Tipi di Dati Astratti

Luca Abeni

April 19, 2017

Dati e Tipi di Dato

- Tipo di dato: concetto di “alto livello”
  - Macchina fisica: unico tipo di dato $\rightarrow$ sequenze di bit
  - Macchine Astratte: tipi di dato più complessi
- Tipo di dato: specificano che valori attribuire a determinate sequenze di bit e che operazioni si possono compiere su tali valori
  - Non solo insieme di valori, ma anche operazioni!
- Tipo di dati opaco: la rappresentazione interna non è “visibile”
  - Esempio: lista ordinata. Conosco le operazioni su tale lista, ma non so come la lista “è implementata”

Definire Nuovi Tipi di Dato

- Molti linguaggi permettono di “definire nuovi tipi” o di definire sinonimi per tipi esistenti...
- Ma si tratta realmente di nuovi tipi di dati?
  - Si possono “creare” nuovi valori...
    - Forse... :)
  - Ma la struttura interna di un tipo è direttamente accessibile al programmatore
- Esempio: tipo `struct lista` in C. Cosa impedisce di accedere direttamente ai campi della struttura, invece di usare solo specifiche operazioni?
  - Vorremmo manipolare liste usando solo `inserisci`, `stampa`, ...

Tipi di Dato “Veri”

- Per “creare veramente” un nuovo tipo di dato...
  - ...Sarebbe necessario “nascondere” la struttura interna di un tipo e “legare” le operazioni al tipo
- Ricordiamo astrazione sul controllo: funzioni
  - Posso usare una funzione conoscendone il prototipo (dichiarazione)

Astrazione sui Dati - 1

- Sarebbe bello definire un tipo di dato in termini di interfaccia, evitando di rendere pubblica l'implementazione
  - Interfaccia: tipi e funzioni accessibili all'utente
  - Implementazione: struttura interna dei dati e delle funzioni che agiscono su di essi
- Esempio: lista in C. Interfaccia:
  - `struct lista`
  - `struct lista *inserisci(int n, struct lista *l),`
  - `void stampa(struct lista *l),`
  - ...

Astrazione sui Dati - Lista in C

- Non voglio rendere nota l'implementazione
  - `struct lista {int val; struct lista *next; }`
  - Implementazione delle funzioni `inserisci()`, `stampa()`, ...
- Specifica: dice cosa le varie operazioni fanno sui dati
  - Espressa non in termini del tipo (non fa riferimento a `val` o `next`, etc...), ma tramite relazioni generali (esempio: `inserisci()` inserisce un numero naturale in una lista ordinata), ...
- Teoricamente, se l'implementazione cambia i programmi che usano l'interfaccia non dovrebbero risentirne...

Riassumendo...

- Tipo di dato: insieme di valori ed operazioni su di essi
- Interfaccia: cosa è “visibile al programmatore”
  - Nomi, valori, prototipi, ...
- Implementazione: dettagli “interni”
  - Rappresentazione dei dati, definizione delle operazioni, ...
- Specifica: Funzionamento “inteso” del tipo, espresso mediante proprietà osservabili attraverso linterfaccia
  - Senza riferimenti a dettagli implementativi

Abstract Data Type

- Tipo di dato: insieme dei valori ed operazioni su tali valori
- Tipo di dato astratto (ADT): “protegge” (rende inaccessibile al programmatore) la rappresentazione interna del dato e tutta l’implementazione
- Un ADT è caratterizzato da:
  1. Il *nome* del tipo
  2. Una rappresentazione dei valori del tipo
  3. Un insieme di operazioni
  4. La loro implementazione
  5. Un meccanismo per separare nomi del tipo ed operazioni dalle loro rappresentazioni / implementazioni

Interfaccia vs Implementazione

- Interfaccia (visibile): nomi, valori e prototipi chiaramente separata da implementazione (non visibile ne' accessibile!)
- Valori: interfaccia
- Insieme delle operazioni sui valori (nomi e prototipi): interfaccia
- Rappresentazione dei valori del tipo: implementazione
- Implementazione di ogni operazione: basata sulla rappresentazione dei dati di cui sopra

Abstract Data Type - Supporto Linguistico

- Astrazione sul controllo: “nasconde” il corpo delle subroutine (funzioni e procedure)
  - Esempio: separazione fra prototipo e definizione
- Astrazione sui dati: “nasconde” la rappresentazione interna dei dati
  - Esempio: separazione fra dichiarazione e definizione di una struttura!
- Che supporto linguistico ci forniscono i vari linguaggi?

Abstract Data Type - Supporto Linguistico

- Differenti linguaggi hanno differenti livelli di supporto per ADT
- In generale:
  - Sintassi per separare *signature* ed implementazione
  - Generica keyword `abstype {type ... signature ... operations}`
- Linguaggio C: forma di supporto "più primitiva"
  - header file (`.h`) = interfaccia/signature;
  - implementazione in file `.c` separati
- Java, C++: OO $\rightarrow$ ADT on steroid
- ML: vedremo...

Indipendenza dalla Rappresentazione

- Principio di indipendenza dalla rappresentazione
  - “Due implementazioni corrette di una stessa specifica (ed interfaccia) di un ADT sono osservabilmente indistinguibili da parte dei clienti di quel tipo”
- Esempio: potrei reimplementare la `struct lista` usando internamente un array...
- Problemi con specifiche informali / non completamente definite
  - Out of memory
  - Stampa di lista vuota?
- Versione “rilassata” del principio di indipendenza dalla rappresentazione, basata solo su signature

Programmazione in Piccolo ed in Grande

- Programmazione “in piccolo”: sviluppo di piccoli programmi “di dimensione trattabile”
  - Possono essere organizzati in un singolo modulo di compilazione
  - Tutto il codice assieme; no separazione in unità logicamente uniformi
  - In questo caso, ADT per incapsulare dati con relative operazioni
- Programmazione “in grande”: sviluppo di programmi “complessi”
  - Necessità di dividere in codice in più file separati
  - ...

Programmazione in Grande

- Programmazione “in grande”: sviluppo di programmi più “complessi”
  - Necessità di dividere in codice in file separati
  - *Moduli* di compilazione o *package*
  - Possibilità di raggruppare dati e codice in base alle funzionalità
- Dati ed operazioni suddivisi in moduli / package
- Meccanismo utilizzabile per incapsulare dati...

Moduli vs ADT

- Diversi costrutti per funzionalità simili
  - Alcuni linguaggi forniscono `abstype` (o simile), altri moduli (esempio: C)
- I moduli (o package) dal punto di vista pratico forniscono a volte più funzionalità
  - Perché progettati per sviluppare programmi più complessi (programmazione in grande)
- Come un ADT, un modulo ha interfaccia ed implementazione
  - Possibilità di definire tipi privati nell'implementazione
- L'interfaccia di un modulo può essere *importata* da chi vuole usare il modulo