Matrici, Struct e Tipi User-Defined

Informatica B AA 2017 / 2018

Giacomo Boracchi

18 Ottobre 2017

giacomo.boracchi@polimi.it

Warm up

- Scrivere un programma che richiede due stringhe all'utente
- Il programma controlla se le due stringhe contengono le stesse vocali nello stesso ordine

- Hint: estrarre, da ogni stringa, una stringa contenente solo le vocali

Richiamo sui Tipi in C

Tipi di dato

- Classificazione sulla base della struttura:
  - **Tipi semplici**, informazione logicamente **indivisibile** (e.g. `int, char, float..`)
  - **Tipi strutturati**: aggregazione di variabili di tipi semplici
    - array

- Altra classificazione:
  - **Built in**, tipi già presenti nel linguaggio base
  - **User defined,** nuovi tipi creati nei programmi «componendo» variabili di tipo built in

Tipi di dato

- Classificazione sulla base della struttura:
  - **Tipi semplici**, informazione logicamente **indivisibile** (e.g. `int, char, float..`)
  - **Tipi strutturati**: aggregazione di variabili di tipi semplici
    - array
    - **enum**
    - **struct**
- Altra classificazione:
  - **Built in**, tipi già presenti nel linguaggio base
  - **User defined,** nuovi tipi creati nei programmi «componendo» variabili di tipo built in

`enum`

- Tipo Enumerativo

Tipi Enumerativi

- Rappresentano un alternativa alla dichiarazione di costanti intere

```
enum{nome1, nome2,…, nomeN};
```

avremo N variabili intere e a ciascuna sarà assegnato un valore da **0** a **N-1**

- Esempio:

```
enum{falso, vero};
```

definisce due costanti intere **falso=0** e **vero=1** e nel codice posso far riferimento a **falso** e **vero** come se fossero costanti.

Tipi Enumerativi: a che servono?

- Servono per facilitare lo sviluppo del codice e per far riferimento a varibili qualitative (associando dei valori quantitativi).
- *Esempio*

```
enum{quardi,cuori,fiori,spade};

enum{biondi,neri,castani,rossi,bianchi};
```

- Posso assegnare valori arbitrari agli elementi della **enum**.

```
enum{lunedi=1, martedi=2,…, domenica=7};
```

- Se i valori sono sequenziali basta specificare il primo

```
enum{lunedi=1, martedi ,…, domenica};
```

Tipi Enumerativi: come li uso?

- Vengono usati come una qualsiasi costante intera:
  - espressioni aritmetiche
  - espressioni logiche

Tipi Enumerativi: come li uso?

- Vengono usati come una qualsiasi costante intera:
  - espressioni aritmetiche
  - espressioni logiche
- Ha senso:

```
enum{falso, vero};
int a = 6;
if ((a>0) == vero)
        printf("\n a e' positiva");
```

Tipi Enumerativi: come li uso?

- Vengono usati come una qualsiasi costante intera:
  - espressioni aritmetiche
  - espressioni logiche

- Ha senso:

```
enum{falso, vero};
int a = 6;
if ((a>0) == vero)
        printf("\n a e' positiva");

enum{lunedì=1, martedì=2,…, domenica=7};
if (martedi == lunedi + 1)
        printf("\n giorni consecutivi");
if (martedi < giovedi)
        printf("\n precede");
```

`struct`

- Tipi strutturati

Struct vs Array

- Gli **array** permettono di aggregare variabili **omogenee** in una sequenza
- Le `struct` permettono di aggregare variabili **eterogenee** in una sola variabile
  - Le `struct` è una sorta di "contenitore" per variabili disomogenee di tipi più semplici.
  - Le variabili aggregate nella struct sono dette **campi** della struct
- *Esempio: variabile per contenere anagrafica di impiegati*
  - *nome, cognome, codice fiscale, indirizzo, numero di telefono, stipendio, data di assunzione etc.*
  - *Non posso metterli in un array, sono variabili diverse, è molto sconveniente metterle in variabili separate, specialmente se ho diversi impiegati*

Dichiarazione di una Struttura

- Sintassi:

```
struct {
 tipo1 nomeCampo1;
 tipo2 nomeCampo2;
 …
 tipoN nomeCampoN;} nomeStruct;
```

- Dichiara una variabile **struct** chiamata **nomeStruct**
- I nomi dei campi della struttura sono **nomeCampo1**…
- Dichiarazione compatta per campi dello stesso tipo

```
struct {
 tipo1 nomeCampoA,nomeCampoB;
…
tipoN campoN;} nomeStruct;
```

Dichiarazione di una Struttura

- È possibile dichiarare due o più variabili dalla stessa struttura

```
struct {tipo1 nomeCampo1;
tipo2 nomeCampo2;
…
tipoN nomeCampoN;}nomeStruct1, nomestruct2;
```

- **NB:** la dichiarazione di una struttura va nella **parte dichiarativa** del programma, **nel `main()`**
- **NB**: i campi **non** sono **necessariamente** di **tipo built-in**, possono essere array o user defined (vedremo a breve)

Esempi

```
struct {
      float reale;
      float immaginaria;
      } numeroComplesso;


struct {
      int numero;
      char seme[10];
      } cartaDaGioco;
```

Esempi

```
struct {
      char Nome[30];
      char Cognome[30];
      int Stipendio;
      char CodiceFiscale[16];
      } dip1, dip2;

struct
{
   char marca[30];
   char modello[100];
   int anno;
   int cilindrata;
   int prezzo;
} miaAuto, tuaAuto;
```

Accedere ai campi di una `struct`

- Per accedere ai campi si usa l'operatore ***dot*** (i.e., il punto)
  Sintassi:

```
nomeStruct.nomeCampo;
```

- Quindi, **`nomeStruct.nomeCampo`** diventa, a tutti gli effetti, **una «normale» variabile** del tipo di **`nomeCampo`**.
  - **Ai campi** di una struttura applicabili tutte le **operazioni caratteristiche** del tipo di appartenenza
  - In questo senso, il *dot* è l'omologo di **`[indice]`** per gli array

Esempio

```
struct {char nome[30];
        char cognome[30];
        int stipendio;
        char codFiscale[16];
        } dip1, dip2;
// accedere ai campi di tipo semplice
dip1.stipendio = 30000;
dip2.stipendio = 2*(dip1.stipendio - 2000);
// accedere ai campi array
dip1.codiceFiscale[0] = 'K';
// copia del valore da un campo array all'altro
for(i = 0 ; i < 16 ; i++)
        dip2.codFiscale[i]=dip1.codFiscale[i];
// copia il nome di un dipendente nell'altro
strcpy(dip2.nome, dip1.nome);
dip1.cognome = dip2.cognome; // sbagliato!
```

Acquisizione e Stampa per Strutture

- Non esistono caratteri speciali che permettano di usare **`printf`** e **`scanf`** direttamente su strutture.
- Occorre lavorare campo per campo!

```
struct {char nome[30];
char cognome[30];
int stipendio;
} dip1;
printf("\nInserire Nome 1: ");
scanf("%s", dip1.nome);
printf("\nInserire Cognome 1: ");
scanf("%s", dip1.cognome);
printf("\nInserire Stipendio 1: ");
scanf("%d", &dip1.stipendio);
printf("%s %s, guadagna %d $",
dip1.nome,dip1.cognome,dip1.stipendio);
```

Esempio

Definire una struttura atta a contenere una data (con mese testuale) e dichiarare due variabili **`dataNascita`** e **`dataLaurea`**.

1. Richiedere all'utente l'inserimento della data di nascita
2. Visualizzare a schermo la data di nascita
3. Definire la presunta data di laurea come
   - Giorno = giorno della nascita
   - Mese = mese della nascita
   - Anno = all'età di 24 anni
4. Stampare la presunta data di laurea

Esempio

```
#include<stdio.h>
void main()
{struct {
        int giorno;
        char mese[20];
        int anno;} N, L;
 printf("\nInserire giorno");
 scanf("%d", &N.giorno);
 printf("\nInserire mese");
 scanf("%s", N.mese);
 printf("\nInserire anno");
 scanf("%d", &N.anno);
 printf("Nato il %d  %s %d",N.giorno, N.mese, N.anno);
 L.giorno = N.giorno;
 strcpy(L.mese, N.mese);
 L.anno = N.anno + 24;
 printf("\nTi laurerai il %d  %s %d",L.giorno, L.mese,
 L.anno);}
```

Assegnamento tra Strutture

È possibile applicare **operazioni globali di assegnamento** tra **strutture identiche**.

```
struct {
    char nome[30];
    char cognome[30];
    int  stipendio;
    char codiceFiscale[16];
    } dip1, dip2;

dip1 = dip2;
```

Con l'assegnamento globale anche i valori nei campi di tipo array vengono copiati

Assegnamento tra Strutture

- **L'assegnamento** è possibile **solo se** la **strutture sono identiche**, se cambia anche solo l'ordinamento dei campi non è possibile.
- L'assegnamento globale **NON** è possibile con gli **array**
  - Però, campi di strutture identiche che sono array (come nel caso di **`dip1`** e **`dip2`**) vengono assegnati correttamente!
- Anche per struct, come per array, **NON** applicabili operazioni di **confronto** (**`==`**, **`!=`**)

Esempio

```
#include<stdio.h>
void main()
{struct {
        int giorno;
        char mese[20];
        int anno;} N, L;
 printf("\nInserire giorno");
 scanf("%d", &N.giorno);
 printf("\nInserire mese");
 scanf("%s", N.mese);
 printf("\nInserire anno");
 scanf("%d", &N.anno);
 printf("Nato il %d  %s %d",N.giorno, N.mese, N.anno);
 L = N;
 L.anno += 24;
 printf("\nTi laurerai il %d  %s %d",L.giorno, L.mese,
  L.anno);}
```

Assegnamento globale, possibile solo se **L** ed **N** sono strutture identiche.

Esempio

```
#include<stdio.h>
void main()
{struct {
        int giorno;
        char mese[20];
        int anno;} N, L;
 printf("\nInserire giorno");
 scanf("%d", &N.giorno);
 printf("\nInserire mese");
 scanf("%s", N.mese);
 printf("\nInserire anno");
 scanf("%d", &N.anno);
 printf("Nato il %d  %s %d",N.giorno, N.mese, N.anno);
 L = N;
 L.anno += 24;
 strcpy(L.mese, "dicembre\0");
 printf("\nTi laurerai il %d  %s %d",L.giorno, L.mese,
  L.anno);}
```

Nel caso volessi cambiare il mese non posso fare assegnamento tra stringhe ma devo ricorrere ad una strcpy

Tipi di Dato User-Defined

- Definire nuovi tipi

Tipi di dato

- Classificazione sulla base della struttura:
  - **Tipi semplici**, informazione logicamente **indivisibile** (e.g. `int, char, float..`)
  - **Tipi strutturati**: aggregazione di variabili di tipi semplici
    - array
    - enum
    - struct
- Altra classificazione:
  - **Built in**, tipi già presenti nel linguaggio base
  - **User defined,** nuovi tipi creati nei programmi «componendo» variabili di tipo built in

Nuovi tipi

- La keyword **`typedef`** permette di definire nuovi tipi in C
- Sintassi:

```
typedef nomeTipo NuovoNomeTipo;
```

- Es:

```
typedef int Anno;
typedef unsigned int TempAssoluta;
typedef unsigned int Eta;
```

- È possibile dichiarare nuovi tipi per
  - Un **tipo semplice** (**ridefinizione di tipo**)
  - Un **tipo strutturato**
- **NB** La dichiarazione di nuovi tipi va **prima** di **`void main()`**, nel corpo del **`main`** potrò dichiarare variabili utilizzando **`NuovoNomeTipo`** con la solita sintassi

Definizione di Nuovi Tipi Strutturati

- Se si combina **typedef** con un costruttore **struct** o **array** i vantaggi diventano più evidenti.

```
typedef struct {int giorno;
                char mese[20];
                int anno;} Data;
```

- Quando si associa un nuovo tipo ad una struttura è possibile:
  1. dichiarare **altre strutture** come variabili del nuovo tipo
  2. dichiarare **array** di strutture come array del nuovo tipo
  3. utilizzare il nuovo tipo come **campo** di altre **strutture**
  4. utilizzare il nuovo tipo come **tipo base per nuovi tipi**

Definizione di Nuovi Tipi Strutturati

- Dichiarare **altre strutture** (i.e., variabili del nuovo tipo)

```
Data oggi, domani, dopoDomani;
```

Definizione di Nuovi Tipi Strutturati

- Dichiarare **altre strutture** (i.e., variabili del nuovo tipo)

```
Data oggi, domani, dopoDomani;
```

- Dichiarare **array** del **nuovo tipo** (i.e., array di strutture)

```
Data calendario[365];
Data settimana[7];
Data andataRitorno[2];
// popolare andataRitorno[0] per l'andata
andataRitorno[0].giorno = 12;
strcpy(andataRitorno[0].mese, "dicembre");
andataRitorno[0].anno = 2012;
// ritorno è come l'andata
andataRitorno[1] = andataRitorno[0];
// posticipo di 10 giorni il ritorno
andataRitorno[1].giorno += 10;
```

Definizione di Nuovi Tipi Strutturati

- Utilizzare il nuovo tipo come **campo** di altre **strutture**

```
struct {   char nome[30];
           char cognome[30];
           int stipendio;
           char codiceFiscale[16];
           Data dataDiNascita;} dip1;
```

Definizione di Nuovi Tipi Strutturati

- Utilizzare il nuovo tipo come **campo** di altre **strutture**

```
struct {   char nome[30];
           char cognome[30];
           int stipendio;
           char codiceFiscale[16];
           Data dataDiNascita;} dip1;
```

- Utilizzare il nuovo tipo come **tipo base per nuovi tipi**

```
typedef struct {char nome[30];
        char cognome[30];
        int stipendio;
        char codiceFiscale[16];
        Data dataDiNascita;
        } Dipendente;
```

Definizione di Nuovi Tipi da Array

Posso definire un nuovo tipo per variabili array

```
typedef double PioggeMensili[12];

PioggeMensili piogge87, piogge88, piogge89;

typedef double IndiciBorsa[12];

IndiciBorsa  indici87, indici88, indici89;
```

È più comprensibile dell'omologo senza definizione di tipo

```
double  piogge87[12], piogge88[12],
piogge89[12],

double indici87[12], indici88[12],
indici89[12];
```

Definizione di Nuovi Tipi da Array

Altro esempio classico

```
typedef char Stringa[30];
```

A questo punto posso

```
typedef struct {
     Stringa nome;
     Stringa cognome;
     int stipendio;
     Stringa codFiscale;
     Data dataNascita;
     } Dipendente
```

Al posto di :

```
typedef struct {
 char nome[30];
 char cognome[30];
 int stipendio;
 char codiceFiscale[30];
 Data dataNascita;
     } Dipendente
```

È possibile dichiarare tipi used-defined a partire da altri tipi user-defined

Definizione di Nuovi Tipi da `enum`

- È anche possibile

```
typedef enum{gennaio = 1,…, dicembre} Mese;
```

- È quindi possibile dichiarare variabili di tipo **Mese**

```
Mese meseCorrente;
```

```
typedef struct {int giorno;
                Mese mese;
                int anno;} Data;
```

- **NB** In questo caso la dichiarazione riguarda **una variabile** che assumerà solo i valori ammissibili nella **enum**

Ridefinizione di Tipi Semplici: a che serve?

- Rende più leggibile e generale il codice.
- *Es* `typedef float MieiDati;`

  Se dichiaro tutte le variabili pensate per contenere i dati di tipo **`MieiDati`** il programma è facilmente estendibile a gestire dati a precisione maggiore. Basterà sostituire

  ```
  typedef double MieiDati;
  ```

- *Es* `typedef unsigned int TempAssoluta;`

  Usare **`TempAssoluta`** per dichiarare una variabile rende il codice più leggibile.

Una Buona Regola

- Utilizzare notazioni differenti per i tipi e per le variabili
- Ad esempio:
  - I tipi user defined iniziano con la lettera maiuscola, le variabili con la lettera minuscola

```
typedef char Stringa[30];
Stringa stringa;
```

tipo variabile

  - Usare un prefisso/suffisso per i tipi, ad esempio

```
typedef char stringa_t[30];
stringa_t stringa;
```

tipo variabile

Assegnamento tra Variabili di Tipo User-Defined

Valgono le **linee guida per l'assegnamento globale per struct e per array**:

- **NON** è possibile l'assegnamento tra due variabili dello stesso tipo quando sono **array**
- **È possibile** associare variabili dello stesso tipo se queste sono di tipo **struct** (anche se contengono array nei loro campi)
- **Non** è possibile eseguire **conversioni intrinseche** tra tipi definiti dall'utente (come avviene tra i tipi built in)

Esercizio

1. Utilizzare i nuovi tipi di dato recentemente utilizzati per definire un nuovo tipo atto a descrivere un libro (con autore, titolo, costo, data di pubblicazione) ed uno scaffale di libri.
2. Scrivere un frammento di codice in cui si esegue l'acquisizione dei dati relativi ad un libro e quindi popolare uno scaffale.
3. Stampare i dati relativi a tutti i libri presenti sullo scaffale.
4. Scrivere un frammento di codice che
   a. Calcola il costo di tutti i libri presenti sullo scaffale (assumendo non vi siano più copie dello stesso libro)
   b. Copia tutti i libri con autore che ha il cognome che inizia per 'B' in una seconda variabile di tipo scaffale.

Array Multidimensionali

- Matrici

Esempio Acquisizione di una Matrice

```
for(i = 0; i < r; i++)
      for(j = 0; j < c ; j++)
      {
            printf("Inserire elemento posizione
                  [%d][%d]" , i+1 , j+1);
            scanf("%d", &M[i][j]);
      }
```

Stampa di una Matrice

```
for(i = 0; i < r; i++)
      {
            for(j = 0; j < c ; j++)
                printf("%5d" , M[i][j]);
            printf("\n");
      }
```

TODO: Esercizio su Matrici

- Si scriva un programma che prende in ingresso una matrice quadrata e controlla che sia simmetrica.
- Una matrice è simmetrica se per ogni elemento vale la seguente proprietà: L'elemento alla riga i, colonna j coincide con l'elemento alla riga j colonna i
- *Es di matrice simmetrica*

1	12	1
12	0	3
1	3	23

TODO: Esercizio su Matrici

- Scrivere un programma che richiede l'inserimento di una matrice di interi **`M`** e di un intero **`n`**.
- Il programma conta il numero di occorrenze di **`n`** in ogni riga di **`M`**.
- Il programma stampa con un istogramma (verticale) il numero di occorrenze di **`n`** per ogni riga di **`M`**.

Array Multidimensionali

- Osservazioni

Array di Array

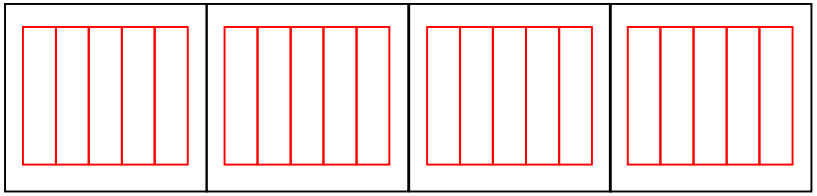
Gli elementi degli array possono essere di qualsiasi tipo (predefinito, definito dall'utente, semplice o strutturato)

⇒ **è possibile costruire "array di array"**

Esempio:

```
typedef  int Vettore[5];
```

```
typedef  Vettore Matrice4Per5[4];
```



```
Matrice4Per5 matrice1;
```

Array di Array

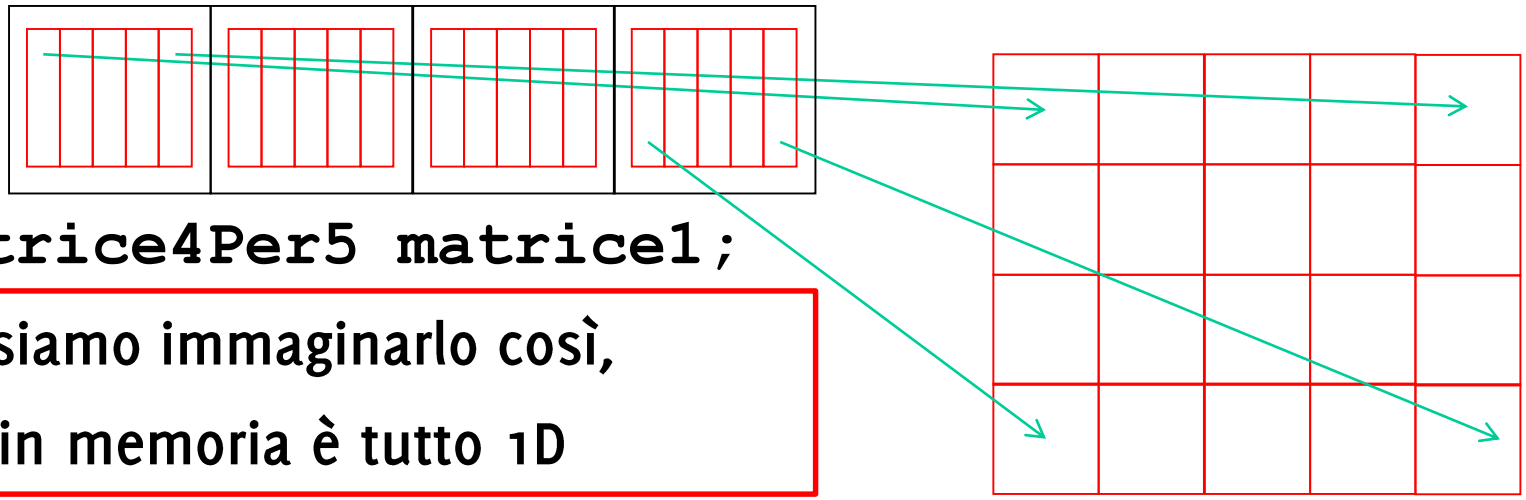
Gli elementi degli array possono essere di qualsiasi tipo (predefinito, definito dall'utente, semplice o strutturato)

⇒ **è possibile costruire "array di array"**

Esempio:

```
typedef   int Vettore[5];
```

```
typedef   Vettore Matrice4Per5[4];
```



```
Matrice4Per5 matrice1;
```

Possiamo immaginarlo così,

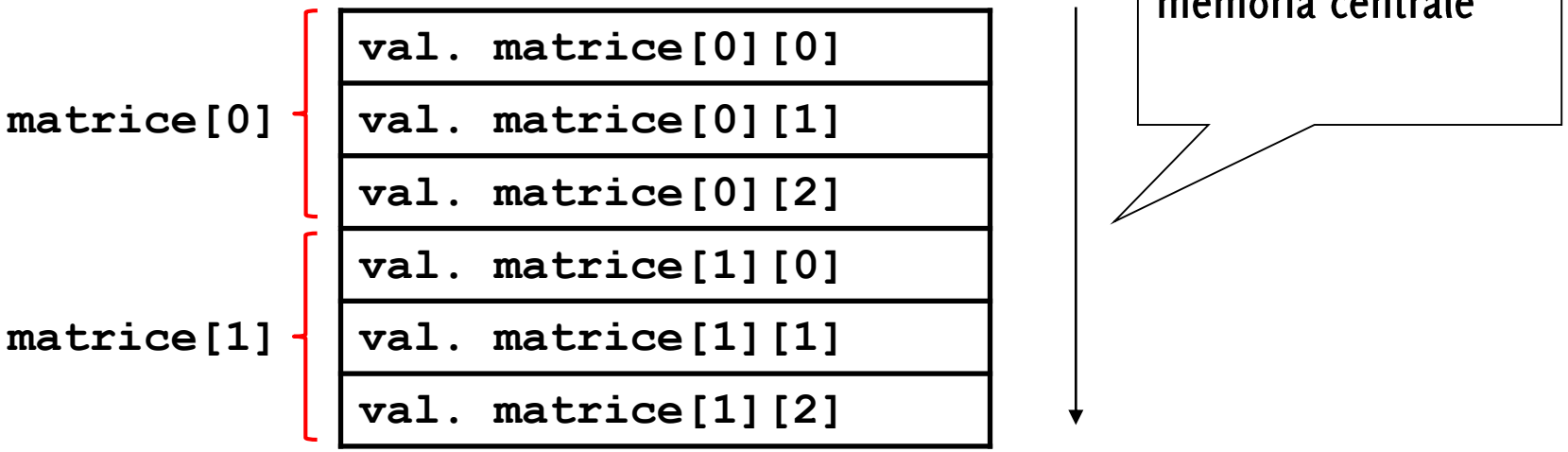
ma in memoria è tutto 1D

Mappa di Memorizzazione di un array 2D

Cioè la rappresentazione in memoria di un array a 2 dimensioni:

- array memorizzato riga per riga, per indice di riga crescente, e, all'interno di ogni riga, per indice di colonna crescente

```
int   matrice [2][3];
```



Mappa di Memorizzazione di un array 2D

- Data la seguente dichiarazione di un vettore

```
int  vett [Dim1];
```

Vale la seguente uguaglianza

```
&vett[i] == &vett[0] + i
```

definiscono entrambe l'indirizzo dell'elemento **i**-simo

- Per le matrici invece, data

```
int  matrice[Dim2][Dim1];
```

Vale la seguente uguaglianza

```
&matrice[i][j]== &matrice[0][0]+ i *Dim1 +j
```

definiscono entrambe l'indirizzo dell'elemento alla riga **i** e colonna **j** in **matrice**

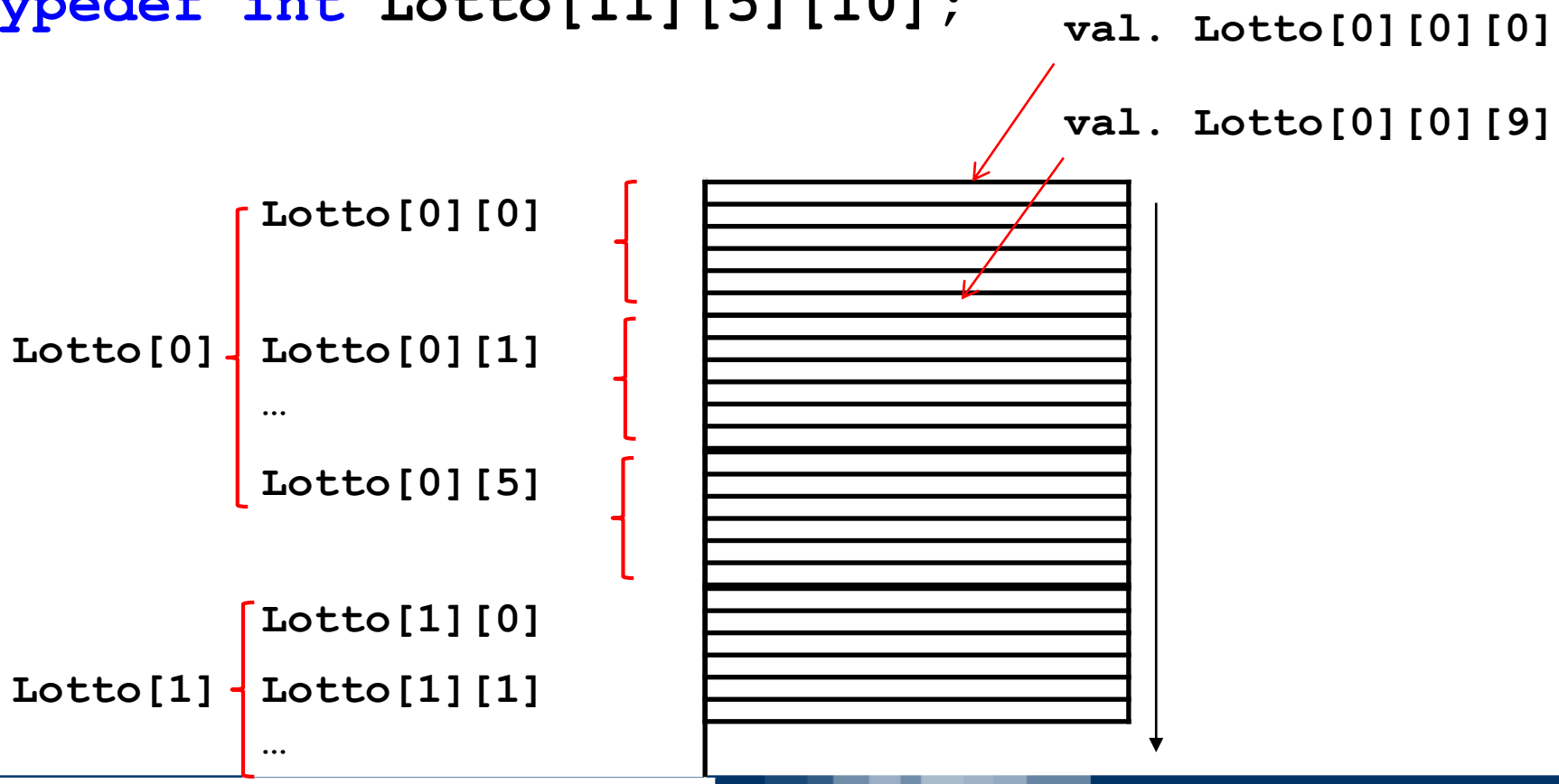
Mappa di Memorizzazione di un array 3D

Definire un tipo di dato atto a contenere i numeri estratti nelle ultime 10 giocate su tutte le 11 ruote (vengono estratti 5 numeri per giocata)

Mappa di Memorizzazione di un array 3D

Definire un tipo di dato atto a contenere i numeri estratti nelle ultime 10 giocate su tutte le 11 ruote (vengono estratti 5 numeri per giocata)

```
typedef int Lotto[11][5][10];
```



Matrici: Array di Array

- dichiarazione più sintetica e per la variabile **M**

```
int M[4][5];
```

- Quindi per il tipo **Matrice4Per5**

```
typedef int Matrice4Per5[4][5];
Matrice4Per5 M;
```

- assegnamento a un elemento della matrice:

```
M[1][3]=7;
```

- *Esempio:* matrice tridimensionale:

```
int matrice3D[10][20][30];
```

per accedere agli elementi di **matrice3D**

```
matrice3D [2][8][15]
```

Conversioni

Compatibilità e Conversioni

- Criterio adottato da C per espressioni e assegnamenti:
  - se costanti e variabili dello **stesso tipo**, applica **operazione associata a quel tipo**
  - se di **tipi diversi** applica, se possibile **regole di conversione**
- In questo corso tratteremo solo regole di **conversione implicita**, non quelle di conversione esplicita (casting).
  - La conversione viene eseguita dal compilatore.
  - Ci sono casi in cui non è possibile eseguire conversioni implicite, ad esempio quando provo ad assegnare una variabile strutturata ad una non strutturata o variabili strutturate differenti.

Espressioni Aritmetiche tra Elementi Eterogenei

- Ogni espressione aritmetica è caratterizzata da:
  - **valore del risultato**
  - **tipo del risultato**
- Il **tipo** degli **operandi determina l'operazione** eseguita:
  - se x ed y sono dello stesso tipo, nessun problema
  - se x ed y sono di tipo diverso,
    - valuto se è possibile la conversione implicita
    - se non è possibile la conversione implicita l'operazione non è eseguibile
- La fattibilità delle operazioni è valutata a compile-time: **tipizzazione forte** del C

Espressioni Aritmetiche: Conversione Implicita

- Espressioni del tipo

  `x op y`

  con **`x`** e **`y`** di tipi diversi

- in C i tipi sono ordinati in base alla precisione

`char < short < int < long < float < double < long double`

- **Regola Generale**
  1. I valori del tipo meno preciso sono implicitamente convertiti a valore del tipo più preciso (**promozione**).
  2. L'operazione è definita dal tipo più preciso.
  3. Il risultato è un valore del tipo più preciso.

Esempio

Esempio:

```
short x; int y,z;
```

Cosa avviene quando valuto l'espressione `x + y` ?

- Il valore di `x` viene convertito **temporaneamente** in `int`
- viene applicata operazione di somma tra `int`
- Il risultato è `int`

NB la variabile `x` continua a restare `short`. La **promozione riguarda solamente il valore letto, non viene quindi assegnato il valore "promosso" alla variabile**

NB. conversioni, terreno scivoloso ... attenzione a quanto accade con gli `unsigned` ($\Rightarrow$ evitarli)

Assegnamenti: Conversione Implicita

Lo stesso ordinamento in base alla precisione

`char < short < int < long < float < double < long double`

viene utilizzato per assegnamenti tra variabili eterogenee

Esempi

- Siano **`double d; int i;`**
- **`d = i;`** valore di **`i`** convertito a **`double`** e assegnato a **`d`**
  $\Rightarrow$ non c'è perdita di informazione
- **`i = d;`** **`d`** convertito a **`int`** (troncandolo), valore intero assegnato a **`i`** $\Rightarrow$ perdita informazione

Esempio

```
// conversione da gradi Fahrenheit a Celsius
#include<stdio.h>
void main()
{
  int  Ftemp;
  float Ctemp;
  printf("Inserire gradi    Fahrenheit\n");
  scanf("%d", &Ftemp);

  Ctemp = (5.0 / 9.0 ) * (Ftemp - 32);

  printf("Celsius %2.2f" , Ctemp);
}
```

Cosa succede? `Ftemp` è `int` `Ctemp` è `float`

Cosa stampano le seguenti istruzioni se **`Ftemp = 50`**?

1. **`Ctemp = (5.0 / 9.0) * (Ftemp - 32);`**
2. **`Ctemp = (5 / 9) * (Ftemp - 32);`**
3. **`Ctemp = (5.0 / 9) * (Ftemp - 32);`**
4. **`Ctemp = 1.0 * (5 / 9) * (Ftemp - 32);`**
5. Se avessi dichiarato in **`int Ctemp`** (con **`%d`** in **`printf`**)?

Cosa succede? `Ftemp` è `int` `Ctemp` è `float`

Cosa stampano le seguenti istruzioni se `Ftemp = 50`?

1. `Ctemp = (5.0 / 9.0) * (Ftemp - 32);`
2. `Ctemp = (5 / 9) * (Ftemp - 32);`
3. `Ctemp = (5.0 / 9) * (Ftemp - 32);`
4. `Ctemp = 1.0 * (5 / 9) * (Ftemp - 32);`
5. Se avessi dichiarato in `int Ctemp` (con `%d` in `printf`)?

1) `Celsius = 10.0` (`5.0` e `9.0` sono `float` ⇒ `/` è divisione tra `float`, `Ftemp` variabile `int` e `32` e costante `int` ⇒ risultato sottrazione `int` ma la moltiplicazione causa conversione implicita (promozione) di (`Ftemp - 32`) a `float`)
2) `Celsius = 0` (`5` e `9` sono `int` ⇒ `/` diventa divisione tra `int`)
3) `Celsius = 10.0` (`5.0` è `float`, `9` è promosso a `float`).
4) `Celsius = 0` (`1.0` viene moltiplicato per `0`)
5) `Celsius = 10` (risultato `float` assegnato alla `Ctemp`, che è `int` ⇒ possibile perdita di informazione)

Qualche esercizio

- Preso dai TDE

Esercizio (TDE 07/08)

Le seguenti dichiarazioni definiscono tipi di dati relativi alla categoria degli impiegati di un'azienda (gli impiegati possono essere di prima, seconda, …, quinta categoria), agli uffici occupati da tali impiegati, all'edificio che ospita tali uffici (l'edificio è diviso in 20 piani ognuno con 40 uffici).

Esercizio (TDE 07/08)

```
/* definizioni dei tipi */
typedef struct { char nome[20], cognome[20];
      int cat; // contiene valori tra 1 e 5
      int stipendio;
} Impiegato;

typedef enum {nord, nordEst, est, sudEst, sud,
sudOvest, ovest, nordOvest} Esposizione;

typedef struct { int superficie; /*in m^2*/
      Esposizione esp;
      Impiegato occupante;
} Ufficio;

/* definizioni delle variabili */
Ufficio torre[20][40]; /* rappresenta un
edificio di 20 piani con 40 uffici per piano */
```

Esercizio

Si scriva un frammento di codice, che includa eventualmente anche le dichiarazioni di ulteriori variabili, che, per tutte e sole le persone che occupano **un ufficio** (tra quelli memorizzati nella variabile torre) **orientato a sud oppure a sudEst e** avente una **superficie compresa tra 20 e 30 metri quadri**, stampi il cognome, lo stipendio e la categoria.

Soluzione punto 1

```
int p, u; /* indice di piano nell'edificio e di
ufficio nel piano */
for (p=0; p<20; p++)
   for (u=0; u<40; u++)
      if (( torre[p][u].esp == sudEst ||
      torre[p][u].esp == sud) &&
       (torre[p][u].superficie >=20 &&
      torre[p][u].superficie<=30))
      {
      printf("\n il Signor %s è impiegato di
      categoria %d",
      torre[p][u].occupante.cognome,
      torre[p][u].occupante.cat);
       printf (" e ha uno stipendio pari a %d euro
      \n", torre[p][u].occupante.stipendio);
   }
```

Soluzione punto 1

```
int p, u; /* indice di piano nell'edificio e di
ufficio nel piano */
for (p=0; p<20; p++)
   for (u=0; u<40; u++)
      if ((torre[p][u].esp == sudEst ||
      torre[p][u].esp == sud) &&
       (torre[p][u].superficie >=20 &&
      torre[p][u].superficie<=30))
       {
      printf("\n il Signor %s è impiegato di
      categoria %d",
      torre[p][u].occupante.cognome,
      torre[p][u].occupante.cat);
       printf (" e ha uno stipendio pari a %d euro
      \n", torre[p][u].occupante.stipendio);
   }
```

Scorro l'array torre come una normale matrice

Soluzione punto 1

```
int p, u; /* indice di piano nell'edificio e di
ufficio nel piano */
for (p=0; p<20; p++)
   for (u=0; u<40; u++)
      if (( torre[p][u].esp == sudEst ||
      torre[p][u].esp == sud)  &&
       (torre[p][u].superficie >=20 &&
      torre[p][u].superficie<=30))
       {
      printf("\n il Signor %s è impiegato di
      categoria %d",
      torre[p][u].occupante.cognome,
      torre[p][u].occupante.cat);
       printf (" e ha uno stipendio pari a %d euro
      \n", torre[p][u].occupante.stipendio);
   }
```

Non è == "sudEst", perchè è una enum!

Esercizio

- Si scriva un frammento di codice, che includa eventualmente anche le dichiarazioni di ulteriori variabili, che, per tutte e sole le persone che occupano **un ufficio** (tra quelli memorizzati nella variabile torre) **orientato a sud oppure a sudEst e** avente una **superficie compresa tra 20 e 30 metri quadri**, stampi il cognome, lo stipendio e la categoria.
- [aggiunto] Visualizzi a schermo i numeri dei piani che non hanno neanche un ufficio esposto a nord.

Soluzione punto 2

```
int uffNord; /* uffNord fa da flag*/
for (p=0; p<20; p++)
{
      uffNord = 0;//flag = 0 in ogni piano
      for (u=0; u<40 && uffNord == 0; u++)
            if(torre[p][u].esposizione == nord)
                  uffNord = 1;
/* se qui vale ancora 0 vuol dire che non ci
sono uffici a nord*/
      if (uffNord == 0);
            printf("il piano %d  non ha edifici
esposti a nord", p);
}
```

Soluzione punto 2

```
int uffNord; /* uffNord fa da flag*/
for (p=0; p<20; p++)
{
     uffNord = 0;//flag = 0 in ogni piano
      for (u=0; u<40 && uffNord == 0; u++)
           if(torre[p][u].esposizione == nord)
                uffNord = 1;
/* se qui vale ancora 0 vuol dire che non ci
sono uffici a nord*/
      if (uffNord == 0);
           printf("il piano %d  non ha edifici
esposti a nord", p);
}
```

Sto usando una variabile di flag per vedere se non ci sono uffici che affacciano a nord

Esercizio

- Si scriva un frammento di codice, che includa eventualmente anche le dichiarazioni di ulteriori variabili, che, per tutte e sole le persone che occupano **un ufficio** (tra quelli memorizzati nella variabile torre) **orientato a sud oppure a sudEst e** avente una **superficie compresa tra 20 e 30 metri quadri**, stampi il cognome, lo stipendio e la categoria.
- [aggiunto] Visualizzi a schermo i numeri dei piani che non hanno neanche un ufficio esposto a nord
- **[aggiunto] Dire in che piano ed in che ufficio si trova Boracchi**
- **[aggiunto] Copiare in un array tutti gli uffici occupati da dipendenti di categoria 5**
- **[aggiunto] Copiare in un array tutti i dipendenti di categoria 5**

Esercizio

- [aggiunto] Come modificare le strutture dati (ed i codici) precedenti per rappresentare un edificio avente un diverso numero (max 40) di uffici per piano?

Esempio Matrici (TDE 2008)

Si scriva un programma C che acquisisce una matrice di interi di nome matr e di dimensione NxN (con N definito come costante) e due interi X e Y da standard input.

Se X e Y non sono indici ammissibili della matrice, il programma termina.

In caso contrario, il programma stampa la stringa “successo!” se l’elemento matr[X][Y] è uguale a X*Y.

Se quest’ultima condizione non è verificata, il programma considera gli elementi della riga X in matr e controlla se tra questi quelli maggiori di matr[X][Y] sono di più di quelli minori di matr[X][Y]. Se questo è il caso, il programma stampa il valore contenuto in matr[X][Y].

Ad esempio, se la riga X di matr è costituita dagli elementi {12, 7, 15, 5, 3} e l’elemento che stiamo considerando è quello di posizione 1 (il valore 7), possiamo concludere che due elementi della riga sono minori di tale numero e altri due sono maggiori di esso. Di conseguenza, la condizione non è verificata e quindi il programma non stampa nulla.