedef struct clientes* Clientes;
```

Tal como no módulo anterior, implementou-se uma tabela de clientes com 26 posições onde cada linha corresponde a uma letra do abecedário, de forma a facilitar tanto a inserção como a procura de clientes. Definimos também "Clientes"como o apontador para a estrutura em si.

### 3.2.2 Funções

```
int search_C(Clientes c, char id[]);
```

Em semelhança às funções do módulo dos Produtos, esta função procura um id numa estrutura Clientes, através de uma procura binária, devolvendo se encontrou ou não.

```
Clientes init_Clientes(int num[6]);
```

A função inicia a estrutura Clientes lendo o ficheiro de Clientes, validando, e escrevendo um novo ficheiro todos os Clientes validados. Escreve no array "num"os Clientes que estão na estrutura e o número de Clientes escritos no ficheiro.

## 3.3 Vendas

No módulo vendas, é definida a estrutura Vendas, recorre ao modulo Arrayd para a construção de uma venda, através da leitura e escrita de dados deste tipo. Definimos "Clientes"como um apontador para a estrutura em si.

### 3.3.1 Estruturas Aplicadas

```
struct vendas
{
  Strings vendas;
};

// Funcao que inicializa as estruturas
Vendas init_Vendas(int* num, Produtos p, Clientes c);
```

### 3.3.2 Funções

A função da API inicializa a estrutura recorrendo à validação do ficheiro Vendas.txt, através da analise de cada segmento da linha, permite separar os campos da informação, escrevendo-a depois no novo ficheiro de vendas válido.

## 3.4 Faturação

### 3.4.1 Estruturas Aplicadas e funções

```

//Adiciona a letra correspondente ao produto
Strings meteletra(Produtos p, char l1);

//Devolve o total faturado tanto N como P
double totalFaturado(Strings v);

//Devolve o total faturado de N e P separados
void totalFaturadoNP(Strings v, double* num);

//Determinar as vendas de um produto numa filial + mes
Strings getVendasProdFM(char* path, char* code, int mes, int f);

//Determina todas as vendas
Strings getvendas_PinM(char* path, char* code, int mes);

//Calcula o total de vendas (num[0]) e o faturado (num[1])
void totalV_F(char* path, int inicio, int fim, double n[2]);
```

O modulo faturação contém as funções responsáveis pela resolução das queries que envolvem os produtos, e as receitas geradas com as suas vendas.

Na resolução da query 2 foi usada a primeira função que adiciona a letra correspondente ao produto.

As funções seguintes são utilizadas pela query 3 para calcular o total faturado global e dividido por tipos (Normal e promoção).

A ultima função é utilizada na query 8 para obter o número total de vendas num intervalo fornecido pelo utilizador e o total faturado.

## 3.5 Filial

### 3.5.1 Estruturas Aplicadas

```
1
2  //Preenche a estrutura strings com os produtos numa determinada
       filial
3  Strings comprados_Filial(char* path, int f);
4
5  //Determina os produtos que n o foram comprados
6  Strings naoComprado(char* path, Strings comprados);
7
8  //Funcao que realiza procura por filial numa Strings
9  int procura_cliente(Strings s[3], char* cl);
10
11 //Fun  o que realiza insercao por filial numa Strings
12 void insert_stringF(Strings s[3], char* c, int f);
13
14 //Funcao que l  os clientes para um array
15 int le_Clientes(char* path, Strings s);
16
17 //Separa  os clientes v lidos por filial
18 int le_Vendas(char* path, Strings s[3]);
19
20 //Funcao que filtra os clientes que fizeram compras nas 3 filiais
21 void filter_CF(Strings f[3], Strings clientes, Strings f1);
22
23
24 //Determinar as vendas de um produto numa filial
25 Strings getVendasProdFilial(char* path, char* code, int f);
26
27 //Funcao que recolhe as vendas de um cliente num determinado mes
28 Strings_Unidades CinM_comprou(char* path, char* code, int mes);
29
30 //Cria a lista de produtos comprados por um cliente num mes
31 void cria_listaProdutos(char* path, char* ccode, int mes);
```

O modulo filial contém as funções responsáveis pela resolução das queries que envolvem os clientes, a sua relação com os produtos e as compras em cada uma.

A query 4 utiliza as duas primeiras funções para preencher uma estrutura que possui todos os produtos vendidos e posteriormente filtra aquelas que não foram compradas no ficheiros de produtos validos.

A query 5 utiliza da função 3 à função 7 para determinar a lista ordenada de códigos de clientes que realizaram compras em todas as filiais.

A query 9 utiliza a função getVendasProdFilial para determinar as vendas de um produto numa filial.

A query 10 utiliza a função CinM e a função crialistaProdutos que determina as compras de um cliente num determinado mês.

# Capítulo 4

# Resolução de Queries

Para a resolução das queries e acesso aos dados foi definida uma API simples que permite:

1. Query 1: Inicialização das estruturas Clientes, Produtos e Vendas, bem como a validação das mesmas. Grau de complexidade de inserção: O(n*log(n)) Grau de complexidade de procura: Olog(n)

2. Query 2: Determinação dos produtos cujo código se inicia por uma determinada letra; Grau de complexidade da leitura: O(1)

3. Query 3: Determinação do número de vendas, total faturado com estas, e distinção do resultado por meses e por filial; Grau de complexidade da leitura: O(1) Grau de complexidade da inserção: O(1)

4. Query 4: Distinção do tipo de produtos (Promoção ou Normal); Grau de complexidade: indeterminado

5. Query 5: Determinação de códigos de produtos não comprados e de clientes registados que não efetuaram compras; Grau de complexidade: indeterminado

6. Query 8: Verificação do registo de vendas num determinado intervalo de tempo; Grau de complexidade da leitura: O(1)

7. Query 9: Determinação de um código de Clientes que comprou um Produto distinguido por N e P; Grau de complexidade

8. Query 10: Determinar a lista de Produtos que um Cliente comprou num determindado mês, por quantidade e por ordem decrescente.

Com estas funções a resolução da maioria das queries mostrou-se trivial.

# Capítulo 5

# Conclusão

Em suma, o grupo considera que o projeto teve algum sucesso, apesar de não ter sido concluído. Através da validação dos ficheiros conseguimos extrair a informação relevante para o projeto, de uma maneira otimizada, passando-a para estruturas de dados pensadas e feitas por nós na totalidade.