Universidade do Minho
Mestrado Integrado em Engenharia Informática

Laboratórios de Informática III

Relatório

Sistema de Gestão de Vendas

Grupo 29

João Nuno Costa Neves
A81366

Gonçalo Pinto Nogueira
A86617

Carlos Manuel Marques Afonso
A82529

Índice

Conteúdo

1. Introdução 3
2. Descrição dos Módulos 3
2.1 Módulos de Dados 3
2.1.1Catálogo de Clientes 3
2.1.1.1 Typedef's(.h) 4
2.1.2 Catálogo de Produtos 4
2.1.2.1 Typedef's(.h) 5
2.1.3 Gestão de Filial. 5
2.1.3.1 Typedef's(.h) 6
2.1.4 Faturação 6
2.1.4.1 Typedef's(.h) 7
2.2 Controller 7
2.3 View 7
3. Testes de desempenho 8
4. Interpretação dos resultados e dificuldades encontradas 9
5. Conclusão e reflexão final 10

1. Introdução

Este projeto foi nos solicitado pelos docentes da unidade curricular de Laboratórios de Informática III, e tem como principal objetivo a realização de um programa que faça a gestão de vendas de uma cadeia de distribuição com três filiais.

O principal desafio deste projeto foi a programação em larga escala, obrigado a utilizar conhecimentos de outras unidades curriculares, tais como programação imperativa e algoritmos e complexidade.

Foi nos também introduzido novos princípios de programação, tais como a Modularidade e o Encapsulamento de dados.

2. Descrição dos Módulos

Neste projeto são 3 as componentes mais relevantes da aplicação, os módulos de dados os quais designados por “Modelo”, o módulo de controlo designado por “Controller” e finalmente o módulo de apresentação designado por “View”.

2.1 Módulos de Dados

Os módulos de dados dividem-se em quatro módulos, sendo estes, o Catálogo de Clientes, o Catálogo de Produtos, a Faturação Global, e por último, a Gestão de Filial. Todos estes módulos estão relacionados com a interface, onde o utilizador pode usufrui das funcionalidades.

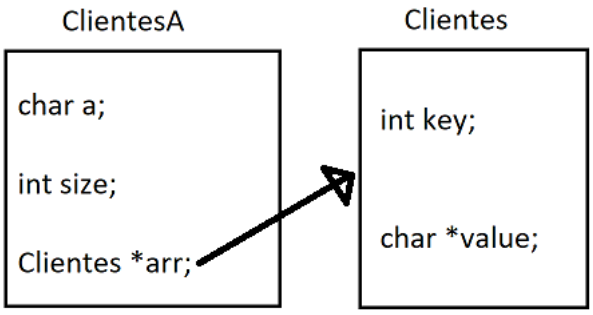
2.1.1Catálogo de Clientes

O Catálogo de Clientes é o módulo que armazena toda a informação relativa ao ficheiro Clientes.txt.

Uma vez que este ficheiro contém muitos dados, optamos por criar uma estrutura:

Struct ClientesA[NUM_ALPHA],

que corresponde a um array com 26 posições, devido às 26 letras do abecedário (cada letra é a letra do código de cliente), onde cada índice contém uma hashtable, e cada chave é o número de um cliente.



2.1.1.1 Typedef's(.h)

-Typedef struct cliente* ClientesA;
-Typedef struct Clientes;

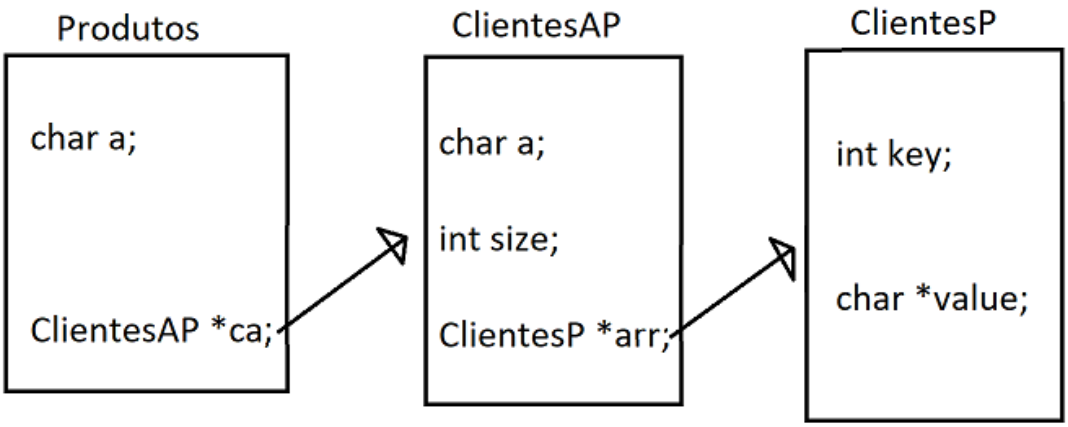
2.1.2 Catálogo de Produtos

O Catálogo de Produtos é o módulo que armazena toda a informação relativa ao ficheiro Produtos.txt.

Tal como o Catálogo de Clientes, também optamos por criar uma estrutura semelhante:

Struct Produtos[NUM_ALPHA],

que corresponde a um array com 26 posições, devido às 26 letras do abecedário (cada índice corresponde a primeira letra do código do produto), onde cada índice contém um outro array com 26 posições (cada posição corresponde à segunda letra do código de produto). Em cada índice existe uma hashtable, cuja chave corresponde ao número do produto.



2.1.2.1 Typedef's(.h)

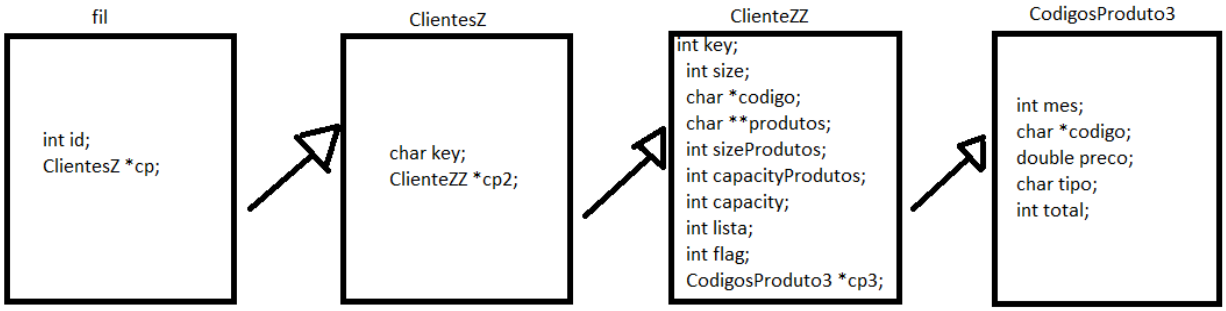
-Typedef struct Produto* Produtos;
-Typedef struct ClientesAP;
-Typedef struct ClientesP;

2.1.3 Gestão de Filial.

Este módulo contém as estruturas responsáveis pelo armazenamento da informação relativa às compras feitas pelos clientes. Para responder a este requesito, criamos a seguinte estrutura,

Struct Filial[3]

Esta estrutura consiste num array com 3 posições, cujo cada índice corresponde a cada filial. Em cada filial, existe um array de 26 posições, onde cada índice corresponde à primeira letra do cliente. A partir daqui, para cada índice, temos uma hashtable, onde a chave é referente ao número do cliente, dentro desta teremos varias informações relativas à existência do código indexado, tais como ¨Existe na lista de codigos¨ ou ¨foi comprado¨ e também um array de strings com os códigos de produto que o cliente em questão comprou, sendo que este ultimo array não contei valores repetidos o que facilita a resposta a algumas querys.Em suma, este modulo, tal como o da faturação, termina com uma hashtable, indexada pelo mês, facilitando também a reposta a algumas querys.



2.1.3.1 Typedef's(.h)

-Typedef struct fil* Filial;

-Typedef struct Zat_Array;

-Typedef struct Zat_Array2;

-Typedef struct Resultados;

-Typedef struct Zat_Numeros;

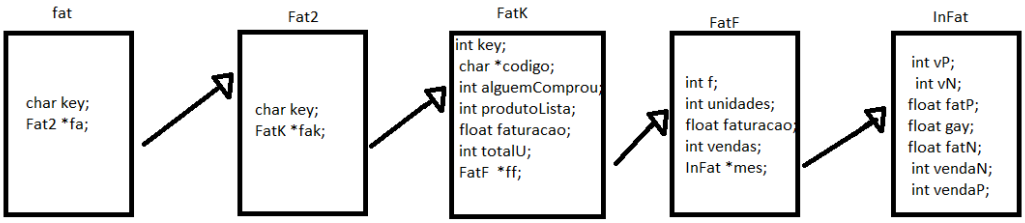
2.1.4 Faturação

Por fim, este módulo de dados contém toda a informação relativa às vendas de produtos, tais como a faturação destes, e número de vendas, distinguindo sempre entre vendas normais, promoção, vendas por filial ou na globalidade.
De forma a guardar estes dados, foi criada a seguinte estrutura,

Struct Faturacao[NUM_ALPHA],

Esta estrutura consiste numa hashtable com 26 posições, cujo cada índice corresponde à primeira letra do produto, onde cada índice contém uma outra hastable com 26 posições (cada posição corresponde à segunda letra do código de produto). A partir daqui cada índice tem uma hashtable, cuja chave referencia o número do produto. Existe ainda uma outra hashtable, onde cada chave referencia o número da filial

correspondente, sendo que por fim teremos uma hashtable final indexada pelo mês correspondente.



2.1.4.1 Typedef's(.h)

-Typedef struct fat* Faturacao;
-Typedef struct Tres;
-Typedef struct Oito;
-Typedef struct Quatro;
-Typedef struct OnzeP;
-Typedef struct Salao;
-Typedef struct Fi4;

2.2 Controller

Esta componente da aplicação serve como motor do programa que coordena tudo o que o mesmo faz. É, por isso, o responsável por toda a interligação de entre os módulos conseguindo comunicar com as outras duas componentes da aplicação realizando pedidos e fornecendo informações sobre o que o utilizador pretende fazer através da recolha das ações do utilizador com o "Scanf". Neste modulo está também implementado o sistema de navegação das listas de resultados das querys que depois envia página a página para a View apresentar ao utilizador.

2.3 View

Esta componente é a responsável pela apresentação de informações ao utilizador.
Entre as várias informações que este módulo pode apresentar estão os diferentes tipos de menus existentes na aplicação, tanto o menu principal como os sub-menus de cada

query, pode apresentar também as listas de resultados das querys quer como páginas de uma lista de resultados quer como tabelas.

3. Testes de desempenho

Queries	Ficheiro 1M	Ficheiro 3M	Ficheiro 5M
6	0.000000 s	0.000000 s	5.000000 s
7	0.000000 s	0.000000 s	0.000000 s
8	0.000000 s	1.000000 s	0.000000 s
9	0.000000 s	0.000000 s	7.000000 s
10	0.000000 s	0.000000 s	0.000000 s
11 (top 100)	6.000000 s	17.000000s	28.000000 s
12	0.000000 s	0.000000 s	0.000000 s

4. Interpretação dos resultados e dificuldades encontradas

Ao longo deste projeto, tentamos sempre atingir o melhor resultado possível, a nível de tempos de execução das queries. Implementamos várias hashtables de forma a ter o tempo de procuras o mais rápido possível. Ficamos agradados com os resultados finais, no entanto, reconhecemos que a estrutura implementada pode não ter sido a melhor, uma vez que a nossa modulação de estruturas pode ser considerada "pesada", pois conseguimos observar uma lentidão durante o carregamento dos ficheiros de 3M e de 5M. Isto poderia ter sido evitado talvez usando uma biblioteca, como por exemplo o glib, mas preferimos tomar esta forma de trabalho.

Relativamente às otimizações feitas, de forma a não alocar memoria a mais, tentamos, sempre que nos foi possível, alocar memoria dinamicamente, fazendo uso de vários reallocs. Não quer isto dizer que não haja memoria alocada estaticamente, uma vez que há, mas tentamos sempre fugir a esse método.

5. Conclusão e reflexão final

De acordo com o objetivo pretendido do projeto, isto é, gerir um rápido acesso a um número bastante elevado de informações, concluímos que esse objetivo foi cumprido, pois os resultados alcançados demonstram uma quantidade de tempo razoavelmente baixa na maioria dos casos.

Ao longo do trabalho, fomos conseguindo diminuir gradualmente o tempo de execução das funcionalidades do programa até a um ponto onde, de facto, não conseguimos aprimorar mais os nossos tempos de execução.

Apesar de todas as dificuldades que o projeto nos colocou, conseguimos afirmar que o resultado é satisfatório, pois conseguimos responder a todas as queries propostas, independentemente do tempo de execução.

Após uma analise na globalidade da eficiência do projeto em questão, chegamos à conclusão que nem tudo pode ficar o máximo de eficiente possível, ou seja, por vezes ao melhorar algumas coisas, outras pioram e vice-versa, para alem disto, temos a consciência que apesar da resposta eficiente à maior parte das queries impostas, existem outras formas de gerir a memoria, que facilitariam a vida ao nosso sistema, um exemplo disso é a forma como guardamos os dados, que apesar de eficiente, coloca-nos numa posição em que abrimos alguns ¨buracos¨ na nossa memoria, tendo consciência que possivelmente essa não fosse a forma ótima de abordar o problema.

Concluindo, com a elaboração deste projeto foram muitas as competências adquiridas e aprimoradas relativamente á linguagem de programação imperativa C. Para alem disto, a grande melhoria foi a forma diferente que tivemos de pensar com o intuito de resolver problemas em larga escala, tornando todos os elementos do projeto mais racionais e obedecendo a regras de forma a manter a integridade e a abstração do maior numero de dados, permitindo nos uma maior preparação para ambientes futuros, ou seja, mais profissionais e complexos.