Algoritmi e Strutture Dati

Università di Camerino

Corso di Laurea in Informatica

(12 CFU)

I periodo didattico

Emanuela Merelli
email:emanuela.merelli@unicam.it

Argomenti della lezione

- Elementi di un linguaggio di programmazione
  - Tipi di dato
    - Tipi di dato elementari
    - Tipi di dato strutturati
    - Tipi puntatore
  - *Concetto di*
    - *Tipi di dato astratti*
      - *Pile*
      - *code*
      - *alberi*

Dato

Dato è tutto ciò su cui agisce un calcolatore

A livello hardware tutti i dati sono rappresentati come sequenza di cifre binarie

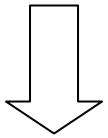
Linguaggi ad alto livello ci permettono di usare **astrazioni** tramite il concetto di **tipo di dato**

Tipo di Dato

Il **tipo di dato** determina l'insieme dei valori che una variabile può assumere

Il tipo del valore rappresentato da una variabile può derivare dalla sua dichiarazione

Ogni operatore accetta argomenti di un tipo di fissato e produce risultati di un tipo fissato



Un tipo implicitamente determina l'insieme di operatori che possono essere applicati alla variabile di tale tipo

Cardinalità di un Insieme

Il numero di valori distinti che appartengono ad un tipo T

$$\text{Type } T = (c_1, c_2, \ldots c_n)$$

$$\text{Card}(T) = n$$

$$\text{Type } T_1 = \text{Integer}$$

$$\text{Card}(T_1) = \aleph_0$$

Insieme delle parti di un insieme

Insieme di tutti i sottoinsiemi di elementi di un insieme T

Type T = (c_1, c_2, c_3, c_4)

$\mathcal{P}$(T)= {[], [c_1], [c_2], [c_3], [c_4], [c_1 c_2], [c_1 c_3], [c_1 c_4] …}

Card($\mathcal{P}$(T)) = $\boxed{2^n}$

A cosa serve il Tipo di Dato

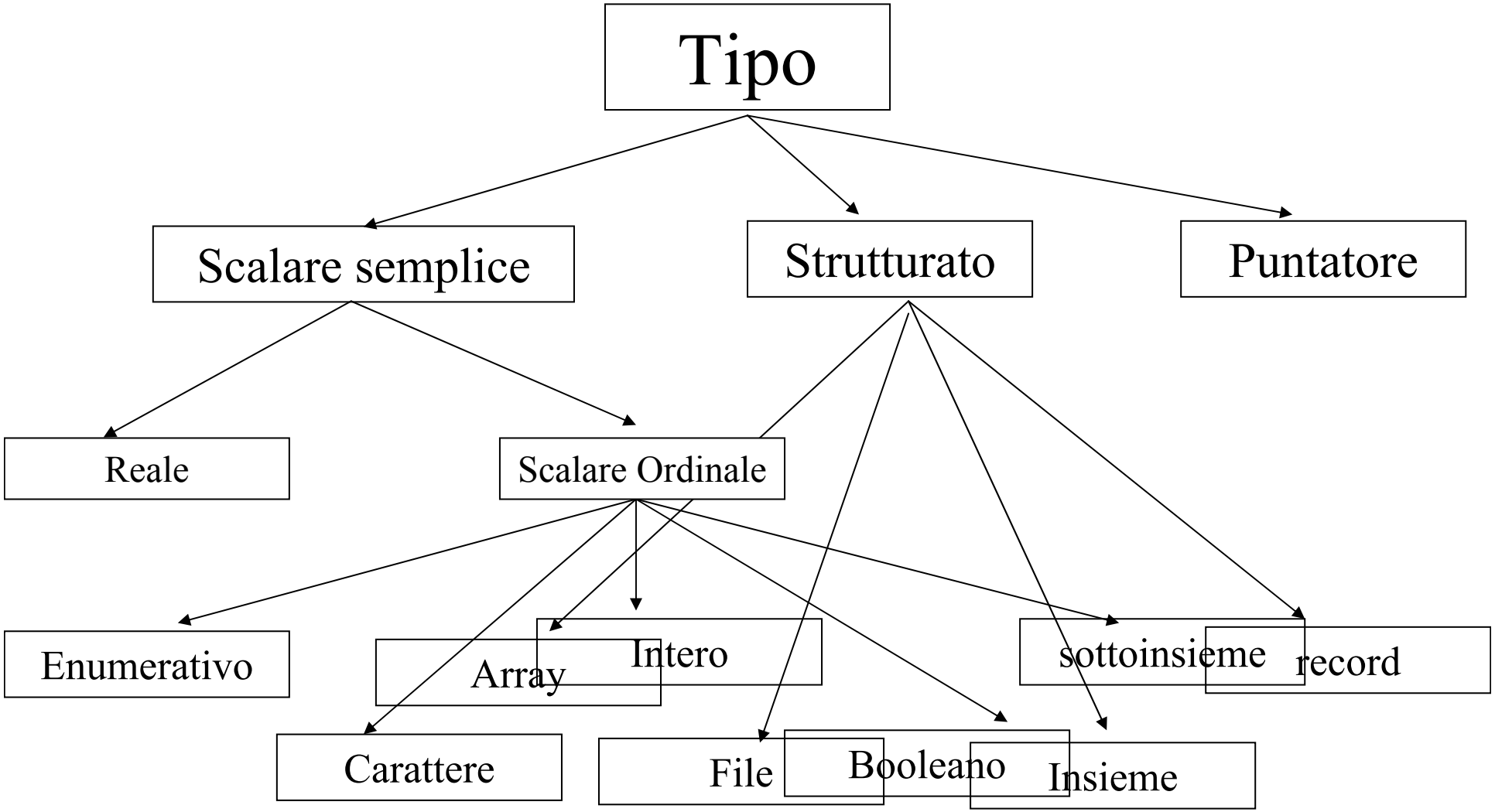
In fase dichiarativa serve a determinare la caratteristiche di una variabile

In fase compilativa serve a controllare la compatibilità e la legalità dei costrutti

Serve al compilatore per allocare la memoria necessaria a rappresentare l'oggetto associato alla variabile

Classificazione dei tipi di dato

di un linguaggio di programmazione



Operatori

Di base

- Confronto

  x > y
- Assegnamento

  x = y
- Trasferimento

  x $\rightarrow$ y

Tipi strutturati

- Costruttori

  volori $\rightarrow$ struttura
- Seletori

  struttura $\rightarrow$ valori

Integer

Valori: Sottoinsieme dei numeri interi

Operatori: + - * div

Real

Valori: Sottoinsieme dei numeri reali

Operatori: + - * /

Boolean

Valori: True, False

Operatori: and (Λ) or (V) not ($\neg$)

Char

Valori: Insieme di tutti i caratteri che possono essere stampati

Operatori: ord(c) chr(n)

Esempio:

Ord(chr(i)) = i chr(ord(c))= c

Subrange

Valori: Insieme dei valori di un intervallo di un tipo T

Operatori: tutti quelli ereditati dal tipo T

Esempio:

type T_1 = 1900 .. 1999

type T_2 = ‘A’ .. ‘Z’

Card(T_1) = ? Card(T_2) = ?

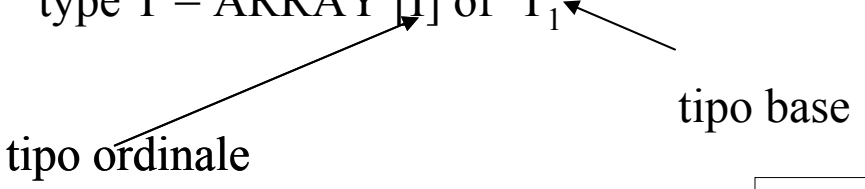
Costruttore Array

L'Array permette di definire un tipo di dato T costituito da n elementi tutti associati ad un tipo base T_1

Struttura omogenea e ad accesso casuale

Esempio:

type T = ARRAY [I] of T_1



var C : T

Operatori: Selettore (C[i])

$$Card(T) = Card(T1)^{Card(I)}$$

Costruttore Record

Il Record permette di definire un tipo di dato T costituito dall'unione di altri tipi di dati T_1, T_2, .. T_n

Esempio:

type T = RECORD

$s_1 : T_1$

s2 : T_2

…

$s_n : T_n$

Operatori: Selettore ($C.s_1$)

Var C : T

$$Card(T) = Card(T_1) * Card(T_2) \ \ldots \ Card(T_n)$$

Costruttore Set

Il Set permette di definire un tipo di dato T costituito dall'insieme delle parti di un altro tipo di dato T_1

Esempio:

type T = SET of T_1

$$Card(T) = 2^{Card(T1)}$$

Operatori:

- Intersezione (*)
- unione (+)
- differenza (-)
- appartenza (in)

Costruttore di sequenze: File

Il File permette di definire un tipo di dato T costituito da elementi tutti associati ad un tipo base T_1

Struttura omogenea e ad accesso sequenziale

Esempio:

type T = FILE of T_1

$Card(T_1) = \aleph_0$

Card(T) = ?

Operatori:

Rewrire

Put

Reset

Get

Variabili Statiche

L’uso dei tipi di dati permette di conoscere a priori la cardinalità dell’insieme dei dati, così che le variabili di quel tipo di dato potranno essere rappresentate in memoria con una **allocazione statica**

Variabili Dinamiche

Variabili la cui struttura varia durante l'esecuzione del programma: **struttura ricorsiva**

Variabili dinamiche implicano un **allocazione dinamica** della memoria

Tipi di dati ricorsivi

- *record varianti*
- puntatori

Puntatore

Permette di definire un tipo di dato Tp i cui valori sono puntatori (riferimenti) a valori di un altro tipo di dato T

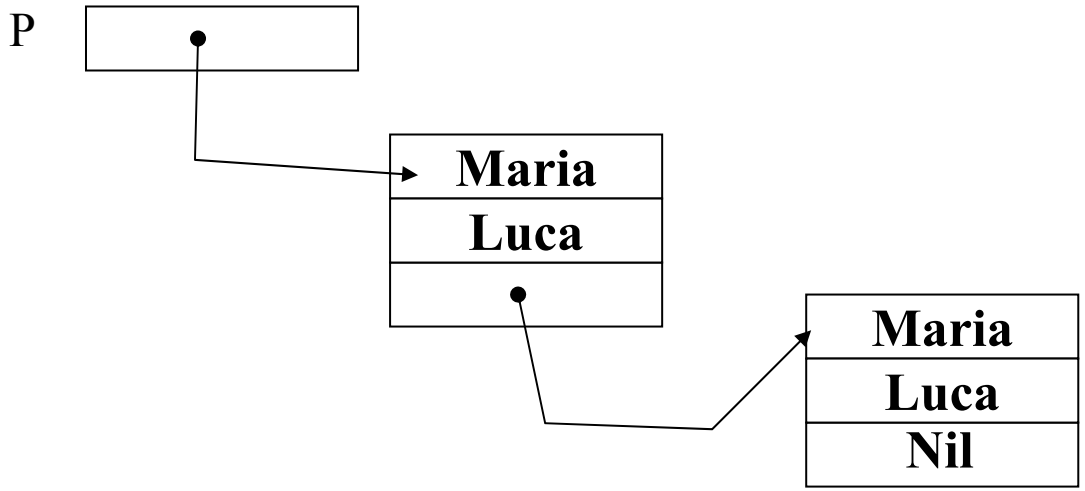
Esempio:

type Tp = ↑ T

Var P :Tp

Begin

New(P)



Esercizio 1

Data una lista di elementi che contengono informazioni relative al nome, cognome e stipendio dei dipendenti del comune di Macerata, scrivere un programma che stampi, in ordine decrescente per valore dello stipendio, l'elenco dei dipendenti.

Scegliere un algoritmo e una struttura dati idonea a risolvere il problema avendo precedentemente fissato alcune ipotesi sulla base di considerazioni fatte sulle caratteristiche del problema.

Quali Scelte fare?

Prima di scrivere il programma che risolva il problema, dobbiamo scegliere l'algoritmo di ordinamento e la struttura dati da utilizzare per memorizzare la lista

Nota sull'algoritmo: se consideriamo di scegliere "InsertionSort" dobbiamo controllare che i dati non siano già ordinati a crescere per stipendio, in quanto saremmo in presenza del caso pessimo $O(n^2)$

Nota sulla struttura dati: la scelta della struttura dati dipenderà dalla variabilità dei dati. L'uso di un array è giustificato se il numero dei dipendenti non subisce incremento o decremento consistente, viceversa è giustificato l'uso di una lista.

Considerazioni

- Quanti dipendenti ci sono al Comune di Macerata?
- Il numero dei dipendenti è soggetto a variazioni consistenti?
- La lista data, è memorizzata in memoria principale o secondaria?
- I dati contenuti nella lista sono correlati?

1ª Ipotesi

- Quanti dipendenti ci sono al Comune di Macerata? 100
- Il numero dei dipendenti è soggetto a variazioni consistenti? No
- La lista data, è memorizzata in memoria principale o secondaria? Memoria principale
- I dati contenuti nella lista sono correlati? Random

Scelte concrete

Algoritmo di ordinamento → *InsertionSort*

Struttura dati → Array di 100 elementi

Utilizzo di due procedure ausiliari

1. Read-array
2. Write-array

Notazione

Per descrivere i programmi delle esercitazioni utilizziamo il Pascal e

- i termini scritti in maiuscolo indicano le parole chiave del linguaggio
- i termini che iniziano con una maiuscola indica qualsiasi identificatori dichiarato nel programma (di tipo, di variabile, di procedura o funzione)

```
PROGRAM Dipendenti;
  CONST Ndip=100;
  TYPE Stringa = ARRAY[1..25] OF CHAR;
       Elemento = RECORD
                    Nome : Stringa
                    Cognome : Stringa
                    Stipendio : INTEGER END;
      Lista = ARRAY[1..Ndip] of Elemento;
  VAR A : Lista,
  BEGIN
   READ(A);
   InsertionSort(A,Ndip)
   WRITE(A);
  END.
```

Domanda

La procedura “InsertionSort” invocata nel programma principale **`Dipendenti`** può la stessa definita nella lezione N.3?

```
InsertionSort(A, n) {
  for i = 2 to n {
     key = A[i]
     j = i - 1;
     while (j > 0) and (A[j] > key) {
          A[j+1] = A[j]
          j = j - 1
     }
     A[j+1] = key
  }
}
```

NO

Perché?

- La procedura “InsertionSort” opera su un array il cui singolo elemento può essere confrontato con qualsiasi altro **`A[i] < key`**
- Il tipo **`Lista`** dichiarato nel programma **`Dipendenti`** ha come elemento base un tipo strutturato, precisamente un record di tre attributi. Quindi il confronto diretto tra due record in questo esempio non ha senso. Inoltre l’ordinamneto richiesto dovrà confrontare solo l’attributo stipendio.

Adattiamo la procedura al programma

```
InsertionSort(A:Lista, N:INTEGER);
 VAR Key: Elemento;
 BEGIN
      FOR i := 2 TO N
       BEGIN
         Key := A[i]
         j := i - 1;
         WHILE (j > 0) AND (A[j].Stipendio >
                key.Stipendio)
          BEGIN
            A[j+1] = A[j]
            j = j - 1
          END
         A[j+1] := key
       END
 END
```

Adattiamo il programma alla procedura

```
PROGRAM Dipendenti;
   CONST Ndip=100;
   TYPE Stringa = ARRAY[1..25] OF CHAR;
         Elemento = RECORD
                  Nome : Stringa
                  Cognome : Stringa
                  Stipendio : INTEGER END;
        Lista = ARRAY[1..Ndip] of Elemento;
        Vettore = ARRAY[1..Ndip] of INTEGER;
   VAR A : Lista, B: Vettore
   BEGIN
    READ(A);
   …
   InsertionSort(B,Ndip)
   …
   WRITE(A);
   END.
```

Il costo computazionale è giustificato?

```
InsertionSort(A:Vettore, N:INTEGER);
 VAR Key: INTEGER;
 BEGIN
         FOR i := 2 TO N
          BEGIN
            Key := A[i]
            j := i - 1;
            WHILE (j > 0) AND (A[j] > key)
             BEGIN
               A[j+1] = A[j]
               j = j - 1
            END
            A[j+1] := key
         END
 END
```

InsertionSort in Pascal

```
InsertionSort(A:Vettore, N:INTEGER);
 VAR Key: INTEGER;
 BEGIN
        FOR i := 2 TO N
         BEGIN
           Key := A[i]
           j := i - 1;
           WHILE (j > 0) AND (A[j] >
   key)
            BEGIN
              A[j+1] = A[j]
              j = j - 1
            END
           A[j+1] := key
         END
 END
```