LABORATÓRIOS DE INFORMÁTICA III

2013/2014
LEI

2º ANO - 2º SEM

F. Mário Martins (fmm@di.uminho.pt)
João Luís Sobral (jls@di.uminho.pt)

DI/UM

OBJECTIVOS

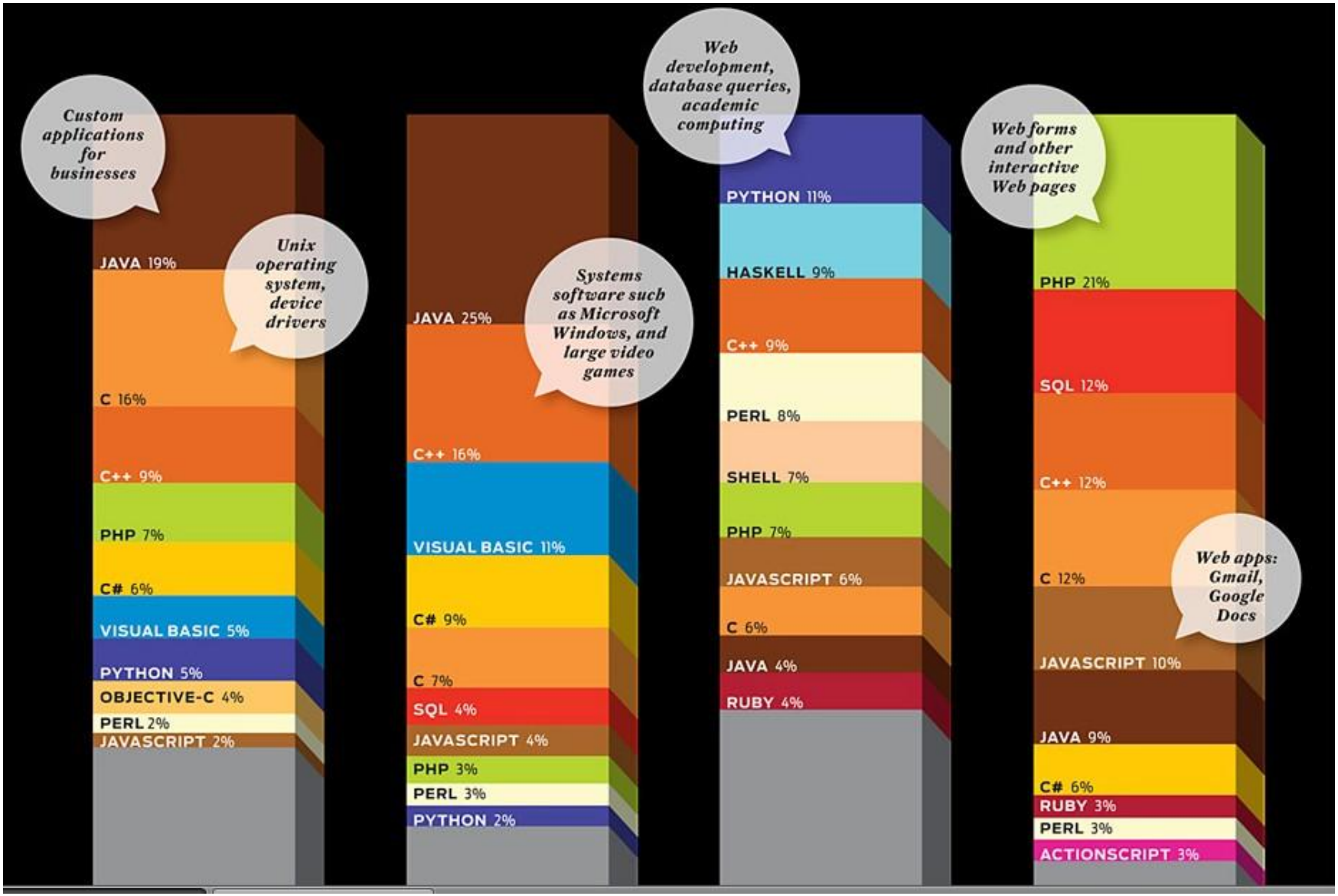
▣ **Conhecer os princípios fundamentais da Engenharia de Software**, designadamente modularidade, reutilização, encapsulamento e abstracção de dados, e saber implementá-los em diferentes linguagens/paradigmas de programação: (imperativo em **C** - 1º projecto e POO em **Java** - 2º projecto);

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

▣ **Desenhar (conceber), codificar e testar software**, realizando dois projectos concretos de média dimensão,

- **1º projeto** - **Linguagem C**: modularidade, reutilização, encapsulamento, estruturas de dados, manipulação de ficheiros, etc.;
- **2º projeto** - **Linguagem Java:** classes, packages, herança, reutilização de código, polimorfismo, colecções e streams de I/O;

PORQUÊ C e JAVA ?



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 facultativa.**

► **Os alunos realizarão dois projectos práticos obrigatórios.**

- O 1º projecto de C será de dimensão média e realizado em grupo (máx. 3 alunos) e terá apenas a submissão final e avaliação presencial.

- O 2º projecto, de JAVA, será realizado em grupo (máx. 3 alunos) e terá apenas a submissão final 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

TURNOS EFECTIVOS

ACESSO BB : LI3CJAVA_1

TURNOS DISPONÍVEIS (inscrições são desnecessárias):

3ª. Feira, 18H00-20H00 (PL3, CPI, 314)

5ª. Feira, 18H00-20H00 (PL2 + PL4, CPII, 303)

6ª. Feira, 16H00-18H00 (PL5, CPI, 310)

► Inscrições nos grupos práticos a realizar no BB (foram criados 80 grupos).

CALENDÁRIO

CALENDÁRIO LI3 - Versão 1
2013-2014

Semana	2a.feira	3a.feira	4a. Feira	5a.feira	6a.feira	sábado	
17/02 a 22/02							**Semana de LEI**
24/02 a 01/03			COMUM				**Aula comum de apresentação de LI3 (TP de C)**
03/03 a 08/03							
10/03 a 15/03							
17/03 a 22/03			COMUM				
24/03 a 29/03							
31/03 a 05/04							
07/04 a 12/04							
14/04 a 19/04							
21/04 a 26/04			COMUM				**TP de Java + Entrega electrónica do TP de C**
28/04 a 03/05							**Avaliações do TP de C**
05/05 a 10/05							
12/05 a 17/05							Semana da queima
19/05 a 24/05			COMUM				
26/05 a 31/05							
02/06 a 07/06							
09/06 a 14/06							**Entrega electrónica do TP de Java**
16/06 a 21/06							**Avaliações do TP de Java**
23/06 a 28/06							**Lançamento das Notas Finais**
30/06 a 05/07							
07/07 a 12/07							

- 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

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 TRANSFORMANDO-OS OU GERANDO RESULTADOS.**

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

Exºs:

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

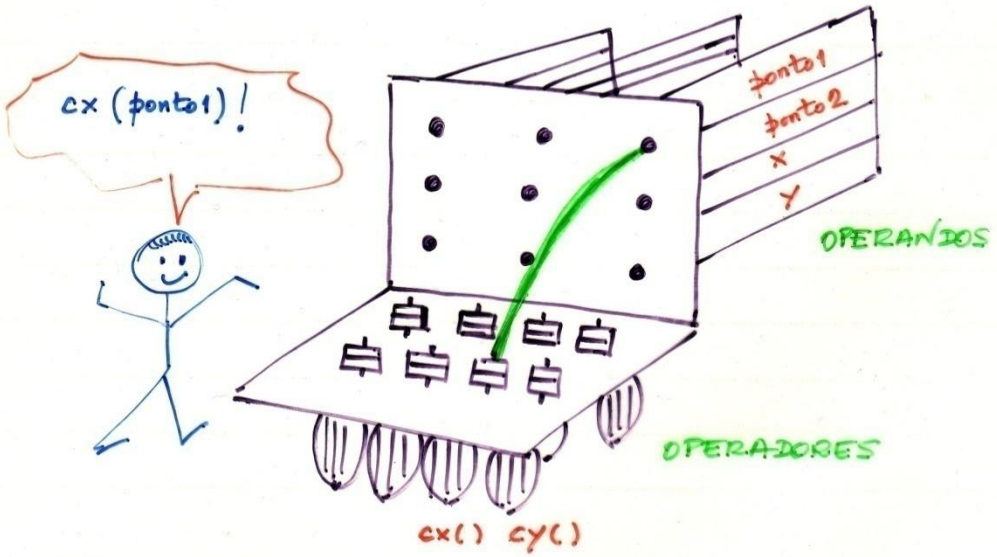
Em POO teremos que passar a pensar que dados e operações se definem de forma ligada; os dados possuem as suas próprias operações.

modelo **x.f()**

PARADIGMA IMPERATIVO

- ESTE MODELO, ORIGINÁRIO DOS PRIMÓRDIOS DA COMPUTAÇÃO, EM QUE COMPUTADORES ERAM VISTOS COMO SUPER-CALCULADORAS, REALIZANDO POIS OPERAÇÕES SOBRE OPERANDOS, É VISÍVEL AINDA AOS MAIS DIVERSOS NÍVEIS:

NÍVEL MÁQUINA:	INSTRUÇÕES + DADOS
NÍVEL LINGUAGEM:	EXPRESSÕES + VARIÁVEIS
NÍVEL PROGRAMA:	SUBROTINAS + ARGUMENTOS
NÍVEL LING. COM.:	COMANDOS + FICHEIROS



- **Dados e operações são entidades separadas;**
- **Dados são entidades passivas Sem operações directamente associadas;**
- **Programamos as ligações, ou seja, os f(x);**

MODULARIDADE: Problemas

Questão1: Como dividir os programas em módulos reutilizáveis ?

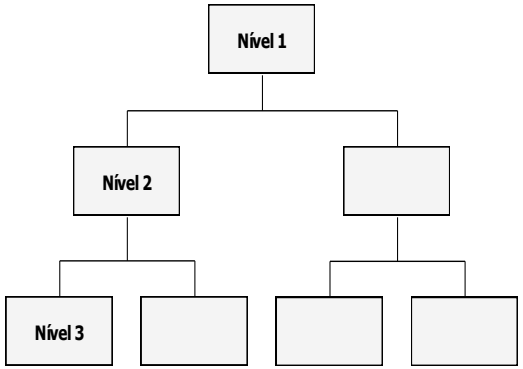
- para não estar sempre a reinventar a roda e para << $$

Questão 2: Como controlar erros e modificações ?

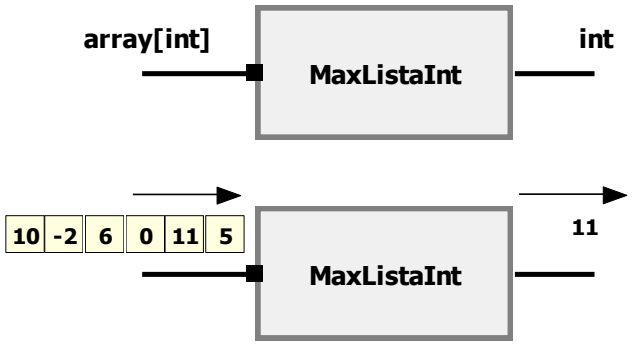
- os programas nunca estão prontos; estão sempre prontos

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

Soluções tradicionais:

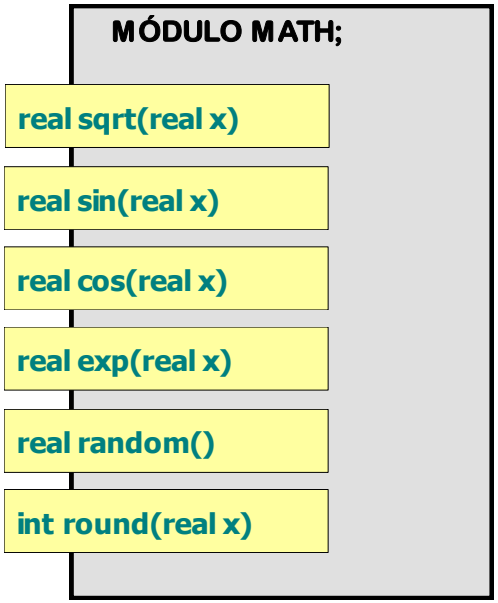


Refinamento Top-Down



Abstracção de Instruções (Procedimental)

MODULARIDADE: Problemas



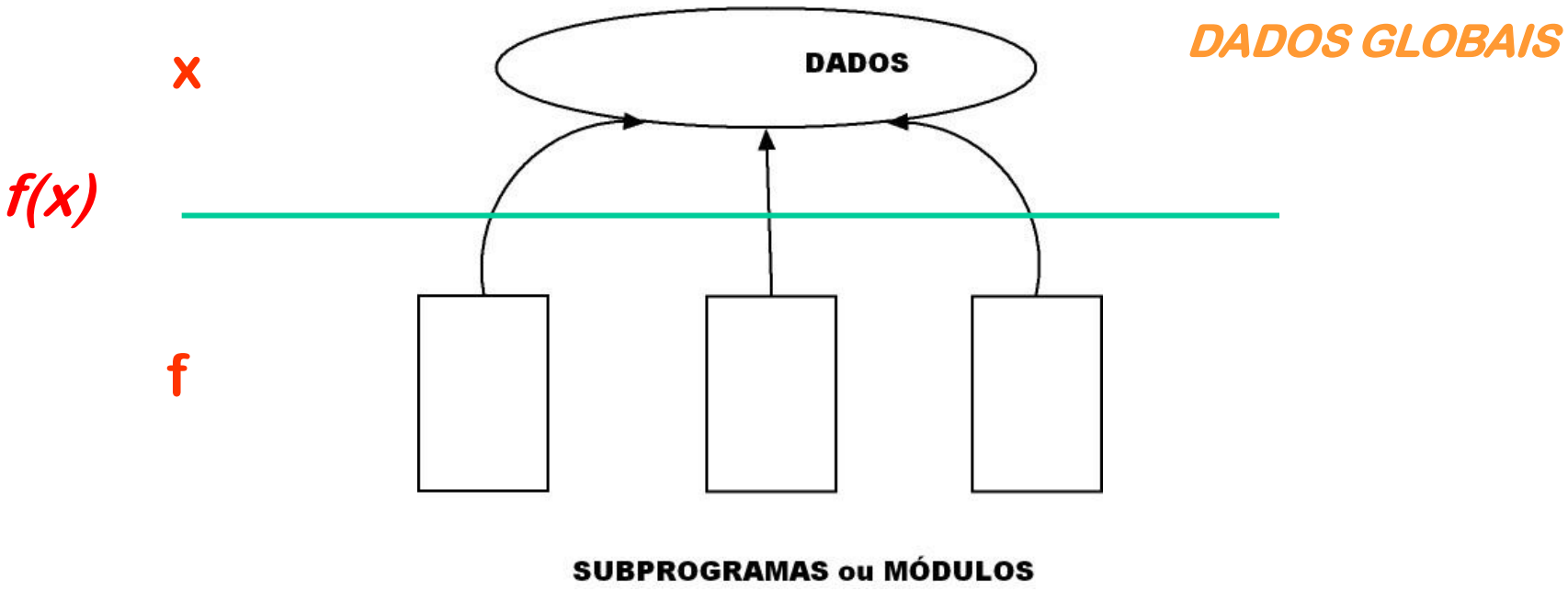
Módulos como abstracções de instruções, tal como em device drivers, módulo de cálculos matemáticos, de I/O, etc.

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 !

MODULARIDADE: Exemplo em C

Calculadora
(usa uma stack)

calc.h:

```
#define NUMBER '0'
void push(double);
double pop(void);
int getop(char []);
int getch(void);
void ungetch(int);
```

definições e declarações comuns

main.c:

```
#include <stdio.h>
#include <stdlib.h>
#include "calc.h"
#define MAXOP 100
main() {
    ...
}
```

getop.c:

```
#include <stdio.h>
#include <ctype.h>
#include "calc.h"
getop() {
    ...
    sp++;
}
```

stack.c:

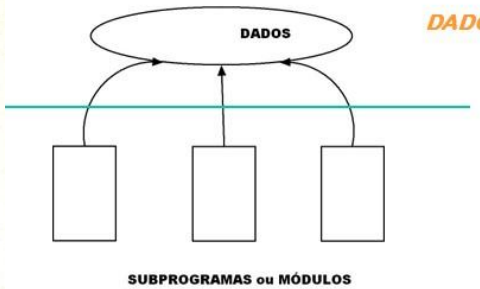
```
#include <stdio.h>
#include "calc.h"
#define MAXVAL 100
int sp = 0;
double val[MAXVAL];
void push(double) {
    ...
}
double pop(void) {
    ...
}
```

getch.c:

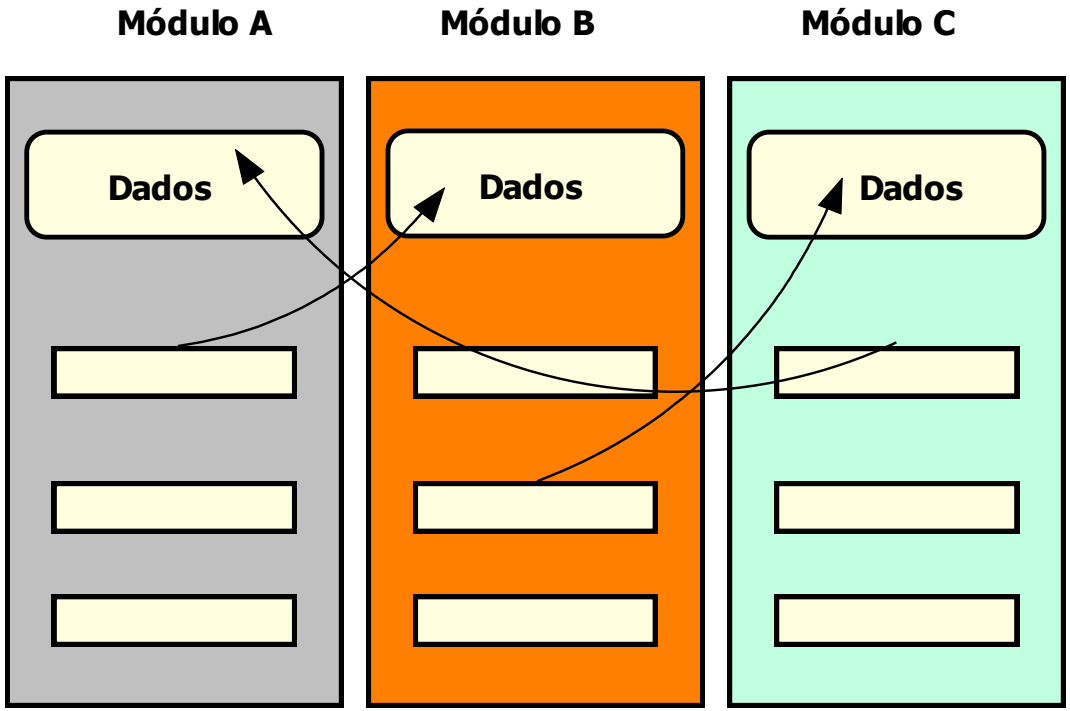
```
#include <stdio.h>
#define BUFSIZE 100
char buf[BUFSIZE];
int bufp = 0;
int getch(void) {
    ...
}
void ungetch(int) {
    ...
}
```

- 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.

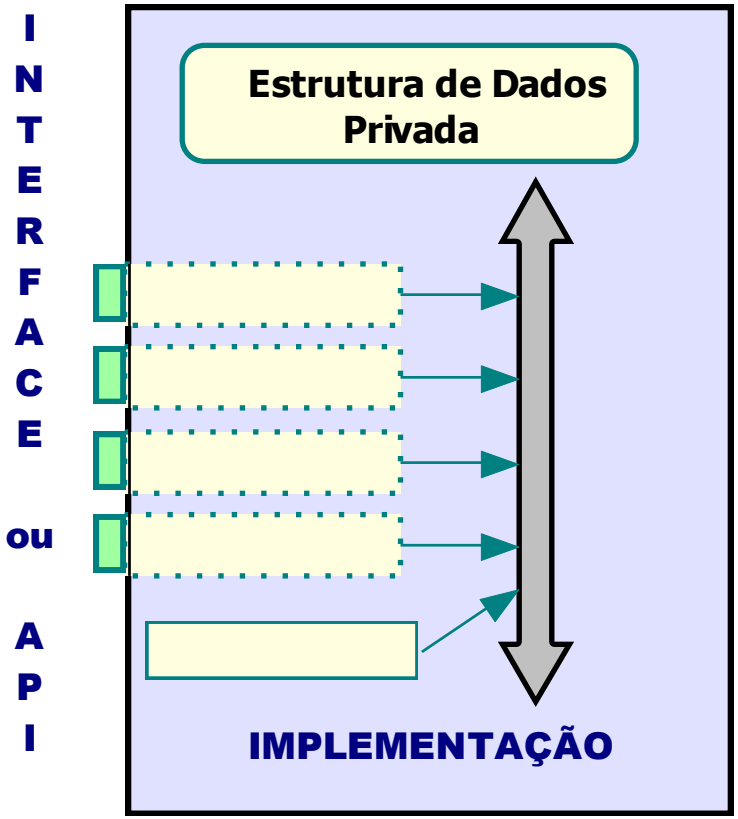
- Estes módulos não são independentes;
- Dados de uns são acedidos por módulos externos;

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.

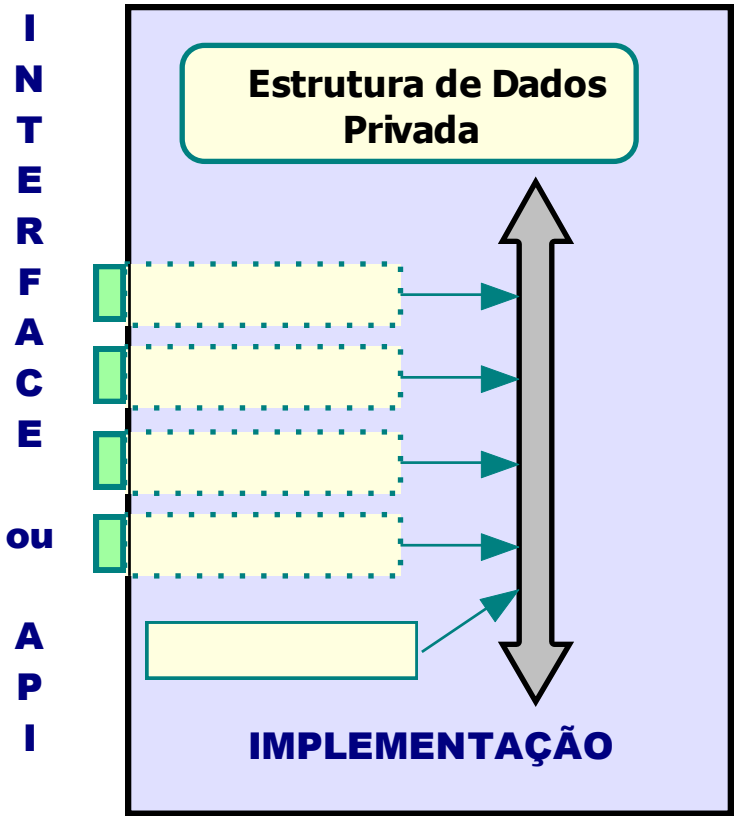
ENCAPSULAMENTO DE DADOS

- **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**;

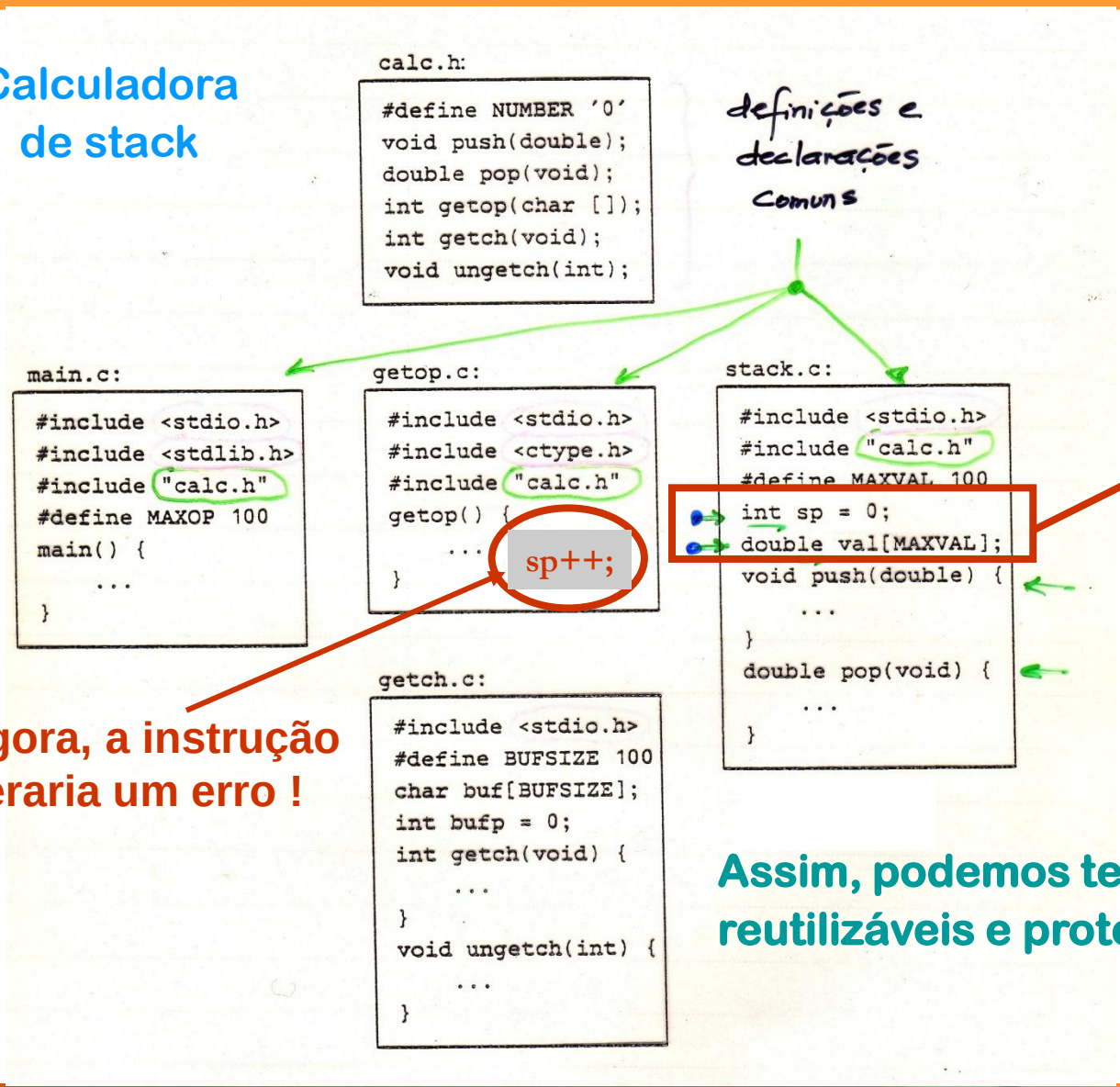
- **ERROS**: Apenas o código interior ao módulo pode provocar erros nos dados (Sherlock Holmes tem agora a vida muito facilitada);

- **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



Agora, a instrução geraria um erro !

O encapsulamento pode ser garantido se as variáveis forem declaradas static

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

ENCAPSULAMENTO EM C

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

Static storage class designation can also be applied to external variables. The only difference is that static external variables can be accessed as external variables only in the file in which they are defined. No other source file can access static external variables that are defined in another file.

```
/* File: xxx.c */
     static int count;
     static char name[8];
     main()
     {
          ... /* program body */
     }
```

Variáveis static external

- Ler documento sobre Modularidade em C (pasta de Conteúdos do BB) para mais informações importantes.

DESENVOLVIMENTO EM LARGA ESCALA

DESENVOLVIMENTO DE SOFTWARE EM LARGA ESCALA

Compiladores não garantem verificação destas propriedades !!

CONCEITOS FUNDAMENTAIS

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

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

2013/2014
LEI

TRABALHO PRÁTICO DE C

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

GESTAUTS: Criação, Gestão e Consulta de um Catálogo de Autores

☑ 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 um ficheiro **.txt** indicado (por exemplo, o ficheiro **publicx.txt**) que será o primeiro previamente disponibilizado para o projecto;

☑ Cada linha de tal ficheiro possui **uma lista de um ou mais nomes de autores, separados por vírgulas, e o ano da respectiva publicação**, cf. os exemplos:

```
Kim-Sang, Hu Xiang, Alan C. Carter, 2001
John Smith, Gillermo Paz, 1991
K. Dix, 2013
```

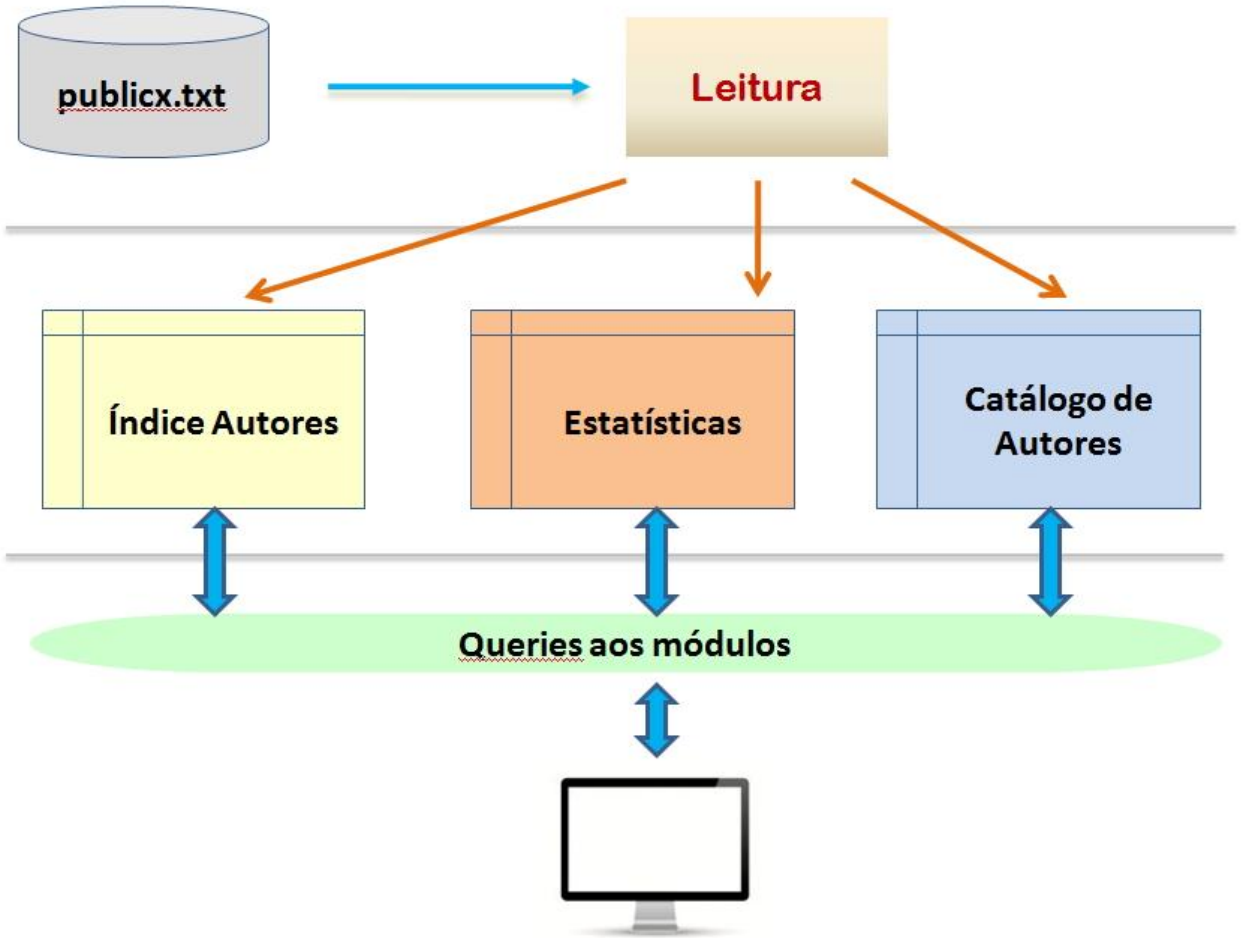
☑ Todas as linhas estão bem formadas e todos os anos poderão ser convertidos para um inteiro.

Projecto de C: GESTAUTS

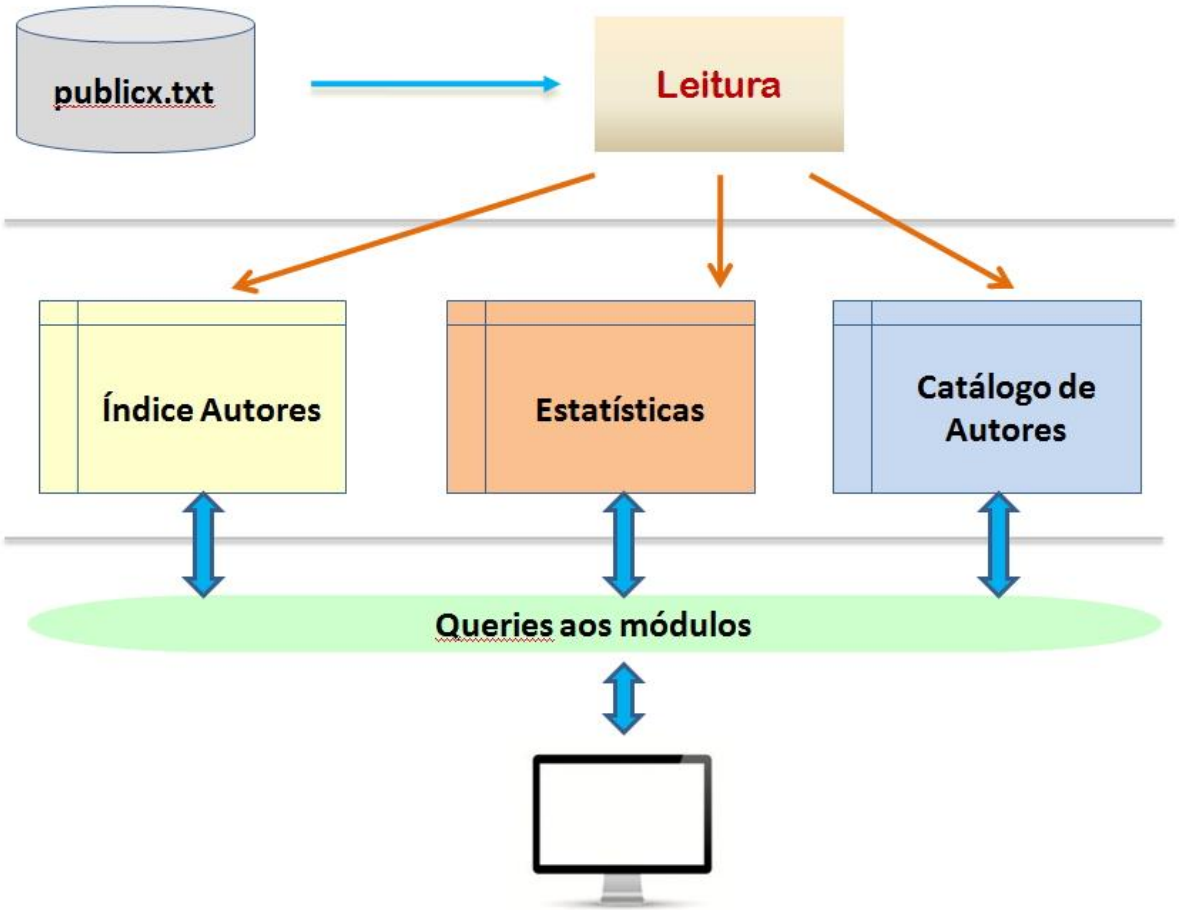
☑ Antes de qualquer outro processamento, o programa deverá ser capaz de ler as mais de 120.000 linhas, e mais de 300.000 nomes de autores deste ficheiro **publicx.txt**;

☑ No sentido de se deixar desde já uma orientação de base quanto à arquitectura possível 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 os seguintes módulos:

Arquitectura da aplicação



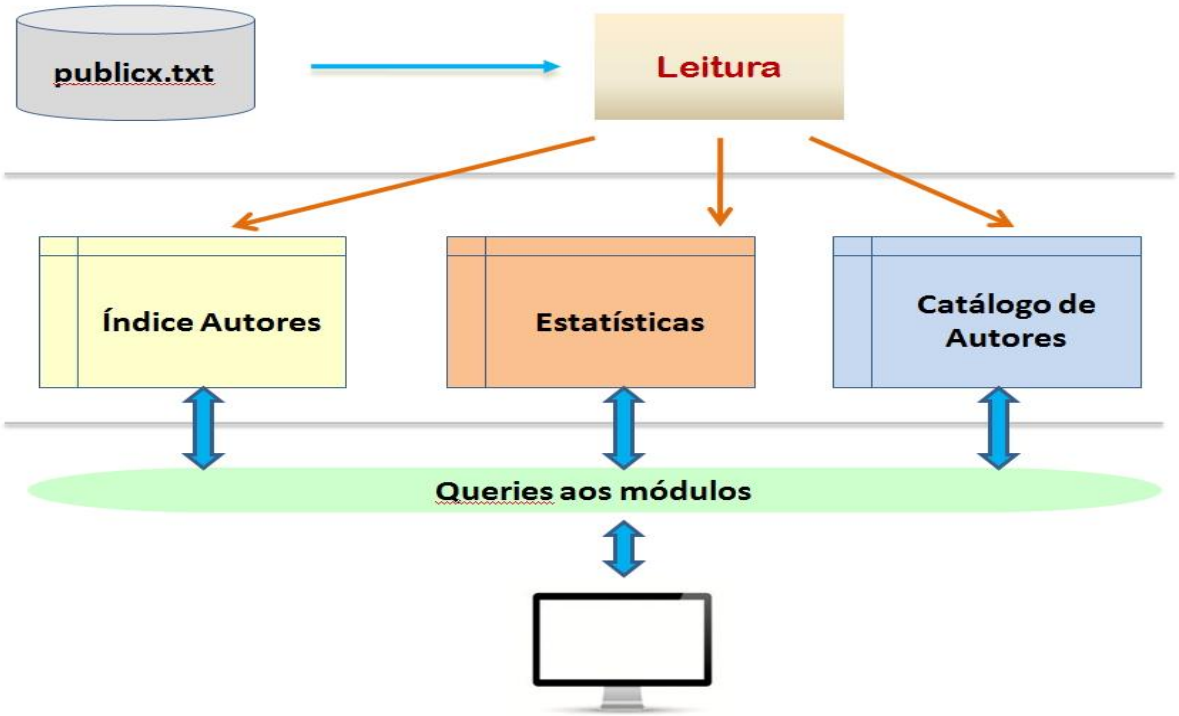
Projecto de C: GESTAUTS



Leitura: função ou parte do código de **main()** no qual é realizada a leitura e eventual contabilização básica das linhas do ficheiro **publicx.txt;**

Índice de Autores: módulo de dados onde deverá ser criado um catálogo de autores por índice alfabético, que irá permitir, de forma eficaz, saber quais são os autores cujos nomes começam por uma dada letra do alfabeto, quantos são, etc.;

Projecto de C: GESTAUTS



Estatísticas: módulo de dados que irá conter as estruturas de dados responsáveis pela resposta eficiente a questões quantitativas como anos de publicação, artigos publicados num dado ano, e nº de autores, etc.;

Catálogo de Autores: módulo de dados que conterá as estruturas de dados adequadas à representação dos dados quantitativos por autor, cf. Artigos publicados em cada ano, e relacionamentos existentes entre autores, designadamente, questões como, quantos autores publicaram com dado autor, quem publicou com quem, quantos artigos dois autores publicaram em conjunto num dado ano, etc.;

Projecto de C: GESTAUTS

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 seu projecto tendo por base o ficheiro **publix.txt**, deverá realizar alguns testes de *performance* e apresentar os respectivos resultados. Pretende-se comparar os tempos de execução dos *queries* 8, 9 e 12, usando os ficheiros, **publicx.txt**, **publicx_x4.txt** e **publicx_x6.txt**. Todos os ficheiros serão fornecidos numa pasta disponibilizada via BB.

Projecto de C: GESTAUTS

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. 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.

Apresentação do projecto e Relatório.

O projecto será submetido por via electrónica num *site* do DI a indicar oportunamente (bem como o formato da pasta e a data e hora limite de submissão). Tal *site* garantirá quer o registo exacto da submissão quer a prova da mesma a quem o submeteu (via e-mail). Tal garantirá extrema segurança para todos.

O código submetido na data de submissão será o código efectivamente avaliado. A *makefile* deverá gerar o código executável, e este deverá executar correctamente. Projectos com erros de *makefile*, de compilação ou de execução serão de imediato rejeitados.