Matrici, Struct e Tipi User-Defined

Informatica B a.a. 2021/2022

Francesco Trovò

francesco1.trovo@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**, ad esempio `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

Tipi Enumerativi

`enum`

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**

Es. `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 Cosa Servono?

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

Es.

```
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

Es.

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

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

Dati Strutturati

`struct`

Struct vs Array

- Gli **array** permettono di aggregare variabili **omogenee** in una sequenza
- Le `struct` permettono di aggregare variabili **eterogenee** in una sola variabile:
  - La `struct` è una sorta di «contenitore» per variabili disomogenee di tipi più semplici
  - Le variabili aggregate nella `struct` sono dette **campi** della `struct`

Es. variabile per contenere anagrafica di impiegati, ovvero che contiene nome, codice fiscale, stipendio, data di assunzione

NB: Non posso metterli in un array, sono variabili diverse ed è molto scomodo metterle in variabili separate

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, ..., nomeCampoN
```

- 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, ovvero all'interno del **`main()`**

NB: i campi **non** sono **necessariamente** di **tipo built-in**, possono essere array o user defined

Esempi

Numero Immaginario

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

Carta da gioco

```
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**, ovvero il punto

Sintassi:

```
nomeStruct.nomeCampo;
```

- Quindi, **`nomeStruct.nomeCampo`** diventa, a tutti gli effetti, **una variabile** del tipo di **`nomeCampo`**
- **Ai campi** di una struttura posso applicare tutte le **operazioni caratteristiche** del tipo di appartenenza
- In questo senso, l'operatore **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 primo elemento di di un campo 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 acquisirle 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);
```

Esercizio

- Dichiarare una struttura atta a contenere una data (con mese testuale) e dichiarare due variabili **`dataNascita`** e **`dataLaurea`**
  - Richiedere all'utente l'inserimento della data di nascita
  - Visualizzare a schermo la data di nascita
  - Definire la presunta data di laurea come
    - Giorno = giorno della nascita
    - Mese = mese della nascita
    - Anno = all'età di 24 anni
  - Stampare la presunta data di laurea

Soluzione

```
#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

NB: 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 le **`struct`**, come per array, **NON** sono applicabili le 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

Tipi di Dato

- Classificazione sulla base della struttura:
  - **Tipi semplici**, informazione logicamente **indivisibile**, ad esempio `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()`**, invece 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

1. Dichiarare **altre strutture** del nuovo tipo

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

2. Dichiarare **array** del **nuovo tipo**

```
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

3. Utilizzare il nuovo tipo come **campo** di altre **strutture**

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

4. 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

- Esempio classico:

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

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

```
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 usare gli enum per creare nuovi tipi

```
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`

NB: Nella variabile posso comunque inserire dei valori interi qualunque

Ridefinizione di Tipi Semplici: a Cosa 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:
  1. I tipi user-defined iniziano con la lettera maiuscola, le variabili con la lettera minuscola

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

tipo variabile

  2. 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

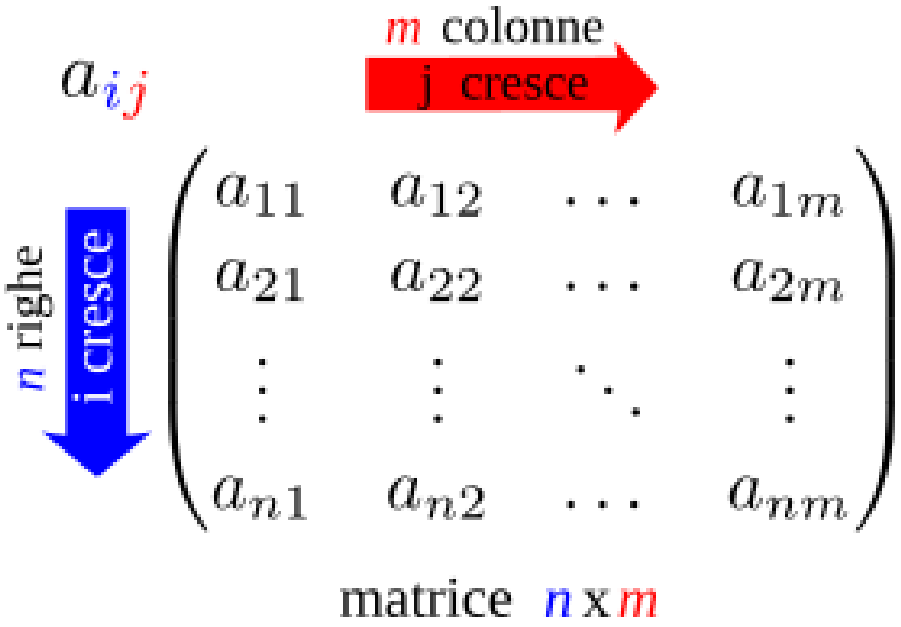
Homeworks

- 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
- Scrivere un frammento di codice in cui si esegue l'acquisizione dei dati relativi ad un libro e quindi popolare uno scaffale
- Stampare i dati relativi a tutti i libri presenti sullo scaffale
- Scrivere un frammento di codice che:
  - Calcola il costo di tutti i libri presenti sullo scaffale (assumendo non vi siano più copie dello stesso libro)
  - Copia tutti i libri con autore che ha il cognome che inizia per 'T' in una seconda variabile di tipo scaffale

Matrici

Definizione di Matrice

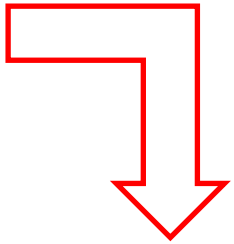
- Tabella ordinata di elementi

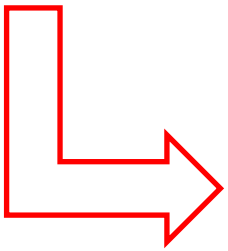


Utilizzo delle Matrici

- Da buoni ingegneri vi serviranno per risolvere problemi di algebra lineare

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
&\vdots \\
a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m
\end{aligned}
$$





$$
\begin{pmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{m1} & a_{m2} & \cdots & a_{mn}
\end{pmatrix}
\begin{pmatrix}
x_1 \\ x_2 \\ \vdots \\ x_n
\end{pmatrix}
=
\begin{pmatrix}
b_1 \\ b_2 \\ \vdots \\ b_m
\end{pmatrix}
$$

Array Multidimensionali

- È possibile definire una matrice in C come array con più di una dimensione
- Avremo un **insieme** di variabili **omogenee** ed **indicizzate**
- Sintassi dichiarazione di una matrice (array 2D) mediante il costruttore array:

```
tipo nomeArray[dim1][dim2];
```

  - `tipo` la keyword di un tipo (built in o user-defined)
  - **`nomeArray`** è il nome della variabile
  - **`dim1`** e **`dim2`** sono **numeri** che stabiliscono il valore massimo rispettivamente del primo (righe) e del secondo (colonne) indice

Esempio

- Acquisire da tastiera i valori 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]);
      }
```

Esempio

- Stampare a video i valori di una matrice

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

Accedo agli elementi di una matrice con la doppia parentesi quadra

Homeworks

- 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 che l'elemento alla riga i e colonna j (a_{ij}) coincide con l'elemento alla riga j e colonna i (a_{ji})

Es.

1	**12**	**1**
12	**0**	**3**
1	**3**	**23**

Homeworks

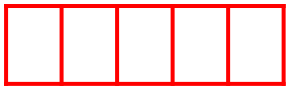
- Scrivere un programma che richiede l'inserimento di una matrice di interi **`M`** e di un intero **`n`**
- Il programma conta il numero di occorrenze di **`n`** in ogni riga di **`M`**
- Il programma stampa con un istogramma (verticale) il numero di occorrenze di **`n`** per ogni riga di **`M`**

Array Multidimensionali

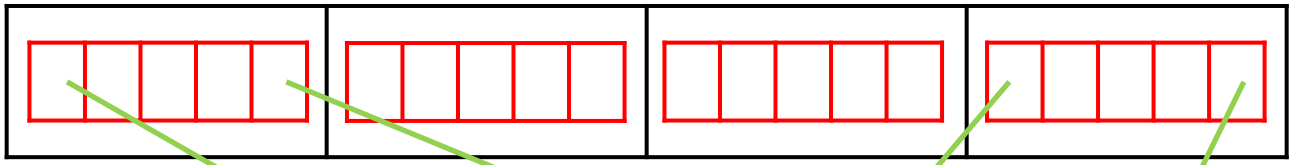
Array di Array

Es.

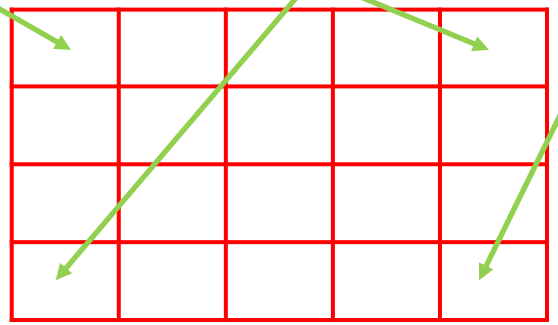
```
typedef  int Vettore[5];
```



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



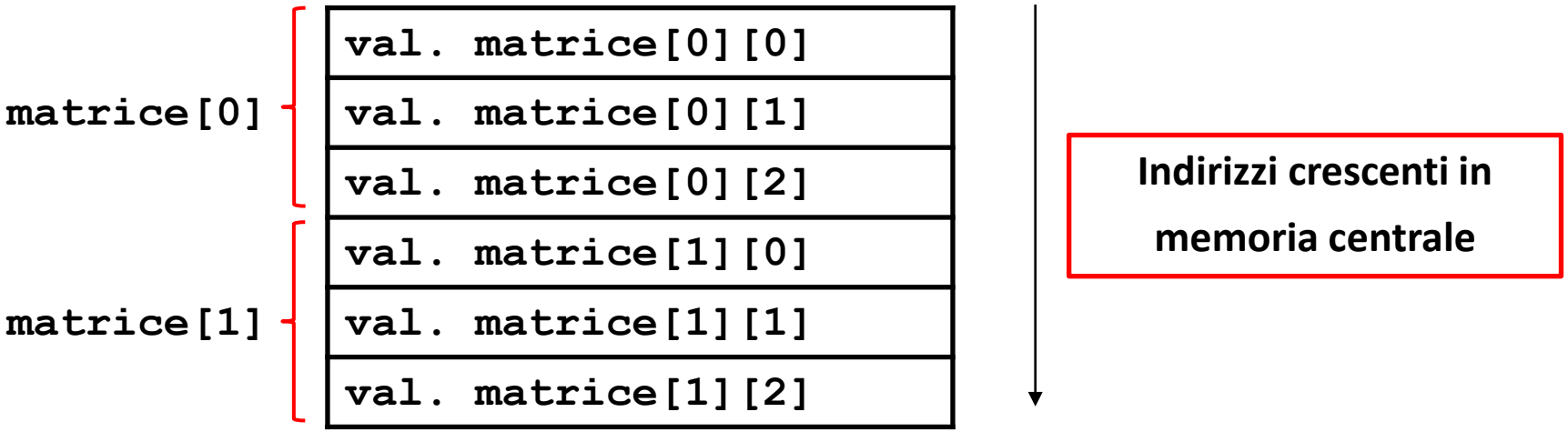
```
Matrice4Per5 matrice1;
```



Memorizzazione di un Array 2D in Memoria

- Un array a 2 dimensioni è memorizzato **riga per riga**, per indice di riga crescente, e, all'interno di ogni riga, per indice di colonna crescente

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



Indirizzo un Array

- 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:

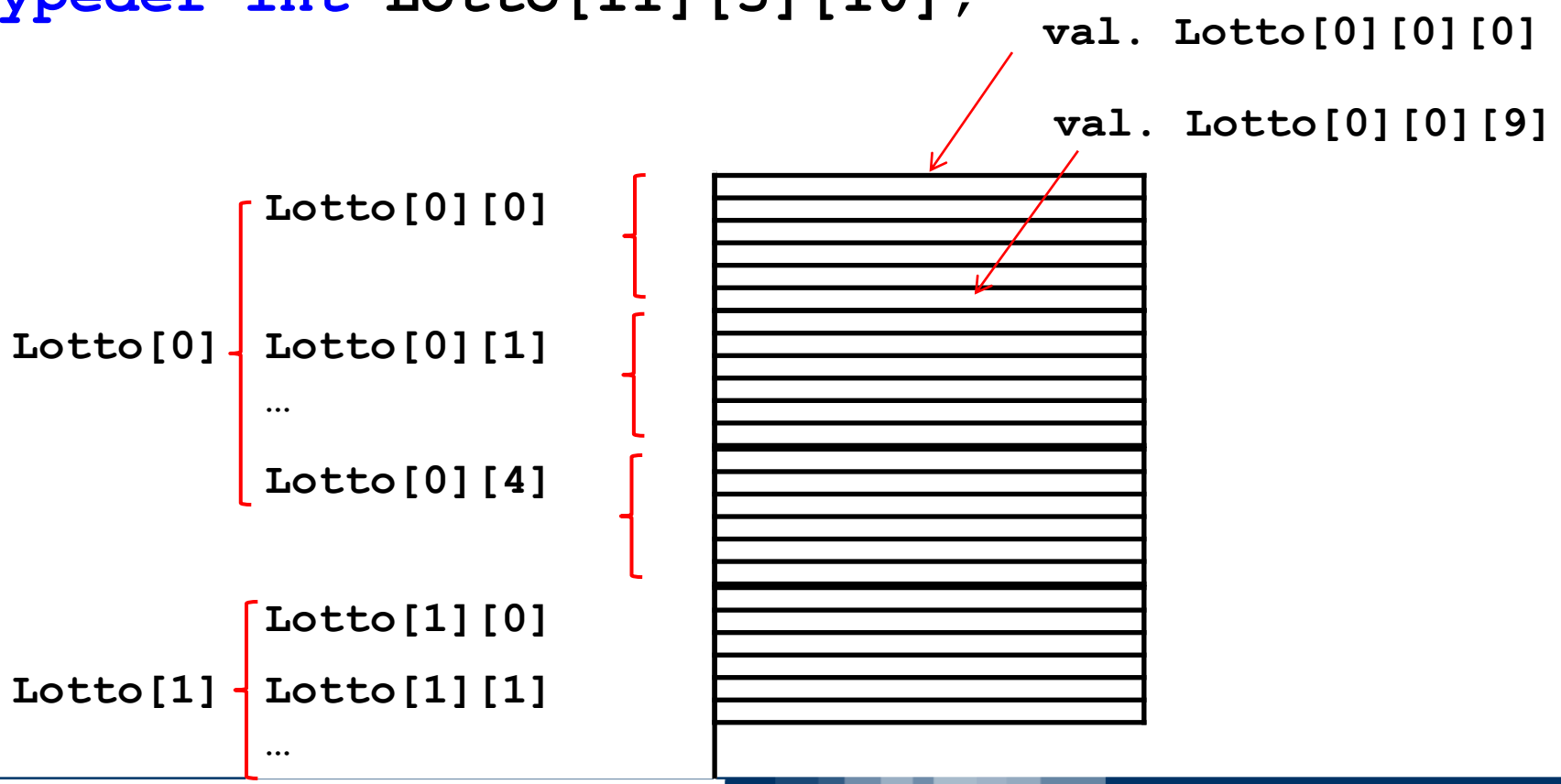
```
&matrice[i][j] == &matrice[0][0]
                        + i * Dim1 + j
```

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

Memorizzazione di un Array 3D in Memoria

- 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];
```



Definizione, Dichiarazione ed Accesso Array 3D

- Dichiarazione di una matrice:

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

- Accesso agli elementi:

```
a = M[0][2];
```

- Assegnamento a un elemento:

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

- Dichiarazione di una matrice tridimensionale:

```
int matrice3D[5][6][3];
```

- Accesso agli elementi:

```
a = matrice3D[2][1][1];
```

- Assegnamento a un elemento:

```
matrice3D[2][4][0] = 5;
```

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

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, il che è dato dalla **tipizzazione forte** del C

Espressioni Aritmetiche: Conversione Implicita

- La conversione implicita avviene nle caso di 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 < int < long int
                    < float < double < long double
```

Regole Generali

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

Es. `short int 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`, perchè la **promozione riguarda solamente il valore letto**

NB: le conversioni sono un terreno scivoloso; fate attenzione usando gli `unsigned` ($\Rightarrow$ evitare di fare conversioni)

Assegnamenti: Conversione Implicita

Lo stesso ordinamento in base alla precisione

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

viene utilizzato per assegnamenti tra variabili eterogenee

Es. `double d; int i;`

```
d = i;
```

- Valore di `i` convertito a `double` e assegnato a `d` e nell'operazione **non c'è perdita di informazione**

```
i = d;
```

- Valore di `d` convertito a `int` (troncandolo), valore intero assegnato a `i` e ho **perdita di informazione**

Esempio (Memento)

```
// 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 Stampa? (`int Ftemp, float Ctemp`)

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

Testo Esercizio (TDE 07/08)

- Le seguenti dichiarazioni definiscono tipi di dati relativi:
  - alla categoria degli impiegati di un'azienda, ovvero prima, seconda, fino a quinta categoria
  - agli uffici occupati da tali impiegati
  - all'edificio che ospita tali uffici, dove l'edificio è diviso in 20 piani ognuno con 40 uffici

Testo Esercizio (TDE 07/08)

```
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 */
```

Richiesta (1) Esercizio (TDE 07/08)

- 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 (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

Non è == "sudEst", perchè è una enum!

Richiesta (2) Esercizio (TDE 07/08)

- Visualizzare a schermo i numeri dei piani che non hanno neanche un ufficio esposto a nord

Soluzione (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

Richiesta (3) Esercizio (TDE 07/08)

- Dire in che piano ed in che ufficio si trova Boracchi
- Copiare in un array tutti gli uffici occupati da dipendenti di categoria 5
- Copiare in un array tutti i dipendenti di categoria 5
- 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 strettamente 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