PROGRAMMAZIONE

I COSTRUTTORI DI TIPO (II)

- **SCHEDA - struct del C (record del Pascal)**
- **PUNTATORI - pointer**

ESEMPI

- Data
  - Giorno, Mese, Anno
- Indirizzo
  - Via, N. civico, CAP, Città, Provincia
- Scheda anagrafica di un cittadino
  - Nome, Cognome, Data di nascita, Stato civile
- Scheda descrittiva di un libro
- Scheda descrittiva di un impiegato

UN NUOVO COSTRUTTORE DI TIPO

- In molte situazioni, il tipo array può non essere adeguato per modellare le entità da trattare
  - Vincolo della omogeneità
- I linguaggi di programmazione offrono un costruttore di tipo per aggregare attributi eterogenei: il tipo **scheda**
  - In C*:* ***struct***
  - In Pascal*:* ***record***

ESEMPI

- Data
  - Giorno, Mese, Anno
  - TIPO **data** SCHEDA

    giorno: 1..31;

    mese: (gen, feb, … , dic);

    anno: 1900..2010

    FINE_SCHEDA

  Dominio: terne (giorno, mese, anno)

  Cardinalità: 31 x 12 x 111 possibili terne

- Indirizzo
  - Via, N. civico, CAP, Città, Provincia
  - TIPO **indirizzo** SCHEDA

    via: stringa(20);

    n.civico: intero;

    cap: stringa(5);

    citta, provincia: stringa(10)

    FINE_SCHEDA

  Dominio: quintuple

  Cardinalità: n.ro dei possibili valori

COSTRUTTORE **Scheda**

- Prodotto cartesiano di *n* insiemi (Modello matematico)
- Aggrega in una *n*-pla di dati le caratteristiche rilevanti di una entità
  - Struttura non lineare (non sono in ordine)
  - A dimensione fissa (componenti prestabilite)
- Insieme di dati non omogenei
- Operazioni consentite
  - Lettura (selezione)
    - Reperimento del valore di un elemento
  - Scrittura (sostituzione)
    - Sostituzione del valore di un elemento con un nuovo valore

SCHEDA
Specifica

- Occorre specificare per ogni singola componente (detta *campo*):
  - Identificatore per accedervi (*selettore*)
    - Ogni campo è denotato ed indirizzato esplicitamente tramite un nome
  - Tipo della componente
    - Limiti di variabilità del valore che può assumere
- Numero fissato di componenti

SCHEDA

scheda *campo1* : *tipo1*;

…

campon : *tipon;*

Card(scheda)= Card(tipo1)x Card(tipo2)x…

…x Card(tipon)

- Accesso a qualunque componente
  - Specifica del nome della variabile scheda seguito dal selettore del campo

    nome_variabile.nome_campo **(dot notation)**
- Un elemento di una scheda può essere a sua volta di un tipo strutturato

STRUCT IN C (1/2)

- *typedef **struct** { int Giorno;*
  int Mese;
  int Anno;
  } Data; / identificatore del tipo */*

  Data Data_prestito, Data_odierna; / variabili */*

  OPPURE

- *struct { int Giorno;*
  int Mese;
  int Anno;
  *} Data_prestito, Data_odierna;/*variabili*/*

STRUCT IN C (2/2)

- Operatori globali
  - Assegnamento
    - Esempio: *Data_prestito = Data_odierna*

- Qualunque altra operazione deve essere eseguita campo per campo
  - Confronto
    - *if (Data_odierna = = Data_prestito)* non consentito
  - Lettura
  - Stampa

SCHEDA
Rappresentazione

- Componenti allocate in posizioni di memoria contigue
- Rispetto dell'ordine di dichiarazione
- Occupazione di memoria complessiva
  - Dipende dall'occupazione di ciascun campo
    - Somma delle occupazioni individuali
    - Le dimensioni dei campi sono note al tempo della compilazione
- Posizione di memoria di un campo
  - Alla posizione iniziale del record vanno sommate le dimensioni dei campi precedenti (***offset***)
  - Esempio: Data_odierna.Mese

SCHEDA

scheda *campo1* : *tipo1*;

…

campon : *tipon;*

indirizzo_campo$_2$*: ind._iniziale* + S_1

……..

Indirizzo campo$_n$ *: ind._iniziale* + $S_1 + S_2 + \ldots S_{n-1}$

S_j: dimensione del *j*-esimo campo

NOTA

Per un array il calcolo dell'indirizzo deve essere fatto a run-time, a[i +k]

ARRAY vs. STRUCT

- Dimensione fissa
- Tipi omogenei

- Sequenza
  - Ordine
- Accesso diretto
  - Indice
    - Uso di espressioni
      - Flessibilità

- Dimensione fissa
- Tipi eterogenei
  - Più generale
- Aggregato

- Accesso diretto
  - Identificatore di campo

ARRAY & STRUCT

- E’ perfettamente legale definire
  - strutture costituite da campi di tipo array
  - array aventi elementi di tipo struct
  - Esempi
    - Registro di classe (Chiave: nominativo dello studente)
    - Orario Ferroviario (Chiave: numero del treno)
    - Tabelle

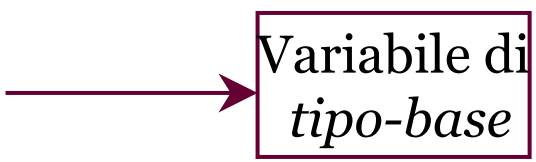
CONCETTO DI PUNTATORE (pointer)

- I puntatori sono variabili atte a contenere indirizzi
  - Utili per implementare il riferimento **indiretto** ad una risorsa
  - Utili per manipolare risorse **dinamiche** (liste, code, pile,…)

TIPO PUNTATORE

- Non è consentito definire variabili che siano puntatori "universali"
  - Occorre specificare il tipo a cui si può puntare
  - Lo spazio di memoria da riservare dipende dal tipo
- Il costruttore di tipo puntatore in C
  - **typedef** *tipo-puntatore* **tipo-base*

  Il tipo *tipo-puntatore* punta a variabili di *tipo-base*
- Si usa rappresentare graficamente il valore di un puntatore tramite una freccia



TIPO PUNTATORE

- Dominio
  - Insieme dei valori degli indirizzi
- Costanti
  - NULL in C (NIL in Pascal) quando non punta a nessun oggetto
  - Valore di inizializzazione "universale"
- Operazioni
  - Operatori unari **&** ed * (sono complementari)
  - Tutte le operazioni aritmetiche consentite sugli indirizzi (tengono conto del tipo di dato a cui fa riferimento il puntatore)
  - Copia (tramite assegnamento tra pointer), Inizializzazione (NULL o 0), Confronto (uguali se puntano alla stessa locazione)

Il compilatore controlla la tipizzazione delle variabili puntate

PUNTATORI

– Tipo esplicito

typedef *tipo-puntatore* *****tipo-base*

tipo-puntatore p;

tipo-base x;

OPPURE

– Direttamente la dichiarativa di variabile di tipo puntatore anonimo

*tipo-base *p*

ESEMPIO

```
int *p1, *p2;       /* puntatori a intero */
float *x;           /* puntatore a float */
char *punt_car; /* puntatore a carattere */
```

OPERATORI (IN C)

- **&** (ampersand) restituisce l'indirizzo di una variabile
- * restituisce il contenuto dell'oggetto puntato (operatore di dereferenziazione)
  - Se una variabile puntatore vale NULL, *p è undefined

```
int a, b, *pa;     /* pa è un puntatore a int */
a = 3;
pa= &a;  /* pa conterrà l'indirizzo di a */
b = *pa;     /* b conterrà 3 */
```

OPERATORI (IN C)
Esempio

- *int a, *pa;*

  pa = &a;

  Se la variabile ***a*** è stata memorizzata nella locazione 2000 e occorrono 2 byte per *int*, allora ***pa +1*** punterà all'indirizzo 2002 (non a 2001!).

NOTA

Sono possibili più livelli di indirizzamento indiretto: **p

ARRAY E PUNTATORI
uno stretto legame…

- Se la variabile riferita ***a*** è un array o una struttura, ***pa*** punta alla prima locazione.
- Il nome di un array in C è sempre associato ad un puntatore "fisso"
  - che punta alla base della zona di memoria in cui sono memorizzati gli elementi dell'array

    &nome_array[0]

ARRAY E PUNTATORI

…uno stretto legame

- In C gli array sono sempre passati attraverso il loro indirizzo (puntatore all'array)
- Le eventuali modifiche all'array effettuate all'interno della funzione sopravvivono all'esecuzione della funzione
- Il compilatore converte tutti i parametri di funzione che rappresentano un vettore mono-dimensionale dalla notazione int v[ ] a quella con puntatore int *v.
- v[i] corrisponde ad *(v+i)

Sono equivalenti:

```
int leggi_vet (int v[ ] , int dim);
int leggi_vet (int *v , int dim);
```

OSSERVAZIONI

- L'uso di un puntatore ad una variabile come parametro di una funzione rende di fatto disponibile alla funzione la memoria allocata per la variabile (legame per riferimento)
  - Ciò permette la comunicazione (in entrambe le direzioni) tra chiamante e chiamato.
- L'uso dei puntatore espone a rischi
  - Effetti collaterali
  - I valori dei puntatori in esecuzione possono cambiare!
- Per rincorrere l'efficienza spesso si abusa nell'uso dei puntatori