

Introduzione agli Abstract Data Type (ADT)

La nozione di tipo di dato astratto

- Sappiamo già che quando si affrontano problemi complessi è necessario procedere in due fasi:
 - ➔ Specifica dell'algoritmo
 - ➔ Implementazione dell'algoritmo in un linguaggio di programmazione (LdP)
- La prima fase coinvolge la specifica di due aspetti:
 - ➔ Specifica dei dati che l'algoritmo deve manipolare
 - ➔ Specifica delle operazioni da eseguire per realizzare l'algoritmo
- Per esprimere i dati in maniera indipendente dai vincoli imposti da un LdP si utilizzano i tipi di dato astratti

La nozione di tipo di dato astratto

- Un **tipo di dato astratto** è costituito da tre componenti:
 - ➔ Il **dominio di interesse D**, che rappresenta l'insieme dei valori che costituiscono il tipo astratto, ed eventuali altri domini, che rappresentano altri valori coinvolti
 - ➔ Un insieme di **costanti**, che denotano valori del dominio di interesse
 - ➔ Un insieme di **operazioni** che si effettuano sugli elementi del dominio di interesse, utilizzando eventualmente elementi degli altri domini:
 - ➔ **funzioni**: calcolano valori sul dominio D
 - ➔ **predicati**: calcolano proprietà vere o false su D
- Tutte e tre le componenti sono indipendenti dalla rappresentazione e dall'uso del tipo stesso nei LdP (la fase di specifica è indipendente dai vincoli imposti dal LdP)

Specifica dei tipi di dato astratti

- La concettualizzazione di un tipo di dato astratto richiede di specificare separatamente le tre componenti:
 - Gli insiemi che rappresentano **dominio di interesse** e gli eventuali **altri domini**
 - l'insieme delle **costanti** del tipo
 - le **operazioni**, attraverso un insieme di **funzioni**:
 - la **segnatura** della funzione (ovvero il suo nome e il tipo dei parametri di ingresso ed uscita) specifica di cosa necessita un'operazione e cosa restituisce
 - il **significato** di un'operazione può venire specificato algebricamente in modo rigoroso. Noi descriveremo il significato delle operazioni in modo informale.

Specifica dei tipi di dato astratti

- *Esempio: Concettualizzazione del tipo astratto Booleano.*
 - dominio di interesse: $\{vero, falso\}$
 - costanti: *true* e *false*, che denotano rispettivamente vero e falso
 - operazioni:
 - and : $Booleano \times Booleano \rightarrow Booleano$
 - or : $Booleano \times Booleano \rightarrow Booleano$
 - not : $Booleano \rightarrow Booleano$

Implementazione dei tipi di dato astratti

- Il tipo di dato astratto deve essere realizzato usando i costrutti del LDP:
 - ➔ 1. rappresentazione dei **domini** usando i **tipi concreti** del LDP
 - ➔ 2. codifica delle **costanti** attraverso i **costrutti** del LDP
 - ➔ 3. realizzazione delle **operazioni** attraverso opportune **funzioni** del LDP
- Osservazioni:
 - ➔ Per uno stesso tipo astratto si possono avere **più realizzazioni** diverse.
 - ➔ Una realizzazione potrebbe avere delle **limitazioni** rispetto al tipo di dato astratto.

Tipi di dato astratto in C

- In C, un *ADT* si costruisce definendo:
 - *il nuovo tipo con typedef*
 - *una funzione per ogni operazione*
- *Esempio: il contatore*
una entità caratterizzata da un valore intero

`typedef int counter;`

con operazioni per

- resettare il contatore a zero `reset(counter*);`
- incrementare il contatore `inc(counter*);`

Organizzazione di un ADT in C

La struttura di un ADT comprende quindi:

1. *un file header*, contenente

- *la typedef*
- *la dichiarazione delle funzioni*

2. *un file di implementazione*, contenente

- *una direttiva #include per includere il proprio header (per importare la definizione di tipo)*
- *la definizione delle funzioni*

ADT counter

1. *file header* counter.h

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

Definisce in astratto **cos'è** un counter e 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)++; }
```

Specifica **come funziona** (quale è l'implementazione) un counter

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

- Gli ADT così realizzati funzionano, 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
- La struttura interna dell'oggetto è *visibile a tutti* (nella typedef)

Superare questi limiti sarà uno degli obiettivi della programmazione ad oggetti