Laboratórios de Informática III (LI3)

MIEI – 2º ano – 2º semestre – Universidade do Minho – 2019/2020

Projecto de C: Sistema de Gestão de Vendas - SGV

Sistema de Gestão das Vendas de uma Distribuidora com 3 Filiais

1.- Introdução e Objectivos.

O projecto de C da disciplina de LI3 de LEI tem por objetivo fundamental ajudar à consolidação experimental dos conhecimentos teóricos e práticos adquiridos nas UCs anteriores. A estes conhecimentos acrescenta-se a introdução de novos conceitos fundamentais da área de Engenharia de Software e o desafio de processar grandes volumes de dados.

Os objetivos de LI3, e deste trabalho, não se restringem apenas a aumentar os conhecimentos dos alunos na linguagem C, o que seria até questionável, mas, e talvez fundamentalmente, apresentar aos alunos de forma pragmática, os desafios que se colocam a quem concebe e programa **aplicações software** (em qualquer linguagem), quando passamos a realizar a designada **programação em larga escala**, ou seja, aplicações com grandes volumes de dados e com mais elevada complexidade algorítmica e estrutural.

De facto, quando passamos para tais patamares de complexidade, torna-se imperioso conhecer e usar os **melhores princípios da Engenharia de Software**, de modo a que tais projectos de software, em geral realizados por equipas, possam ser concebidos com melhor estrutura, de modo a que sejam mais facilmente modificáveis, e sejam, apesar da complexidade, o mais optimizados possível a todos os níveis.

Para que tal seja possível teremos que introduzir novos princípios de programação, mais adequados à programação em grande escala, designadamente:

- Modularidade e encapsulamento de dados usando construções da linguagem;
- Criação de código reutilizável;
- Escolha optimizada das estruturas de dados e reutilização;
- Testes de *performance* e até *profiling*.

Este projecto, a desenvolver em trabalho de grupo (de no máximo 3 alunos), visa a experimentação e aplicação destas práticas de desenvolvimento de software **usando a linguagem C,** práticas que são extensíveis a outras linguagens e paradigmas.

O desenvolvimento deste projeto deve ser feito colaborativamente com o auxílio de *GIT* e *GITHUB* (tutorial em https://rogerdudler.github.io/git-guide/index.pt_BR.html). Cada elemento do grupo de trabalho deverá criar um conta (caso não possua) no *GITHUB* e fornecer o respetivo ID aos docentes (mais informações e instruções serão divulgadas em breve via Blackboard). Será criado pelos docentes um repositório para cada grupo de trabalho. Os docentes serão membros integrantes da equipa de desenvolvimento de cada trabalho e irão acompanhar semanalmente a evolução dos projetos. A entrega do trabalho será efetuada através desta plataforma e os elementos do grupo serão avaliados individualmente de acordo com a sua contribuição para o repositório. Serão fornecidas algumas regras que os grupos deverão seguir para manter e estruturar o repositório, de forma a que o processo de avaliação e de execução dos trabalhos possa ser uniforme entre os grupos e possa ser efetuada de forma automática.

2.- Requisitos da aplicação a desenvolver.

O projecto a desenvolver em **C** (usando **gcc**), **SGV**, tem como fonte de dados três ficheiros de texto disponibilizados no BB da disciplina.

No ficheiro **Produtos.txt** **cada linha** representa o **código dos produto** à venda, sendo cada código de produto formado por duas letras maiúsculas e 4 dígitos (que representam um inteiro entre 1000 e 9999), cf. os exemplos,

AB9012
XY1185
BC9190

O ficheiro de produtos conterá cerca de 200.000 códigos de produtos que devem ser validados.

No ficheiro **Clientes.txt** **cada linha** representa o **código de um cliente** identificado no hipermercado, sendo cada código de cliente formado por uma letra maiúscula e 4 dígitos que representam um inteiro entre 1000 e 5000, cf. os exemplos:

F2916
W1219
F2915

O ficheiro de clientes conterá cerca de 20.000 códigos de cliente que devem ser validados.

O ficheiro que será a maior fonte de dados do projecto designa-se por Vendas_1M.txt, no qual **cada linha** representa o **registo de uma venda** efectuada numa qualquer das 3 filiais da Distribuidora. Cada linha (a que chamaremos **compra** ou **venda**, o que apenas depende do ponto de vista) será formada por um **código de produto**, um **preço unitário decimal** (entre 0.0 e 999.99), o **número inteiro de unidades compradas** (entre 1 e 200), a letra **N** ou **P** conforme tenha sido **uma compra Normal ou uma compra em Promoção**, o **código do cliente**, o **mês da compra** (1 .. 12) e a **filial** (de 1 a 3) onde a venda foi realizada, cf. os exemplos seguintes:

```
KR1583 77.72 128 P L4891 2 1
QQ1041 536.53 194 P X4054 12 3
OP1244 481.43 67 P Q3869 9 1
JP1982 343.2 168 N T1805 10 2
IZ1636 923.72 193 P T2220 4 2
```

O ficheiro de vendas inicial, Vendas_1M.txt, conterá 1.000.000 (1 milhão) de registos de vendas realizadas nas 3 filiais da cadeia de distribuição.

Para todos os efeitos, não se deve considerar que todas as linhas foram "bem formadas" pelo programa que as criou. Assim, cada linha deverá ser validada e tal registo apenas deverá ser aceite se for válido.

Em termos estritamente numéricos, teremos portanto um ficheiro com 1 milhão de registos, em que cada linha regista uma venda de uma dada quantidade de um produto a um dado cliente num dado mês e numa dada filial. Note-se que o preço final não está calculado mas, dado que temos a quantidade e o preço unitário, tal poderá será feito de forma simples sempre que necessário.

Tais compras envolvem em geral um número garantidamente elevado de clientes (no máximo 20.000 mas em geral menos pois **nem todos compram**) e um universo máximo de 200.000 produtos (em geral menos pois há sempre **produtos que ninguém compra**).

Independentemente do volume de dados, importantes serão as decisões quanto à forma do seu armazenamento em memória, em função do que pretendermos realizar com tais dados.

2.1.- Arquitectura da aplicação.

No sentido de se deixar desde já uma orientação de base quanto à arquitectura final da aplicação a desenvolver, pretende-se de facto que a aplicação possua uma arquitectura na qual, tal como apresentado na figura seguinte, se identifiquem de forma clara as fontes de dados, a sua leitura e os módulos de dados a construir:

- **Leitura:** função ou parte do código no qual é realizada a leitura (e tratamento) dos dados dos 3 ficheiros;

- **Catálogo de Produtos:** módulo de dados onde deverão ser guardados os códigos válidos de todos os produtos do ficheiro **Produtos.txt**, organizados por índice alfabético, o que irá permitir, de forma eficaz, saber quais são os produtos cujos códigos começam por uma dada letra do alfabeto, quantos são, etc.;

- **Catálogo de Clientes:** módulo de dados onde deverão ser guardados os códigos válidos de todos os clientes do ficheiro **Clientes.txt**, organizados também por índice alfabético;

- **Facturação 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, 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;

- **Gestão de Filial:** módulo de dados que, **para uma Filial**, 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

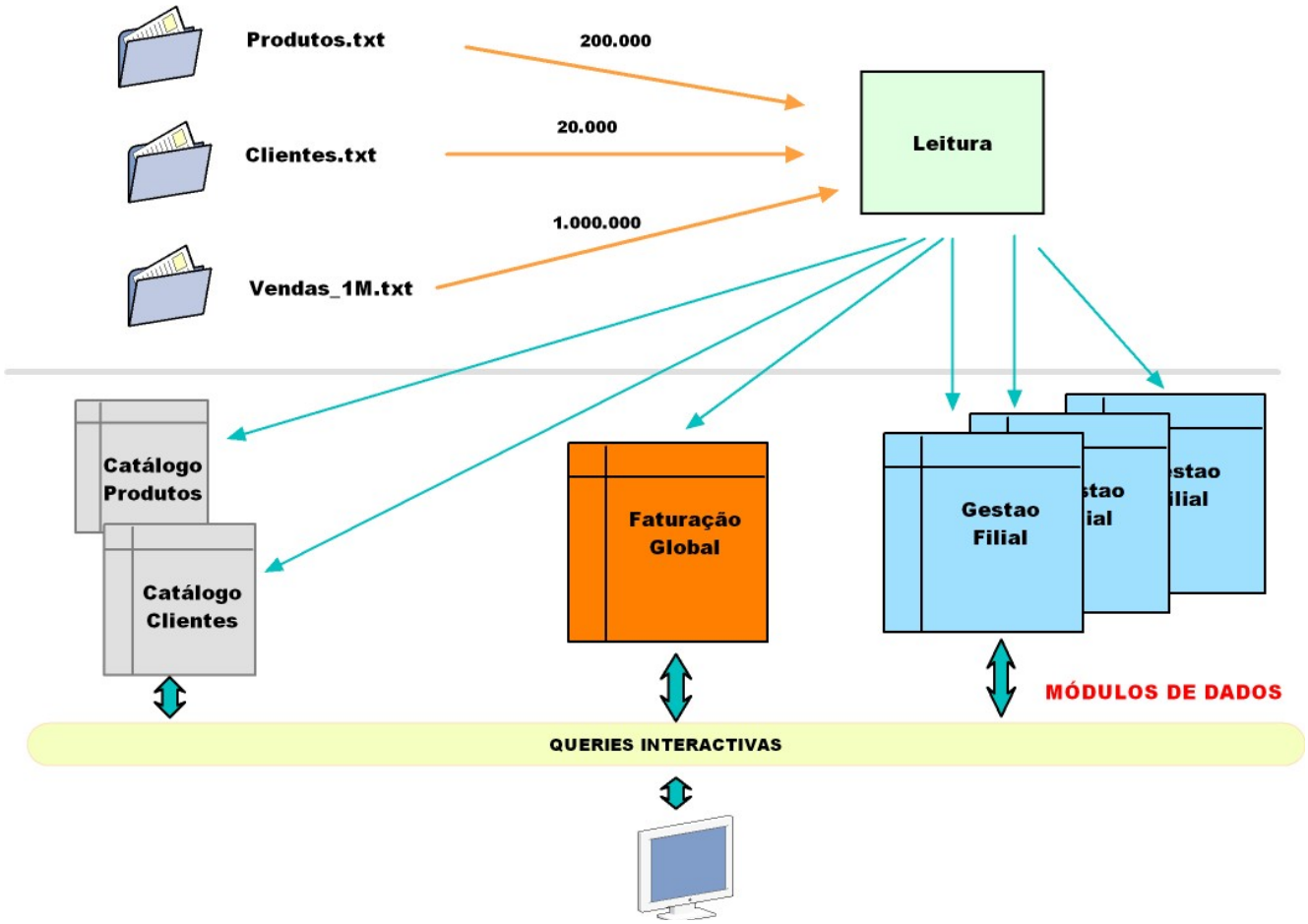
os clientes que o compraram, quantas unidades cada um comprou, em que mês, etc.

Para a estruturação optimizada dos dados destes 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.

Teremos assim, uma primeira arquitectura de base para o projecto, sendo desde já de salientar que esta arquitectura (ou algo equivalente) irá ser construída por fases, módulo a módulo, testando cada fase e cada módulo até se ter a garantia de que podemos **juntar todas as unidades do projecto na aplicação final**.

A **aplicação final terá ela própria uma estruturação particular** (a apresentar e a estudar), usando-se um padrão de software em que as três grandes componentes de uma aplicação serão codificadas de forma totalmente separada e independente:

- camada de dados e algoritmos, que designaremos por **MODELO**;
- camada de interação com o utilizador, cf. menus, entradas de ddaos, listas de resultados, etc, que designaremos por **APRESENTAÇÃO**;
- camada responsável pela ordem de execução correcta da aplicação em função dos desejos dos utilizadores, que designaremos por **CONTROLO DE FLUXO** ou apenas **FLUXO** ou **CONTROLADOR**;



Arquitectura de referência para o SGV

2.2.- Queries interactivas.

Tendo sido apresentada a arquitectura genérica da aplicação, a efectiva estruturação de cada um dos módulos depende, naturalmente, da funcionalidade esperada de cada um deles e da funcionalidade geral da aplicação. Tal é, naturalmente, completamente dependente das *queries* (consultas) que a aplicação deve implementar como funcionalidade disponível para o utilizador final.

Deste modo, e fornecida que foi uma arquitectura de referência, deixa-se ao critério dos grupos de trabalho a concepção das soluções, módulo a módulo e seguindo os princípios de criação de módulos de dados apresentados, que satisfaçam a implementação de cada uma das *queries* que podem ser realizadas pelo utilizador e, até, a forma adequada de apresentação das queries ao utilizador (cf. estruturação de menus, etc.).

Contudo, apesar da concepção da arquitetura e dos módulos ser deixada ao critério do grupo de desenvolvimento, o módulo que representa o sistema SGV deverá obrigatoriamente suportar um conjunto de funcionalidades (correspondentes às *queries)* com a assinatura apresentada na Figura 1. O valor de retorno (identificado com <?>) de cada uma destas consultas fica ao critério do grupo, devendo ser definido tendo em conta aspetos de usabilidade e reutilização de informação. Mais concretamente, estes valores deverão consistir em representações/estruturas genéricas, que não revelem e partilhem a representação interna dos módulos e possam ser (re)utilizadas para adequadamente apresentar a informação ao utlizador do sistema. O programa principal do projeto(i.e. a "*main*") deve importar e utilizar este ficheiro denominado "interface.h", de forma a responder à interação com o utilizador.

```
SGV initSGV();
SGV destroySGV();
SGV loadSGVFromFiles(SGV svg , char* filesFolderPath);
<?> getProductsStartedByLetter(SGV sgv, char letter);
<?> getProductSalesAndProfit(SGV sgv, char* productID, int month);
<?> getProductsNeverBought(SGV sgv, int branchID);
<?> getClientsOfAllBranches(SGV sgv);
<?> getClientsAndProductsNeverBoughtCount(SGV sgv);
<?> getProductsBoughtByClient(SGV sgv, char* clientID);
<?> getSalesAndProfit (SGV sgv, int minMonth, int maxMonth);
<?> getProductBuyers (SGV sgv, char* productID, int branch);
<?> getClientFavoriteProducts(SGV sgv, char* clientID, int month);
<?> getTopSelledProducts(SGV sgv, int limit);
<?> getClientTopProfitProducts(SGV sgv, char* clientID, int limit);
```

Deve notar-se, desde já, que estamos claramente a realizar **programação em larga escala** e com grandes volumes de dados sem apoio de Bases de Dados. Assim, por exemplo uma simples consulta de todos os códigos de produtos começados pela letra A poderá gerar uma **lista resultado com milhares de códigos**, que será necessário saber apresentar ao utilizador, ou seja, **programando os adequados mecanismos de navegação na solução de dados obtida** a partir do módulo adequado e a apresentar no

programa principal (ou numa qualquer sua função). Tal implicará a criação de módulos adicionais genéricos, designados Navegadores ou Cursores para strings.

Assim, e sem qualquer ordem particular, o **SGV** deve ser capaz de dar ao utilizador respostas eficazes às seguintes "questões" cf. as seguintes operações:

1. Ler os 3 ficheiros (Produtos, Clientes e Vendas), cujos nomes poderão ser introduzidos pelo utilizador ou, opcionalmente, assumidos por omissão (sendo neste caso os ficheiros anteriormente referidos). O resultado desta leitura deverá ser a apresentação imediata ao utilizador do nome do ficheiro lido e que vai ser usado, o número total de linhas lidas e validadas. Note-se que qualquer nova leitura destes ficheiros deverá de imediato reiniciar e refazer as estruturas de dados em memória como se de uma reinicialização so tratasse;
2. Determinar a lista e o nº total de produtos cujo código se inicia por uma dada letra (maiúscula);
3. Dado um mês e um código de produto, ambos válidos, determinar e apresentar o número total de vendas (nº de registos de venda) e o total facturado com esse produto em tal mês, distinguindo os totais em modo N e os totais em modo P. O utilizador deverá decidir se pretende o resultado global ou os resultados filial a filial para todas as 3 filiais.
4. Determinar a lista ordenada dos códigos dos produtos (e o seu número total) que ninguém comprou, podendo o utilizador decidir igualmente se pretende valores totais ou divididos pelas filiais.
5. Determinar a lista ordenada de códigos de clientes que realizaram compras em todas as filiais;
6. Determinar o número de clientes registados que não realizaram compras bem como o número de produtos que ninguém comprou.
7. Dado um código de cliente, criar uma tabela com o **número total de produtos comprados** (ou seja a soma das quantidades de todas as vendas do produto), mês a mês (para meses em que não comprou a entrada deverá ficar a 0). A tabela deverá ser apresentada em ecrã organizada por filial.
8. Dado um intervalo fechado de meses, por exemplo [1..3], determinar o total de vendas (nº de registos de venda) registadas nesse intervalo e o total facturado;
9. Dado um código de produto e uma filial, determinar os códigos (e número total) dos clientes que o compraram, distinguindo entre compra N e compra P;

10. Dado um código de cliente e um mês, determinar a lista de códigos de produtos que mais comprou por quantidade e não por facturação), por ordem descendente;
11. Criar uma lista dos N produtos mais vendidos em todo o ano, indicando o número total de clientes e o número de unidades vendidas, filial a filial;
12. Dado um código de cliente determinar quais os códigos dos N produtos em que mais gastou dinheiro durante o ano;

3.- Testes de performance.

Depois de desenvolver e codificar todo o seu projecto tendo por base o ficheiro **Vendas_1M.txt**, deverá realizar alguns testes de *performance* e apresentar os respectivos resultados. Pretende-se comparar os tempos de execução das *queries* 6 a 12, usando os ficheiros, **Vendas_1M.txt**, **Vendas_3M.txt** (3 milhões de vendas) e **Vendas_5M.txt (**5 milhões de vendas**)**. Todos os ficheiros serão fornecidos numa pasta disponibilizada via BB.

Para a obtenção destes tempos pode ser usada a bilbioteca standard de C **time.h** ou outra qualquer equivalente. Os tempos a registar devem ser homogéneos, isto é, ou registamos sempre os designados *elapsed times* (o tempo que o utilizador nota que esperou) ou os *CPU times* (tempos de processamento). Pessoalmente sugiro os *elapsed times*.

4.- Requisitos para a codificação final.

A codificação final deste projecto deverá ser realizada usando a linguagem C e o compilador **gcc**. O código fonte deverá compilar sem erros usando o *switch* **-ansi**. Podem também ser utilizados *switches* de optimização. Deverá existir uma Makefile genérica que irá ser fornecida e que permitirá gerar um executável “program" através do comando “make program". Este comando deverá compilar um ficheiro “main.c" que irá incluir o ficheiro “interface.h". Respeitar esta nomenclatura é importante para garantir que o projeto é validado pelos scripts que irão executar nos repositórios GIT para validar o trabalho após a entrega.

Qualquer utilização de bibliotecas de estruturas de dados em C deverá ser sujeita a prévia validação por parte da equipa docente. Não são aceitáveis bibliotecas genéricas tais como LINQ e outras semelhantes.

Caso se justifique, o código final será sujeito a uma análise usando a ferramenta **JPlag**, que detecta similaridades no código de vários projectos, e, quando a

percentagem de similaridade ultrapassar determinados níveis, os grupos serão chamados a uma clara justificação para tal facto.

5.- Apresentação do projecto e Relatório.

O projeto será extraído automaticamente do repositório de cada grupo imediatamente após a data de entrega. *Commits* posteriores à data de entrega com alterações efetuadas no código do projeto não serão consideradas para fins de avaliação. A *makefile* deverá gerar o código executável, e este deverá executar corretamente. Projectos com erros de *makefile*, de compilação ou de execução serão de imediato rejeitados. . Cada elemento do grupo será avaliado individualmente de acordo com a sua contribuição para o repositório do trabalho, tendo em conta o seu número de *commits* e respetivas alterações efetuadas no código do projeto.

Para além do código do projecto, cada grupo deverá elaborar um relatório sucinto (máx. 10 páginas) no qual descreverá a sua implementação do projecto e todas as decisões de concepção do mesmo, designadamente:

- Descrição de cada módulo em termos de API (.h), respectivas estruturas de dados (apresentar um desenho destas) e justificação objectiva para a escolha das mesmas;
- Arquitectura final da aplicação e razões para tal modularidade;
- Complexidade das estruturas e optimizações realizadas;
- Resultados dos testes realizados.

O relatório será entregue aquando da apresentação presencial do projecto e servirá de guião para a avaliação do mesmo. A avaliação presencial do projecto implica a presença de todos os membros do grupo de trabalho, sob pena de reprovação imediata em caso de ausência injustificada. Para além da análise e avaliação da solução em termos estruturais e de eficácia de execução, e do relatório, alguma avaliação individual poderá ser realizada quando tal se justificar.

Uma avaliação final inferior a 10 valores neste primeiro projecto implica a reprovação imediata à UC de LI3.

Todas as questões adicionais deverão ser esclarecidas ao longo das aulas junto da equipa docente.