Implementação em C de Módulos de Dados

(Técnica dos Tipos de Dados Abstratos ou Opacos)

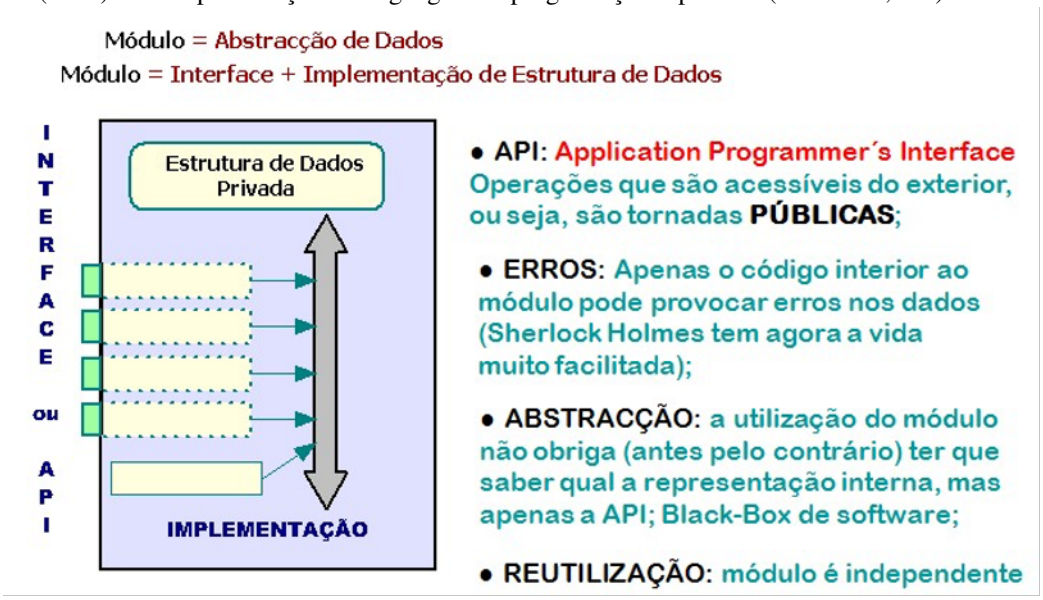
Laboratórios de Informática III
Guião #3 (2/2)

Departamento de Informática
Universidade do Minho
Autor original: Fernando Mário Martins

Outubro de 2022

1 Introdução

Como vimos anteriormente, um *módulo de dados* é uma implementação encapsulada, protegida, segura e robusta de um tipo abstrato 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 Tipo Definido pelo Programador (TDP) ao tipo abstrato de dados (TAD), e Tipo Concreto de Dados (TCD) à sua representação na linguagem de programação específica (neste caso, o C).



Na linguagem 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.

2 Definição incompleta usando estruturas

Em C existem vários tipos incompletamente definidos que são aceites pelo compilador. Bem conhecidos são `void`s, arrays sem dimensão cf. `int a[]`, e `struct`s, `union`s e enumerados sem definição prévia cf. `struct stack stk1;`. Adicionalmente, o 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 efetiva implementação de *módulos de dados encapsulados* passará por usar esta técnica, definindo tipos incompletos no ficheiro `.h` (o ficheiro de cabeçalho com declarações) 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 a construção `p->campo`, tal como pretendíamos.

Regra: `typedef struct TCD* tdp;`

No exemplo anterior de criação do TDP Stack teríamos no `stack.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
```

No ficheiro `stack.c` de implementação do TCD teríamos a definição completa:

```
struct stack {
    int array[MAXSTACKSIZE];
    int stackPointer;
}
```

Teríamos, também, o código das funções que é muito semelhante ao apresentado na segunda versão do módulo de stack.

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

meuTipo.h:

- declaração incompleta do TDP
- declaração abstrata das funções

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.

3 Exercícios

1. Considere a estrutura de dados `Deque` dos guiões passados. Reescreva-a de forma a que módulos externos não consigam aceder à sua declaração interna.

2. Considere um sistema de gestão dos balcões de registos portugueses. O sistema contém um conjunto de balcões, definidos pelo seu id, nome, localização, e horário de atendimento. A cada balcão estão associados vários clientes (um cliente pode estar associado a vários balcões), definidos pelo número de cartão de cidadão, nome, data de nascimento, e morada. Considere ainda que cada balcão pode ter múltiplas filas de espera, uma por cada serviço prestado (definido pelo nome, e.g., "Renovar cartão de cidadão"), sendo que a fila deverá dar prioridade de atendimento a utilizadores com mais de 80 anos. O sistema deverá suportar as seguintes operações:

   - Consultar a informação de um cliente, fornecendo o seu número de cartão de cidadão;
   - Consultar a lista de clientes de um balcão, fornecendo o seu id;
   - Num dado balcão, adicionar um cliente a uma fila de espera, fornecendo o id do balcão, o número de cartão de cidadão do cliente, e o nome do serviço em questão;
   - Num dado serviço de um balcão, chamar (i.e., remover) o primeiro cliente na fila, fornecendo o id do balcão e o nome do serviço.

   Como é que estruturaria o código desta aplicação, i.e., que módulos definiria e quais seriam as interações entre estes?