

Componenti software

Obiettivi

- Introdurre le tipologie di componente software e la nozione di Abstract Data Type
- Mostrare come realizzare un ADT in linguaggio C

Componenti software

- Librerie
- Astrazioni di dato
- Tipo di Dato Astratto (ADT)

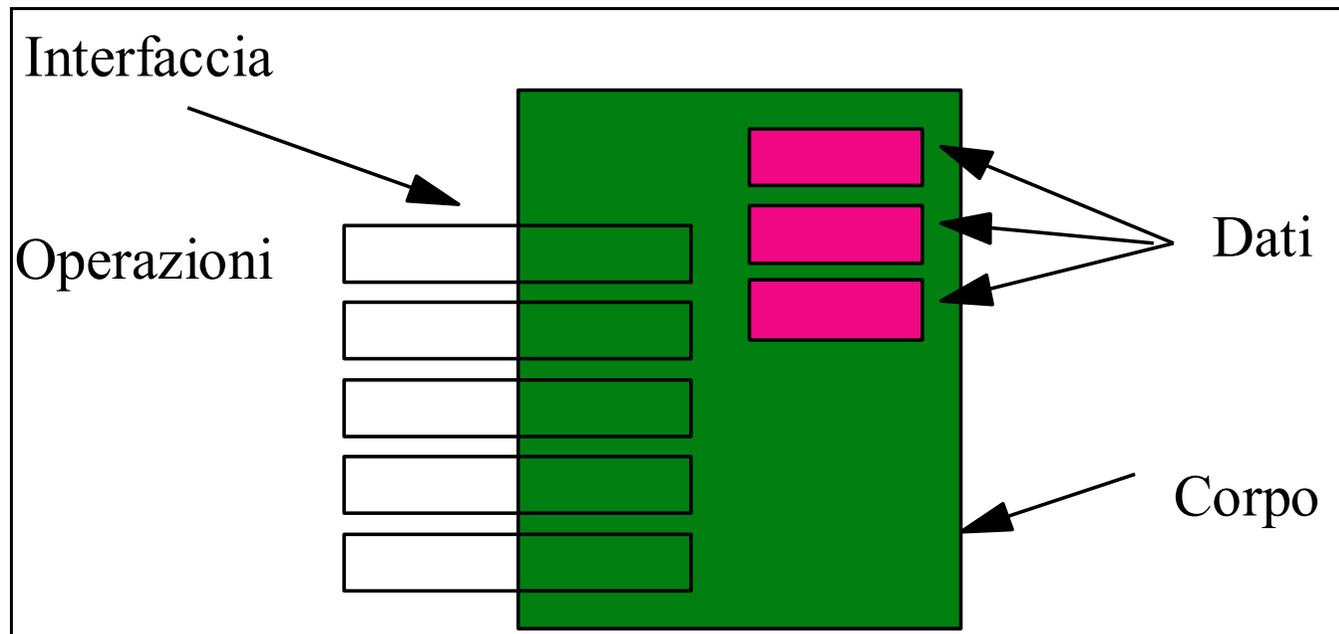
Librerie

- Il componente rende visibili procedure e funzioni che non fanno uso di variabili non locali. Il componente è una collezione di *operazioni* (ad esempio, funzioni matematiche) ed eventuali costanti (globali)
- Di sistema o “user-defined”, ad esempio:
 - `math.h`, `stdio.h`, `mylib_ord.h`
- Per usarle:

```
#include <stdio.h>
#include "mylib_ord.h"
```

Astrazioni di dato

- Il componente ha dati locali (nascosti) e rende visibili all'esterno i prototipi delle operazioni invocabili (procedure e funzioni) su questi dati locali, ma non gli identificatori dei dati. Attraverso una di queste operazioni si può assegnare un valore iniziale ai dati locali nascosti.



Esempio: il contatore

1. *file header contatore.h*

```
void reset(void) ;  
void inc(void) ;  
void stampa(void) ;
```

Definisce cosa si può fare con il contatore

2. *file di implementazione contatore.c*

```
#include <stdio.h>  
static int c;  
void reset(void) { c=0; }  
void inc(void) { c++; }  
void stampa(void) { printf("%d", c); }
```

Incapsula il contatore c (IL dato) e specifica il codice delle funzioni con cui si agisce sul contatore

Esempio: il contatore

- Il cliente:

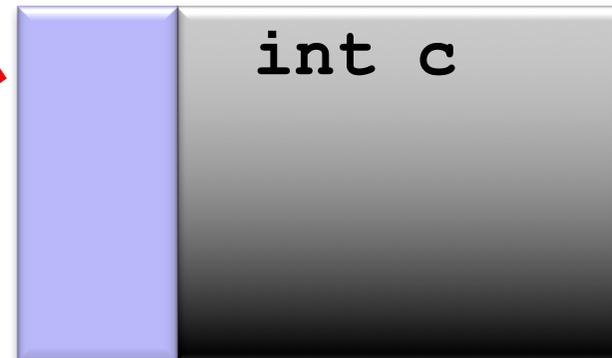
```
#include "contatore.h"
```

```
int main() {  
    reset(); inc(); stampa();  
}
```

main.c

```
c=0; ERRORE!!!
```

contatore.h contatore.c



Esempio: il contatore

1. *file header contatore.h*

```
void reset(void) ;  
void inc(void) ;  
void stampa(void) ;
```

Se l'header non cambia

2. *file di implementazione contatore.c*

```
#include <stdio.h>  
static char c;  
void reset(void) { c='a' ; }  
void inc(void) { c++; }  
void stampa(void) { printf("%c", c) ; }
```

Ma l'implementazione si

Esempio: il contatore

- Il cliente non cambia:

```
#include "contatore.h"
int main() {
    reset(); inc(); stampa();
}
```

- Se il cliente avesse bisogno di gestire due contatori?

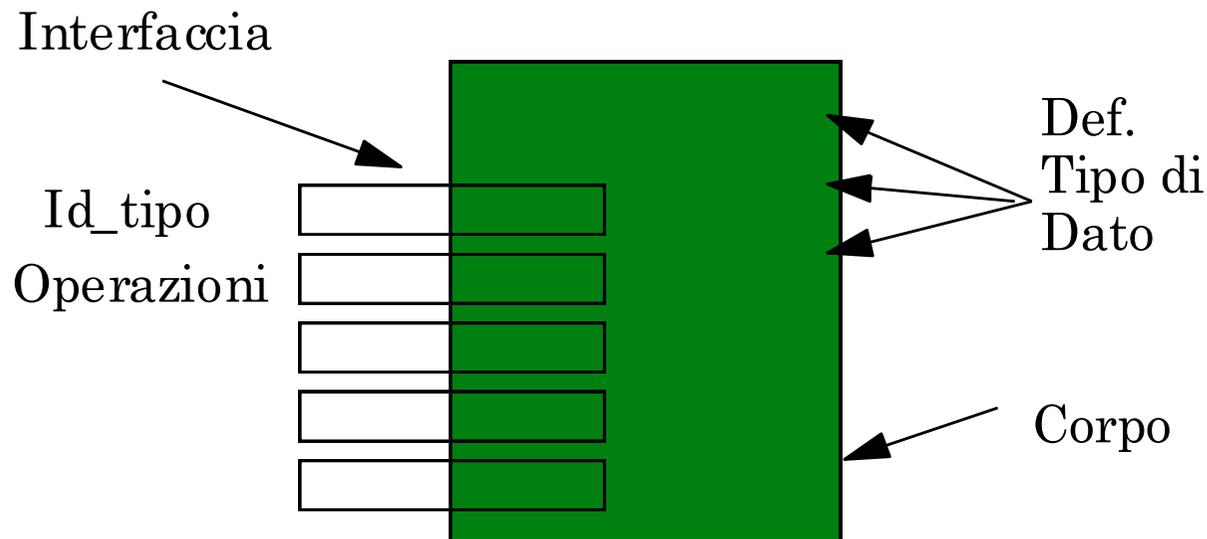
```
contatore1.h      contatore2.h
contatore1.c      contatore2.c
```

Non è la soluzione praticabile !

Tipo di dato astratto

(Abstract Data Type – ADT)

- Il componente esporta un **identificatore di tipo T** ed i prototipi delle operazioni eseguibili su dati dichiarati di questo tipo. I “clienti” del componente dichiarano e controllano quindi il tempo di vita delle variabili di tipo T .



Tipi di dato astratto

Un ***tipo di dato astratto (ADT)*** definisce una categoria concettuale con le sue proprietà:

- una ***definizione di tipo***
 - implica un dominio, D
- un ***insieme di operazioni ammissibili*** su oggetti di quel tipo
 - funzioni: *calcolano valori sul dominio D*
 - predicati: *calcolano proprietà vere o false su D*

TIPI DI DATO ASTRATTO IN C

In C, un **ADT** si costruisce definendo:

- *il nuovo tipo con `typedef`*
- *una funzione per ogni operazione*

Esempio: ADT contatore

una entità caratterizzata da un valore intero

```
typedef int counter;
```

con operazioni per

- *inizializ. contatore a zero* `void reset(counter*);`
- *incrementare il contatore* `void inc(counter*);`

ORGANIZZAZIONE DI ADT IN C

- Un ADT in C può essere realizzato separandone l'interfaccia e l'implementazione in due file:
 1. ***un file header (nomefile.h)***, contenente
 - *typedef*
 - *dichiarazione delle funzioni*
 2. ***un file di implementazione (nomefile.c)***, contenente
 - *direttiva #include per includere il proprio header* (per importare la definizione di tipo)
 - *definizione delle funzioni*

Esempio: ADT counter

1. *file header counter.h*

```
typedef int counter;  
void reset(counter*);  
void inc(counter*);
```

Definisce in astratto che cos'è un counter e che cosa si può fare con esso

2. *file di implementazione counter.c*

```
#include "counter.h"  
void reset(counter *c) { *c=0; }  
void inc(counter* c) { (*c)++; }
```

Specifica come funziona (quale è l'implementazione) di counter

ADT counter: un cliente

Per usare un `counter` occorre:

- *includere il relativo file header*
- *definire una o più variabili di tipo `counter`*
- *operare su tali “istanze” mediante le sole operazioni (funzioni) previste*

```
#include "counter.h"
```

```
int main() {
```

```
    counter c1, c2; c1=0; c1++;
```

```
    reset(&c1); reset(&c2);
```

```
    inc(&c1); inc(&c2); inc(&c2);
```

```
}
```

OPERAZIONI DI UN ADT

Quali operazioni definire per un ADT?

- **costruttori** (*costruiscono un oggetto* di questo tipo, a partire dai suoi “costituenti elementari”)
- **selettori** (restituiscono uno dei “*mattoni elementari*” che compongono l’oggetto)
- **predicati** (verificano la *presenza di una proprietà* sull’oggetto, restituendo *vero* o *falso*)
- **funzioni** (*agiscono* in vario modo sugli oggetti)
- **trasformatori** (*cambiano lo stato* dell’oggetto)

ADT IN C: LIMITI

- L'ADT così realizzato funziona, ma *molto dipende dall'autodisciplina del programmatore*
- **Non esiste alcuna protezione contro un uso scorretto dell'ADT**
 - l'organizzazione *suggerisce* di operare sull'oggetto *solo tramite le funzioni previste*, ma **NON riesce a impedire** di aggirarle a chi lo volesse (ad esempio: `counter c1; c1++;`)
- **La struttura interna dell'oggetto è visibile a tutti** (nella `typedef` che si deve trovare – per le regole di visibilità lessicale del C – nel file `.h` incluso dal cliente)

ADT counter: un cliente “birichino”

```
#include "counter.h"

int main() {
    counter c1, c2;
    reset(&c1); reset(&c2); inc(&c1); c1++;
    inc(&c2); inc(&c2);
}
```

- Il cliente ha la visibilità dell'organizzazione fisica (un `int`) del dato
- Il C non garantisce un livello adeguato di protezione (*information hiding*) e includendo l'header il cliente conosce la struttura del tipo definito (`typedef`)

ADT IN C: LIMITI

Superare questi limiti sarà uno degli obiettivi cruciali della *programmazione a oggetti*, con il linguaggio Java