

# I Tipi di Dato Astratto

# Sommario

- ▶ Cosa sono le Strutture Dati Astratte?
  - ▶ Le strutture dati
  - ▶ Le operazioni
- ▶ Come scegliere fra varie implementazioni?

# Quale è la questione?

- ▶ Come organizzare (*strutturare*) i dati perché sia possibile elaborarli agevolmente tramite algoritmi?
- ▶ **Importanza:**
  - ▶ In alcune applicazioni la scelta della struttura dati è l'**unica** scelta importante
  - ▶ A seconda della scelta della struttura dati, l'implementazione di un algoritmo può risultare più o meno **efficiente**
  - ▶ La scelta di una struttura dati può comportare un guadagno di **tempo** e/o di **spazio**

# Tipo di dato

- ▶ **Definizione:**

Un **tipo di dato** è definito da un insieme di valori e da una collezione di operazioni su questi valori

- ▶ **Esempio:** un tipo di dato è il tipo intero in cui l'insieme di valori è costituito dai numeri naturali e le operazioni dalla somma, sottrazione, moltiplicazione, divisione, etc.

# Verso l'astrazione

- ▶ Preoccupazione principale nello scrivere un programma:
  - ▶ Efficienza
  - ▶ Semplicità del programma (uso e modifica)
  - ▶ Riutilizzo del programma (stabile e affidabile)
  - ▶ Applicazione alla più ampia varietà possibile di situazioni
  - ▶ Astrazione dalle implementazioni per poter lavorare a livelli di complessità maggiore

# L'astrazione

- ▶ Si può lavorare a diversi livelli di astrazione:
- ▶ **Bit** entità di informazione binaria: astrae dal supporto fisico (tecnologia elettronica) con cui è rappresentato
- ▶ **Modello** di calcolatore: astrae dalla rappresentazione dell'informazione
- ▶ **Linguaggi** di programmazione: si astrae dal linguaggio macchina e quindi dal modello di calcolatore
- ▶ **Algoritmi**: si astrae dai linguaggi di programmazione
- ▶ **ADT**: si astrae dalle implementazioni algoritmiche

# Utilità delle astrazioni

- ▶ Lavorare a livelli alti di astrazione permette di lavorare in modo **semplice** su problemi **complessi**
- ▶ Si possono analizzare gli algoritmi **indipendentemente** dai linguaggi con i quali sono poi implementati
- ▶ Si possono realizzare programmi complessi tramite le strutture dati astratte **indipendentemente** dalla loro implementazione algoritmica

# Tipo di dato astratto

- ▶ **Definizione di *Abstract Data Type* (ADT):**

*Un ADT è un tipo di dato accessibile **solo** attraverso una interfaccia specificata*

- ▶ o anche:

*Un ADT e' un insieme di valori e operazioni associate specificate indipendentemente da una particolare implementazione*

- ▶ Si chiama ***programma client*** il programma che usa ADT
- ▶ si chiama ***implementazione*** il programma che specifica il tipo di dato (cioè i valori e le operazioni)

# In sintesi

- ▶ L'idea cardine di un ADT è di **separare** oggetti e operazioni concettuali da una specifica implementazione
- ▶ I benefici che si ottengono da questa separazione sono:
  - ▶ Il codice è facilmente comprensibile: sono visibili solo operazioni di alto livello e non i dettagli
  - ▶ Le implementazioni possono essere cambiate per migliorare l'efficienza lasciando inalterati i programmi clienti
  - ▶ Robustezza: il codice funziona come voluto in altri programmi

# In sintesi

- ▶ Si identificano due parti in un ADT:
  - ▶ pubblica o esterna:
    - ▶ architettura concettuale (cosa è uno specifico ADT)
    - ▶ operazioni concettuali (cosa si può fare con l'ADT)
  - ▶ privata o interna:
    - ▶ rappresentazione dei dati
    - ▶ implementazione delle operazioni (gli algoritmi usati)

# Esempio: il punto

- ▶ Un ADT che rappresenti un punto bidimensionale mette a disposizione delle operazioni come ad es. l'assegnazione, il confronto, la somma
  - ▶ questo viene fatto senza rivelare i dettagli implementativi interni: l'interfaccia maschera l'implementazione
  - ▶ è possibile rappresentare un punto mediante due coordinate cartesiane  $x, y$  oppure mediante coordinate polari  $r, \theta$
  - ▶ si vuole poter cambiare la rappresentazione interna senza che il programma client debba essere modificato



# Esempio: sequenza

- ▶ Un insieme di operazioni definibili per la sequenza potrebbe essere:
  - ▶ **dimensione**: ritorna il numero di oggetti nella sequenza
  - ▶ **aggiungiPrima**: inserisci un oggetto prima della posizione corrente
  - ▶ **aggiungiDopo**: inserisci un oggetto dopo la posizione corrente
  - ▶ **rimuoviCorrente**: elimina dalla sequenza l'oggetto di posizione corrente
  - ▶ **restituisceCorrente**: ritorna l'oggetto di posizione corrente
  - ▶ **inizia**: rendi il primo oggetto l'elemento corrente
  - ▶ **avanza**: incrementa la posizione corrente
  - ▶ **testaCorrente**: vero se esiste un oggetto alla posizione corrente
  - ▶ **trova**: cerca la posizione di un oggetto nella sequenza

# Proprietà degli ADT

- ▶ Gli ADT di interesse descrivono insiemi o *collezioni* di elementi (che a loro volta possono essere ADT)
- ▶ Queste collezioni possono essere *dinamiche*, ovvero il numero di elementi può variare: si possono aggiungere o togliere elementi dalla collezione
- ▶ Gli elementi hanno generalmente una struttura costituita da una *chiave* e (eventualmente) da altri *dati satellite*
- ▶ La chiave ha in genere valori in un insieme *totalmente ordinato* (per cui vale la proprietà di tricotomia cioè per ogni coppia di elementi  $a, b$  nell'insieme deve valere esattamente una delle seguenti relazioni:  $a=b$ ,  $a < b$ ,  $a > b$ )

# Operazioni in un ADT

- ▶ In genere le varie operazioni definibili su di un ADT sono raggruppabili nelle categorie:
  - ▶ Inizializzazione
  - ▶ Aggiunta
  - ▶ Rimozione
  - ▶ Accesso

# Esempio Operazioni per un ADT

## ▶ *Aggiunta*

- *inserimento* di un nuovo elemento
- *unione* di due collezioni

## ▶ *Cancellazione*

- *cancellazione* di uno specifico elemento

## ▶ *Accesso*

- *ricerca* di un elemento avente una chiave specificata
- *minimo* e *massimo* ovvero restituzione dell'elemento con chiave più piccola o più grande
- *successore* e *predecessore* ovvero restituzione dell'elemento con la minore chiave maggiore di una data chiave (o la maggiore chiave minore)
- *selezione* del k-esimo elemento più piccolo
- *ordinamento* ovvero attraversamento della collezione in ordine di chiave

# Quali ADT vedremo?

- ▶ Sequenze, Alberi, Grafi
- ▶ Pile, Code e Code con priorità
- ▶ Dizionari (tabelle hash, alberi binari di ricerca)
- ▶ Ci interesseremo particolarmente delle operazioni di **ordinamento** e di **ricerca**

# ADT di Prima Categoria

- ▶ Per una maggiore flessibilità è necessario garantire di poter utilizzare istanze degli ADT come **parametri** in ingresso o in uscita a funzioni, o averne istanze **multiple** (ad esempio un vettore di istanze)
- ▶ **Definizione:**

Un tipo di dato di prima categoria (o classe) è un tipo di dato del quale possono esistere istanze multiple e che possiamo assegnare a variabili che sono dichiarate in modo specifico per memorizzare queste istanze

# E le implementazioni?

- ▶ La differenza fra due implementazioni algoritmiche delle operazioni che permettono l'uso delle interfacce sta nell'**efficienza**
- ▶ Per poter caratterizzare l'efficienza si ricorre all'**analisi** degli algoritmi
- ▶ L'analisi permette di stabilire quale algoritmo sia migliore in funzione delle **caratteristiche** dei dati su cui lavoriamo
- ▶ **Esempio:** l'algoritmo migliore che implementa l'operazione di ordinamento per collezioni di dati *quasi* ordinate è diverso da quello migliore per collezioni di dati ordinati *casualmente*

# Algoritmi e pseudocodice

- ▶ Le operazioni su un ADT vengono implementate tramite algoritmi
- ▶ Durante l'analisi degli algoritmi conviene astrarsi dallo specifico linguaggio di programmazione
- ▶ Per fare questo si usa un linguaggio detto *pseudocodice*
- ▶ Nello pseudocodice si possono impiegare metodi **espressivi** più chiari e concisi che nei linguaggi di programmazione reali
- ▶ Lo pseudocodice in genere **non** è un linguaggio formale
- ▶ Nello pseudocodice si possono usare **frasi** in linguaggio naturale per sintetizzare procedure complesse ma non ambigue

# Convenzioni sullo pseudocodice

- ▶ **Convenzioni sullo pseudocodice:** stesse convenzioni utilizzate nel libro “Introduzione agli algoritmi” di T.H.Cormen, C.E.Leiserson, R.L.Rivest
- ▶ Le indentazioni indicano la struttura dei blocchi
- ▶ I costrutti iterativi `while`, `repeat` e `for` e quelli condizionali `if`, `then`, `else` hanno la stessa interpretazione dei linguaggi Pascal o C/C++
- ▶ Il simbolo “ ▶ ” indica un commento
- ▶ L'assegnamento si indica con il simbolo ‘←’ come in `i←3`
- ▶ Il test di eguaglianza si indica con il simbolo ‘=’

# Convenzioni sullo pseudocodice

- ▶ Si indica l'accesso all'elemento di posizione  $i$ -esima di un array  $A$  tramite la notazione  $A[i]$
- ▶ Si accede agli attributi o campi di un oggetto usando il nome del campo seguito dal nome dell'oggetto fra parentesi quadre come in  $\text{length}[A]$  per denotare la lunghezza del vettore  $A$ 
  - ▶ Nota: in C++ avremmo invece usato la convenzione  $A.\text{length}$
- ▶ Nelle procedure o funzioni i parametri sono passati per valore (per copia)
- ▶ ..cioe' non verranno mai passati oggetti per alias o indirizzo