Contents

Tipi di dato opaco

Per evitare di dover cambiare il codice quando usiamo una struttura per più file(module) usiamo il tipo opaco.

Se non usassimo questa tecnica, l'accoppiamento(coupling) tra i moduli sarebbe molto alto(i moduli sono dipendenti tra loro).

La coesione (cohesion) esprime una misura della relazione tra le diverse funzioni presenti un modulo e un qualche concetto unificante associato al modulo.

Per avere un codice ben strutturato dobbiamo cercare di avere accoppiamento basso e alta coesione.

Per cercare di ridurre l'accoppiamento dobbiamo implementare l'Abstract Data Type(ADT) che in C è implementato come tipo di dato opaco

Ad esempio per i numeri complessi abbiamo:

```c
typedef struct {
    double re;
    double im;
} Dcomplex;
```

Ma vogliamo evitare di accedere direttamente alla variabile.

Definiamo una struttura opaca nell'header:

```c
typedef struct dcomplex *ADTcomplex; // puntatore a struttura opaca
```

Questa struttura non ha campi esposti quindi non è accessibie dall'esterno.

Poi costruiamo le funzioni per manipolare questa struttura:

```c
ADTcomplex mkADTcomplex(double re, double im);  // Costruisce il numero
 ↪  complesso dalla parte reale e quella immaginaria

/* @brief libera la memoria puntata da *'pc' e setta *'pc' a NULL
 */
void dsADTcomplex(ADTcomplex *pc);
```

```
ADTcomplex sum(ADTcomplex c1, ADTcomplex c2);   // Somma
ADTcomplex product(ADTcomplex c1, ADTcomplex c2);   // Prodotto


double modulus(ADTcomplex c);   // Modulo

/** @brief Restituisce l'argomento nell'intervallo [-pi,+pi] radianti
*/
double agument(ADTcomplex c);

double re(ADTcomplex c);    // Restituisce il numero reale
double im(ADTcomplex c);    // Restituisce la parte immaginaria

/** @brief Controlla se le componenti di due complessi sono uguali a meno di
 ↪  un espilon reale
*/
Bool equal(ADTcomplex c1, ADTcomplex c2, double epsilon);
```

A questo punto chi importa questo header non sa i campi del numero complesso ma deve usare queste funzioni per manipolarlo e usarlo.

In questo modo l'implementazione del numero complesso è indipendente dal client e se viene modificata la libreria, non è necessario modificare il client.

Inoltre, possiamo anche implementare in modi diversi la stessa funzione a seconda dalla feature, ecc…

Inoltre il compilatore evita la modifica dei campi della struttura dai file client, evitando l'introduzione di errori.

Un altro vantaggio è che non bisogna ricompilare i client ma solo la libreria, in caso di cambiamenti.

Composizione di un ADT

Un ADT è quindi composto da: - Un tipo opaco che viene definito attraverso una `typedef` - Un insieme di operazioni che si possono fare su questo tipo, sono contenute nel file header insieme alla definizione di tipo opaco - L'implementazione di queste operazioni e la definizione completa del tipo nel file c che implementa la struttura opaca