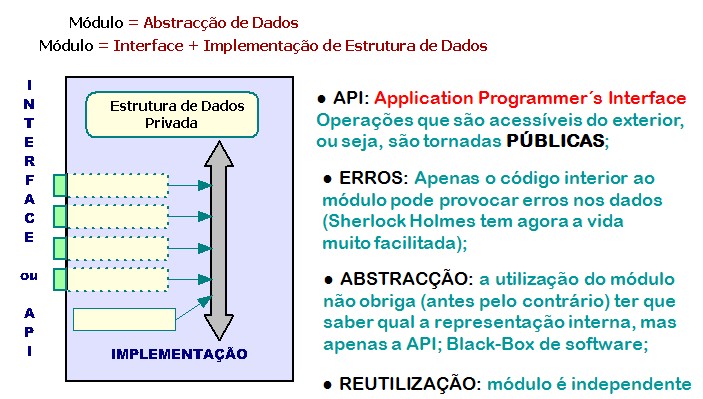
**IMPLEMENTAÇÃO EM C DE MÓDULOS DE DADOS - TIPOS**

**TÉCNICA DOS TIPOS INCOMPLETOS ou OPACOS**

**F. Mário Martins, LI3, 2015-2016 (rev. 2018/2019)**

Como vimos anteriormente, um **módulo de dados** é uma implementação encapsulada, protegida, segura e robusta de um *tipo abstracto de dados* (**TAD**), ou seja, um tipo de dados que pode ser representado de muitas formas mas que deve obedecer a um conjunto de propriedades de comportamento bem definidas. Procurando clarificar o que vamos apresentar em seguida, chamaremos ao tipo abstracto de dados, **TAD ou Tipo Definido pelo Programador (Tdp)**, e à sua representação na linguagem (neste caso C) tipo concreto de dados, **Tcd**.



Em C, a criação deste tipo de módulos de dados requer algum esforço de programação de modo a que propriedades como encapsulamento, robustez, segurança, etc., possam ser garantidas, tal como vimos anteriormente.

Em C, a API é definida no ficheiro .h e a implementação no ficheiro .c. Assim, no ficheiro .h deveremos definir o essencial do **Tdp**, e no ficheiro .c a sua implementação concreta, ou seja, o correspondente **Tcd**.

Como vimos anteriormente com o módulo stack fazer apenas esta divisão não chega e mais algumas regras e técnicas devem ser introduzidas.

**Definição incompleta usando structs**

Em C existem vários tipos incompletamente definidos que são aceites pelo compilador. Bem conhecidos são void, arrays sem dimensão cf. int a[], e structs, unions e enumerados sem definição prévia cf. struct stack stk1;.

Adicionalmente C permite que se declarem apontadores para tipos incompletos, como por exemplo em void\* ptr; ou struct stack\* stk1;. Interessam-nos em particular as **struct**.

Porém, como o tipo é incompleto, os compiladores rejeitam qualquer tentativa de desreferenciar o apontador, quer usando \* quer usando ->. Assim, se uma variável **p** é um apontador para um tipo incompleto, qualquer tentativa de a usar via **\*p** ou **p->** originará um erro de compilação.

No entanto, **num contexto em que o tipo tenha já sido definido e completado**, o compilador já aceitará uma referência para a sua definição e implementação.

A estratégia para a efectiva implementação de **MÓDULOS DE DADOS ENCAPSULADOS** passará por usar esta técnica, definindo tipos incompletos no ficheiro .h (o ficheiro de declaração) e apenas os definindo completamente no ficheiro .c (o ficheiro de implementação). **Assim, fora deste ficheiro .c de implementação, os apontadores não podem ser usados e não darão acesso aos dados, e apenas dentro deste .c se poderá escrever código de acesso à representação do tipo usando p -> campo.** **Tal como pretendíamos**.

**Regra:** **typedef struct Tcd\*** **Tdp**;

No exemplo anterior de criação do **Tdp** **Stack** teríamos no **.h**

**#ifnedef STACK\_H**

**#define STACK\_H**

**typedef struct stack\*** **Stack**;

**Stack initStack();**

**Stack push(Stack, int);**

**int pop(Stack);**

**int isEmpty(Stack);**

**int isFull(Stack);**

...

**#endif**

e no ficheiro **.c** de implementação do **Tcd** teríamos a definição completa,

struct stack {

int array[MAXSTACKSIZE];

int stackPointer;

}

e o código das funções que é muito semelhante ao apresentado em **stack2.c**.

A organização da informação pelos ficheiros C terá sempre uma estrutura clara e uniforme para este tipo de preocupações de abstracção de dados e que é a seguinte:

**Ficheiro meuTipo.h**

. declaração incompleta do **Tdp**

. declaração abstracta das funções

**Ficheiro meuTipo.c**

. #include **"meuTipo.h"**

. #include de estruturas de dados de implementação

. declaração completa do **Tcd** (struct)

. declaração dos tipos auxiliares se existirem

Exemplos comuns de declarações:

**typedef struct stack\*** **Stack**;

**typedef struct catalogoProds\*** **CatProds**;

**typedef struct aluno\*** **Aluno**;

**typedef struct turma\* Turma;**

Note-se finalmente que o posicionamento do \* nestas declarações, sendo todas legais, correspondem a uma preferência pessoal, tratando-se apenas de uma questão de estilo.