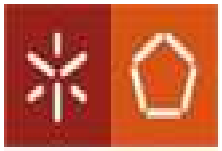


Universidade do Minho
Departamento de Informática

LABORATÓRIOS DE INFORMÁTICA III

MIEI - 2º Ano - 2º Semestre

2018/2019

DI/UM

F. Mário Martins >>> fmm@di.uminho.pt
Francisco Ribeiro
José Nuno Macedo
Rui Rua

OBJECTIVO PRINCIPAL

>>> TRANSIÇÃO PARA A PROGRAMAÇÃO EM LARGA ESCALA

- Milhões de dados; Milhares de linhas de código;
- Problemas de robustez e de manutenção; Modificações permanentes;
- Desenvolvimento em equipa e em contexto de projecto.

☑ **Conhecer os princípios fundamentais da Engenharia de Software**, designadamente **modularidade**, **reutilização**, **encapsulamento** e **abstracção de dados**, **robustez e segurança**, e saber implementá-los em diferentes linguagens/paradigmas de programação: (imperativo em **C** - 1º projecto e POO em **Java** - 2º projecto);

☑ **Complementar experimentalmente os conhecimentos adquiridos** nas Unidades Curriculares de Programação Imperativa, Algoritmos e Complexidade e Programação Orientada aos Objetos;

☑ **Desenhar (conceber), codificar e testar software**, realizando dois projectos concretos de média dimensão em linguagens diferentes.

PROBLEMAS A RESOLVER

☑ **Modularidade** >> **Módulos** >> Como criar módulos de software ?

☑ **Estrutura de uma Aplicação Software** >> Como dividir uma aplicação ?

☑ **Reutilização de Software** >> Usar "builds" e bibliotecas standard;

☑ **Testes de Software** >> Benchmarking.

- 1º projecto - Linguagem C: modularidade em C, encapsulamento, tipos opacos, reutilização de estruturas de dados, leitura de ficheiros de dados, etc.;

- 2º projecto - Linguagem JAVA: Tópicos complementares a POO; Lambdas e Funções; Programação com Colecções e Streams de Dados; Benchmarking.

PORQUÊ C e JAVA ?

TIOBE Index for February 2019

February Headline: Groovy re-enters the TIOBE index top 20

The programming language Groovy has re-entered the TIOBE index top 20. This dynamically typed language that is compatible with Java entered the top 20 for the first time at the end of 2016. At that time it was pushed because scripts for the most popular Continuous Integration tool Jenkins are mainly written in Groovy. Now it is back. Apart from Jenkins also the build automation system Gradle, which is getting popular as well, is using Groovy for its scripting. In general more and more glue software is written in Groovy. Next to Groovy, it is also worth mentioning that Hack has entered the top 50 and TypeScript (temporarily) has fallen back to position 57 this month.

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. It is important to note that the TIOBE index is not about the *best* programming language or the language in which *most lines of code* have been written.

The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found here.

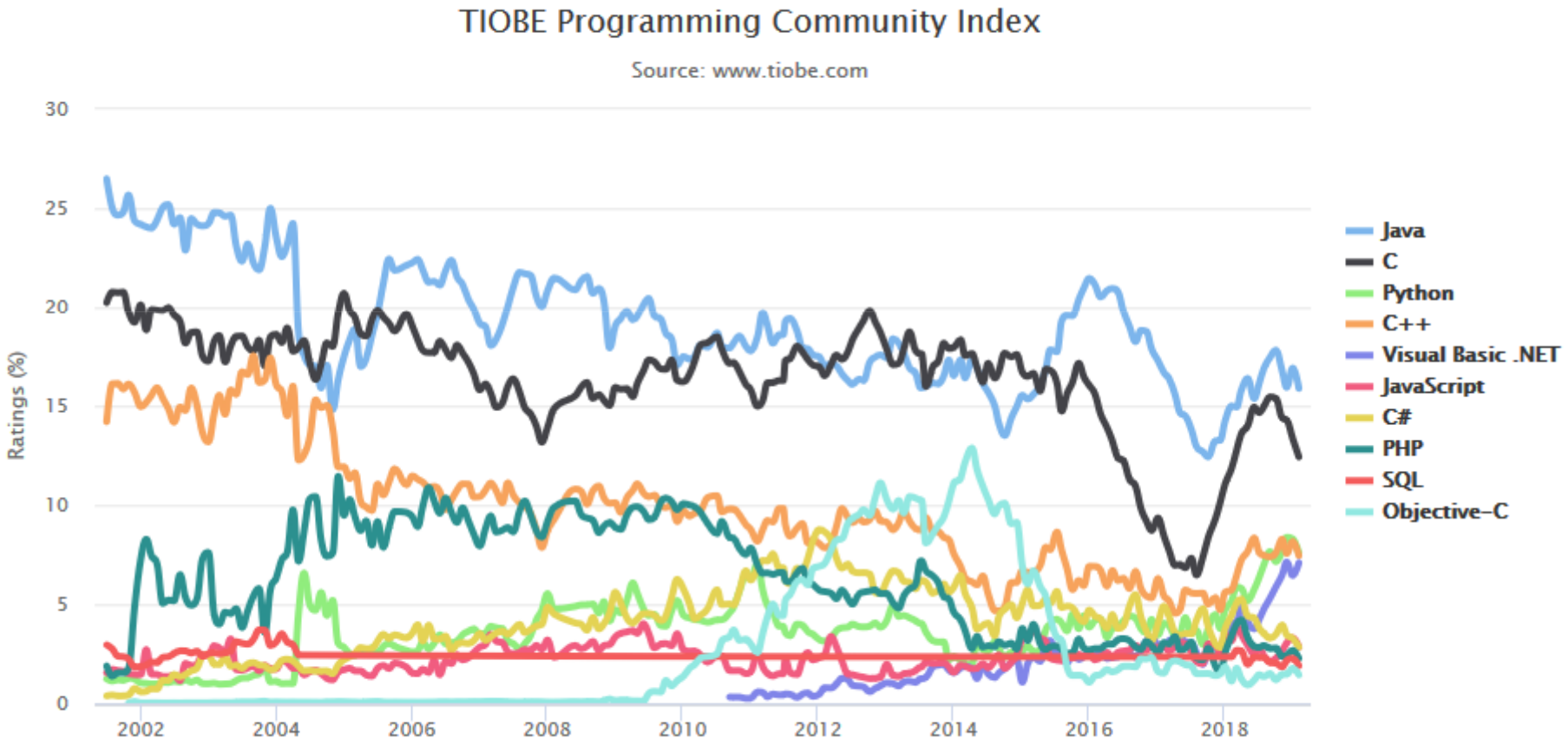
Feb 2019	Feb 2018	Change	Programming Language	Ratings	Change
1	**1**		**Java**	**15.876%**	**+0.89%**
2	2		C	12.424%	+0.57%
3	4	↑	Python	7.574%	+2.41%
4	3	↓	C++	7.444%	+1.72%
5	6	↑	Visual Basic .NET	7.095%	+3.02%
6	8	↑	JavaScript	2.848%	-0.32%
7	5	↓	C#	2.846%	-1.61%
8	7	↓	PHP	2.271%	-1.15%
9	11	↑	SQL	1.900%	-0.46%
10	20	⇈	Objective-C	1.447%	+0.32%

C porque sim.

Java obviamente !!

PORQUÊ C e JAVA ?

Linguagens de Programação



Ver também >> https://insights.stackoverflow.com/survey/2018#most-popular-technologies

FUNCIONAMENTO

▣ **As PLs são momentos reservados a apoio tutorial aos alunos** que necessitem de esclarecer dúvidas e/ou precisem de acompanhamento para a execução dos projectos. **A frequência é obrigatória 2 horas/semana.**

▣ **Um BRP (Boletim de Requisitos de Projecto) definirá semanal ou quinzenalmente as tarefas/objectivos de projecto a atingir nesse tempo.**

► **Neste esquema, os alunos realizarão os dois projectos práticos obrigatórios.**

- O 1º projecto, em C, será de dimensão média e realizado em grupo (de 3 alunos) e terá submissão electrónica e avaliação presencial.
- O 2º projecto, em JAVA8, será realizado em grupo (de 3 alunos) e terá submissão electrónica e avaliação presencial.

► **A fórmula que calcula a nota final pressupõe que ambos os trabalhos foram entregues e ambos possuem avaliação final >= 10:**

Nota Final = 55%*ProjC + 45%*ProjJava

CALENDÁRIO

CALENDÁRIO LI3
2018-2019

Semana	2a.feira	3a.feira	4a. Feira	5a.feira	6a.feira	sábado	
4/2 a 7/2							Semana de SEI
11/2 a 16/2			COMUM				Aula comum de apresentação de LI3 (TP de C)
18/2 a 23/2							
25/2 a 2/3							
4/3 a 9/3		CARNV					
11/3 a 16/3							
18/3 a 23/3							
25/3 a 30/3							
1/4 a 6/4					**		Entrega electrónica do TP de C
8/4 a 13/4							
15/4 a 20/4							*Páscoa*
22/4 a 27/4		JAVA			JAVA		
29/4 a 4/5							
6/5 a 11/5							
13/5 a 18/5							*Semana da queima*
20/5 a 25/5							
27/5 a 1/6							
3/6 a 8/6							Entrega electrónica do TP de Java
10/6 a 15/6		EM		POO			Avaliações do TP de Java
17/6 a 22/6		SO				CP	Lançamento das Notas Finais

	Aula comum em sala a marcar
	entregas electrónicas de trabalhos
	avaliações presenciais dos trabalhos
	laboratórios de C
	laboratórios de Java
	férias feriados
	queima
	Notas finais da UC
**	Aula opcional para todos os alunos

PARADIGMAS

■ EM INFORMÁTICA, E QUALQUER QUE SEJA A PERSPECTIVA, HÁ APENAS DOIS TIPOS DE ENTIDADES COMPUTACIONAIS:

- INFORMAÇÕES;
- TRANSFORMADORES DE INFORMAÇÕES;

■ COMO SÃO CARACTERIZADAS ?

- PELA FORMA ▶ SINTAXE
- PELO SIGNIFICADO ▶ SEMÂNTICA

Passamos a vida a estudar sintaxe e semântica (isto é, linguagens)

PARADIGMA = MODELO COMPUTACIONAL

Um modelo computacional é uma abstracção (simplificação) do processo computacional concreto que se realiza na máquina, que nos permite racionalizar de uma forma simples como é que informações e transformadores interagem para realizar a computação.

PARADIGMA IMPERATIVO

PARADIGMAS TRADICIONAIS: IMPERATIVO, FUNCIONAL, RELACIONAL, POO

▶ **DADOS E OPERAÇÕES SÃO ENTIDADES DISTINTAS E DESLIGADAS, DECLARADAS POR ISSO EM ÁREAS DISTINTAS;**

(relembrar como se faz em ASSEMBLY, PASCAL, C, HASKELL, SQL, etc.)

▶ **PROGRAMAR => APLICAR OPERAÇÕES A DADOS PELA ORDEM CERTA TRANSFORMANDO-OS E GERANDO RESULTADOS.**

este é o modelo f(x) »» operadores aplicados a operandos

Exºs:

```
add x, y;
println( sqrt(lado) );
delete fich1
```

Em POO temos que passar a pensar que dados e operações se definem de forma ligada; os dados possuem as suas próprias operações.

modelo x.f()

MODULARIDADE: Problemas

Questão1: Como dividir os programas em módulos reutilizáveis ?

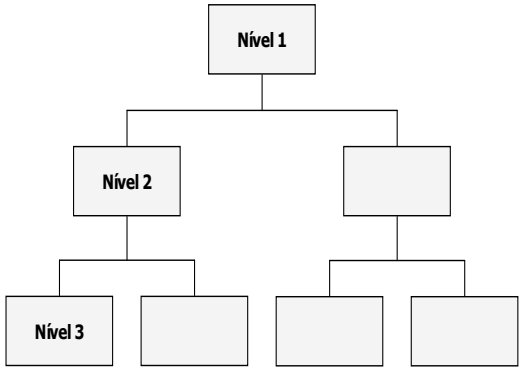
- para não estar sempre a reinventar a roda e para << $$

Questão 2: Como controlar erros e modificações ?

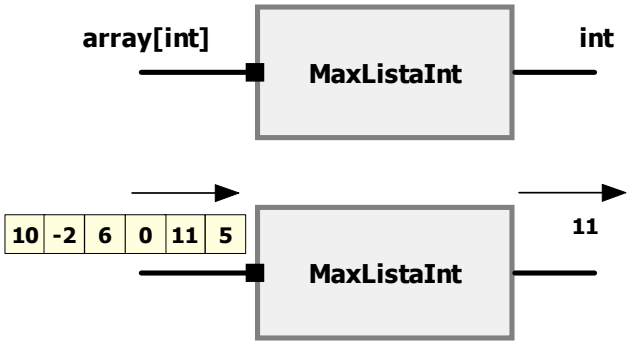
- os programas nunca estão prontos; estão sempre prontos

para serem corrigidos e modificados; fáceis modificações implicam << $$

Soluções tradicionais:

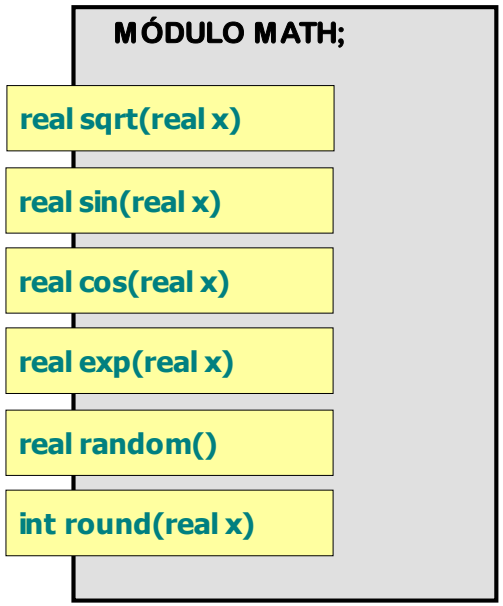


Refinamento Top-Down



Abstracção de Instruções (Procedimental)

MODULARIDADE: Problemas



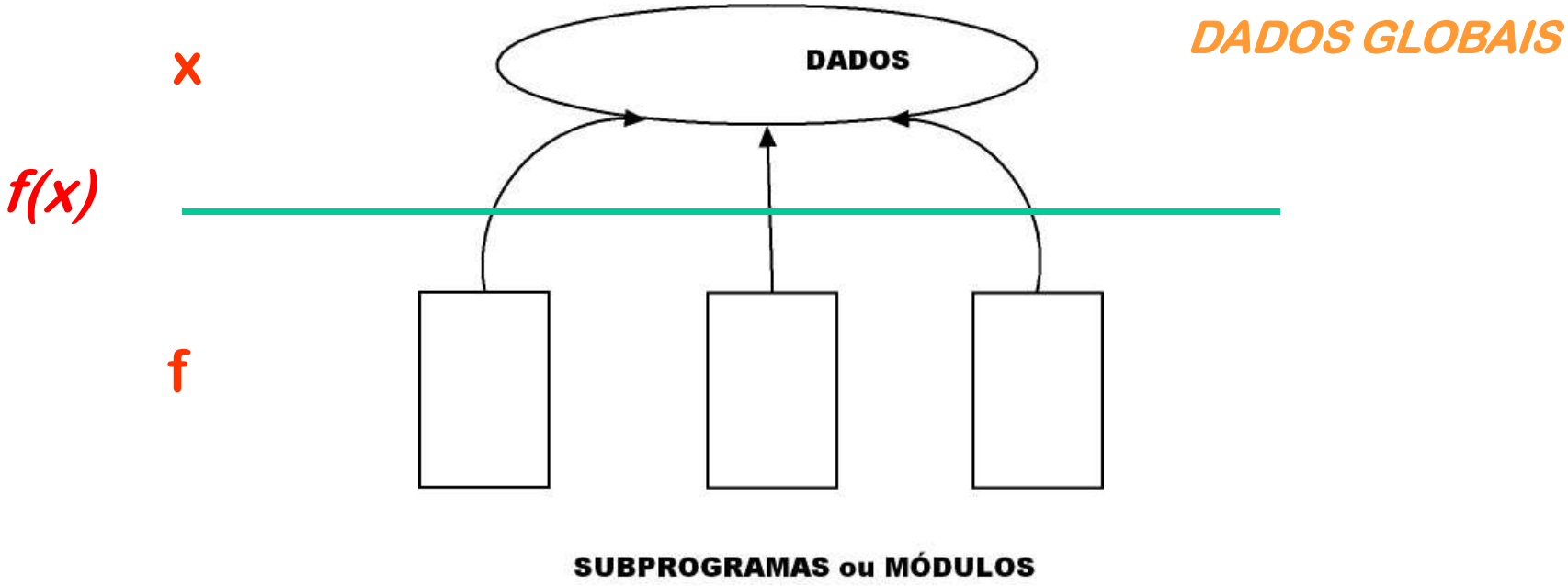
Módulos como abstracções de instruções, tal como em device drivers, módulo de cálculos matemáticos, de I/O, etc.

Assim, originalmente, a noção de **MÓDULO DE SOFTWARE** era a de que :

MÓDULOS = ABSTRACÇÃO DE INSTRUÇÕES ou CONTROLO

PERMITEM: ESTRUTURAÇÃO DE CÓDIGO, REUTILIZAÇÃO DE CÓDIGO, ABSTRACÇÃO, etc., **MAS É PRECISO MAIS …**

MODULARIDADE: Esquema 1

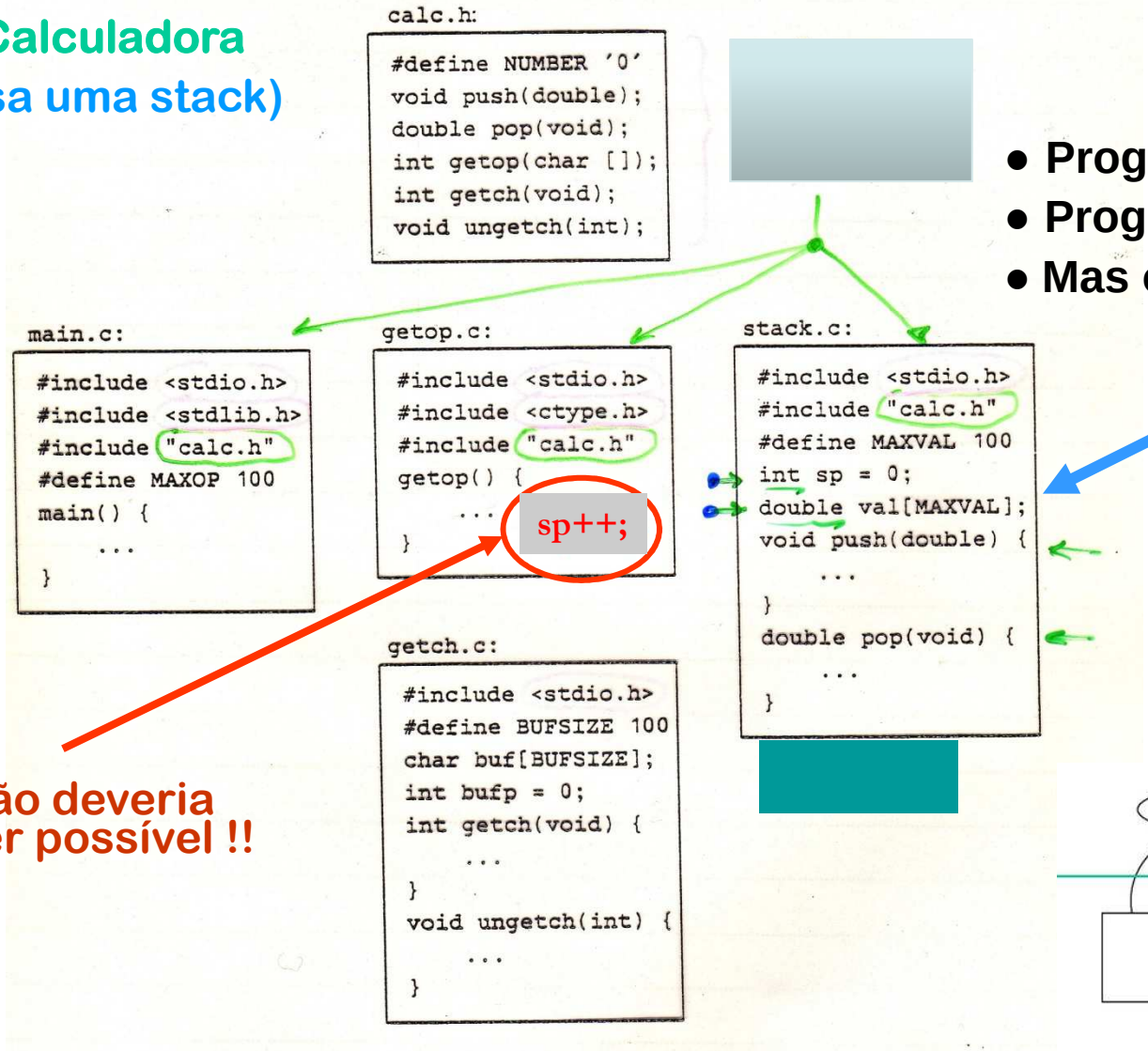


▶ Exemplo estrutural de codificação imperativa típica e exemplo de má modularidade real porque os dados são GLOBAIS !

▶ Princípio de Sherlock Holmes: Erro nos DADOS =>
Qual a instrução suspeita ? Neste exemplo TODAS !

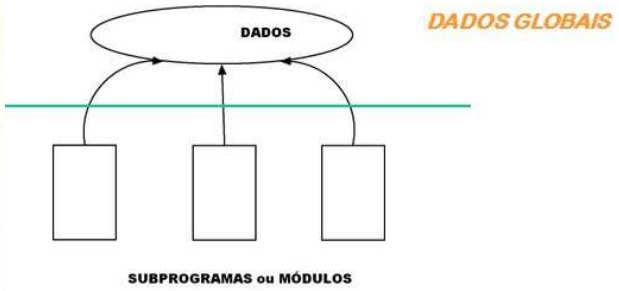
MÁ MODULARIDADE: Exemplo em C

Calculadora
(usa uma stack)

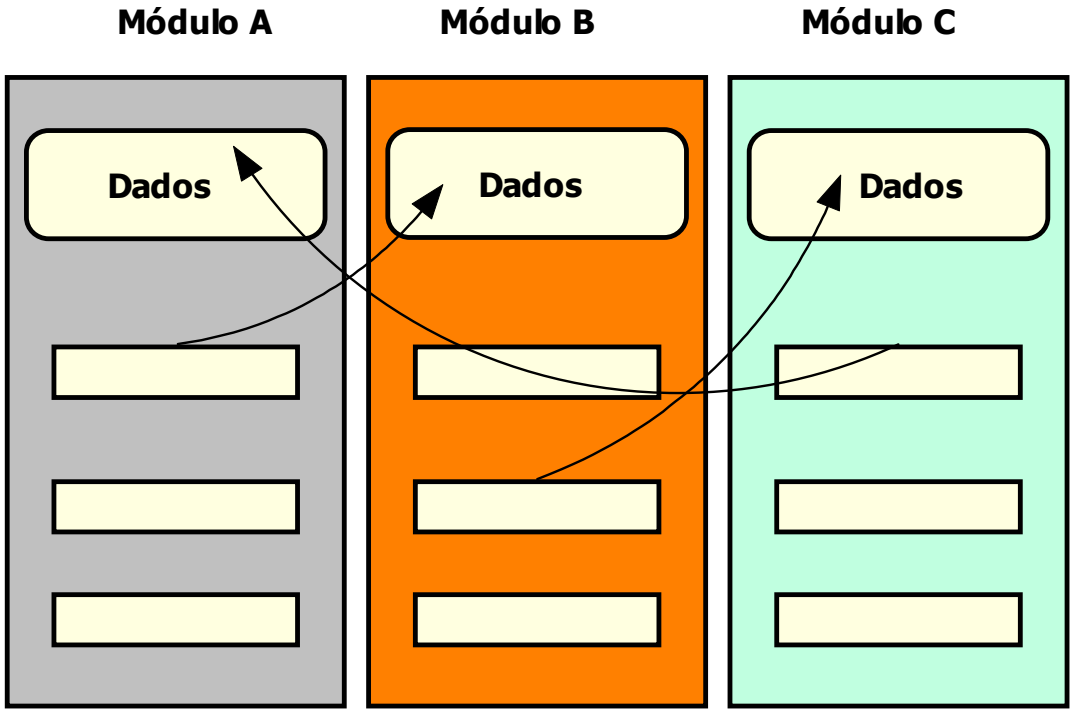


- **Programa está estruturado;**
- **Programa funciona;**
- **Mas os dados são globais !!**

Não deveria ser possível !!



MODULARIDADE: Esquema 2



Se apenas pretendermos usar o módulo A, como A depende de B e B depende de C, teremos que os usar a TODOS.

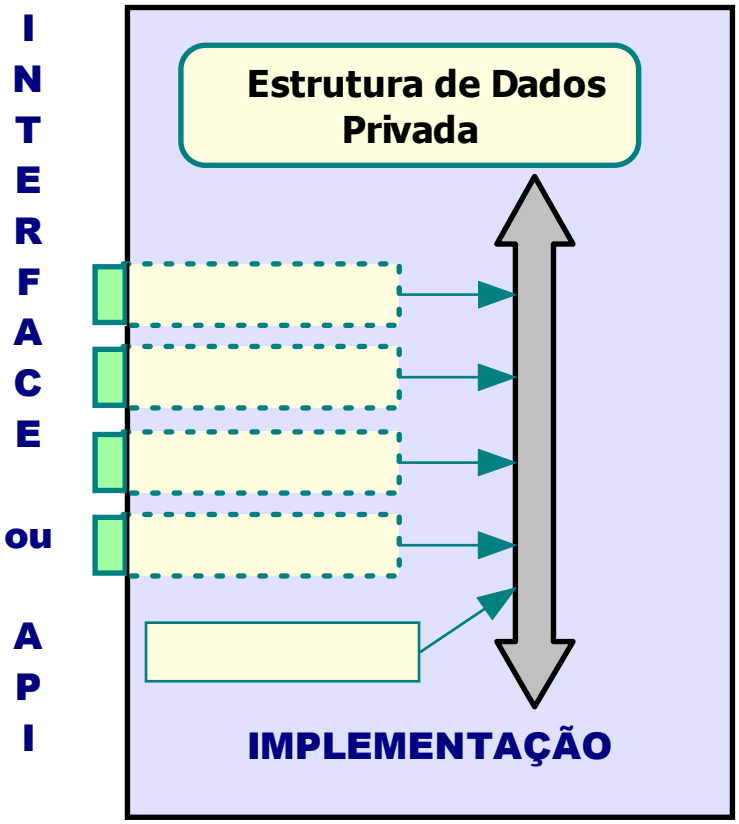
- Estes módulos não são independentes;
- Dados de uns são acedidos por módulos externos;

Solução: Módulo => Estrutura de Dados privada e suas operações

SOLUÇÃO: ENCAPSULAMENTO

Módulo = Abstracção de Dados

Módulo = Interface + Implementação de Estrutura de Dados



MÓDULO É UMA CÁPSULA QUE CONTÉM UMA ESTRUTURA DE DADOS PRIVADA, NÃO ACESSÍVEL DO EXTERIOR, E AS ÚNICAS OPERAÇÕES QUE PODEM ACEDER A TAIS DADOS.

ENCAPSULAMENTO DE DADOS

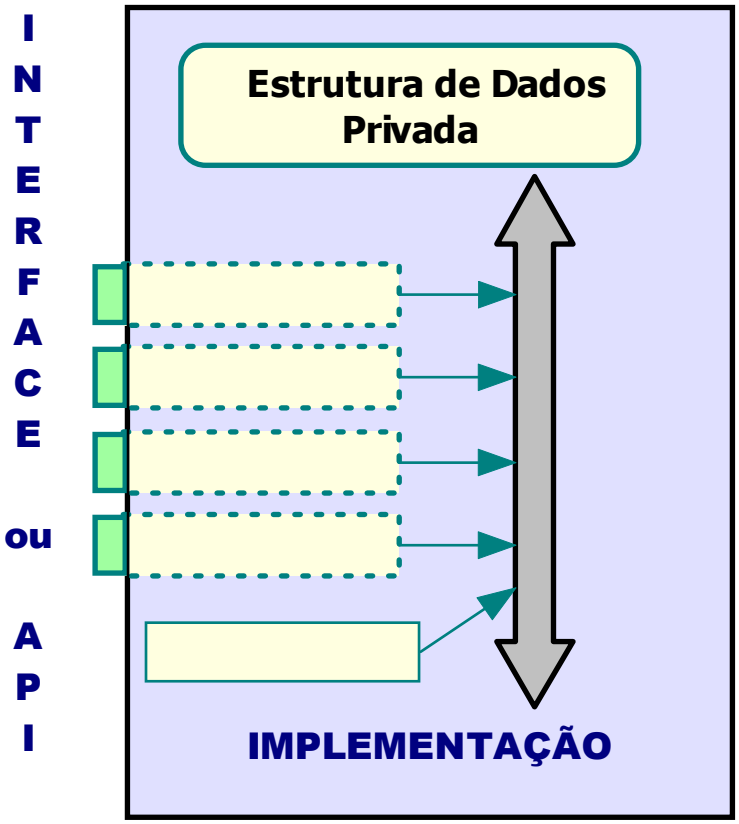
- **Operações podem ser tornadas públicas, ou seja acessíveis do exterior, ou serem apenas internas ao módulo (privadas);**

- **Operações públicas formam a interface do módulo ie. o que pode ser invocado;**

SOLUÇÃO: ENCAPSULAMENTO

Módulo = Abstracção de Dados

Módulo = Interface + Implementação de Estrutura de Dados



- API: Application Programmer´s Interface
Operações que são acessíveis do exterior, ou seja, são tornadas **PÚBLICAS**;

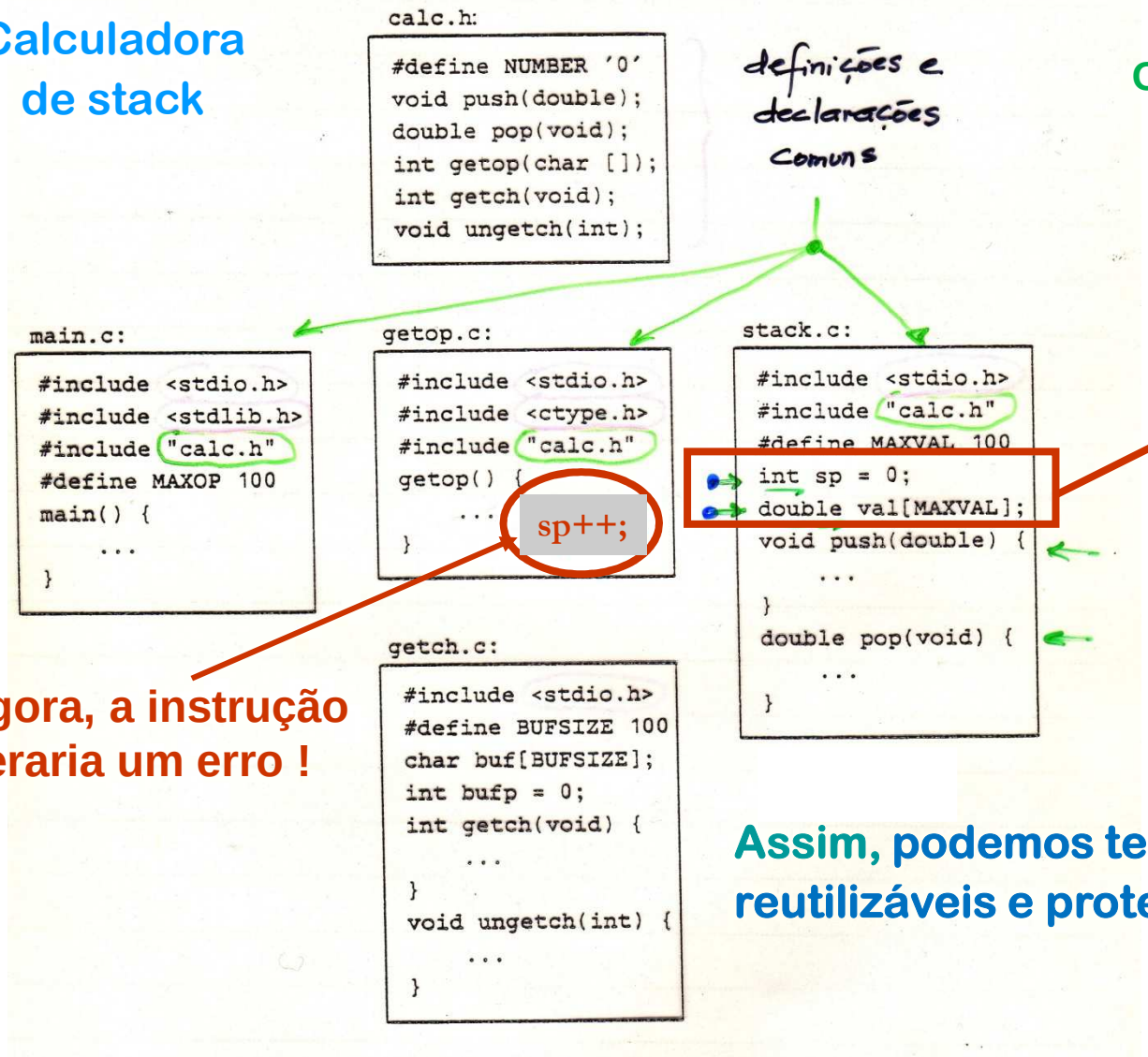
- ERROS: Apenas o código interior ao módulo pode provocar erros nos dados (Sherlock Holmes tem agora a vida muito facilitada);

- ABSTRACÇÃO: a utilização do módulo não obriga (antes pelo contrário) ter que saber qual a representação interna, mas apenas a API; Black-Box de software;

- REUTILIZAÇÃO: módulo é independente e autónomo;

SOLUÇÃO: ENCAPSULAMENTO

Calculadora de stack



O encapsulamento básico pode ser garantido se as variáveis forem declaradas static

Agora, a instrução geraria um erro !

Assim, podemos ter módulos de software reutilizáveis e protegidos, mesmo em C

ENCAPSULAMENTO EM C

Assim, em C, o encapsulamento básico pode ser garantido se as variáveis forem declaradas static tal como sugerido e aconselhado em manuais de C.

► **Uma forma mais elaborada de criação de módulos de dados em C deverá ser estudada, analisada e posteriormente usada, consultando os documentos seguintes colocados no BB de LI3.**

MODULARIDADE EM PROGRAMAS C

F. Mário Martins

LABORATÓRIOS DE INFORMÁTICA III - LEI – 2013/2014

IMPLEMENTAÇÃO EM C DE ABSTRACÇÕES DE DADOS

TÉCNICA DOS TIPOS INCOMPLETOS

F. Mário Martins, LI3, 2015

DESENVOLVIMENTO EM LARGA ESCALA

DESENVOLVIMENTO DE SOFTWARE EM LARGA ESCALA

CONCEITOS FUNDAMENTAIS

- "DATA HIDING"
- "IMPLEMENTATION HIDING"
- ABSTRACÇÃO DE DADOS
- ENCAPSULAMENTO
- INDEPENDÊNCIA CONTEXTUAL

Compiladores não garantem verificação destas propriedades !!

Dados privados e protegidos;

Representação dos dados não deve ser acedida directamente;

Acesso aos dados apenas usando a API;

As operações internas do módulo não devem possuir operações de I/O;

LABORATÓRIOS DE INFORMÁTICA III

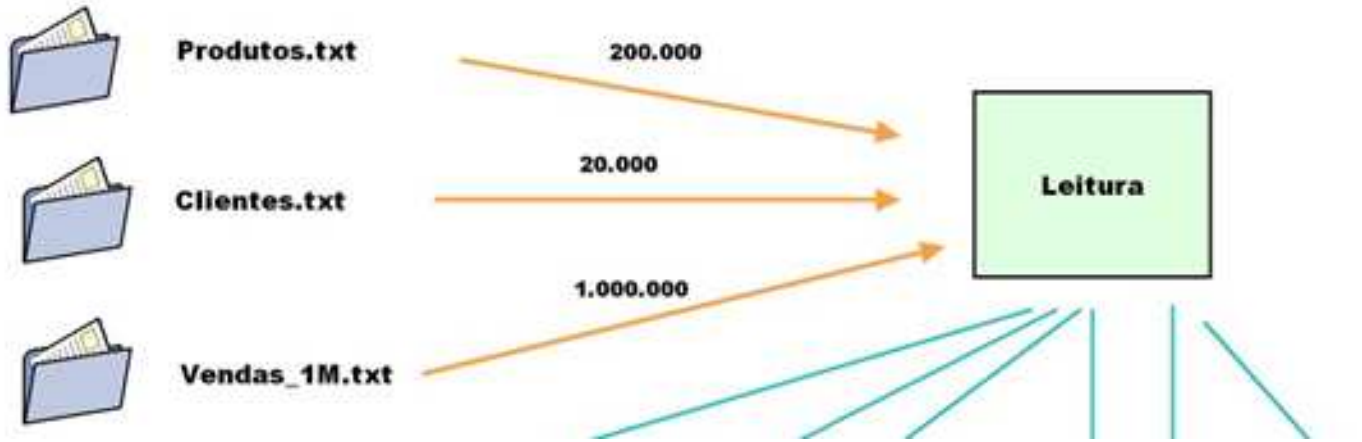
2018/2019
MiEI

TRABALHO PRÁTICO DE C

- ☑ **Enunciado do problema;**
- ☑ **Requisitos de modularidade e funcionalidade;**
- ☑ **Estrutura do Relatório final;**
- ☑ **Avaliação: Critérios gerais.**

SGV: Sistema de Gestão de Vendas de Hipermercado com Filiais

☑ Pretende-se desenvolver uma aplicação em GNU C, com **código standard**, modular e eficiente, quer em termos de algoritmos quer em termos de estruturas de dados implementadas, que seja, antes de mais, capaz de ler e processar as linhas de texto de 3 ficheiros **.txt** indicados, um contendo todos os códigos de **produtos**, outro todos os códigos de **clientes** e o terceiro com registo de todas as **vendas feitas**.



Teremos 200.000 códigos de produtos, 20.000 códigos de clientes e 1.000.000 de compras/vendas.

Projecto de C: SGV

No ficheiro **Produtos.txt** **cada linha** representa o **código de um produto** vendável no hipermercado, sendo cada código formado por duas letras maiúsculas e 4 dígitos (que representam um inteiro entre 1000 e 1999), cf. os exemplos,

AB1012 XY1185 BC1190

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.:

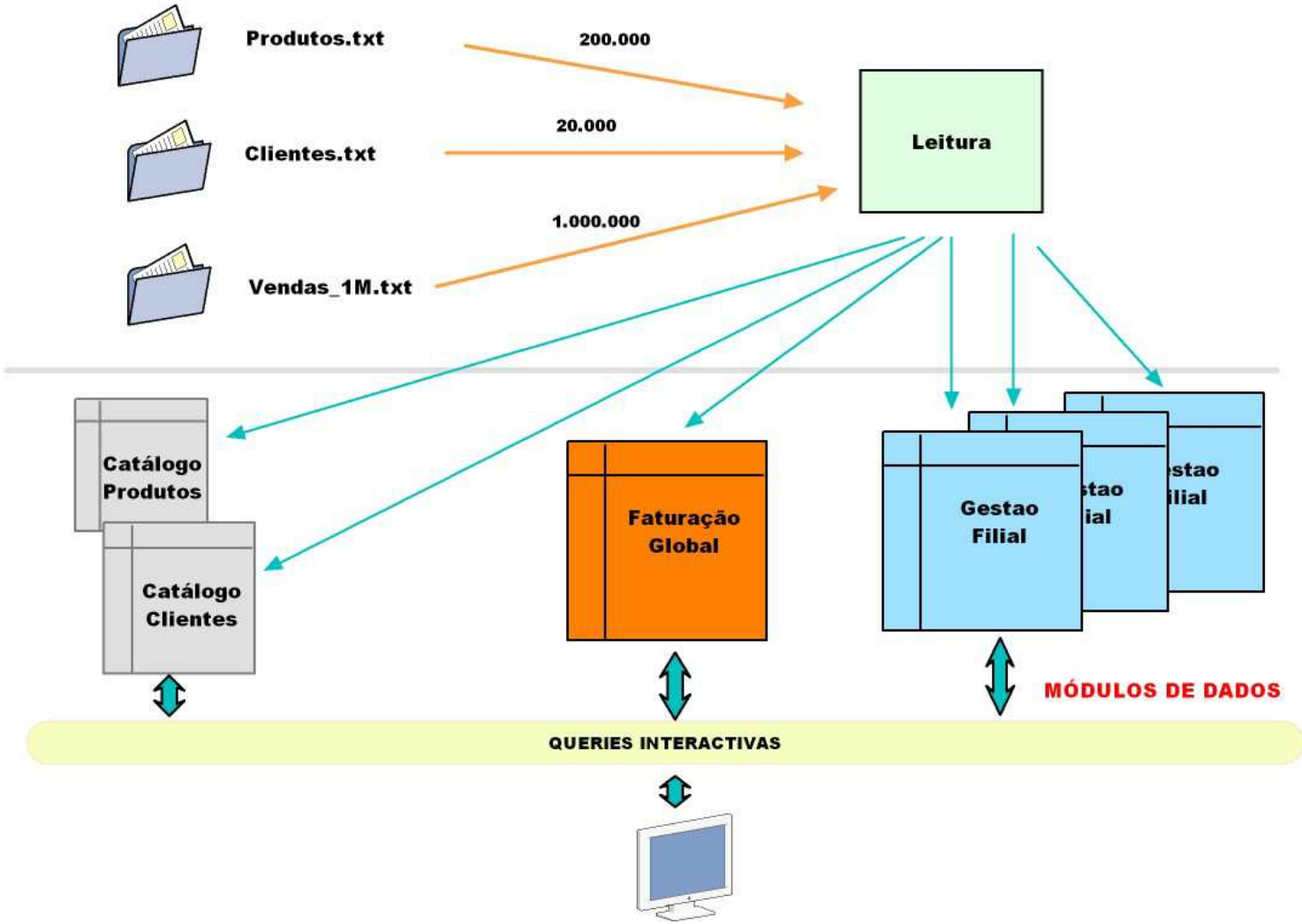
F2916 W1219 F2915

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 compra/venda** efectuada no hipermercado. cf. os exemplos seguintes produto + preço unidade + nº de unidades + **P**romoção ou **N**ormal, código cliente + mês + filial:

KR1583 77.72 128 P L4891 2 1 ← **registo de compra/venda**
QQ1041 536.53 194 P X4054 12 3
OP1244 481.43 67 P Q3869 9 1
JP1982 343.2 168 N T1805 10 2

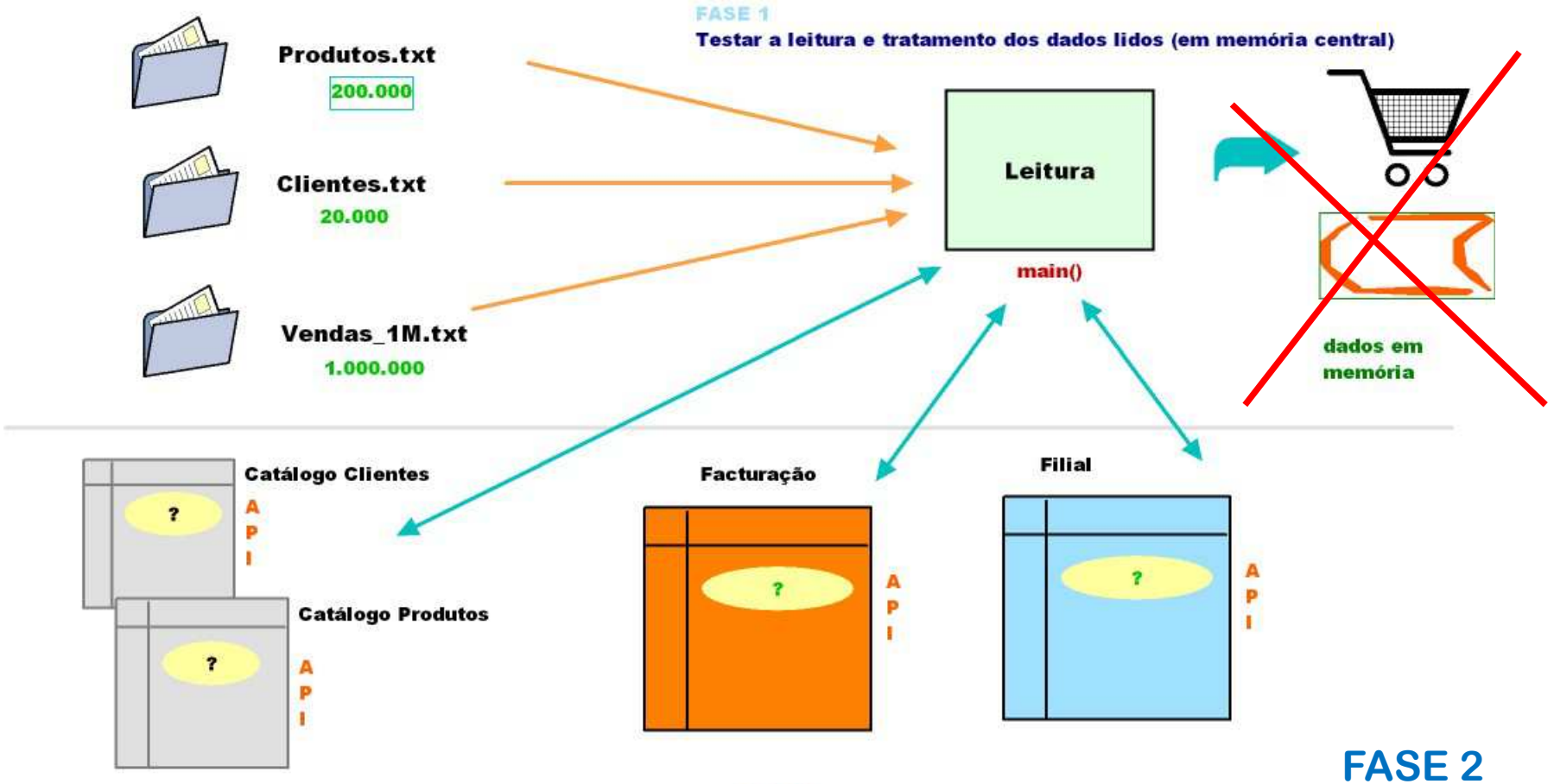
Projecto de C: SGV

Arquitectura da aplicação



Projecto de C (evolução): SGV

FASE 1



FASE 2

FASE 1: Ler e testar valores

____ FIM DA LEITURA DE VENDAS_1M _____

Produtos envolvidos: ……..

Clientes envolvidos: ……..

cf. Boletim Semanal/Quinzenal

Vendas efectivas: …….

Ultimo Cliente: ……..

Numero de Vendas Registadas para este cliente: ??

Numero de Vendas na Filial 1: ???

Numero de Vendas na Filial 2: ???

Linha mais longa: ???

Numero de Clientes com codigo começado por A: ???

Facturação Total registada: ???

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. Tal é, naturalmente, completamente dependente das *queries* que a aplicação deve implementar 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, para a satisfação da implementação de cada uma das *queries* que podem ser realizadas pelo utilizador e, até, a sua adequada estruturação sob a forma de menus, etc.

Testes de performance.

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

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 outros *switches* de optimização, etc (cf. **–pedantic –O2 –Wall**). Para a correcta criação das *makefiles* do projecto aconselha-se a consulta do utilitário **GNU Make** no endereço *www.***gnu***.org/software/***make.**

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.

O código final de todos os grupos 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.