



Matrici, Struct e Tipi User-Defined

Informatica B AA 2021/2022

Luca Cassano

28 Ottobre 2021

luca.cassano@polimi.it



Array Multidimensionali

- Matrici



Array Multidimensionali

- È possibile definire 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 del primo e del secondo indice rispettivamente



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") ;  
    }
```



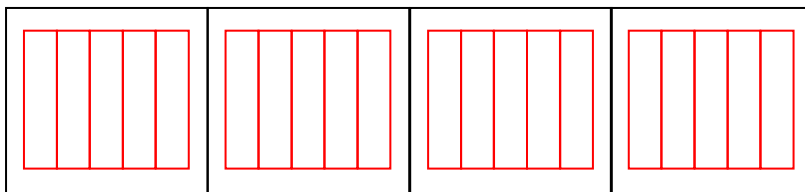
Array di Array

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

⇒ è possibile costruire "array di array"

Esempio:

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





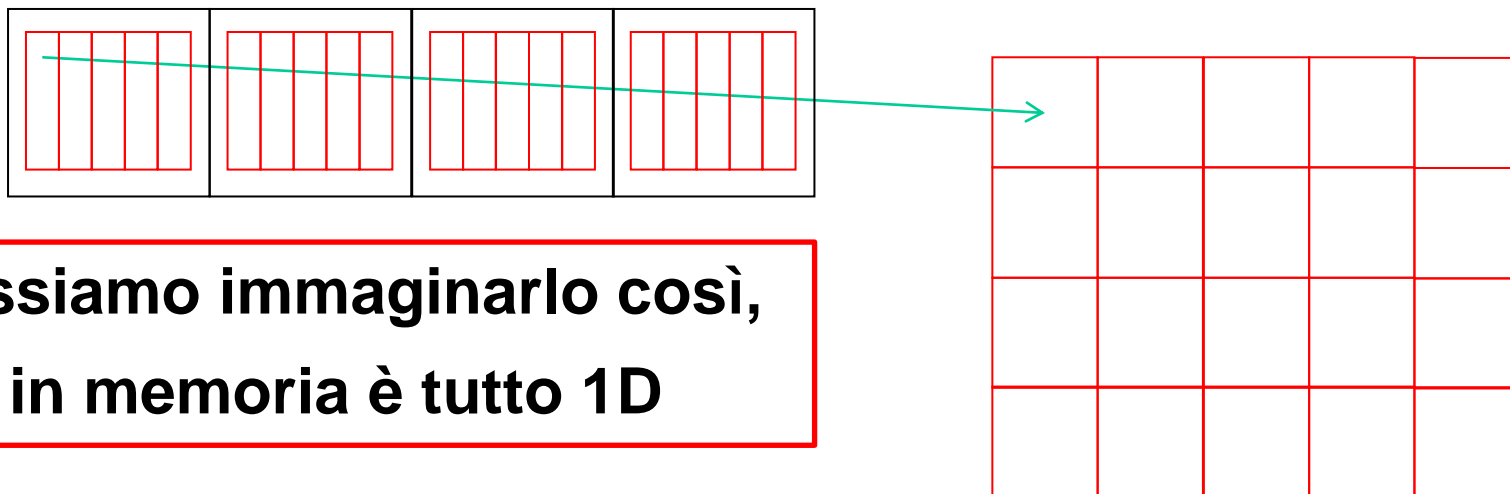
Array di Array

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

⇒ è possibile costruire "array di array"

Esempio:

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



**Possiamo immaginarlo così,
ma in memoria è tutto 1D**



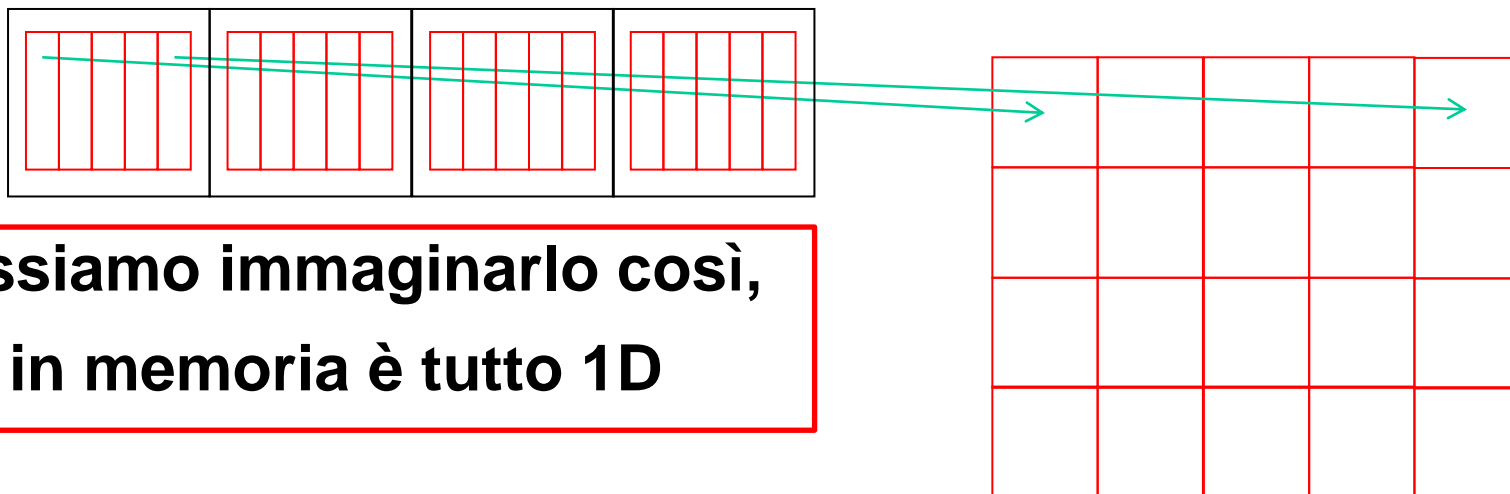
Array di Array

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

⇒ è possibile costruire "array di array"

Esempio:

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





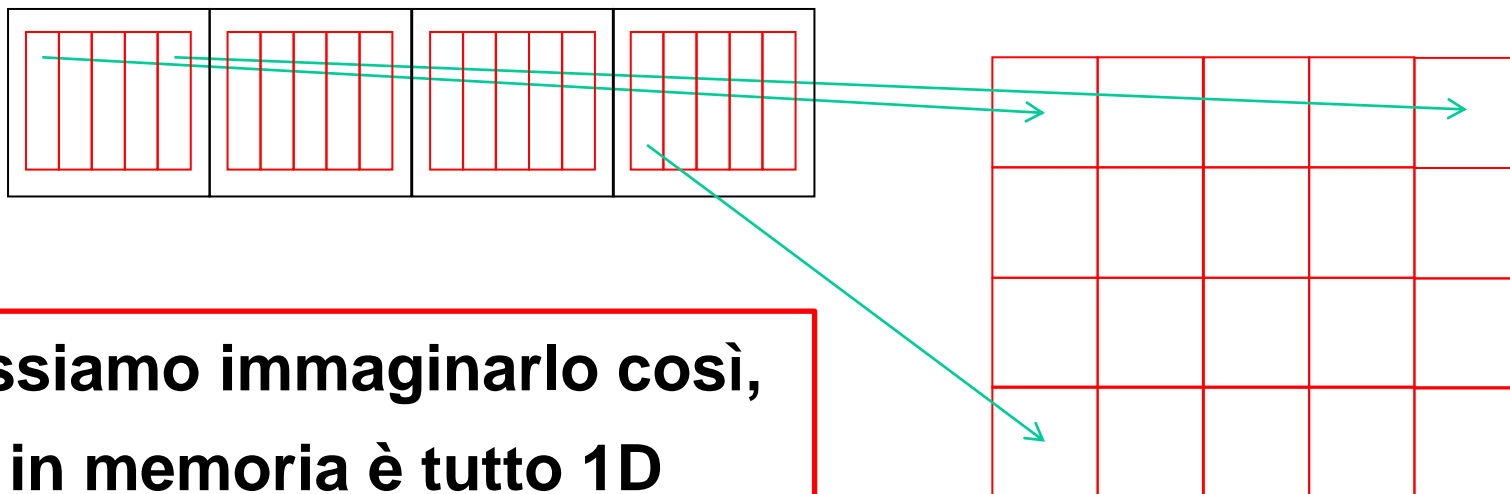
Array di Array

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

⇒ è possibile costruire "array di array"

Esempio:

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





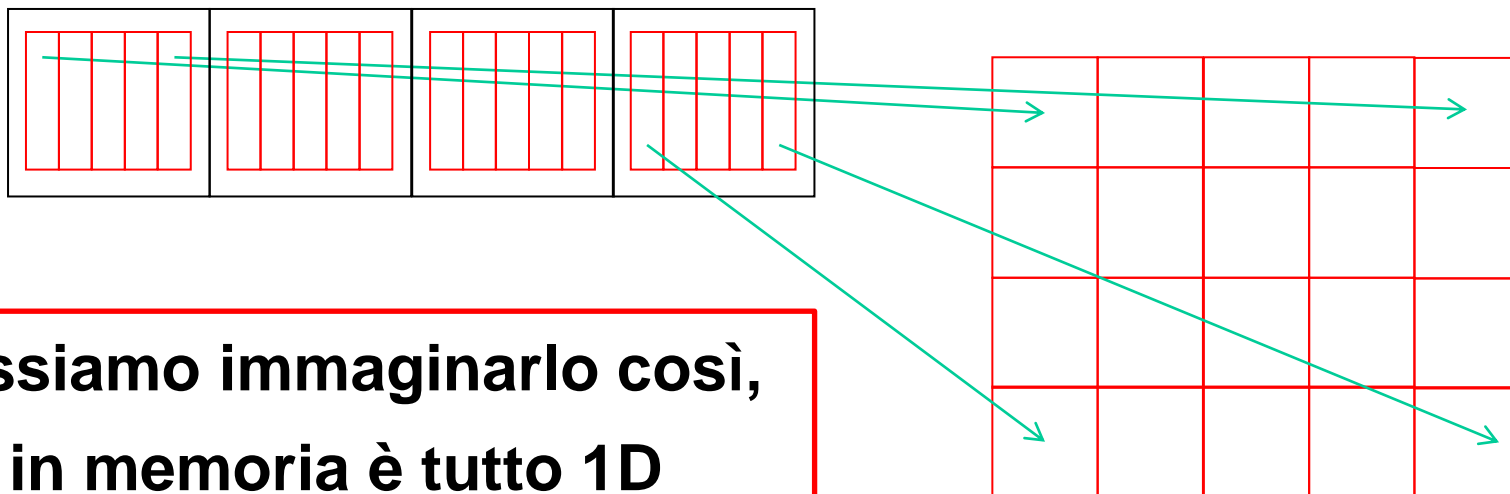
Array di Array

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

⇒ è possibile costruire "array di array"

Esempio:

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



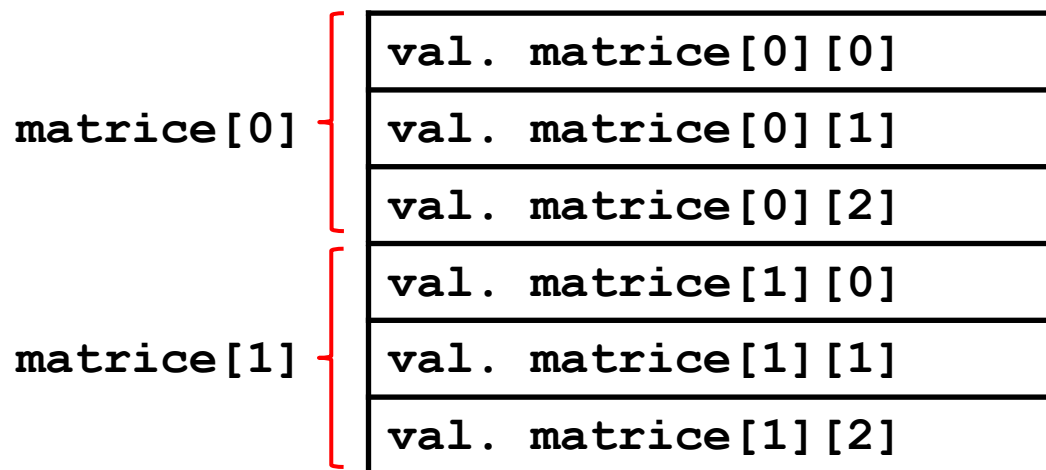


Mappa di Memorizzazione di un array 2D

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];
```



Indirizzi crescenti
in memoria
centrale





Mappa di Memorizzazione di un array 1D

Definire un tipo di dato atto a contenere le velocità medie di un'auto in 10 istanti fissati



Mappa di Memorizzazione di un array 1D

Definire un tipo di dato atto a contenere le velocità medie di un'auto in 10 istanti fissati

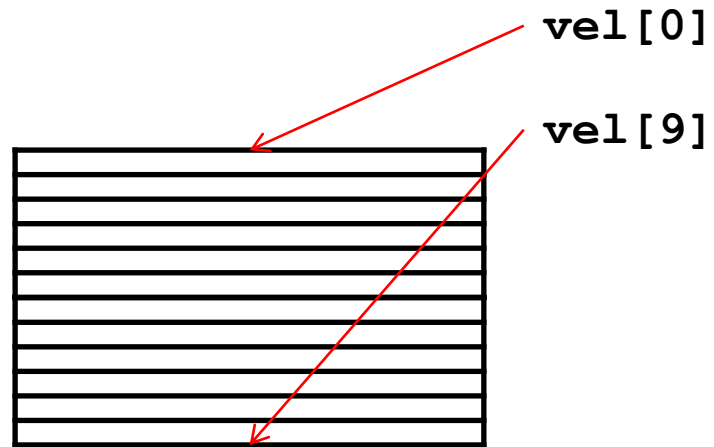
- 10 valori di velocità

```
double vel[10];
```



Mappa di Memorizzazione di un array 1D

```
double vel[10];
```





Mappa di Memorizzazione di un array 2D

Definire un tipo di dato atto a contenere il numero di gol segnati da ciascuna delle 20 squadre di calcio di serie A in ciascuna delle 38 giornate di serie A



Mappa di Memorizzazione di un array 2D

Definire un tipo di dato atto a contenere il numero di gol segnati da ciascuna delle 20 squadre di calcio di serie A in ciascuna delle 38 giornate di serie A

- 20 squadre di serie A
- 38 giornate di serie A

```
int gol[20][38];
```




Mappa di Memorizzazione di un array 2D

```
int go1[20][38];
```

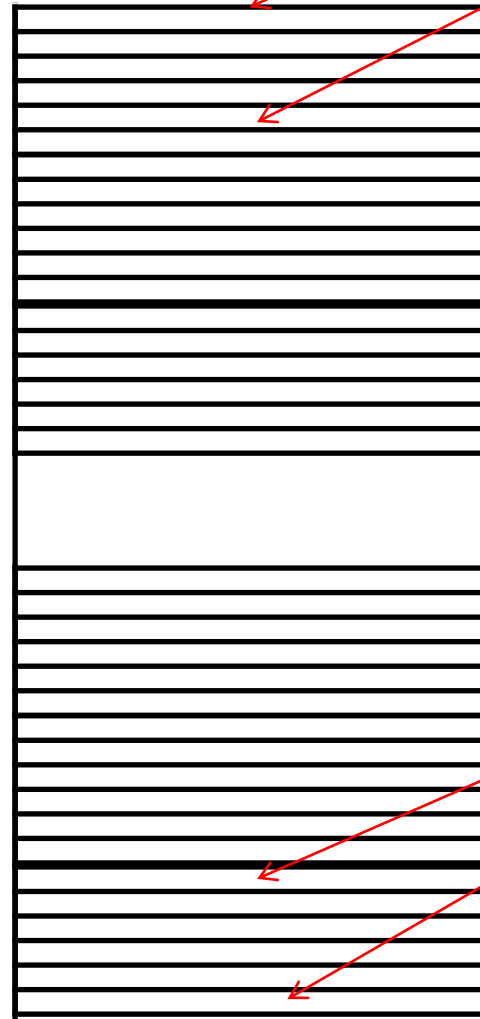
```
go1[0]
```

```
...
```

```
...
```

```
...
```

```
go1[19]
```



```
go1[0][0]
```

```
go1[0][37]
```

```
go1[19][0]
```

```
go1[19][37]
```



Mappa di Memorizzazione di un array 3D

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

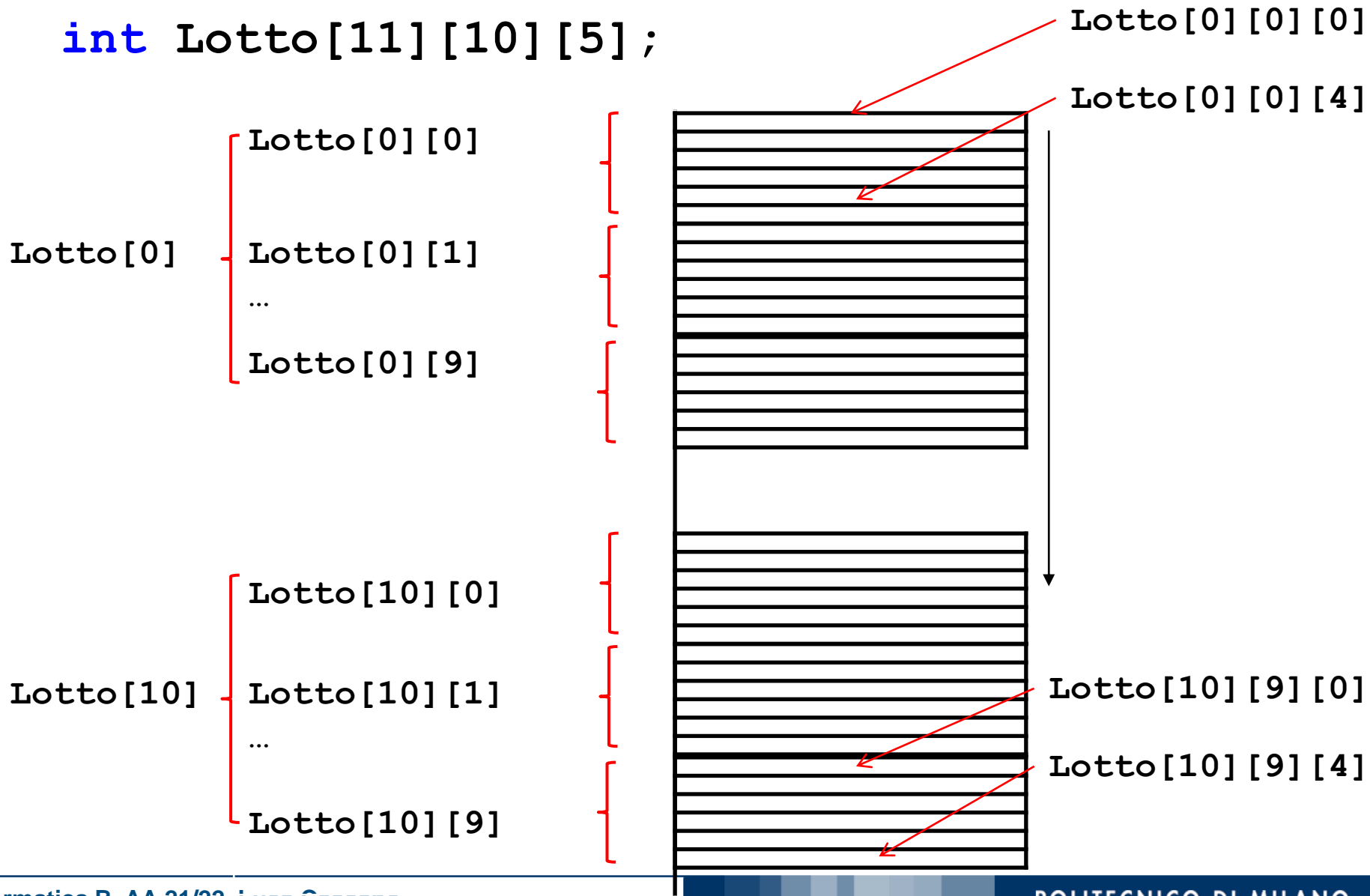
- 11 ruote
- 10 estrazioni per ciascuna ruota
- 5 numeri per ciascuna estrazione

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



Mappa di Memorizzazione di un array 3D

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





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**



Tipi Enumerativi: a che servono?

- Aumentare l'espressività del linguaggio

Esempio:

```
enum{false, vero};
```

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



Tipi Enumerativi: che valori assumono?

- Di default

```
enum{quadri = 0, cuori = 1, fiori = 2,  
picche = 3};
```

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

```
enum{quadri = 5, cuori = -1, fiori = 21,  
picche = 13};
```

- Se i valori sono sequenziali basta specificare il primo

```
enum{quadri = 14, cuori, fiori, picche};
```




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



Dichiarazione di una Struttura

- Sintassi:

```
struct {  
    tipo1 nomeCampo1;  
    tipo2 nomeCampo2;  
    ...  
    tipoN nomeCampoN;  
} nome;
```

- Dichiarare una variabile **struct** chiamata **nome**
- I nomi dei campi della struttura sono **nomeCampo1...**



Dichiarazione di una Struttura

- Dichiarazione compatta per campi dello stesso tipo

```
struct {  
    tipo1 nomeCampoA, nomeCampoB;  
  
    ...  
    tipoN campoN;  
} nome;
```



Dichiarazione di una Struttura

- È possibile dichiarare due o più variabili dalla stessa struttura

```
struct {  
    tipo1 nomeCampo1;  
    tipo2 nomeCampo2;  
    ...  
    tipoN nomeCampoN;  
} nome1, nome2;
```



Struct vs Array

- *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



Struct vs Array

- *Esempio: variabile per contenere anagrafica di impiegati*
 - *nome, cognome, codice fiscale, indirizzo, numero di telefono, stipendio, data di assunzione etc.*

```
struct {  
    char nome [20], cognome[20], CF[20];  
    char indirizzo[20];  
    int stipendio;  
    int dataDiAssunzione;  
}anagrafica;
```



Dichiarazione di una Struttura

- **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 (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];  
} dipendente1, dipendente2;
```

```
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: ");  
scanf("%s", dip1.nome);  
printf("\nInserire Cognome: ");  
scanf("%s", dip1.cognome);  
printf("\nInserire Stipendio: ");  
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 di nascita");
    scanf("%d", &N.giorno);
    printf("\nInserire mese di nascita");
    scanf("%s", N.mese);
    printf("\nInserire anno di nascita");
    scanf("%d", &N.anno);
```



Esempio

```
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



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;`



Nuovi tipi

- 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;
```




Definizione di Nuovi Tipi Strutturati

- Quando si associa un nuovo tipo ad una struttura è possibile:
 1. dichiarare **altre strutture** (i.e., variabili del nuovo tipo)
 2. dichiarare **array** di strutture (i.e., 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 user-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** assegnare 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)



Qualche esempio



Ufficio ed impiegati

Definire le strutture dati necessarie a gestire:

- Un edificio di 20 piani. Ogni piano contiene 40 uffici.
- Ogni ufficio ha un'esposizione (Nord, NordEst, Est ...) ed un impiegato.
- Ogni impiegato ha nome, cognome, categoria e stipendio.



Ufficio ed impiegati

```
typedef struct {  
    char nome[20], cognome[20];  
    int cat;  
    int stipendio;  
} Impiegato;
```



Ufficio ed impiegati

```
typedef struct {  
    char nome[20], cognome[20];  
    int cat;  
    int stipendio;  
} Impiegato;
```

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



Ufficio ed impiegati

```
typedef struct {
    char nome[20], cognome[20];
    int cat;
    int stipendio;
} Impiegato;
```

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

```
typedef struct {
    int superficie;
    Esposizione esp;
    Impiegato occupante;
} Ufficio;
```




Ufficio ed impiegati

```
typedef struct {  
    char nome[20], cognome[20];  
    int cat;  
    int stipendio;  
} Impiegato;
```

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

```
typedef struct {  
    int superficie;  
    Esposizione esp;  
    Impiegato occupante;  
} Ufficio;
```

```
Ufficio torre[20][40];
```



Ufficio ed impiegati

Si scriva un frammento di codice, 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.



Ufficio ed impiegati

```
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++)
```



Ufficio ed impiegati

```
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))

{

}
}
```



Ufficio ed impiegati

```
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) )

{

}
}
```



Ufficio ed impiegati

```
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))
        {

}
```



Ufficio ed impiegati

```
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);
        }
```



Ufficio ed impiegati

- Si scriva un frammento di codice, che visualizzi a schermo i numeri dei piani che non hanno neanche un ufficio esposto a nord.



Ufficio ed impiegati

```
int uffNord; /* uffNord fa da flag*/
for (p=0; p<20; p++)
{
    uffNord = 0; //flag = 0 in ogni piano
}
}
```



Ufficio ed impiegati

```
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++)

}
```



Ufficio ed impiegati

```
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;
}
```



Ufficio ed impiegati

```
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);
}
```



Ufficio ed impiegati

- Si scriva un frammento di codice che visualizzi il piano a cui si trova l'ufficio di Giacomo Boracchi



Ufficio ed impiegati

```
int p, u, nome, cognome;
```




Ufficio ed impiegati

```
int p, u, nome, cognome;
for (p=0; p<20; p++) {
    for (u=0; u<40; u++) {
        nome =
            strcmp(torre[p][u].occupante.nome,
                "Giacomo");
    }
}
```



Ufficio ed impiegati

```
int p, u, nome, cognome;
for (p=0; p<20; p++) {
    for (u=0; u<40; u++) {
        nome =
            strcmp(torre[p][u].occupante.nome,
                "Giacomo");

        cognome =
            strcmp(torre[p][u].occupante.cognome,
                "Boracchi");

    }
}
```



Ufficio ed impiegati

```
int p, u, nome, cognome;
for (p=0; p<20; p++) {
    for (u=0; u<40; u++) {
        nome =
            strcmp(torre[p][u].occupante.nome,
                "Giacomo");

        cognome =
            strcmp(torre[p][u].occupante.cognome,
                "Boracchi");

        if(nome == 0 && cognome == 0)
            printf("Giacomo Boracchi occupa un
ufficio al piano %d", p);
    }
}
```