## Corso di Laurea in INFORMATICA a.a. 2017-2018 Algoritmi e Strutture Dati MODULO 2 Astrazioni e Algebre di dati

Dati e rappresentazioni, requisiti delle astrazioni di dati, costrutti. Astrazioni di dati e dati primitivi. Specifica sintattica e semantica. La realizzazione.



Questi lucidi sono stati preparati da per uso didattico. Essi contengono materiale originale di proprietà dell'Università degli Studi di Bari e/o figure di proprietà di altri autori, società e organizzazioni di cui e' riportato il riferimento. Tutto o parte del materiale può essere fotocopiato per uso personale o didattico ma non può essere distribuito per uso commerciale. Qualunque altro uso richiede una specifica autorizzazione da parte dell'Università degli Studi di Bari e degli altri autori coinvolti.

## Astrazione procedurale

- fornita da tutti i linguaggi ad alto livello
- aggiunge nuove operazioni a quelle della macchina astratta del linguaggio di programmazione
  - per esempio, sqrt sui float
- la specifica descrive le proprietà della nuova operazione



### Un esempio

```
float sqrt (float coef) {
   // PRE: coef > 0
   // POST: ritorna una approssimazione
   // della radice quadrata di coef
   float ans = coef / 2.0; int i = 1;
   while (i < 7) {
    ans = ans-((ans*ans-coef)/(2.0*ans));
    i = i+1;   }
   return ans; }</pre>
```

- precondizione
  - deve essere verificata quando si chiama la procedura
- postcondizione
  - tutto ciò che possiamo assumere valere quando la chiamata di procedura termina, se al momento della chiamata era verificata la precondizione



### Il punto di vista di chi usa la procedura

```
float sqrt (float coef) {
   // PRE: coef > 0
   // POST: ritorna una approssimazione
   // della radice quadrata di coef
   ... }
```

- gli utenti della procedura non si devono preoccupare di capire cosa la procedura fa, astraendo le computazioni descritte dal corpo
  - cosa che può essere molto complessa
- gli utenti della procedura non possono osservare le computazioni descritte dal corpo e dedurre da questo proprietà diverse da quelle specificate dalle asserzioni
  - astraendo dal corpo (implementazione), si "dimentica" informazione evidentemente considerata non rilevante



## Altri tipi di astrazione

#### - iterazione astratta

 permette di iterare su elementi di una collezione, senza sapere come questi vengono ottenuti

#### - gerarchie di tipo

 permette di astrarre da specifici tipi di dato a famiglie di tipi correlati

#### - tipi di dati astratti

 si aggiungono nuovi tipi di dato a quelli della macchina astratta del linguaggio di programmazione



### Iterazione astratta

- non è fornita da nessun linguaggio di uso comune
  - può essere simulata (per esempio, in C++, in Java)
- permette di iterare su elementi di una collezione, senza sapere <u>come</u> questi vengono ottenuti
- evita di dire cose troppo dettagliate sul flusso di controllo all'interno di un ciclo
  - per esempio, potremmo iterare su tutti gli elementi di un insieme senza imporre nessun vincolo sull'ordine con cui vengono elaborati
- astrae (nasconde) il flusso di controllo nei cicli



## Gerarchie di tipo

- fornite da alcuni linguaggi ad alto livello O-O
  - per esempio, C++, Java
- permettono di astrarre gruppi di astrazioni di dati (tipi) a <u>famiglie</u> di tipi
- i tipi di una famiglia condividono alcune operazioni
  - definite nel supertype, di cui tutti i tipi della famiglia sono subtypes
- una famiglia di tipi astrae i dettagli che rendono diversi tra loro i vari tipi della famiglia
- in molti casi, il programmatore può ignorare le differenze



### Astrazione sui dati

- ricalca ed estende il concetto di astrazione procedurale
- fornita da tutti i linguaggi ad alto livello moderni
- aggiunge al linguaggio nuovi tipi di dato e i relativi operatori.
- è cosa diversa da un insieme di astrazioni procedurali

#### Ad esempio, è possibile aggiungere:

- un tipo Lista, con le operazioni inserisci, rimuovi, primolista, etc.
- la rappresentazione dei valori di tipo Lista e le operazioni sono realizzate nel linguaggio di riferimento
- l'utente non deve interessarsi dell'implementazione, ma fare solo riferimento alle proprietà presenti nella specifica
- le operazioni sono astrazioni definite da asserzioni
  - la specifica descrive le relazioni fra le varie operazioni



#### Astrazione sui dati

LE PROPRIETA' CHE CARATTERIZZANO UN TIPO DI DATI DIPENDONO ESCLUSIVAMENTE DALLA SPECIFICA DEL TIPO DI DATO MENTRE DEVONO ESSERE INDIPENDENTI DAL MODO IN CUI I DATI SONO RAPPRESENTATI.

LE OPERAZIONI APPLICABILI SUGLI OGGETTI O SULLE STRUTTURE DATI CHE RAPPRESENTANO SONO ISOLATE DAI DETTAGLI USATI PER REALIZZARE IL TIPO.



#### Strutture dati

- •sono particolari tipi di dato, caratterizzati più dall'organizzazione dei dati più che dal tipo dei dati stessi
- •il tipo dei dati contenuti può essere addirittura parametrico

#### Una struttura di dati è composta quindi da:

- un modo sistematico di organizzare i dati
- un insieme di operatori che permettono di manipolare la struttura



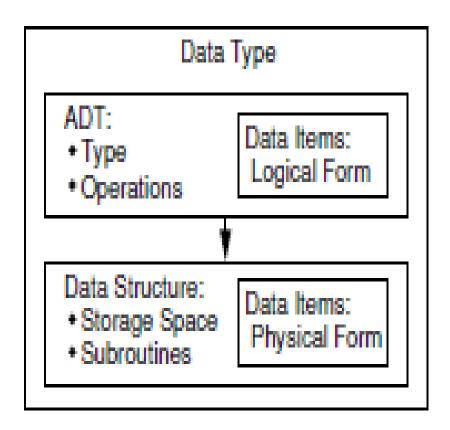
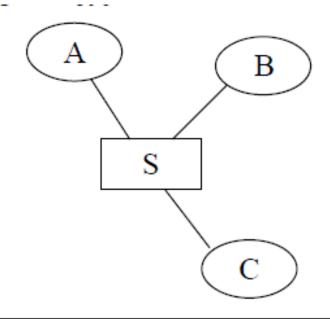


Figure 1.1 The relationship between data items, abstract data types, and data structures. The ADT defines the logical form of the data type. The data structure implements the physical form of the data type.

## Una struttura di dati è necessaria per implementare la forma fisica di un ADT

#### Astrazione sui dati



A, B, C procedure S struttura dati

Per accedere ad S le procedure A, B e C devono conoscerne i dettagli!

Si ha così un'inutile propagazione delle informazioni su S che devono essere resi noti ad A, B e C.



#### Astrazione sui dati

- Problemi relativi ad un accesso alle strutture dati "non disciplinato":
  - Propagazione degli errori difficilmente controllabile e prevedibile
    - ? Un errore provocato da un modulo cliente A potrebbe rivelarsi in corrispondenza di un modulo cliente B che invece partendo da dati consistenti li manipolerebbe correttamente!
  - Difficile manutenzione
    - ? Un cambiamento su S indurrebbe modifiche su tutti i moduli che lo utilizzano)
  - Riuso tipicamente inesistente
    - ? Le strutture dati riflettono aspetti relativi all'applicazione e all'implementazione, e quindi non sono stati "pensati" per essere riusati





Molti linguaggi di programmazione forniscono i mezzi per creare nuovi tipi di dati, ma non tutti consentono di fare ASTRAZIONE DATI

#### UN'ASTRAZIONE DI DATI È COSTITUITA DA:

#### **UNA SPECIFICA:**

HA IL COMPITO DESCRIVERE SINTETICAMENTE IL TIPO DEI DATI E GLI OPERATORI AD ESSI APPLICABILI E CHE LI CARATTERIZZANO

# UNA REALIZZAZIONE o IMPLEMENTAZIONE: STABILISCE COME I DATI E GLI OPERATORI VENGONO RICONDOTTI AI TIPI DI DATI E AGLI OPERATORI GIÀ DISPONIBILI (NATIVI)



## COME ATTUARE UNA CORRETTA ASTRAZIONE DI DATI?

■ BISOGNA IDENTIFICARE I COSTRUTTI DI PROGRAMMAZIONE CHE LA CONSENTONO

SONO NECESSARI MEZZI PER DEFINIRE E CREARE NUOVI TIPI DI DATI

PER GLI OPERATORI, NON E' SUFFICIENTE USARE SOTTOPROGRAMMI CHE LI IMPLEMENTANO, MA BISOGNA ASSICURARE CHE A QUESTI VI SIA UN ACCESSO "CONTROLLATO"

## Un *ADT* (*Abstract Data Type*) è un modello (formale) che generalizza ed estende i tipi di dato primitivi e le operazioni definite su di essi.

In parole povere, un **ADT** è una combinazione di dati e operazioni sugli stessi.

Esempio.

Un numero complesso ha:

• dati:

Parte reale, Parte immaginaria

•operazioni:

moltiplicazione, divisione, modulo

#### Un file ha:

• dati:

nome, posizione

• operazioni:

scrittura, lettura, spostamento

#### Una anatra ha:

• dati:

Numero di riconoscimento, posizione

operazioni:

camminare, nuotare, starnazzare



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

- una definizione di tipo che implica un dominio, D
- un insieme di metodi o operazioni ammissibili su oggetti di quel tipo

funzioni: calcolano valori sul dominio D

predicati: calcolano proprietà vere o false su D

Ad esempio in C, un ADT si costruisce definendo:

il nuovo tipo con typedef una funzione per ogni operazione



#### Un ADT (contatore) è connotato da:

- □ lo stato dell'ADT (es. conteggio);
- □ i metodi che definiscono le operazioni ammesse sul dato astratto (modifica e/o accesso allo stato) (es. crea, elimina, azzera, incrementa, decrementa).

L'ADT consente pertanto di estendere l'insieme dei tipi di dato predefiniti del linguaggio.

#### **Esempio in C: il contatore**

una entità caratterizzata da un valore intero

#### typedef int counter;

con operazioni per inizializzare il contatore a zero
reset(counter\*);

incrementare il contatore inc(counter\*);



#### Inoltre:

- □ Un tipo di dato astratto o ADT (Abstract Data Type) è un tipo di dato le cui istanze possono essere manipolate con modalità che dipendono esclusivamente dalla semantica del dato e non dalla sua implementazione.
- Nei linguaggi di programmazione che consentono la programmazione per tipi di dati astratti, un tipo di dati viene definito distinguendo nettamente la sua interfaccia, ovvero le operazioni che vengono fornite per la manipolazione del dato, e la sua implementazione interna, ovvero il modo in cui le informazioni di stato sono conservate e come le operazioni manipolano tali informazioni al fine di esibire il comportamento desiderato.
- □ La conseguente inaccessibilità dell'implementazione viene spesso identificata con l'espressione incapsulamento (information hiding).



## Tornando all'esempio dato in C rileviamo che:

- Gli ADT così realizzati funzionano, ma molto dipende dall'autodisciplina del programmatore
- ☐ Infatti, non esiste alcuna protezione contro un uso scorretto dell'ADT
- ☐ L'organizzazione del dato realizzata 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)* 



DISPONENDO DI UN LINGUAGGIO CHE CONSENTE LA DEFINIZIONE DI TIPI, PER REALIZZARE GLI ADT E' **NECESSARIO:** 

- UTILIZZARE DATI CHE DEFINIAMO E DICHIARIAMO DIRETTAMENTE, PRESCINDENDO DALLA EFFETTIVA REALIZZAZIONE
- FARE IN MODO CHE ALLE PROCEDURE (OPERATORI) COSTRUITE PER I NOSTRI DATI ABBIANO ACCESSO **ESCLUSIVAMENTE QUEI DATI**

QUESTE CARATTERISTICHE NECESSARIE A UNA BUONA ASTRAZIONE DI DATI SONO NOTE COME



#### I REQUISITI DELLA ASTRAZIONE DI DATI/1

#### <REQUISITO DI ASTRAZIONE>

DÈ VERIFICATO QUANDO I PROGRAMMI CHE USANO UN'ASTRAZIONE POSSONO ESSERE SCRITTI IN MODO DA NON DIPENDERE DALLE SCELTE DI REALIZZAZIONE. NEL CASO DI ALCUNI LINGUAGGI (PASCAL, ADA, MODULA2 etc.) LE DIVERSE REALIZZAZIONI NON PROVOCANO CAMBI NELLE INTESTAZIONI DEI SOTTOPROGRAMMI CHE REALIZZANO GLI OPERATORI.

MANIFESTA CON L'IMPOSSIBILITÀ DI DICHIARARE "DIRETTAMENTE" VARIABILI CON IL NUOVO TIPO DEFINITO. NEL CODICE NON SI PUO' FARE RIFERIMENTO AI NUOVI DATI CREATI PRESCINDENDO DALLA RAPPRESENTAZIONE USATA E DALLA REALIZZAZIONE.



#### I REQUISITI DELLA ASTRAZIONE DI DATI/2

#### <REQUISITO DI PROTEZIONE >

□È VERIFICATO SE SUI NUOVI DATI SI PUÒ LAVORARE ESCLUSIVAMENTE CON GLI OPERATORI DEFINITI ALL'ATTO DELLA SPECIFICA.

LA MANCANZA DEL REQUISITO DI PROTEZIONE SI MANIFESTA CON LA POSSIBILITA' DI LAVORARE CON GLI OPERATORI DEFINITI PER I NUOVI DATI ANCHE SU DATI CON RAPPRESENTAZIONI SIMILI, MA NON <u>OMOGENEI</u> PER TIPO.

NEL CASO DI ALCUNI LINGUAGGI, IL TIPO DI DICHIARATIVE DEI SOTTOPROGRAMMI NON SPECIFICA FORMALMENTE IL TIPO DEI PARAMETRI E CONSENTE UN ACCESSO POCO CONTROLLATO AGLI OPERATORI.

#### I REQUISITI DELLA ASTRAZIONE DI DATI/3

IL REQUISITO DI ASTRAZIONE, IN ACCORDO CON I PRINCIPI DELLA ASTRAZIONE DI DATI, IMPONE CHE I PROGRAMMI SIANO SENSIBILI SOLO A VARIAZIONI DELLA SPECIFICA DEI NUOVI TIPI.

IL REQUISITO DI PROTEZIONE È PARTICOLARMENTE IMPORTANTE, SE VOGLIAMO CHE L'ESECUTORE DEL LINGUAGGIO ASSICURI I CONTROLLI DI CONSISTENZA TRA I TIPI DI DATI E GLI OPERATORI, NON SOLO AI DATI PRIMITIVI, MA ANCHE AI DATI OTTENUTI PER ASTRAZIONE.



#### UN'ASTRAZIONE DI DATI È COSTITUITA DA: UNA SPECIFICA E UNA REALIZZAZIONE

PER LA SPECIFICA, CHE HA IL COMPITO DESCRIVERE SINTETICAMENTE IL TIPO DEI DATI E GLI OPERATORI CHE LI CARATTERIZZANO, E' POSSIBILE DISTINGUERE:

- ☐ UN LIVELLO SINTATTICO
- ☐ UN LIVELLO SEMANTICO

#### A LIVELLO SINTATTICO BASTA DEFINIRE:

- •L'ELENCO DEI NOMI DEI TIPI DI DATO UTILIZZATI PER DEFINIRE LA STRUTTURA, DELLE OPERAZIONI SPECIFICHE DELLA STRUTTURA E DELLE COSTANTI
- •I DOMINI DI PARTENZA E DI ARRIVO, CIOÈ I TIPI DEGLI OPERANDI E DEL RISULTATO PER OGNI NOME DI OPERATORE



## A LIVELLO SEMANTICO VA SPECIFICATO IL SIGNIFICATO DEI NOMI DEI DATI E DEGLI OPERATORI ASSOCIANDO:

- UN INSIEME AD OGNI NOME DI TIPO INTRODOTTO
- UN VALORE AD OGNI COSTANTE
- UNA FUNZIONE AD OGNI NOME DI OPERATORE

PER FARE IN MODO CHE SIA UNIVOCAMENTE DEFINITA L'APPLICABILITA' DI UN OPERATORE E L'EFFETTO, OLTRE A ESPLICITARE I DOMINI DI PARTENZA E DI ARRIVO, VANNO SPECIFICATE:

- 1) UNA PRECONDIZIONE CHE DEFINISCE QUANDO L'OPERATORE È APPLICABILE
- 2) UNA POSTCONDIZIONE CHE STABILISCE COME IL RISULTATO SIA VINCOLATO AGLI ARGOMENTI DELL'OPERATORE



E' NECESSARIO DEFINIRE L'ALGEBRA DEL DATO

#### L'algebra dei dati

- Un tipo astratto di dato può essere formalizzato come un oggetto matematico costituito da tre componenti:
- un insieme di valori, detto dominio del tipo;
- un insieme di operazioni primitive, che si applicano a valori del dominio o che hanno come risultato valori del dominio;
- un insieme di costanti, che denotano valori significativi del dominio.

Per esempio, il tipo astratto boolean potrebbe essere così definito:

- dominio: insieme di valori { true, false }
- operazioni: and, not, or
- costanti: true, false



Le costanti denotano, rispettivamente, i due valori del dominio. In questo esempio ne sono state definite tante quanti sono i valori del dominio.

In altri casi non è così.

Nella seguente definizione di un tipo di dato per i numeri naturali:

- dominio: quello dei numeri naturali
- operazioni: +, \*, =, <, >
- costanti: 0,1

il dominio è infinito e sono definite solo due costanti significative.



Esempio: I NUMERI INTERI E I BOOLEANI

**SPECIFICA** 

**TIPI: INTEGER, BOOLEAN** 

**OPERATORI:** 

+, - : (INTEGER, INTEGER)  $\rightarrow$  INTEGER

Forniscono come risultato somma e differenza

<,>: (INTEGER, INTEGER)  $\rightarrow$  BOOLEAN

Danno VERO se il 1° operatore è minore (maggiore) del 2°

AND, OR: (BOOLEAN, BOOLEAN) → BOOLEAN

Forniscono la congiunzione e la disgiunzione logica dei due operandi

**NOT** :  $(BOOLEAN) \rightarrow BOOLEAN$ 

Fornisce la negazione logica dell'operando

**SEQCIF** :  $() \rightarrow INTEGER$ 

Fornisce l'intero in notazione decimale

**TRUE** : ()  $\rightarrow$  BOOLEAN

Corrisponde al valore di verità VERO

FALSE :  $() \rightarrow BOOLEAN$ 

Corrisponde al valore di verità FALSO

IN QUESTA SPECIFICA IL TIPO DI DATO INTEGER POSSIEDE UN NUMERO INFINITO DI COSTANTI (OGNI POSSIBILE SEQUENZA DI CIFRE DECIMALI È UNA COSTANTE)

MA VI E' UN SOLO MODO DI DEFINIRE UNA SPECIFICA? L'INSIEME DEGLI OPERATORI DEFINIBILI SU INSIEMI DI DATI E'UNICO?



#### **UN'ALTRA POSSIBILE SPECIFICA.**

TIPI: INTEGER, BOOLEAN

#### **OPERATORI:**

**ZERO** :  $() \rightarrow INTEGER$ 

Dà lo zero dei numeri interi

TRUE : ()  $\rightarrow$  BOOLEAN

Corrisponde al valore di verità VERO

**FALSE** :  $() \rightarrow BOOLEAN$ 

Corrisponde al valore di verità FALSO

**SUCC**: (INTEGER)  $\rightarrow$  INTEGER

Dà n+1

PRED : (INTEGER)  $\rightarrow$  INTEGER

Dà n-1

**ISZERO** : (INTEGER)  $\rightarrow$  BOOLEAN

Dà vero se e solo se n è lo zero degli interi



DEFINITE LE SPECIFICHE, LA REALIZZAZIONE SFRUTTA AL MEGLIO LE POSSIBILITA' OFFERTE DALL'AMBIENTE DI PROGRAMMAZIONE.

UN COSTRUTTO DI PROGRAMMAZIONE PROPRIAMENTE ADATTO ALLA REALIZZAZIONE DI

#### **ASTRAZIONE DI DATI**

#### DOVRÀ:

- DICHIARARE ESPLICITAMENTE QUALI SONO
  - I NUOVI TIPI DI DATI E
  - I NUOVI OPERATORI
- DEFINIRE LA <u>RAPPRESENTAZIONE</u> DEI NUOVI DATI IN TERMINI DI DATI PREDEFINITI (NATIVI NEL LINGUAGGIO DI PROGRAMMAZIONE)
- CONTENERE UN INSIEME DI <u>SOTTOPROGRAMMI</u> CHE EALIZZINO GLI OPERATORI DEFINITI SUI NUOVI DATI

#### LA REALIZZAZIONE

RICONDUCE LA SPECIFICA AI TIPI PRIMITIVI E AGLI OPERATORI GIÀ DISPONIBILI; IN PARTICOLARE GLI OPERATORI SONO REALIZZATI CON SOTTOPROGRAMMI.

LE METODOLOGIE DI PROGRAMMAZIONE TENDONO A RENDERE LA REALIZZAZIONE NON VISIBILE ALL'UTENTE, MA ALLA STESSA SPECIFICA POSSONO CORRISPONDERE REALIZZAZIONI PIÙ O MENO EFFICIENTI.

OLTRE A VALUTARE LA EFFICIENZA DELL'ALGORITMO, E' NECESSARIO, PER PROGRAMMI CHE USANO DATI ASTRATTI, VALUTARE L'EFFICIENZA DEI PROGRAMMI CHE REALIZZANO GLI OPERATORI.

## PER VALUTARE L'EFFICIENZA, SI PRESCINDE DALLE CARATTERISTICHE DELLA MACCHINA, E SI FA RIFERIMENTO A:

- **DATI CONTENUTI IN MEMORIA**
- MEMORIA ORGANIZZATA IN CELLE (PAROLA)
- **CELLE ACCESSIBILI TRAMITE INDIRIZZO**
- **□INDIRIZZI CHE SONO INTERI CONSECUTIVI**
- □DATI ELEMENTARI CHE NON DECIDONO LA CAPACITÀ DI UNA CELLA (MAXINT)
- IL CUI ACCESSO RICHIEDE TEMPO COSTANTE.



I LINGUAGGI DI PROGRAMMAZIONE SONO DOTATI DI STRUTTURE STATICHE NATIVE (VETTORI, ARRAY) I CUI ELEMENTI SONO CONTENUTI IN LOCAZIONI CONTIGUE DELLA MEMORIA CENTRALE.

LA STRUTTURA ARRAY E' A DIMENSIONE FISSA, ED È INTESA COME UN INSIEME DI ELEMENTI OMOGENEI IN RELAZIONE D'ORDINE TRA LORO SU CUI POSSONO EFFETTUARSI OPERAZIONI DI:

LETTURA (O SELEZIONE) REPERIMENTO DEL VALORE DI UN ELEMENTO

SCRITTURA (O SOSTITUZIONE) DI UN VALORE DI UN ELEMENTO CON UN NUOVO VALORE

L'ORGANIZZAZIONE IN MEMORIA CONSENTE L'ACCESSO DIRETTO AL SINGOLO COMPONENTE ATTRAVERSO L'INDICE.

## SE VOLESSIMO DEFINIRE LE SPECIFICHE FORMALI DI UN VETTORE COSI' COME E' INTESO NELLA MAGGIORPARTE DEI LINGUAGGI DI PROGRAMMAZIONE IMPERATIVI:

#### **SPECIFICA**

TIPI: VETTORE, INTERO, TIPOELEM

**OPERATORI:** 

**CREAVETTORE** : ()  $\rightarrow$  VETTORE

POST :  $\forall i$ ,  $1 \le i \le n$ , l'i-esimo elemento del vettore V(i) è uguale ad un definito elemento di tipo TIPOELEM

**LEGGI VETTORE**: (VETTORE, INTERO) → TIPOELEM

 $PRE : 1 \le i \le n$ 

POST: e=V(i)

**SCRIVIVETTORE**: (VETTORE, INTERO, TIPOELEM) → VETTORE

 $PRE : 1 \le i \le n$ 

POST: V'(i) = e, V'(j) = V(j) PER OGNI j

TALE CHE  $1 \le j \le n \in j \ne i$ 



### NEL CASO DEL VETTORE LA REALIZZAZIONE FA RIFERIMENTO AD UNA ORGANIZZAZIONE IN MEMORIA COMUNE A QUASI TUTTI I LINGUAGGI DI PROGRAMMAZIONE.

### **TIPICA ORGANIZZAZIONE IN MEMORIA**

PER ESEGUIRE UN ASSEGNAMENTO m = V[i]SI PRESUPPONE CHE I VALORI DELLE TRE VARIABILI m, I E V[i] SIANO CONTENUTI IN TRE CELLE



## NEI LINGUAGGI IL VETTORE È ILTIPO DI DATO PRIMITIVO IN FORTRAN

CREAVETTORE 

⇔ DIMENSION V (n)

LEGGI VETTORE  $(V, I) \Leftrightarrow V(I)$ 

SCRIVIVETTORE (V, I,e) ⇔ V(I)=e



LA CORRISPONDENZA TRA LA SPECIFICA INTRODOTTA E

QUELLA DEL C È:

**CREAVETTORE** 

 $\Leftrightarrow$ 

Oppure int  $V[5] = \{1, 2, 3, 4, 5\}$  float  $V[5] = \{1.4, 3.2, 5.4\}$ 

LEGGI VETTORE (V , I) ⇔ V [I]

SCRIVIVETTORE (V, I,e) ⇔ V[I] =e



IN PASCAL CORRISPONDE ALL'ARRAY.

LA CORRISPONDENZA TRA LA SPECIFICA INTRODOTTA E QUELLA DEL PASCAL È :

CREAVETTORE

⇔ VAR V: ARRAY[1..n] OF

**TIPOELEM** 

LEGGI VETTORE (V , I) 

⇔ V [I]

SCRIVIVETTORE (V, I,e) ⇔ V[I]:=e



### <u>REALIZZAZIONE</u>

#### IN JAVA

int n = Input.readInt(); // legge la dimensione
int[] array = new int[n]; // dichiara l'array

la creazione fa una inizializzazione implicita:

num = new int[10];
con valore 0 per int e double, false per i
boolean.

□ Dopo la creazione, un array ha lunghezza fissata (e non può cambiare nel programma tramite una nuova dichiarazione).



LA MATRICE, INTESA COME TABELLA A DUE O A MOLTE DIMENSIONI, È DI SOLITO UN DATO PRIMITIVO.

PREVEDE UN NOME E DUE O PIÙ INDICI CHE CONSENTONO L'ACCESSO DIRETTO AL SINGOLO ELEMENTO.

DIAMO LE SPECIFICHE PER UNA MATRICE DI nºmºp ELEMENTI (A TRE DIMENSIONI)

#### **SPECIFICA SINTATTICA**

TIPI: MATRICE, INTERO, TIPOELEM

**OPERATORI:** 

CREAMATRICE : ()  $\rightarrow$  MATRICE

**LEGGIMATRICE**: (MATRICE, INTERO, INTERO)

→ TIPOELEM

SCRIVIMATRICE: (MATRICE, INTERO, INTERO,

TIPOELEM)

 $\rightarrow$ 

**MATRICE** 

### **SPECIFICA SEMANTICA**

TIPI:

**INTERO**: INSIEME DI INTERI

MATRICE: INSIEME DI nemep ELEMENTI DI TIPO

**TIPOELEM ∋' L'ELEMENTO** 

CARATTERIZZATO DAGLI INDICI

i,j,k E' QUELLO NELLA

i-esima RIGA

j-esima COLONNA

k-esima DIMENSIONE IN PROFONDITA'

etc. .....

. . . . . .



# NELLA MAGGIOR PARTE DEI LINGUAGGI (C, PASCAL, ...) IL DATO PRIMITIVO MATRICE A PIU' DIMENSIONI È RIPORTATO AL DATO PRIMITIVO ARRAY

CREAMATRICE  $\Rightarrow$  VAR W: ARRAY[1..n,1..m,1..p] OF

**TIPOELEM** 

LEGGIMATRICE (W,I,J,K)  $\Rightarrow$  W[I,J,K]

SCRIVIMATRICE (W,I,J,K,e)  $\Rightarrow$  W[I,J,K]:=e



SI FA RIFERIMENTO AD UNA ORGANIZZAZIONE IN MEMORIA NELLA QUALE LE TABELLE SONO LINEARIZZATE PER RIGHE.

W	
	1
	2
•	
	n
	n + 1
	n + 2
-	
•	
	2 n
•	





### RIASSUMENDO

Alla base dell'astrazione dati c'è il principio che non si può accedere direttamente alla rappresentazione di un dato, qualunque esso sia, ma solo attraverso un insieme di operazioni considerate lecite (principio dell'astrazione dati).

VANTAGGIO: un cambiamento nella rappresentazione del dato si ripercuoterà solo sulle operazioni lecite, che potrebbero subire delle modifiche, mentre non inficerà gli utilizzatori del dato astratto.

## QUALITECNICHE? Information Hiding

Il principio di astrazione suggerisce di nascondere dell'informazione (information hiding), sia perché non necessaria al fruitore dell'entità astratta, sia perché la sua rivelazione creerebbe delle inutili dipendenze e finirebbero per compromettere l'invarianza ai cambiamenti.

Nella astrazione procedurale la tecnica suggerisce di nascondere i dettagli del processo di trasformazione (come esso avviene).

Nella astrazione dati la tecnica identifica nella rappresentazione del dato l'informazione da nascondere.

In entrambi i casi non si dice **COME** farlo.



## QUALI TECNICHE? Incapsulamento

L'incapsulamento (encapsulation) è una tecnica di progettazione consistente nell'impacchettare (racchiudere in capsule) una collezione di cose, creando una barriera concettuale.

### Implica:

- Un processo: l'impacchettamento
- Una entità: il 'pacchetto' ottenuto
- L'incapsulamento NON dice come devono essere le "pareti" del pacchetto, che potranno essere:
  - Trasparenti: permettendo di vedere tutto ciò che è stato impacchettato;
  - Traslucide: permettendo di vedere in modo parziale il contenuto;
  - Opache: nascondendo tutto il contenuto del pacchetto,

# ASTRAZIONE DATI E INCAPSULAMENTO

- La combinazione del principio dell'astrazione dati con la tecnica dell'incapsulamento suggerisce che:
- 1. La rappresentazione del dato va nascosta
- L'accesso al dato deve passare solo attraverso operazioni lecite
- Le operazioni lecite, che ovviamente devono avere accesso alla informazione sulla rappresentazione del dato, vanno impacchettate con la rappresentazione del dato stesso.



## I principi che consentono di mettere in pratica uno stile di programmazione rigoroso

- Astrazione dei dati: non serve conoscere la rappresentazione e l'implementazione interna di un oggetto usarlo
- Modularità/modificabilità: l'universo è strutturato in oggetti di cui è facile cambiare la definizione
- ☐ Riutilizzabilità: poiché un oggetto è definito dal proprio comportamento, grazie ad un'interfaccia esplicita, è facile includerlo in una libreria da riutilizzare all'occorrenza
- Leggibilità/comprensibilità: l'incapsulamento, la possibilità di overloading e la modularità aumentano la leggibilità di un programma

### Object Oriented Programming

- Grazie all'astrazione dati e all'incapsulamento un oggetto contiene ("incapsula") al suo interno gli attributi (dati) e i metodi (procedure) che accedono ai dati stessi.
- Lo scopo principale è dare accesso ai dati incapsulati solo attraverso i metodi definiti, nell'interfaccia, come accessibili dall'esterno.
- E' possibile vedere l'oggetto come una scatola nera della quale, attraverso l'interfaccia, sappiamo cosa fa e come interagisce con l'esterno ma non come lo fa.
- I vantaggi principali portati sono: robustezza, indipendenza e l'estrema riusabilità degli oggetti creati

### I Linguaggi ad oggetti

- Una possibile distinzione [Masini, Napoli, Colnet, Léonard, Tombre, 1991]
- Linguaggi che offrono solo astrazione dati e incapsulamento (Ada, Modula-2)
- Linguaggi che raggruppano gli oggetti in classi (CLU, Simula, Smalltalk-80)
- □Linguaggi ad oggetti che aggiungono all'astrazione la nozione di ereditarietà (C++)



## I Linguaggi ad oggetti

Un particolare ambiente C++ può fornire una biblioteca (library) che include una classe List. La forma logica della lista è definita dalle *Public Functions*, i relativi *input* e *output* che definiscono la classe. Questo potrebbe essere tutto quello che l'utente sa circa l'implementazione della classe lista e potrebbe essere tutto e solo quello che l'utente deve conoscere.

All'interno della classe è possibile una grande varietà di implementazioni fisiche della lista.

