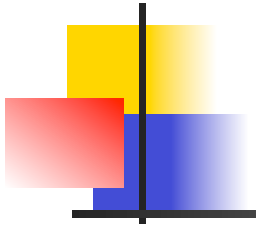
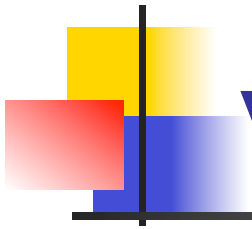


Puntatori e strutture dati dinamiche: allocazione della memoria e modularità in linguaggio C

Capitolo 2: La dualità puntatore-vettore

G. Cabodi, P. Camurati, P. Pasini, D. Patti, D. Vendraminetto

MAGGIOLI EDITORE APOGEO education 2016



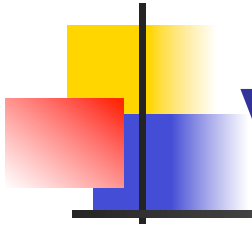
Vettori e puntatori

Vettori e puntatori sono **duali** e consentono l'accesso ai dati in 2 forme:

- **vettoriale** con indici e [ ]
- con **puntatori** mediante &, * e aritmetica dei puntatori

Il **nome** della variabile che identifica il vettore corrisponde formalmente al **puntatore** al primo elemento del vettore stesso:

<id vettore> ⇔ &<id vettore>[0]

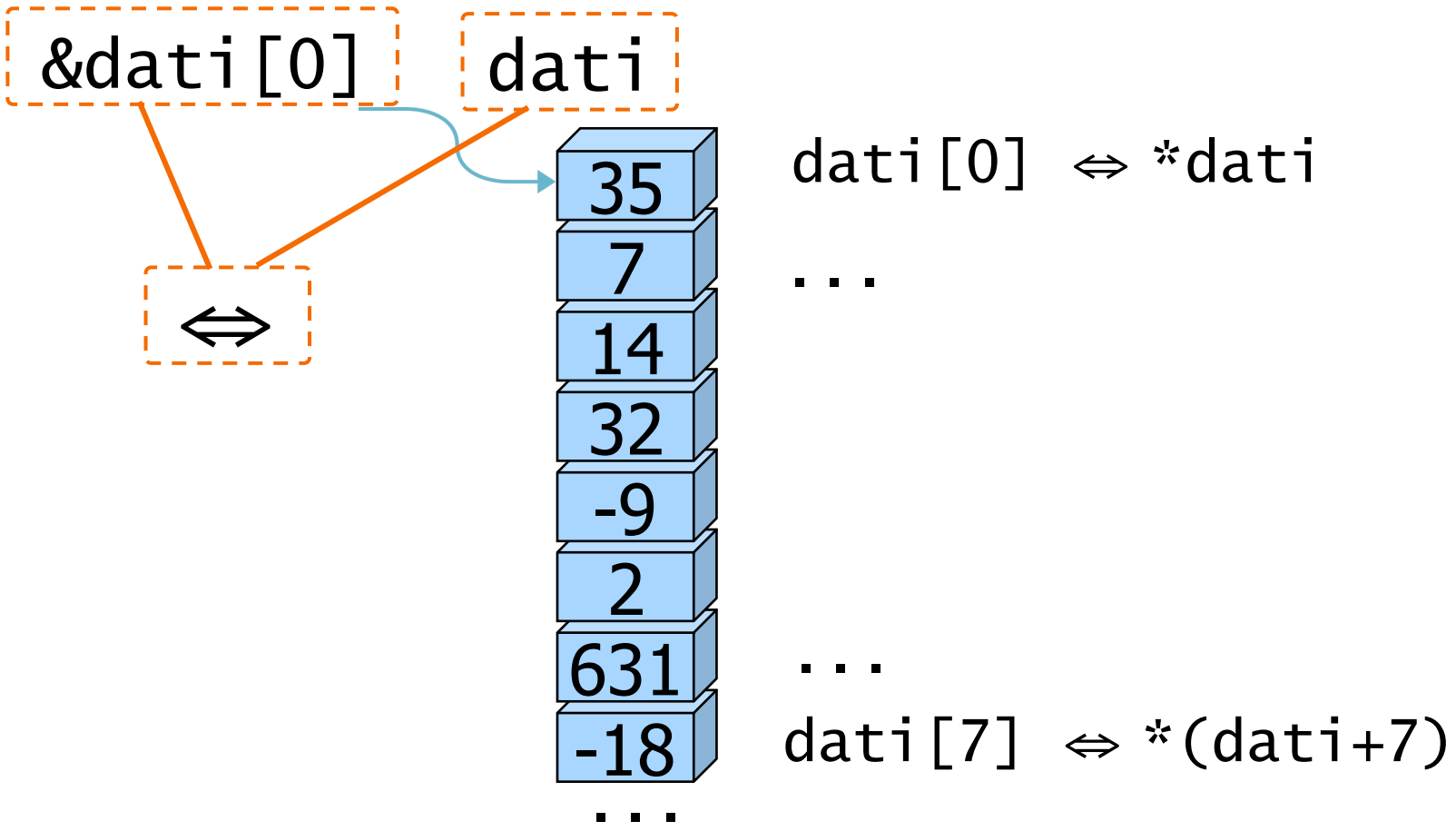


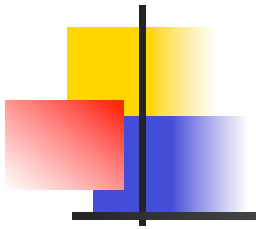
Vettori e puntatori

Esempio: data una variabile di tipo vettore

```
int dati[100];
```

- `dati` ⇔ `&dati[0]`
- `*dati` ⇔ `dati[0]`
- `*(dati+i)` ⇔ `dati[i]`


&dati[0]
dati
⇔
35
7
14
32
-9
2
631
-18
...
dati[0] ⇔ *dati
...
...
dati[7] ⇔ *(dati+7)

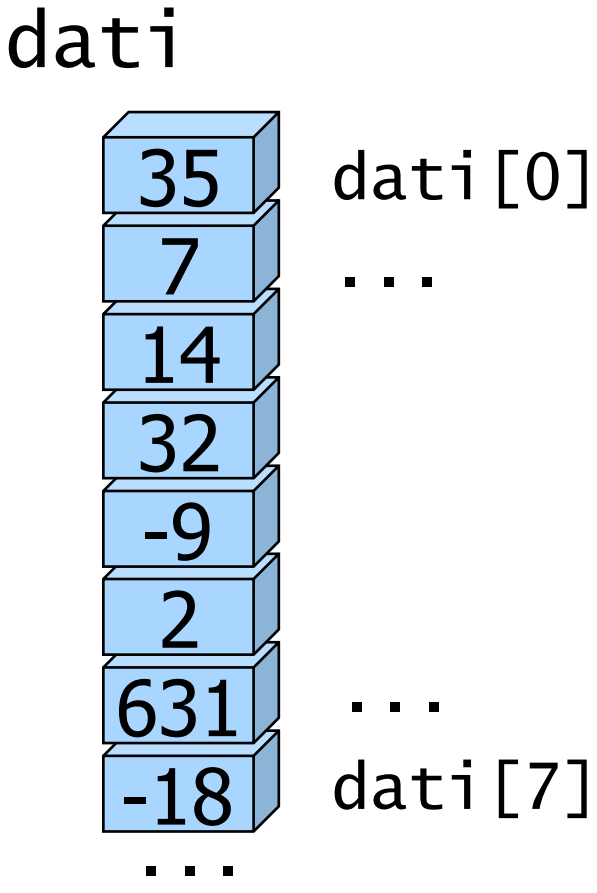


Esempio: lettura di un vettore di 100 interi:

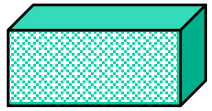
1. come vettore, con notazione vettoriale [ ] scorrendo le caselle mediante un indice
2. con puntatore a intero `int *p`, inizializzato a `&dati[0]` e aritmetica dei puntatori (somma tra puntatore e indice intero)
3. con puntatore a intero `int *p`, inizializzato a `&dati[0]` e scansione dei dati direttamente mediante puntatore.

Modo 1:

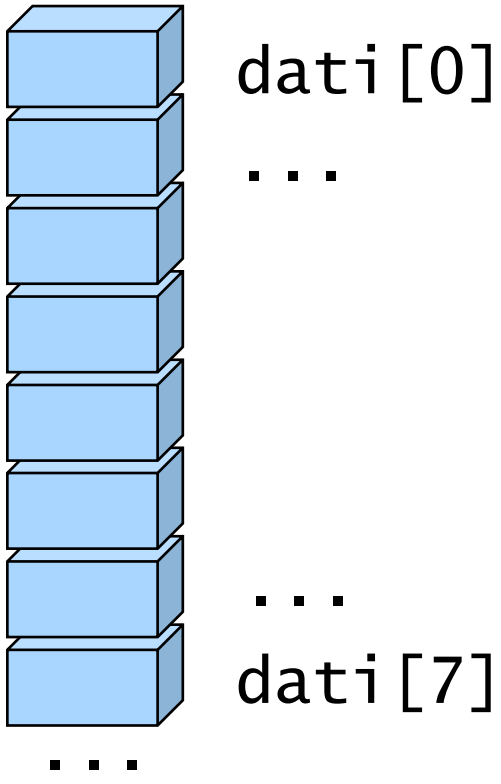
```
int dati[100];
...
for (i=0;i<100;i++)
  scanf("%d“,&dati[i]);
```



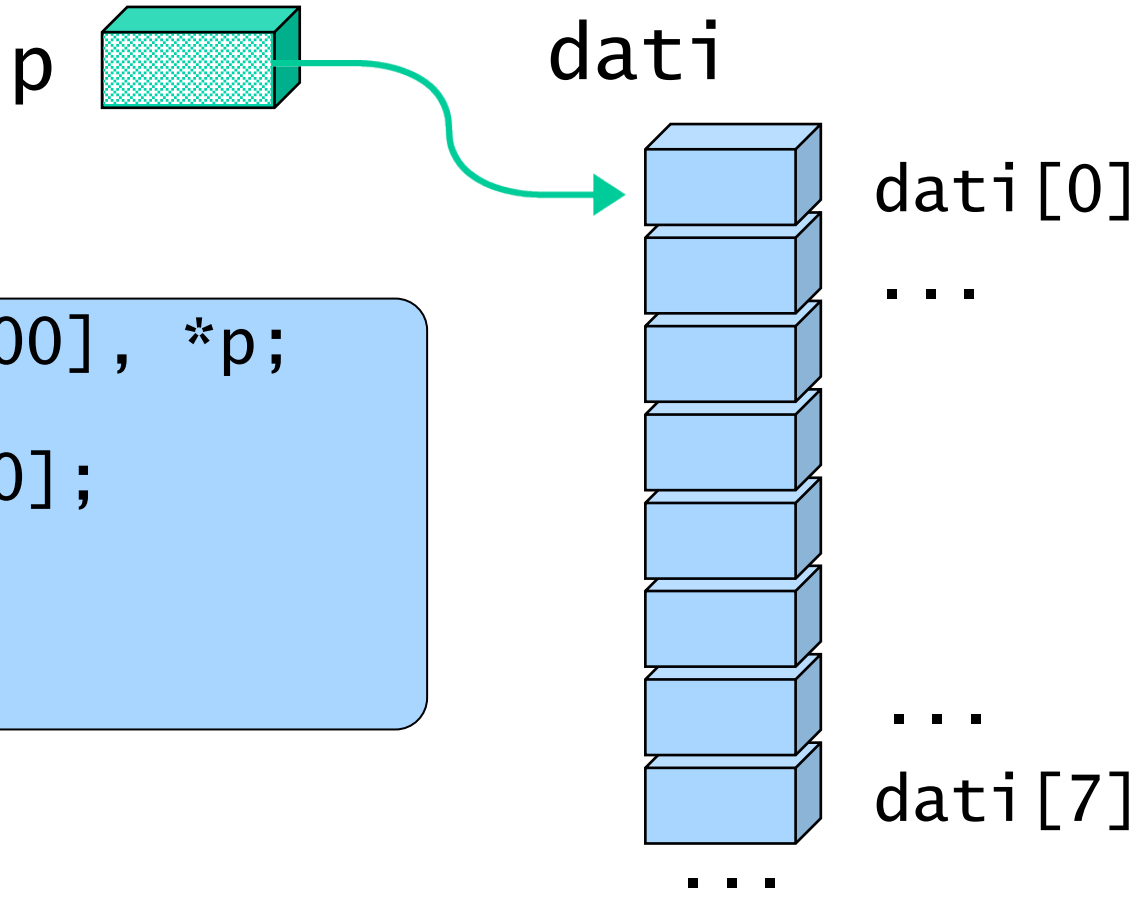
Modo 2:




```
int dati[100], *p;
...
```

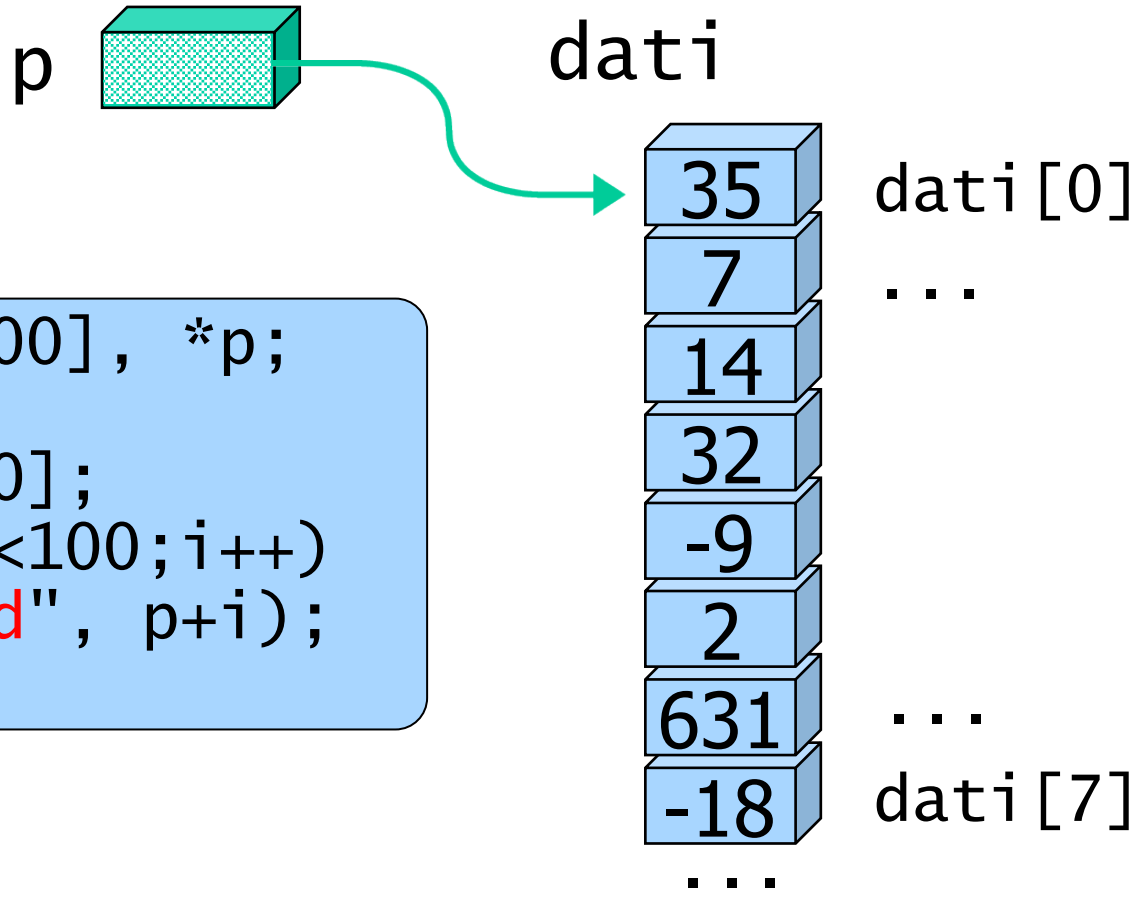


Modo 2:




```
int dati[100], *p;
...
p = &dati[0];
```

Modo 2:

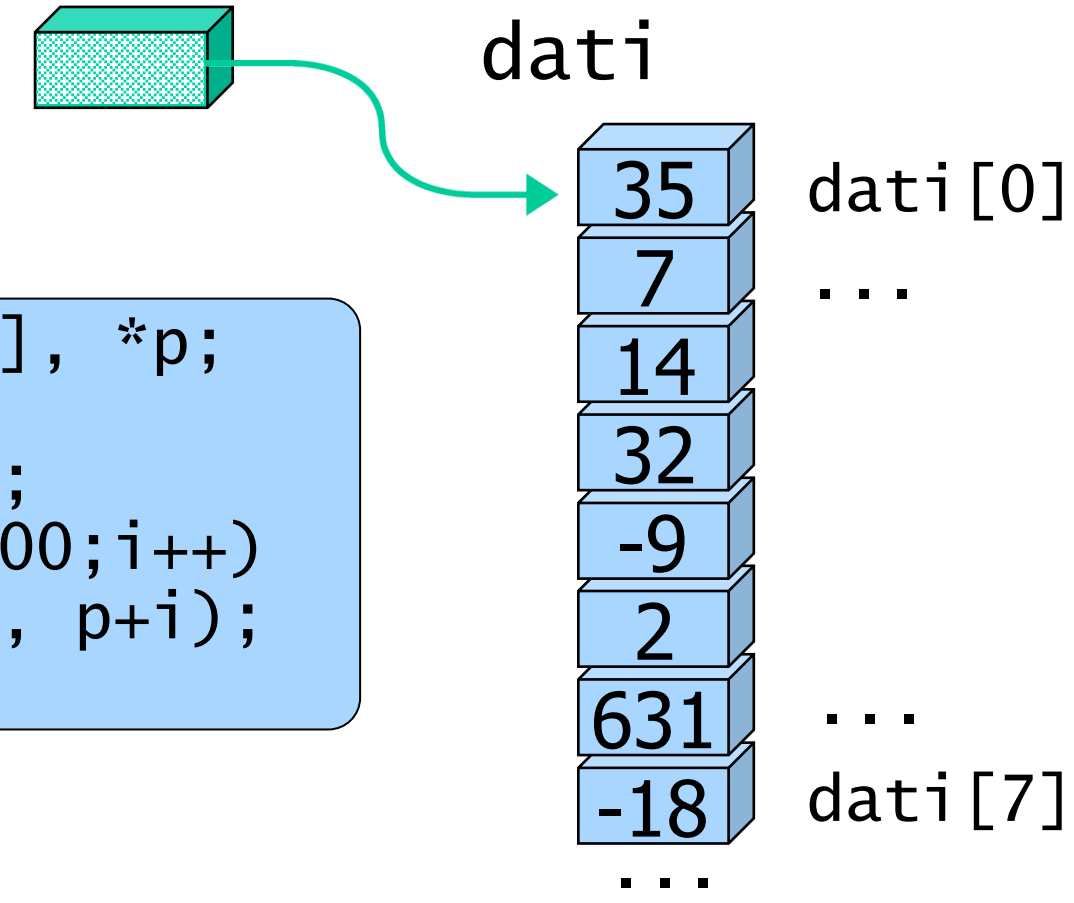



```
int dati[100], *p;
...
p = &dati[0];
for (i=0;i<100;i++)
  scanf("%d", p+i);
```

Modo 2:

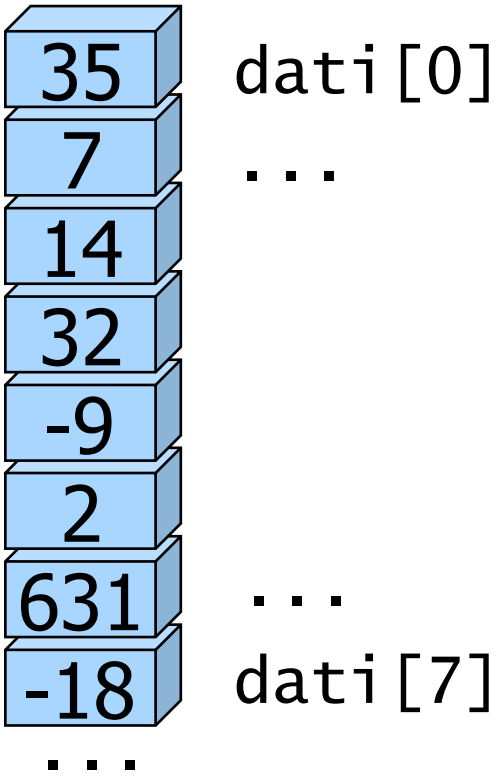
dati = &dati[0]

```
int dati[100], *p;
...
p = &dati[0];
for (i=0;i<100;i++)
  scanf("%d", p+i);
```



Modo 2:

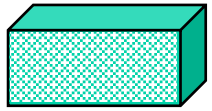
dati = &dati[0]



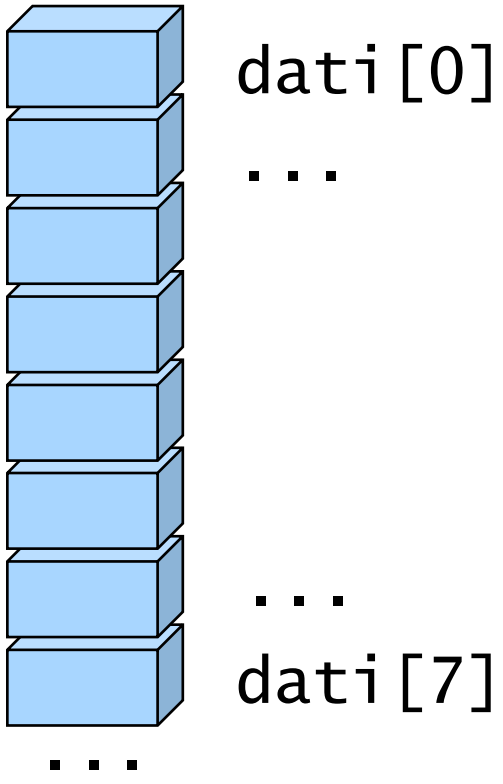

```
int dati[100], *p;
...
p = &dati[0];
for (i=0;i<100;i++)
  scanf("%d", p+i);
```

p+i = &dati[i]

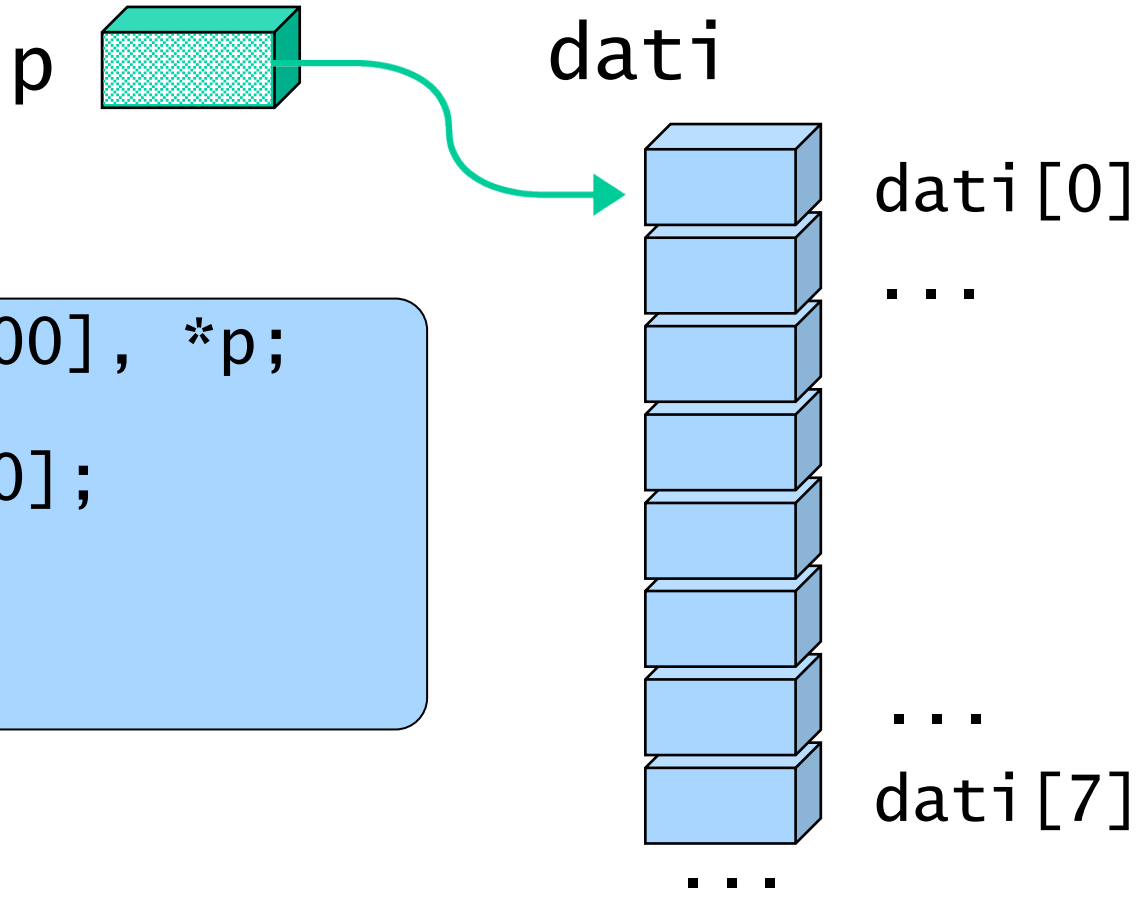
Modo 3:




```
int dati[100], *p;
...
```

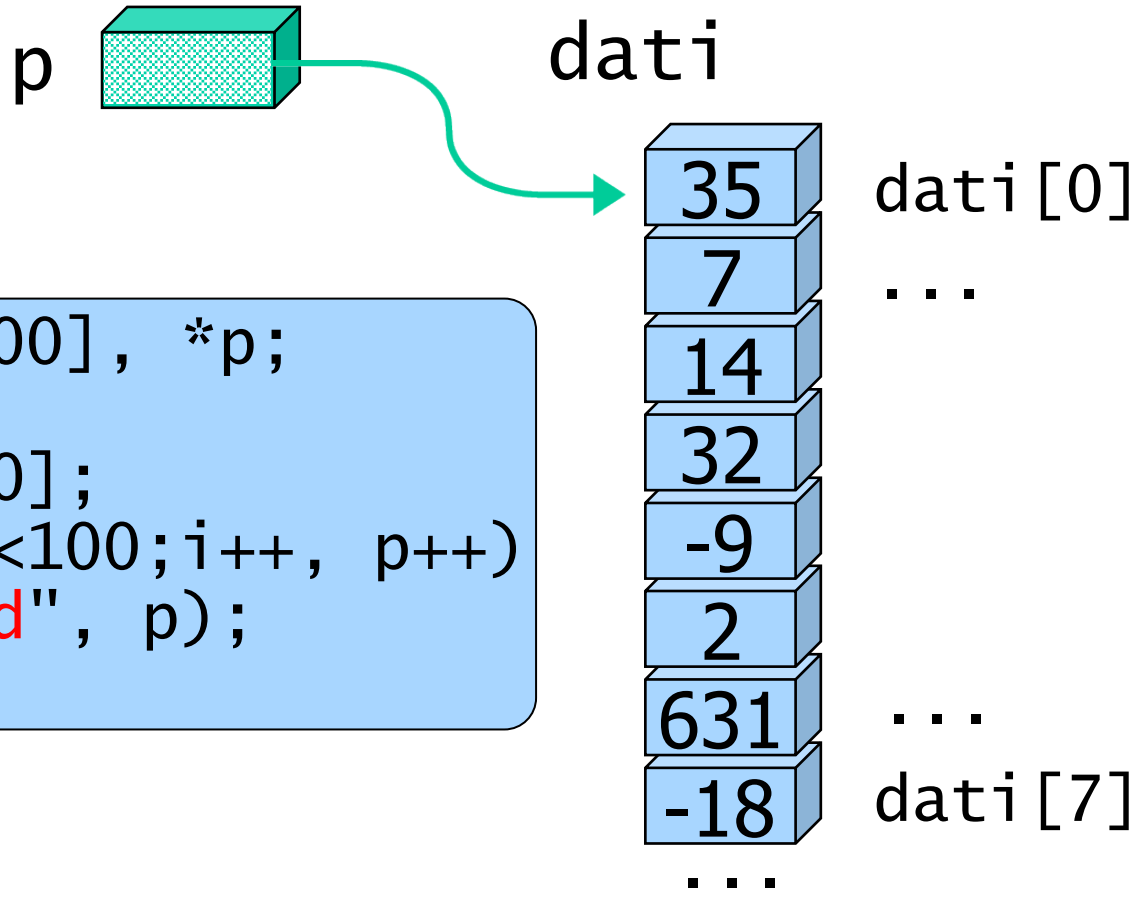


Modo 3:

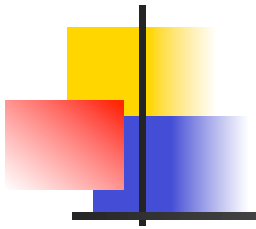



```
int dati[100], *p;
...
p = &dati[0];
```

Modo 3:




```
int dati[100], *p;
...
p = &dati[0];
for (i=0;i<100;i++, p++)
  scanf("%d", p);
```



Sono lecite notazioni miste:

- puntatore a intero `int *v`, inizializzato a `dati` (=`&dati[0]`) e usato con notazione vettoriale

```
int *v = dati;
for (i=0; i<100; i++)
  scanf("%d", v[i]);
```

- vettore di interi `dati` utilizzato come puntatore

```
for (i=0; i<100; i++)
  scanf("%d", dati+i);
```

Limite alla dualità:

- il nome del vettore corrisponde ad una costante puntatore, non ad una variabile, quindi non può essere incrementato per scandire il vettore

```
for (i=0; i<100; i++, dati++)
  scanf("%d", dati);
```

Puntatori e sottovettori

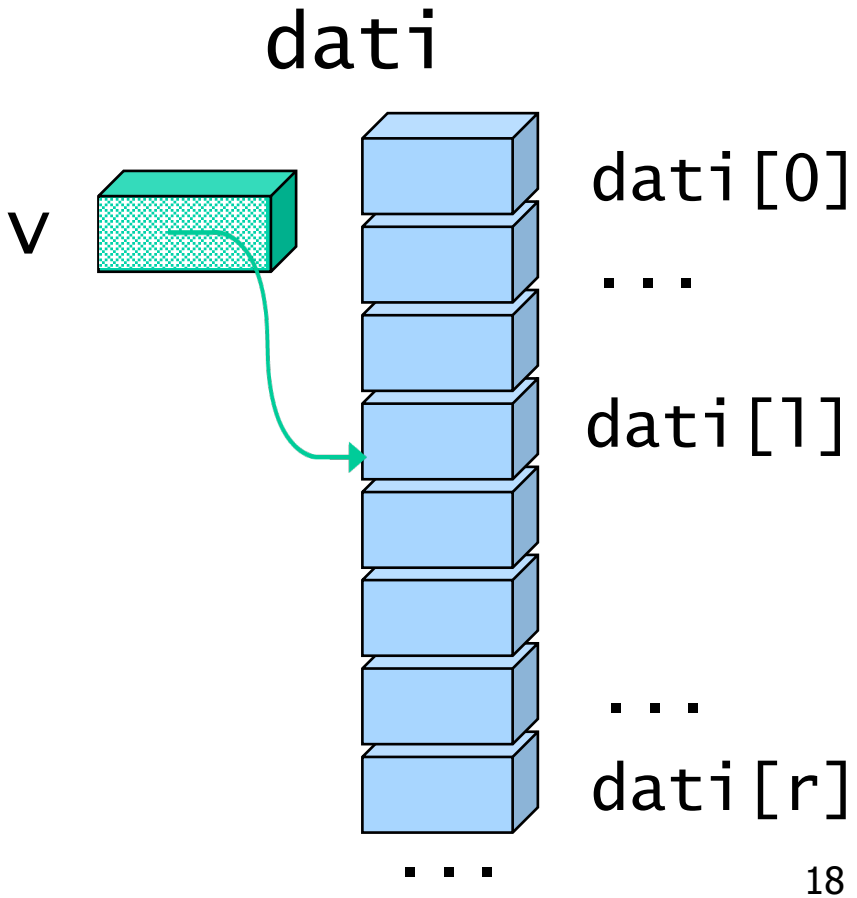
Per identificare un sottovettore compreso tra indici `l` e `r` di un vettore dato:

- si limita tra `l` e `r` l'indice `i` (identificazione implicita del sottovettore):

```
for (i=l; i<=r; i++)
  scanf("%d", &dati[i]);
for (i=l; i<=r; i++)
  printf("%d", dati[i]);
```

- si identifica esplicitamente un sottovettore tramite un puntatore alla sua casella di indice `l`:

```
int dati[100], *v, i;
v = &dati[l];
n = r - l + 1;
for (i=0; i<n; i++)
  scanf("%d", &v[i]);
for (i=0; i<n; i++)
  printf("%d", v[i]);
```

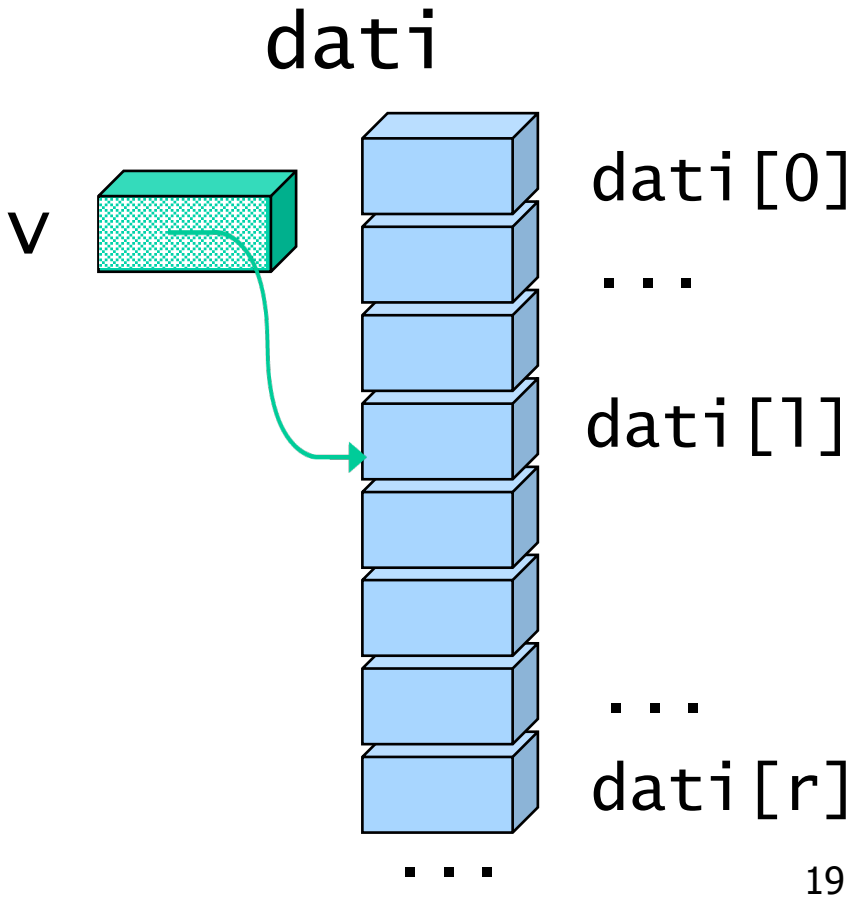


Modo 1

- si identifica esplicitamente un sottovettore tramite un puntatore alla sua casella di indice `l`:

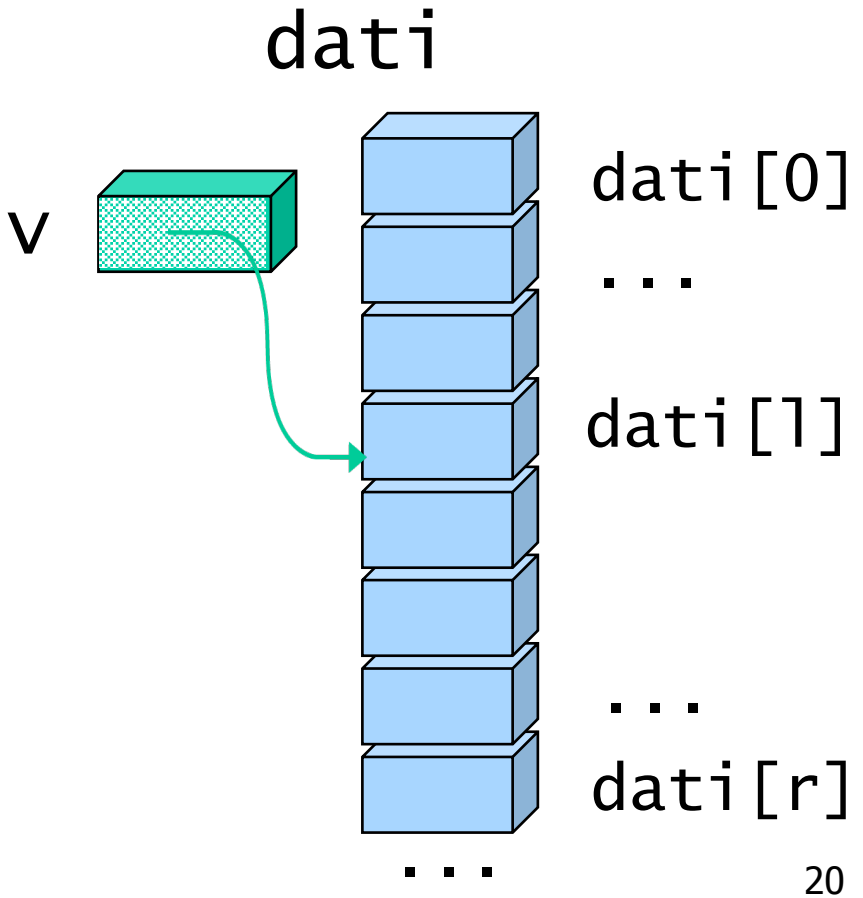
```
int dati[100], *v, i;
v = &dati[l];
n = r - l + 1;
for (i=0; i<n; i++)
   scanf("%d", &v[i]);
for (i=0; i<n; i++)
   printf("%d", v[i]);
```

n è il numero di caselle del sottovettore



- si identifica esplicitamente un sottovettore tramite un puntatore alla sua casella di indice `l`:

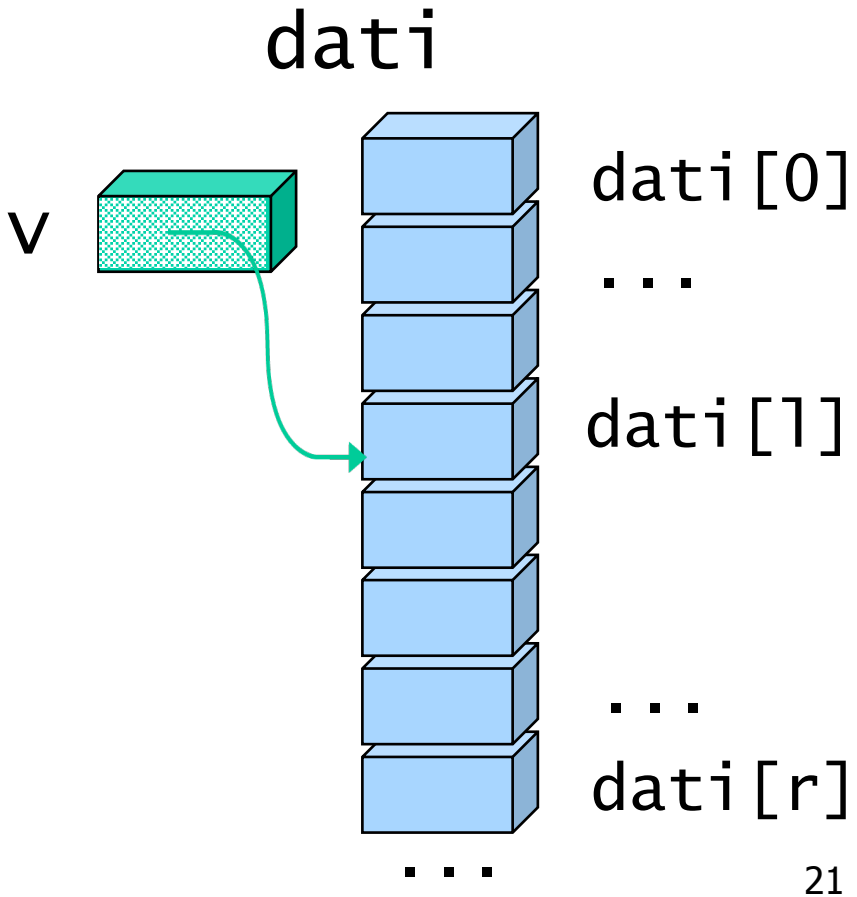
```
int dati[100], *v, i;
v = &dati[l];
n = r - l + 1;
for (i=0; i<n; i++)
  scanf("%d", v++);
v = dati+l;
for (i=0; i<n; i++)
  printf("%d", *v++);
```



Modo 2

- si identifica esplicitamente un sottovettore tramite un puntatore alla sua casella di indice `l`:

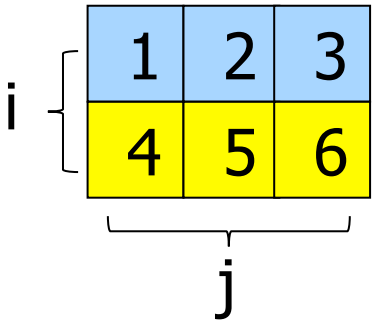
```
int dati[100], *v, i;
v = dati + l;
n = r - l + 1;
for (i=0; i<n; i++)
  scanf("%d", v+i);
for (i=0; i<n; i++)
  printf("%d", *(v+i));
```

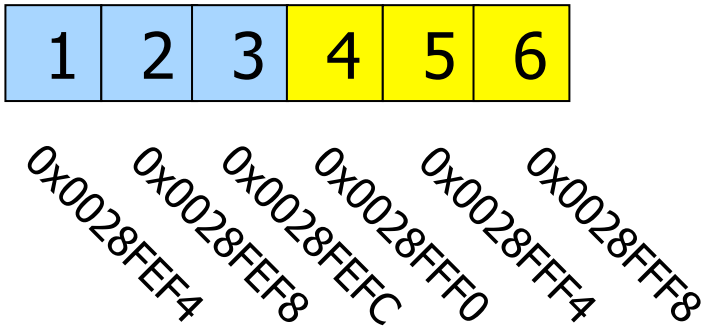


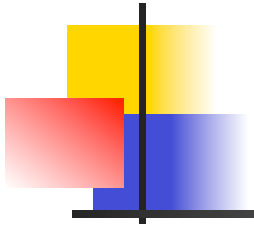
Modo 3

Decomposizione di matrici

- Le matrici (bidimensionali e multidimensionali) si possono decomporre per righe (o sotto-matrici)
- Le matrici sono memorizzate con la tecnica row-major
  - matrice bidimensionale come vettore di righe
  - casella di una riga adiacenti e righe in sequenza





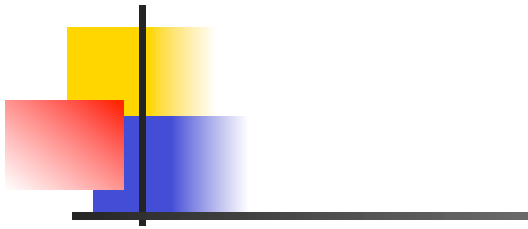


Esempio:

prodotto scalare di una matrice M di NR righe per NC colonne per un vettore di NC elementi.

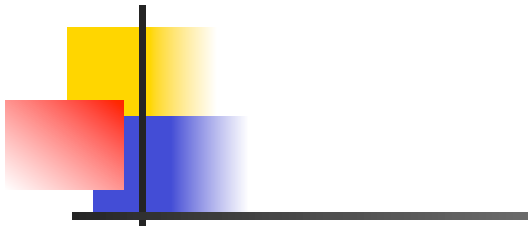
Il risultato è un vettore di NR elementi:

1	2	x	5	=	17	39
3	4		6			



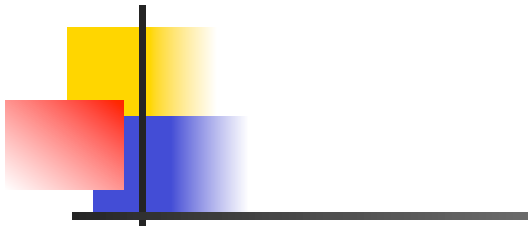
Modo 1: matrice bidimensionale con accesso a riga mediante iterazione su colonne:

```
float M[NR][NC], V[NC], Prod[NR];
int r, c;
...
for (r=0; r<NR; r++) {
  Prod[r] = 0.0;
  for (c=0; c<NC; c++)
    Prod[r] = Prod[r] + M[r][c]*V[c];
}
```



Modo 2: righe della matrice identificate grazie ad un puntatore `riga` (possibile grazie al row-major):

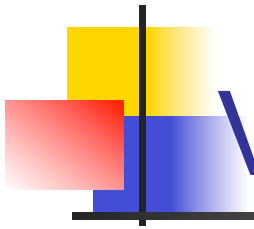
```
float M[NR][NC], V[NC], Prod[NR], *riga;
int r, c;
...
for (r=0; r<NR; r++) {
  Prod[r] = 0.0;
  riga = M[r];
  for (c=0; c<NC; c++)
    Prod[r] = Prod[r] + riga[c]*V[c];
}
```



Modo 2: righe della matrice identificate grazie ad un puntatore `riga` (possibile grazie al row-major):

```
float M[NR][NC], V[NC], Prod[NR], *riga;
int r, c;
...
for (r=0; r<NR; r++) {
  Prod[r] = 0.0;
  riga = M[r];
  for (c=0; c<NC; c++)
    Prod[r] = Prod[r] + riga[c]*V[c];
}
```

Equivale a riga= &(M[r][0]);



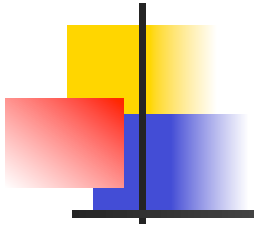
Vettori e matrici come parametri

Vettori e matrici passati come parametri non vengono generati all'interno della funzione:

- si passa solo il puntatore alla prima casella.

La dualità puntatore ⇔ vettore è:

- totale per vettori (monodimensionali)
- parziale per matrici (vettori multidimensionali)

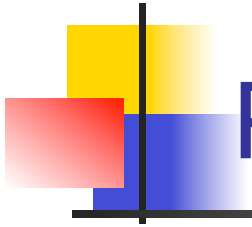


Sono ridondanti le seguenti informazioni:

- dimensione di un vettore
- prima dimensione di una matrice

Esempi: prototipi compatibili di funzioni di libreria

```
size_t strlen(const char s[]);
int strcmp(const char s1[], const char s2[]);
char *strcpy (char dest[], const char src[]);
```



Parametri formali

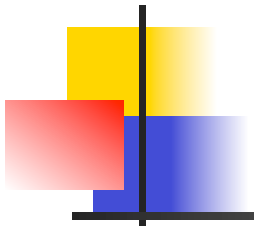
Notazione a vettore o a puntatore sono interscambiabili nei parametri formali

Esempi: prototipi compatibili di funzioni di libreria:

```
size_t strlen(const char *s);
int strcmp(const char *s1, const char *s2);
char *strcpy (char *dest, const char *src);
```

Esempi: prototipi reali di funzioni di libreria:

```
size_t strlen(const char *s);
int strcmp(const char *s1, const char *s2);
char *strcpy (char *dest, const char *src);
```



- Esempio: argomenti al `main`: il vettore di puntatori a carattere argv è dichiarato come:

vettore adimensionale di puntatori

```
int main(int argc, char *argv[])
```

puntatore a puntatore

```
int main(int argc, char **argv)
```

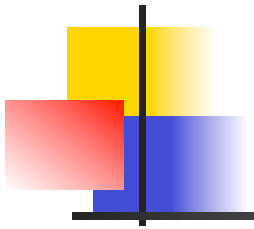
Dimensione come parametro

Sovente si passa come parametri:

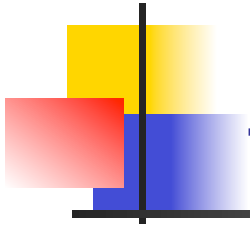
- il vettore adimensionato
- la sua dimensione effettiva

per realizzare funzioni che si adattano ad essa:

```
int leggi(int v[], int maxDim);

int main (void) {
  int v1[DIM1], v2[DIM2];
  int n1, n2;
  n1 = leggi(v1,DIM1);
  n2 = leggi(v2,DIM2);
  ...
```



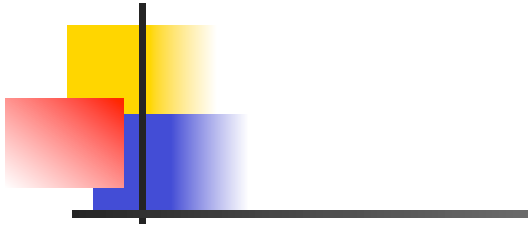
```c
int leggi(int v[], int maxDim) {
  int i, fine=0;
  for (i=0; !fine && i<maxDim; i++) {
    printf("v[%d] (0 per terminare): ", i);
    scanf("%d",&v[i]);
    if (v[i]==0) {
      fine = 1;
      i--; // trascura lo 0
    }
  }
  return i;
}
```



Corrispondenza parametri formali-attuali

Ad un parametro formale vettore può corrispondere un parametro attuale puntatore

- consente di generare un vettore (per una funzione) da un sotto-vettore, oppure da un puntatore (a memoria contigua)



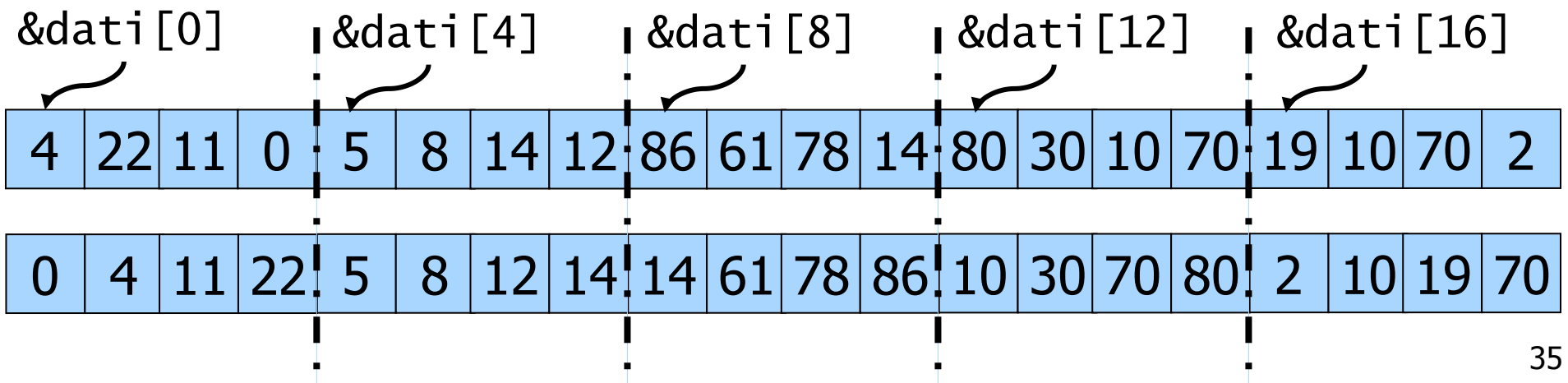
Esempio: ordinamento di vettore per gruppi

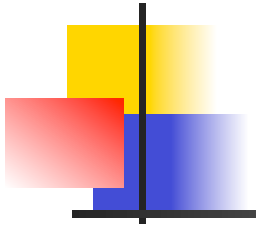
```
void ordinaInt(int v[], int n);
...
int dati[20];
...
for (i=0;i<20;i+=4)
  ordinaInt(&dati[i],4);
```

Esempio: ordinamento di vettore per gruppi

```
void ordinaInt(int v[], int n);
...
int dati[20];
...
for (i=0;i<20;i+=4)
  ordinaInt(&dati[i],4);
```

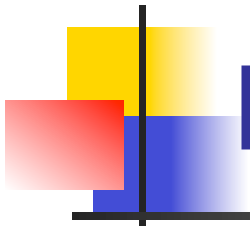
Ordinamento applicato a sotto-vettori di 4 elementi





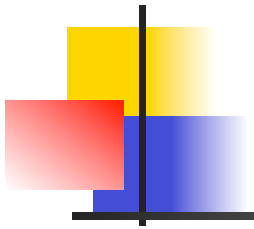
- Ad un parametro formale puntatore può corrispondere un parametro attuale vettore
  - Il puntatore, a sua volta, può essere trattato internamente come vettore (purché punti a dati contigui in memoria)

```
int leggi(int *v, int maxDim);
...
int sim, dati[100];
...
dim = leggi(dati, 100);
```



Puntatori e stringhe

- In C le stringhe non sono un tipo
- Una stringa è (formalmente) un vettore di caratteri (terminato da '\0')
- Le stringhe possono essere manipolate:
  - come vettori
  - con i puntatori e la loro aritmetica



Esempio: `strlen` con stringa come vettore:

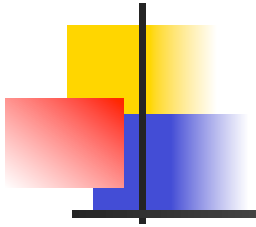
```
int strlen0(char s[]){
  int cnt=0;
  while (s[cnt]!='\0')
    cnt++;
  return cnt;
}
```

`cnt` funge sia da indice che da contatore

Esempio: `strlen` con stringa come vettore, manipolata mediante puntatore:

```
int strlen1(char s[]){
  int cnt=0;
  char *p=&s[0];
  while (*p !='\0')
    cnt++;
  return cnt;
}
```

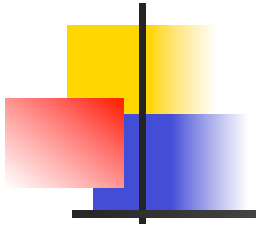
`cnt` funge da contatore, lo scorrimento avviene mediante puntatore p



Esempio: `strlen` con stringa rappresentata (e manipolata) mediante puntatori:

```
int strlen2(char *s){
  int cnt=0;
  while (*s++ != '\0')
    cnt++;
  return cnt;
}
```

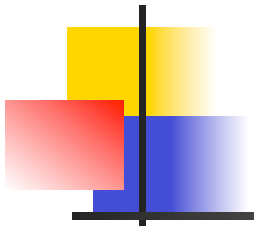
`cnt` funge da contatore, lo scorrimento avviene mediante puntatore `s`



- Esempio: `strlen` con stringa rappresentata mediante puntatori e manipolata mediante aritmetica dei puntatori:

```
int strlen3(char *s){
     char *p=s;
     while (*p != '\0')
        p++;
     return p-s;
  }
```

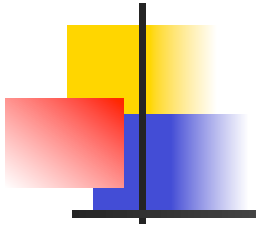
non serve un contatore esplicito



Esempio: `strcmp`

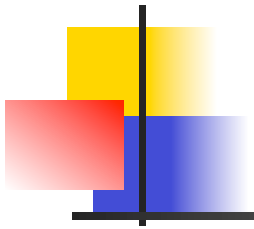
- Date due stringhe, confrontarne i contenuti, ritornando:
  - 0 se le stringhe sono uguali
  - <0 se la prima stringa precede la seconda
  - >0 se la prima stringa segue la seconda

  Se le stringhe differiscono si ritorna la differenza tra i codici dei primi caratteri diversi.



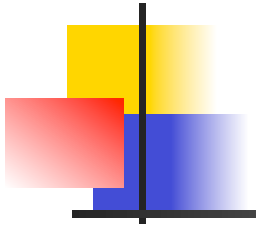
- Stringhe come vettori
- Strategia:
  - iterazione confrontare i caratteri sino alla prima differenza, oppure al terminatore di stringa
  - si ritorna la differenza tra i caratteri diversi (o i terminatori)

```
int strcmp0(char s0[], char s1[]){
  int i=0;
  while (s0[i]==s1[i] && s0[i]!='\0')
    i++;
  return (s0[i]-s1[i]);
}
```



- Stringhe come puntatori
- Stessa strategia ma iterazione con avanzamento dei puntatori

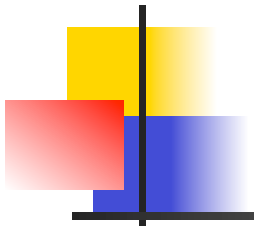
```
int strcmp1(char *s0, char *s1) {
  while ((*s0==*s1) && (*s0!='\0')){
     s0++;
     s1++;
  }
  return (*s0-*s1);
}
```



Esempio: `strncmp`

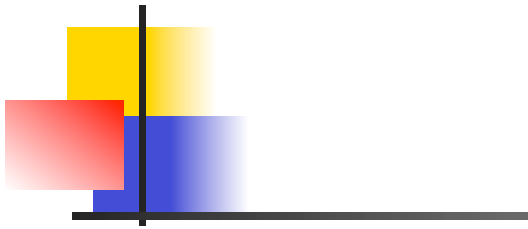
- Come `strcmp0`, ma il confronto si limita ai primi n caratteri

```
int strncmp0(char s0[], char s1[], int n){
  int i=0;
  while (s0[i]==s1[i] && s0[i]!='\0')
    if (i<n)
      i++;
    else
      return 0;
  return (s0[i]-s1[i]);
}
```



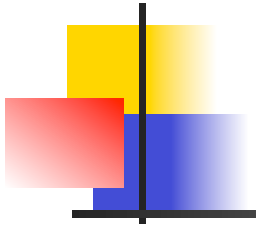
Esempio: `strstr`

- Date due stringhe, cercare la prima occorrenza della seconda stringa all'interno della prima, ritornando un puntatore:
  - `NULL` se non viene trovata la seconda stringa all'interno della prima
  - al primo carattere della sottostringa trovata



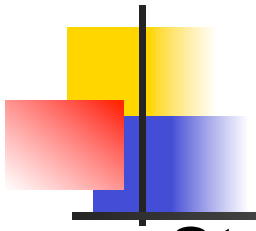
- Strategia:
  - iterazione che, per l'i-esimo carattere della prima stringa, determina se si tratta dell'inizio della sotto-stringa cercata (usando `strncmp`)

```
char *strstr0(char s[], char cerca[]){
  int i, lun_s, lun_c;
  lun_s=strlen(s);
  lun_c=strlen(c);
  for (i=0; i<=lun_s-lun_c; i++)
     if (strncmp(&s[i],c,lun_c)==0)
        return (&s[i]);
   return (NULL);
}
```

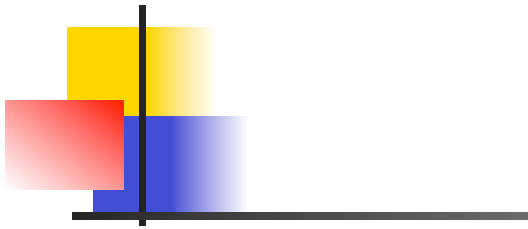


Esempio: input mediante `sscanf`

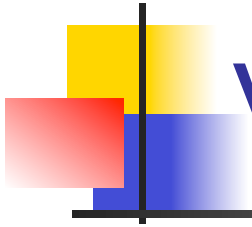
- Acquisire da una riga di testo (`stdin`) una riga (di non più di `MAX` caratteri), contenente un numero non noto di numeri interi separati da spazi
- Calcolare e stampare la media aritmetica dei valori letti.



- Strategia:
  - Non si può usare un input formattato (%d) in quanto non si sa quanti sono i campi e il %d non discrimina tra spazi e a-capo.
  - Input in due fasi:
    - `gets` (o `fgets`), per acquisire (come stringa) una riga
    - lettura degli interi dalla stringa (con `sscanf`). PROBLEMA: come iterare `sscanf(riga, "%d", …)` ripartendo ogni volta dal punto in cui si era rimasti ?
      - formato %n (dice quanti caratteri letti)
      - puntatore per identificare sotto-stringa



```
...
int i, x, cnt = 0;
float somma = 0.0;
char riga[MAX], *s;
...
fgets(riga,MAX,stdin);
s=riga;
while (sscanf(s, "%d%n", &x, &i)>0) {
   s = s+i; // oppure s = &s[i];
   somma += x;
   cnt++;
}
printf("La media e': %f\n", somma/cnt);
```



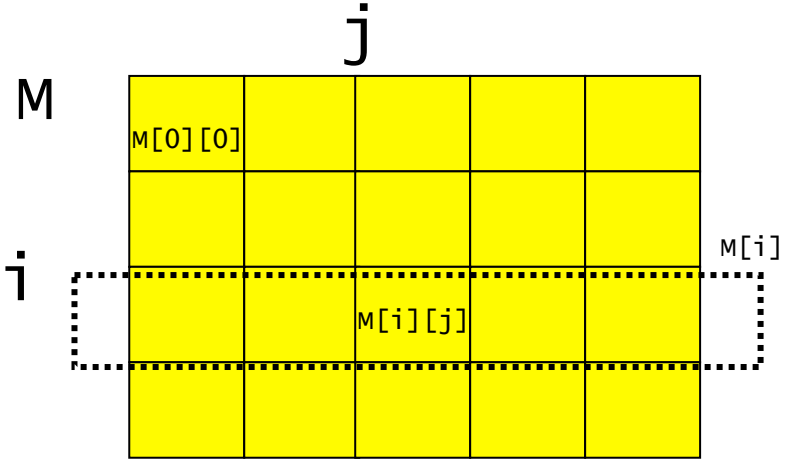
Vettori di puntatori

- Essendo il puntatore un dato
  - possono esistere vettori di puntatori
- Siccome un puntatore può corrispondere ad un vettore
  - Allora un vettore di puntatori può corrispondere a un vettore di vettori (una matrice)

Vettori di vettori e matrici

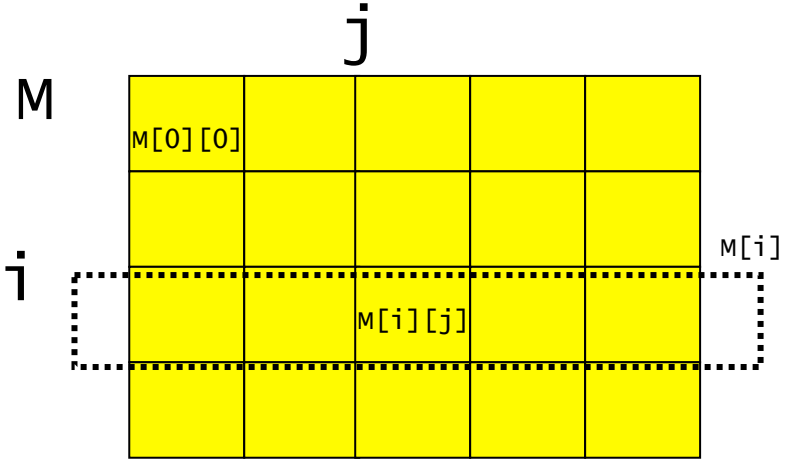
Esempio: matrice come vettore di righe

```
#define NR 4
#define NC 5
float M[NR][NC];
...
```



Esempio: matrice come vettore di righe

```
#define NR 4
#define NC 5
float M[NR][NC];
...
```



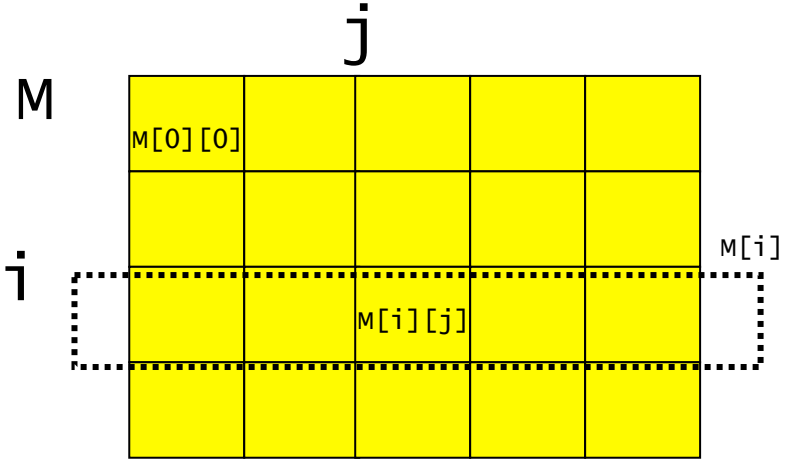

```
printf("dim. matr.: %d\n", sizeof(M));
printf("dim. riga: %d\n", sizeof(M[0]));
printf("dim. elem.: %d\n", sizeof(M[0][0]));
```

80 = NR x NC x sizeof(float)

float su 4 byte

Esempio: matrice com ore di righe

```
#define NR 4
#define NC 5
float M[NR][NC];
...
```

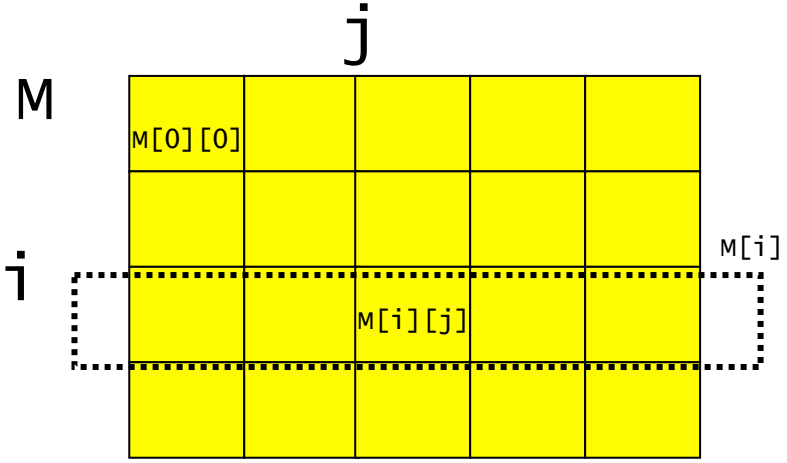



```
printf("dim. matr.: %d\n", sizeof(M));
printf("dim. riga: %d\n", sizeof(M[0]));
printf("dim. elem.: %d\n", sizeof(M[0][0]));
```

```
20 = NC x sizeof(float)
```

Esempio: matrice con ttore di righe

```
#define NR 4
#define NC 5
float M[NR][NC];
...
```

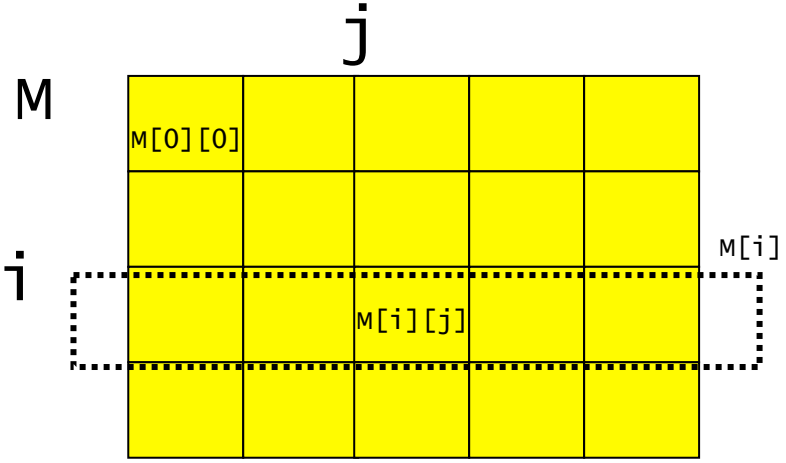



```
printf("dim. matr.: %d\n", sizeof(M));
printf("dim. riga: %d\n", sizeof(M[0]));
printf("dim. elem.: %d\n", sizeof(M[0][0]));
```

```
4 = sizeof(float)
```

Esempio: matrice co vettore di righe

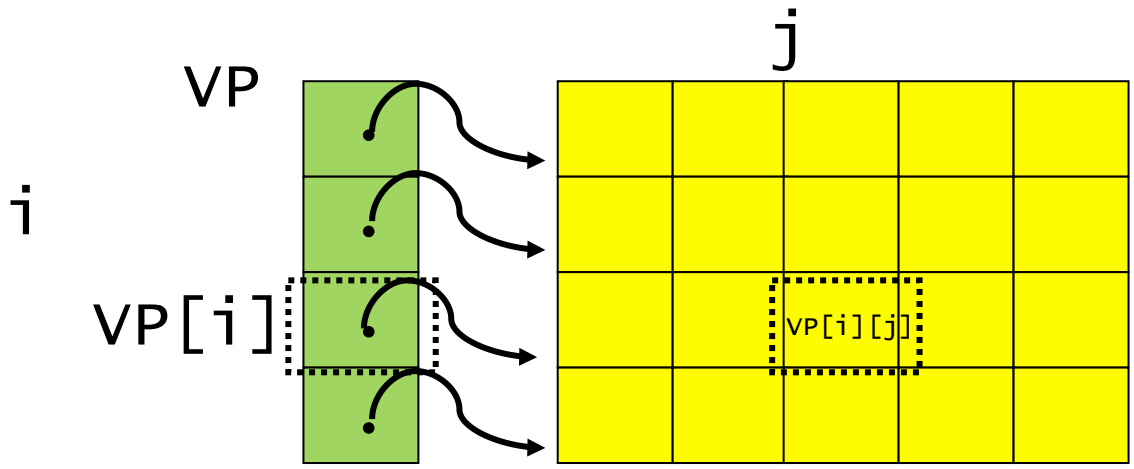
```
#define NR 4
#define NC 5
float M[NR][NC];
...
```




```
printf("dim. matr.: %d\n", sizeof(M));
printf("dim. riga: %d\n", sizeof(M[0]));
printf("dim. elem.: %d\n", sizeof(M[0][0]));
```

Esempio: matrice come vettore di puntatori a vettori

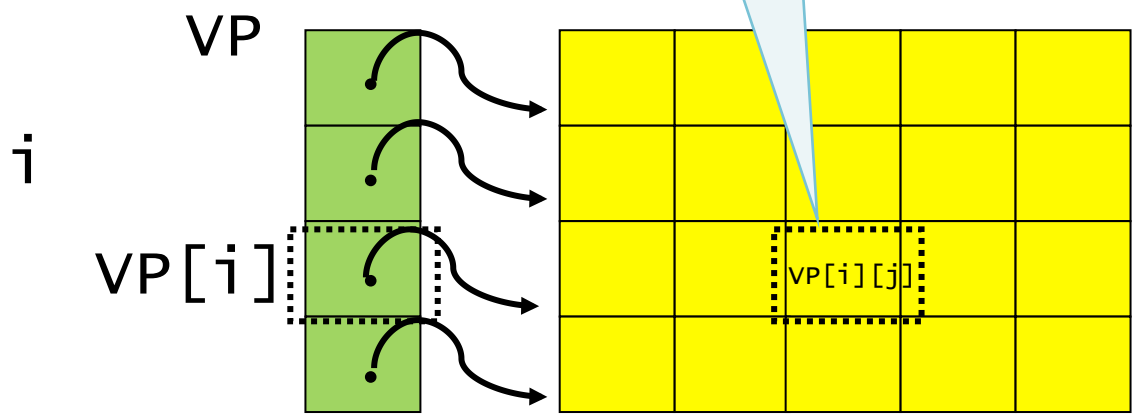
```
#define NR 4
#define NC 5
float R0[NC],R1[NC],R2[NC],R3[NC];
float *VP[NR] = {R0,R1,R2,R3};
...
```



Notazione matriciale per
VP[i][j] ⇔(VP[i])[j]

Esempio: matrice come vet puntatori a vettori

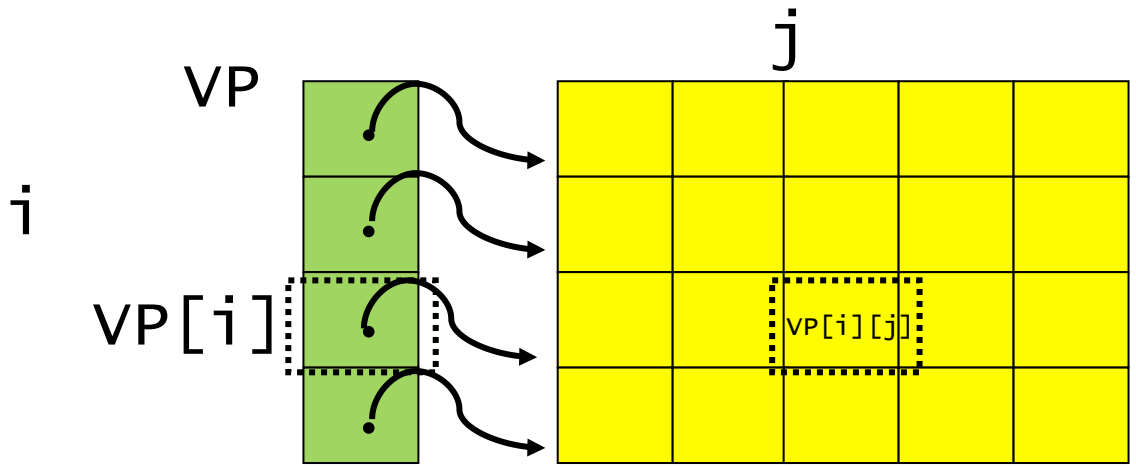
```
#define NR 4
#define NC 5
float R0[NC],R1[NC],R2[NC  3[NC];
float *VP[NR] = {R0,R1,R2,  ;
...
```



puntatore su 8 byte

Dimensioni:

- 32 byte per `VP` (vettore di 4 puntatori)
- 8 byte pe `VP[i]` (è un puntatore)
- 4 byte per `VP[i][j]` (è un float)



Vettori di vettori a dimensione variabile

Grazie ai vettori di puntatori (vettori di vettori) con notazione matriciale si realizzano matrici con righe di dimensione variabile.

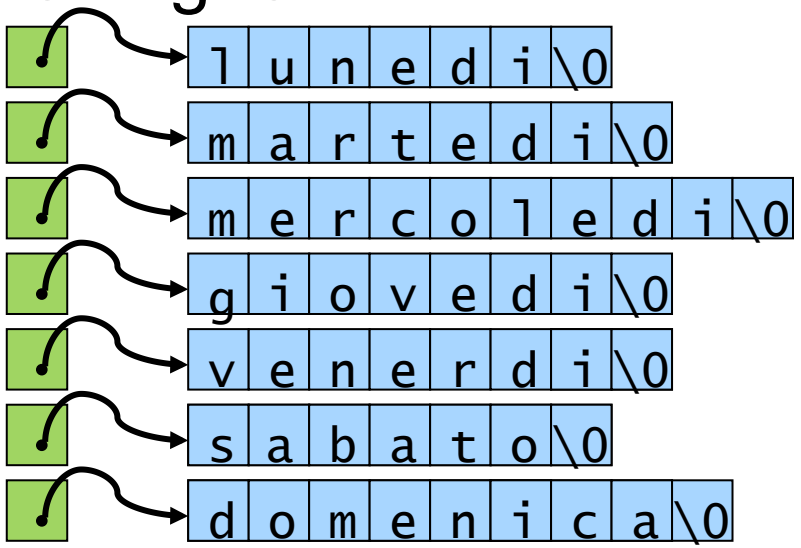
- opportunità sfruttata sovente nei vettori di stringhe, accessibili anche come matrici di caratteri

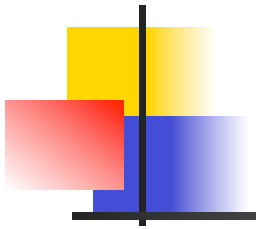
Esempio: vettore con i nomi di giorni della settimana e stampa dell'i-esimo carattere del nome se esiste.

Esempio: vettore con i nomi di giorni della settimana e stampa dell'i-esimo carattere del nome se esiste:

- soluzione 1: matrice di caratteri
- soluzione 2: vettore di stringhe

l	u	n	e	d	i	\0			
m	a	r	t	e	d	i	\0		
m	e	r	c	o	l	e	d	i	\0
g	i	o	v	e	d	i	\0		
v	e	n	e	r	d	i	\0		
s	a	b	a	t	o	\0			
d	o	m	e	n	i	c	a	\0	





Matrice di caratteri

```
void main (void) {
  int i,g;
  char giorni[7][10]={"lunedì","martedì",
                      "mercoledì","giovedì",
                      "venerdì","sabato",
                      "domenica"};
  printf("quale carattere (1-6)? ");
  scanf("%d",&i);
  for (g=0; g<7; g++)
    if (i<strlen(giorni[g]))
      printf("%c ", giorni[g][i-1]);
    else printf("_ ");
  printf("\n");
}
```

Matrice di caratteri

Inizializzazione con costanti stringa

```
void main (void) {
  int i,g;
  char giorni[7][10]={"lunedì","martedì",
                      "mercoledì","giovedì",
                      "venerdì","sabato",
                      "domenica"};
  printf("quale carattere (1-6)? ");
  scanf("%d",&i);
  for (g=0; g<7; g++)
    if (i<strlen(giorni[g]))
      printf("%c ", giorni[g][i-1]);
    else printf("_ ");
  printf("\n");
}
```

Matrice di caratteri

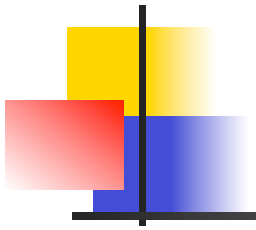
Riga generica identificabile con un solo indice

```
void main (void) {
  int i,g;
  char giorni[7][10]={"l    dì","martedì",
                      "mer    edì","giovedì",
                       "ve    dì","sabato",
                       "do    nica"};
  printf("quale carat    e (1-6)? ");
  scanf("%d",&i);
  for (g=0; g<7; g++)
    if (i<strlen(giorni[g]))
      printf("%c ", giorni[g][i-1]);
    else printf("_ ");
  printf("\n");
}
```

Matrice di caratteri

Singolo carattere identificabile con due indici

```
void main (void) {
  int i,g;
  char giorni[7][10]={"lun      ","martedì",
                      "mercol    ","giovedì",
                       "venerd   "sabato",
                       "domenic  };
  printf("quale carattere (1-  )? ");
  scanf("%d",&i);
  for (g=0; g<7; g++)
    if (i<strlen(giorni[g]))
      printf("%c ", giorni[g][i-1]);
    else printf("_ ");
  printf("\n");
}
```



Vettore di puntatori a stringa

```
void main (void) {
  int i,g;
  char *giorni[7]={"lunedì","martedì",
                   "mercoledì","giovedì",
                   "venerdì","sabato",
                   "domenica"};
  printf("quale carattere (1-6)? ");
  scanf("%d",&i);
  for (g=0; g<7; g++)
    if (i<strlen(giorni[g]))
      printf("%c ", giorni[g][i-1]);
    else printf("_ ");
  printf("\n");
}
```

Vettore di puntatori a char

Vettore di puntatori a string

```
void main (void) {
  int i,g;
  char *giorni[7]={"lunedì","martedì",
                   "mercoledì","giovedì",
                   "venerdì","sabato",
                   "domenica"};
  printf("quale carattere (1-6)? ");
  scanf("%d",&i);
  for (g=0; g<7; g++)
    if (i<strlen(giorni[g]))
      printf("%c ", giorni[g][i-1]);
    else printf("_ ");
  printf("\n");
}
```

Inizializzazione con puntatori a stringhe (costanti)

Vettore di puntatori a stringa

```
void main (void) {
  int i,g;
  char *giorni[7]={"lunedì","martedì",
                   "mercoledì","giovedì",
                   "venerdì","sabato",
                   "domenica"};
  printf("quale carattere (1-6)? ");
  scanf("%d",&i);
  for (g=0; g<7; g++)
    if (i<strlen(giorni[g]))
      printf("%c ", giorni[g][i-1]);
    else printf("_ ");
  printf("\n");
}
```

Vettore di puntatori a stringa

giorni[g]: stringa di ordine g

```
void main (void) {
  int i,g;
  char *giorni[7]={"lunedì","martedì",
                   "mercoledì","giovedì",
                   "venerdì","sabato",
                   "domenica"};
  printf("quale carattere (1-6)? ");
  scanf("%d",&i);
  for (g=0; g<7; g++)
    if (i<strlen(giorni[g]))
      printf("%c ", giorni[g][i-1]);
    else printf("_ ");
  printf("\n");
}
```

Vettore di puntatori a stringa

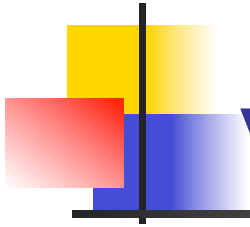
```
void main (void) {
  int i,g;
  char *giorni
                          "venerdì",      o",
                          "domenica"
  printf("quale carattere (1-6      );
  scanf("%d",&i);
  for (g=0; g<7; g++)
    if (i<strlen(giorni[g]))
      printf("%c ", giorni[g][i-1]);
    else printf("_ ");
  printf("\n");
}
```

giorni[g][i-1]: i-esimo carattere della stringa di ordine g

Vettore di puntatori a stringa

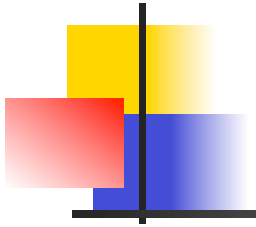
```
void main (void) {
  int i,g;
  char *giorn
                        "venerdì","
                        "domenica"}
  printf("quale carattere (1-6
  scanf("%d",&i);
  for (g=0; g<7; g++)
    if (i<strlen(giorni[g]))
      printf("%c ", giorni[g][i-1]);
    else printf("_ ");
  printf("\n");
}
```

giorni[g][i-1]: vettore di stringhe utilizzato come matrice di caratteri



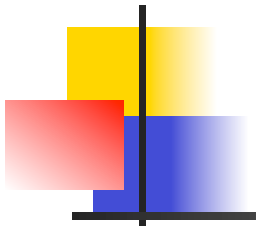
Vettore di stringhe

- Un vettore di stringhe può essere realizzato come:
  - matrice di caratteri: vettore bidimensionale (righe, colonne). Le righe hanno tutte la stessa lunghezza (vanno sovradimensionate sulla stringa più lunga)
  - vettore di puntatori a stringhe: ogni elemento del vettore punta a una stringa distinta. Le stringhe possono avere lunghezze diverse
- Con entrambi i metodi si può utilizzare la notazione matriciale



Esempio: ordinamento di stringhe

- Leggere da tastiera delle stringhe:
  - al massimo 20
  - ognuna al massimo di 50 caratteri
  - la somma delle lunghezze delle stringhe è <= 500
  - l'input termina con una stringa vuota
- Ordinare le stringhe in ordine crescente (secondo `strcmp`)
- Visualizzarle (secondo l'ordine precedente) su video

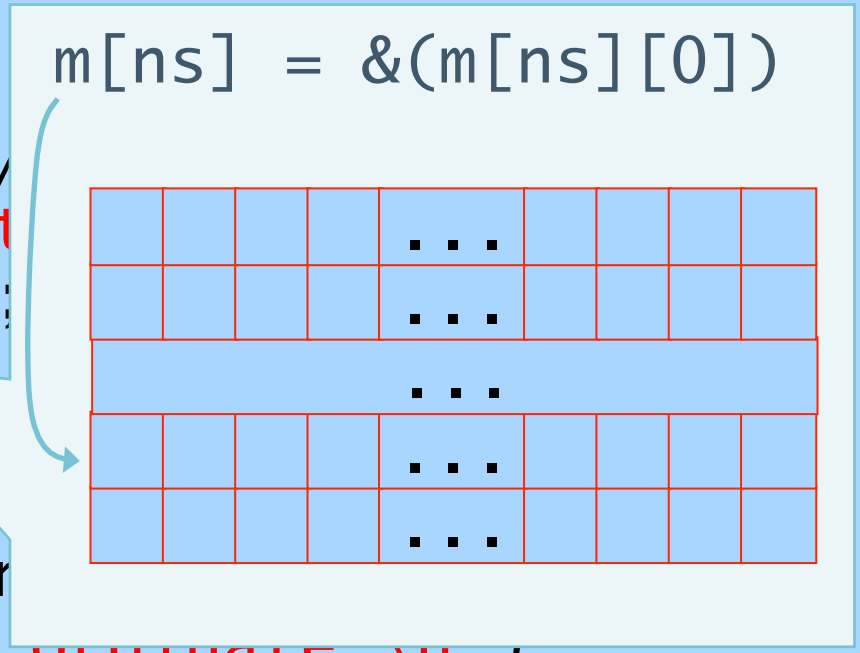


Matrice di caratteri

```
void main (void){
   int i,ns;
   char m[20][51]; // 51 colonne per i \0
   printf("scrivi stringhe:\n");
   for (ns=0; ns<20; ns++) {
      gets(m[ns]);
      if (strlen(m[ns])==0) break;
   }
   ordinaMatrice(m,ns);
   printf("stringhe ordinate:\n");
   for (i=0; i<ns; i++)
      printf("%s\n", m[i]);
}
```

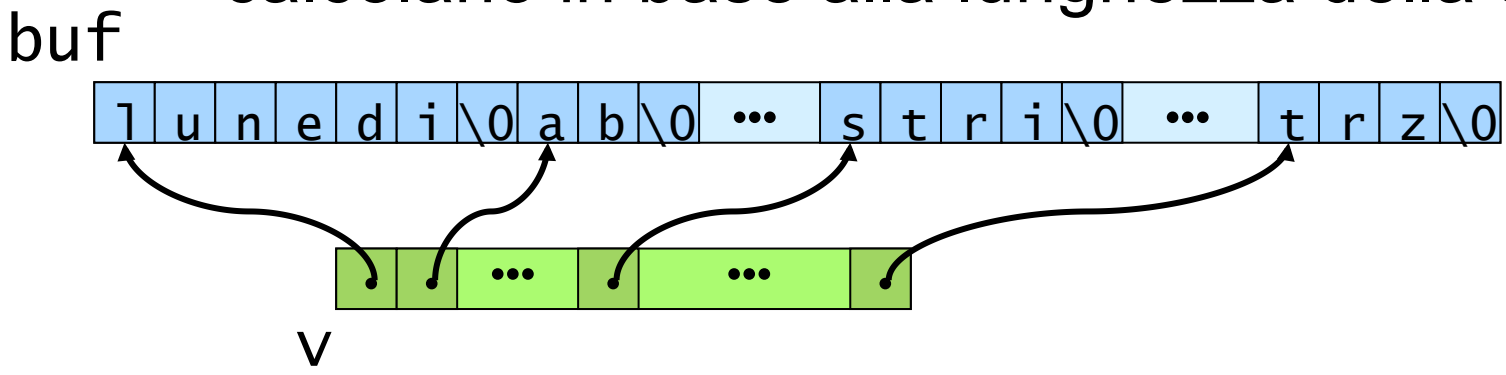
```
void main (void){
   int i,ns;
   char m[20][51]; // 51 colonne per i \0
   printf("scrivi stringhe:\n");
   for (ns=0;
      gets(m[ns
      if (strle
   }
   ordinaMatri
   printf("str
   for (i=0; i
      printf("%
}
```

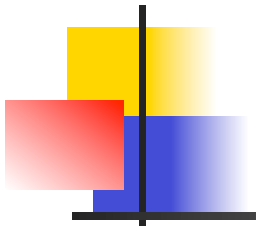
```
void main (void){
   int i,ns;
   char m[20][51]; /
   printf("scrivi st
   for (ns=0; ns<20;
      gets(m[ns]);
      if (strlen(m
   }
   ordinaMatrice(m,n
   printf("stringhe ordinate:\n");
   for (i=0; i<ns; i++)
      printf("%s\n", m[i]);
}
```



Vettore
Doppio livello di memorizzazione

- vettore `buf` di 520 elementi che contiene le stringhe (terminatore incluso) una dopo l'altra
- vettore `v` di 20 puntatori al primo carattere di ogni stringa
  - `v[0]` coincide con `buf`, gli altri puntatori si calcolano in base alla lunghezza della stringa

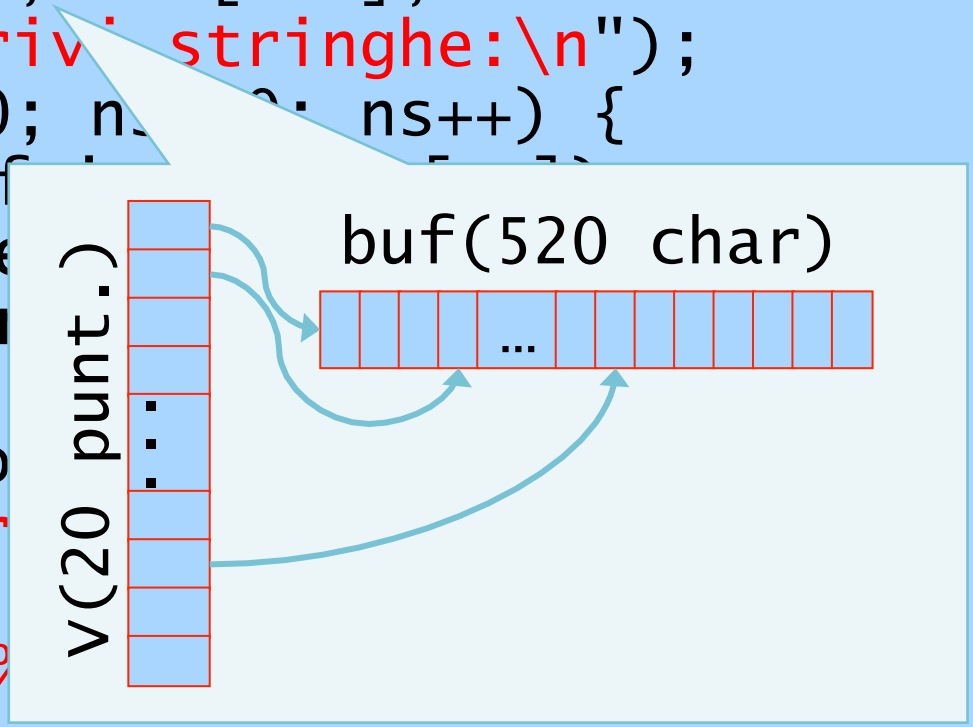




Vettore

```
void main (void) {
   int i,ns;
   char *v[20], buf[520];
   printf("scrivi stringhe:\n");
   for (ns=i=0; ns<20; ns++) {
      v[ns]=buf+i; gets(v[ns]);
      if (strlen(v[ns])==0) break;
      i = i+strlen(v[ns])+1;
   }
   ordinaVettore(v,ns);
   printf("stringhe ordinate:\n");
   for (i=0; i<ns; i++)
      printf("%s\n", v[i]);
}
```

```
void main (void) {
   int i,ns;
   char *v[20], buf[520];
   printf("scrivi stringhe:\n");
   for (ns=i=0; n        ns++) {
      v[ns]=buf
      if (strle
      i = i+str
   }
   ordinaVetto
   printf("str
   for (i=0; i
      printf("%
}
```

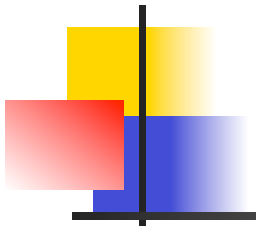


```
void main (void) {
   int i,ns;
   char *v[20], buf[520];
   printf("scrivi stringhe:\n");
   for (ns=i=0; ns<20; ns++) {
      v[ns]=buf+i; gets(v[ns]);
      if (strlen(v[ns])==0) break;
      i = i+strlen(v[ns])+1;
   }
   ordinaVettore(v,ns);
   printf("stringhe ordinate:\n");
   for (i=0; i<ns; i++)
      printf("%s\n", v[i]);
}
```

buf+i = &buf[i]

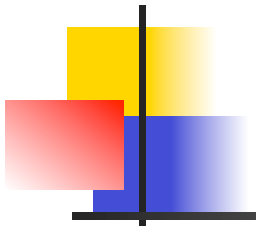
```
void main (void) {
   int i,ns;
   char *v[20], buf[520];
   printf("scrivi stringhe:\n");
   for (ns=i=0; ns<20; ns++) {
      v[ns]=buf+i; gets(v[ns]);
      if (strlen(v[ns])==0) break;
      i = i+strlen(v[ns])+1;
   }
   ordinaVettore(
   printf("string
   for (i=0; i<ns
      printf("%s\n", v[i]);
}
```

Avanza in buf saltando stringa corrente più terminatore ('\0')



Confronto tra le soluzioni:

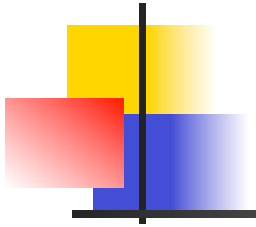
- Matrice di caratteri
  - 20 righe: massimo numero di stringhe
  - 51 colonne: massima lunghezza di stringa
  - 20*51 = 1020 caratteri: dimensione matrice
- Vettore di puntatori a stringhe
  - 20 puntatori: dimensione vettore di puntatori
  - 520 caratteri: dimensione di caratteri contenente le stringhe (500 caratteri per le stringhe + 20 terminatori)



Confronto tra le soluzioni:

- Matrice di caratteri
  - 20 righe: massimo numero di stringhe
  - 51 colonne: massima lunghezza di stringa
  - 20*51 = 1020 caratteri: dimensione matrice
- Vettore di puntatori a stringhe
  - 20 punt[illegible]ri: dimensione vettore di puntatori
  - 520 cara[illegible]ensione di caratteri contenente le[illegible]) terminatori)

20 puntatori + 520 caratteri < 1020 caratteri !

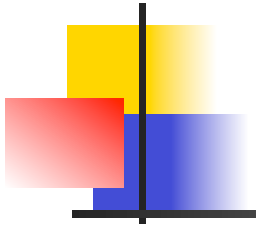


Ordinamento con matrice

```
void ordinaMatrice (char m[][51], int n) {
  int i, j, min; char tmp[51];
  for (i=0; i<n-1; i++) {
    min = i;
    for (j=i+1; j<n; j++)
      if (strcmp(m[min],m[j])>0)
        min = j;
    strcpy(tmp,m[i]);
    strcpy(m[i],m[min]);
    strcpy(m[min],tmp);
  }
}
```

```
void ordinaMatrice (char m[][51], int n) {
  int i, j, min; char tmp[51];
  for (i=0; i<n-1; i++) {
    min = i;
    for (j=i+
      if (str
        min = j;
    strcpy(tmp,m[i]);
    strcpy(m[i],m[min]);
    strcpy(m[min],tmp);
  }
}
```

Con la matrice di caratteri gli scambi di stringa sono realizzati mediante copia



Ordinamento con vettore

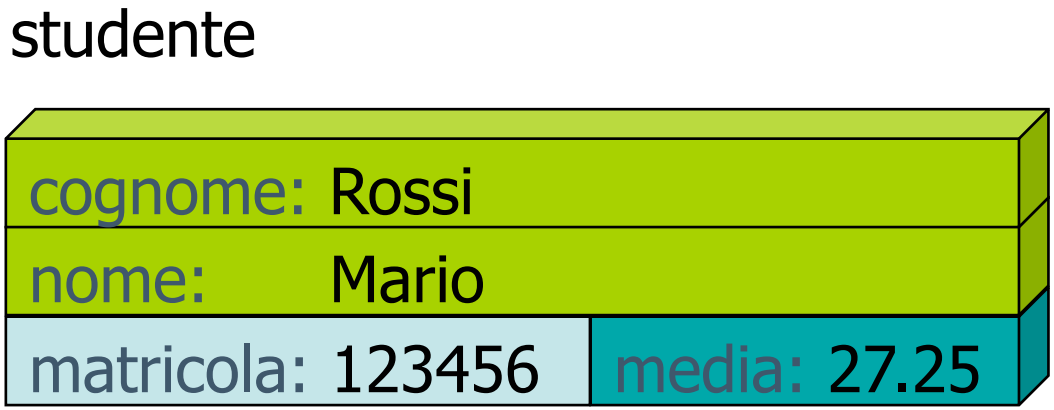
```
void ordinaVettore (char *m[], int n){
  int i, j, min; char *tmp;
  for (i=0; i<n-1; i++) {
    min = i;
    for (j=i+1; j<n; j++)
      if (strcmp(m[min],m[j])>0)
        min = j;
    tmp = m[i];
    m[i] = m[min];
    m[min] = tmp;
  }
}
```

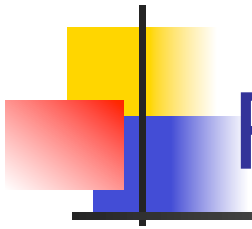
```
void ordinaVettore (char *m[], int n){
  int i, j, min; char *tmp;
  for (i=0; i<
    min = i;
    for (j=i+1
      if (strcmp(m
        min = j;
    tmp = m[i];
    m[i] = m[min];
    m[min] = tmp;
  }
}
```

Con il vettore di puntatori gli scambi di stringa sono realizzati mediante scambio di puntatori

`Struct`, puntatori e vettori

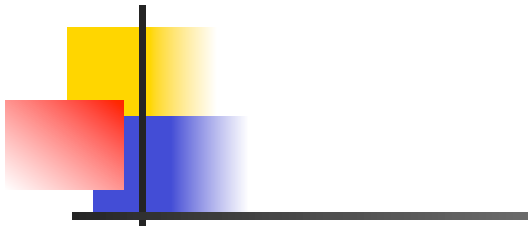
- Più informazioni eterogenee possono essere unite come parti (campi) di uno stesso dato dato (aggregato)





Richiami sui tipi `struct`

- Il dato aggregato in C è detto `struct`. In altri linguaggi si parla di `record`
- Una `struct` (struttura) è un dato costituito da campi:
  - i campi sono di tipi (base) noti (eventualmente altre `struct` o puntatori)
  - ogni campo all'interno di una `struct` è accessibile mediante un identificatore (anziché un indice, come nei vettori)



Esempio:

```
struct studente {
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
```

```
struct studente {
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
```

Nuovo tipo di dato

- Il nuovo tipo definito è `struct studente`
- La parola chiave `struct` è obbligatoria

```
struct studente {
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
```

Nuovo tipo di dato

Nome del tipo aggregato

- Stesse regole che valgono per i nomi delle variabili
- I nomi di `struct` devono essere diversi da nomi di altre struct (possono essere uguali a nomi di variabili)

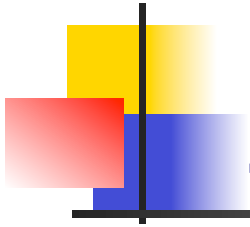
```
struct studente{
   char cognome[MAX], nome[MAX];
   int matricola;
   float media;
};
```

Campi (eterogenei)

Nuovo tipo di dato

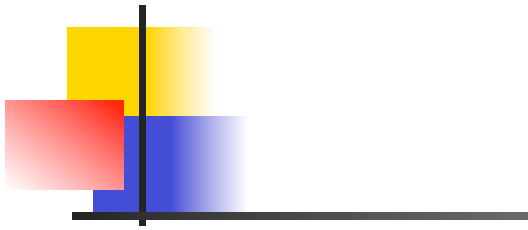
Nome del tipo aggregato

- I campi corrispondono a variabili locali di una `struct`
- Ogni campo è quindi caratterizzato da un tipo (base) e da un identificatore (unico per la struttura)



`struct` e vettori

- Analogia:
  - sono entrambi tipi di dati aggregati
- Differenze:
  - dati eterogenei (`struct`) / omogenei (vettori)
  - accesso per nome (`struct`) / indice (vettori)
  - parametri per valore (`struct`) / per riferimento (vettori)
  - accesso parametrizzato non ammesso (`struct`) / ammesso (vettori)



Accesso parametrizzato a vettori:

- i vettori sono frequentemente utilizzati per accesso parametrizzato a dati numerati (es. in costrutti iterativi)

Esempio

```
for (i=0; i<N; i++) {
   ...
   dati[i] = ...;
   ...
}
```

Accesso parametrizzato a `struct`:

- ai campi di una `struct` NON si può accedere in modo parametrizzato

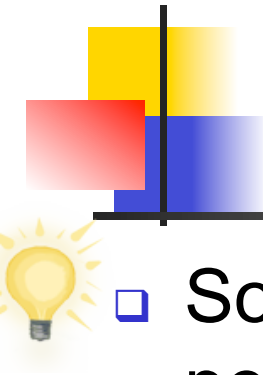

```
char campo[20];
...
scanf("%s",campo);
printf("%s",s.campo);
```

Accesso parametrizzato a `struct`:

- ai campi di una `struct` NON si può accedere in modo parametrizzato

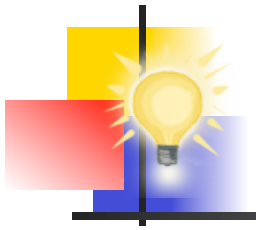

```
char campo[20];
...
scanf("%s",campo);
printf("%s",s.campo);
```

campo è una variabile !

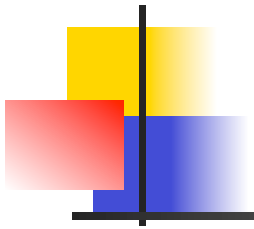


- Soluzione: utilizzare una funzione (da realizzare) per il passaggio da identificatore (variabile) di campo a struttura

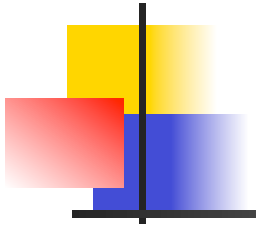
```
char campo[20];
...
scanf("%s",campo);
stampaCampo(s,campo);
```



```
void stampaCampo(
   struct studente s, char id[]) {
   if (strcmp(id,”cognome”)==0)
      printf("%s",s.cognome);
   else if(strcmp(id,”nome”)==0)
      printf("%s",s.nome);
   else if(strcmp(id,”matricola”)==0)
      printf("%d",s.matricola);
   ...
}
```



- Per dati omogenei (es. punto come elenco delle coordinate, numero complesso, …)
- meglio `struct` o vettore ?
  - Soluzione 1: `struct` consigliata se:
    - pochi campi
    - meglio identificarli per nome
    - non serve accesso parametrizzato
    - si vuole poter trattare la `struct` come un unico dato (es. per assegnazioni a variabili, parametro unico o valore di ritorno da funzioni)

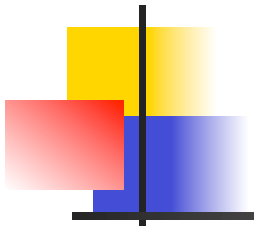


Vettori come campi di `struct`:

- Una `struct` può avere uno o più vettori come campi

  Esempio: cognome e nome in `struct` studente

- Attenzione!
  - una `struct` viene passata per valore a una funzione, mentre un vettore sarebbe passato per riferimento
  - se una `struct` ha come campo un vettore di N elementi, passare la `struct` come parametro richiede tempo O(N) (per copiare dati)



Soluzione 2: vettore di `struct` consigliato per:

- collezioni (numerabili) di aggregati eterogenei
  - Esempio: gestione di un elenco di studenti
- Attenzione!
  - una vettore (anche se di `struct`) viene passato per riferimento a una funzione
  - una funzione può quindi modificare il contenuto di un vettore (ad esempio ordinare i dati)

```
int main(void) {
  struct studente elenco[NMAX];
  int i, n;

  printf("quanti studenti(max %d)? ",NMAX);
  scanf("%d",&n);
  for (i=0; i<n; i++) {
    elenco[i] = leggiStudente();
  }
  ordinaStudenti(elenco,n);
  printf("studenti ordinati per media\n");
  for (i=0; i<n; i++) {
    stampaStudente(elenco[i]);
  }
}
```

```
int main(void) {
  struct studente elenco[NMAX];
  int i, n;

  printf("quanti studenti(max %d)? ",NMAX);
  scanf("%d",&n);
  for (i=0; i<n; i++) {
    elenco[i] = leggiStudente();
  }
  ordinaStudenti(elenco,n);
  printf("studenti ordinati per media\n");
  for (i=0; i<n; i++) {
    stampaStudente(elenco[i]);
  }
}
```

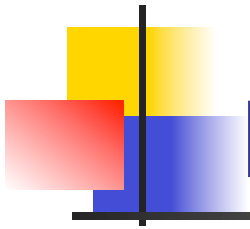
vettore di `struct`

```
int main(void) {
  struct studente elenco[NMAX];
  int i, n;

  printf("quanti stud      (max %d)? ",NMAX);
  scanf("%d",&n);
  for (i=0; i<n; i++)  {
    elenco[i] = lege Studente();
  }
  ordinaStudenti(elenco,n);
  printf("studenti ordinati per media\n");
  for (i=0; i<n; i++) {
    stampaStudente(elenco[i]);
  }
}
```

Passaggio per riferimento

```
int main(void) {
   struct studente elenco[NMAX];
   int i, n;

   printf("quanti studenti(max %d)? ",NMAX);
   scanf("%d",&n);
   for (i=0; i<n; i++) {
      elenco[i] = leggiStude
   }
   ordinaStudenti(elenco,n
   printf("studenti ordina i per media\n");
   for (i=0; i<n; i++) {
      stampaStudente(elenco[i]);
   }
}
```

Accesso al singolo dato

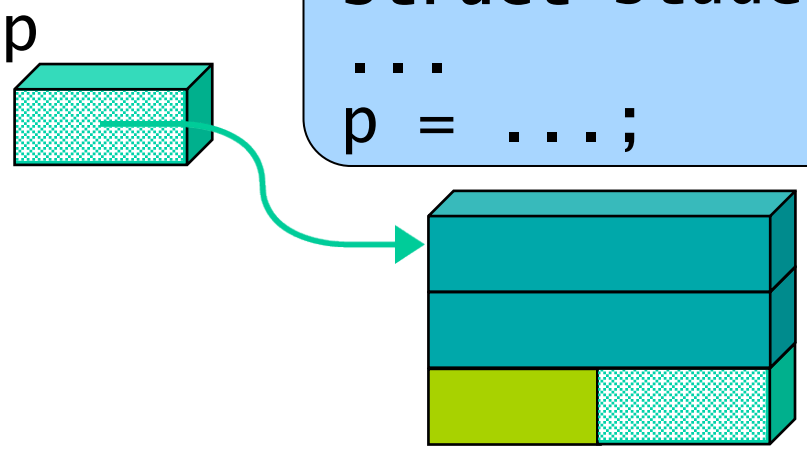


Puntatore a `struct`

- Per accedere a una struttura (tipo `struct`), utilizzando puntatori, si utilizzano le stesse regole viste per gli altri tipi
- Si noti che un puntatore può:
  - puntare a una struttura intera
  - puntare a un campo di struttura
  - essere un campo di una struttura

Puntatore a struttura (intera):

```
struct studente{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
...
struct studente *p;
...
p = ...;
```

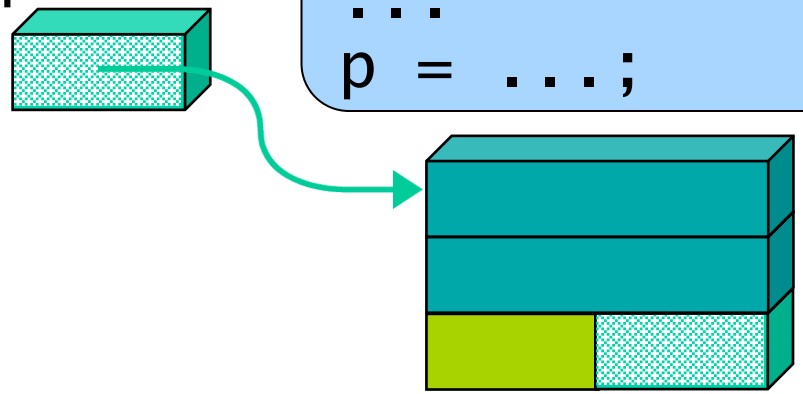


Puntatore a struttura (intera):

```
struct studente{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
...
struct studente *p;
...
p = ...;
```

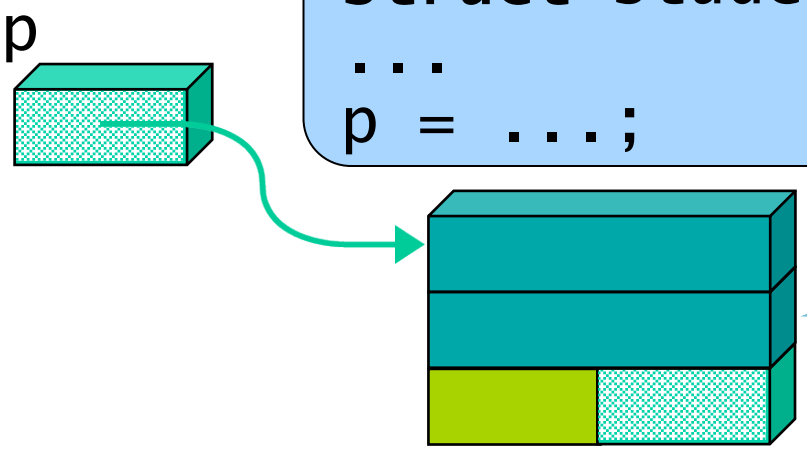
Variabile puntatore

p



Puntatore a struttura (intera):

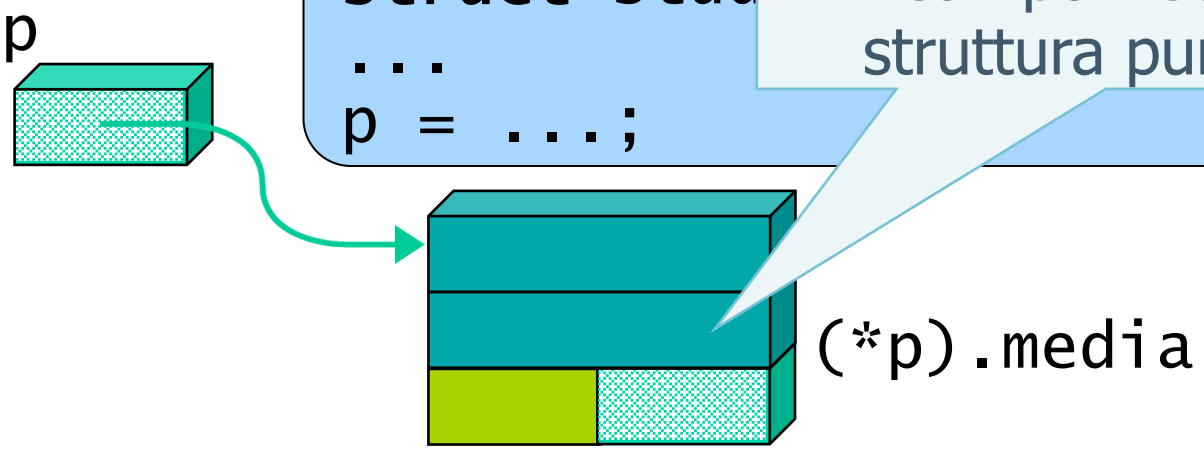
```
struct studente{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
...
struct studente *p;
...
p = ...;
```



Puntatore a struttura (intera):

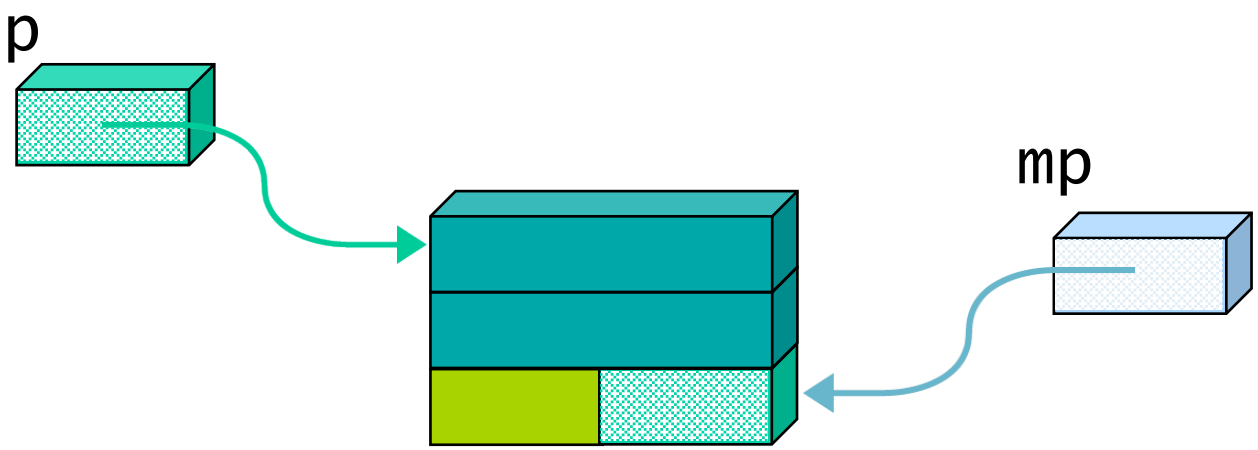
```
struct studente{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
...
struct stud
...
p = ...;
```

Campo media della struttura puntata da p



Puntatore a campo di struttura

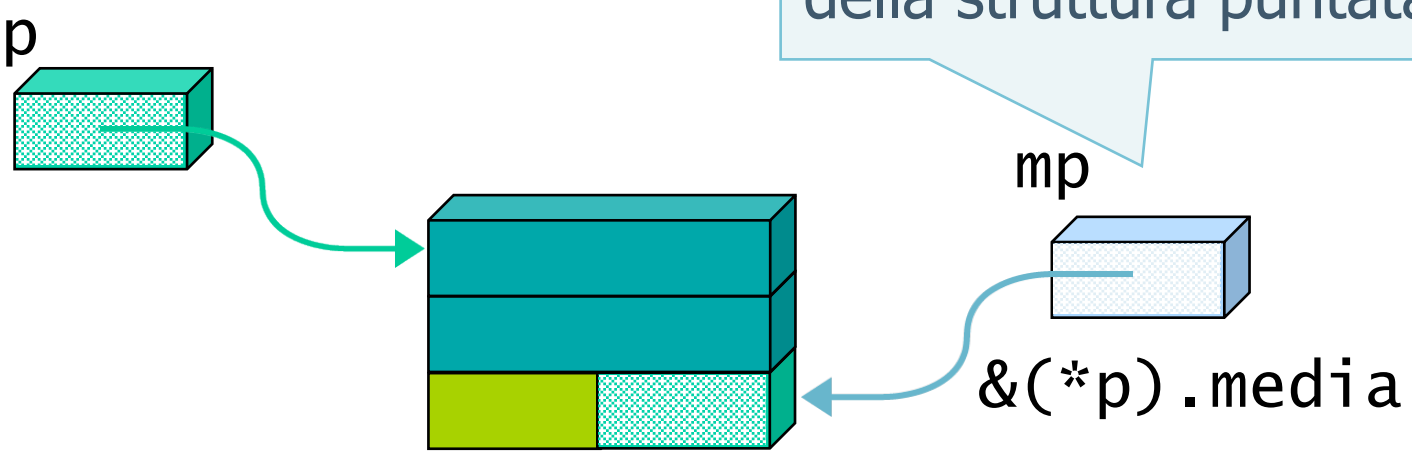
```
...
struct studente *p;
float *mp;
...
p = ...;
mp = &(*p).media;
```



Puntatore a campo di struttura

```
...
struct studente *p;
float *mp;
...
p = ...;
mp = &(*p).
```

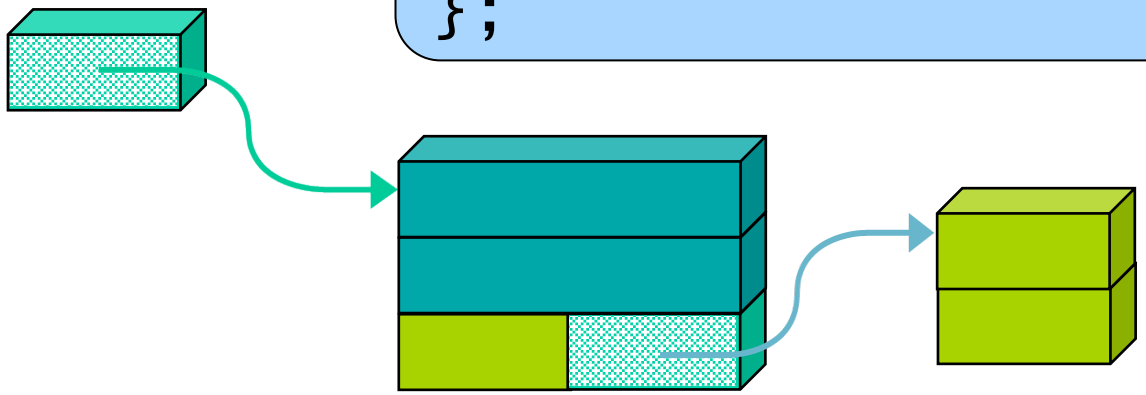
Puntatore a campo media della struttura puntata da p



Puntatore come campo di struttura

```
struct esame {
    int scritto, orale;
};
struct studente {
    char cognome[MAX], nome[MAX];
    int matricola;
    struct esame *es;
};
```

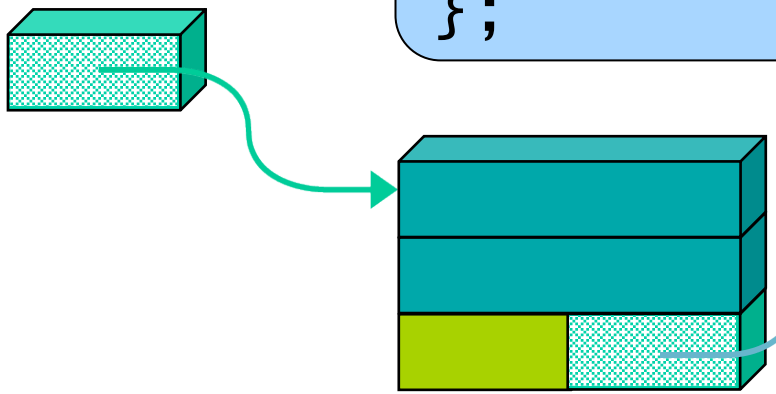
p



Puntatore come campo di struttura

```
struct esame {
    int scritto, orale;
};
struct studente {
    char cognome[MAX], nome[MAX];
    int matricola;
    struct esame *es;
};
```

p

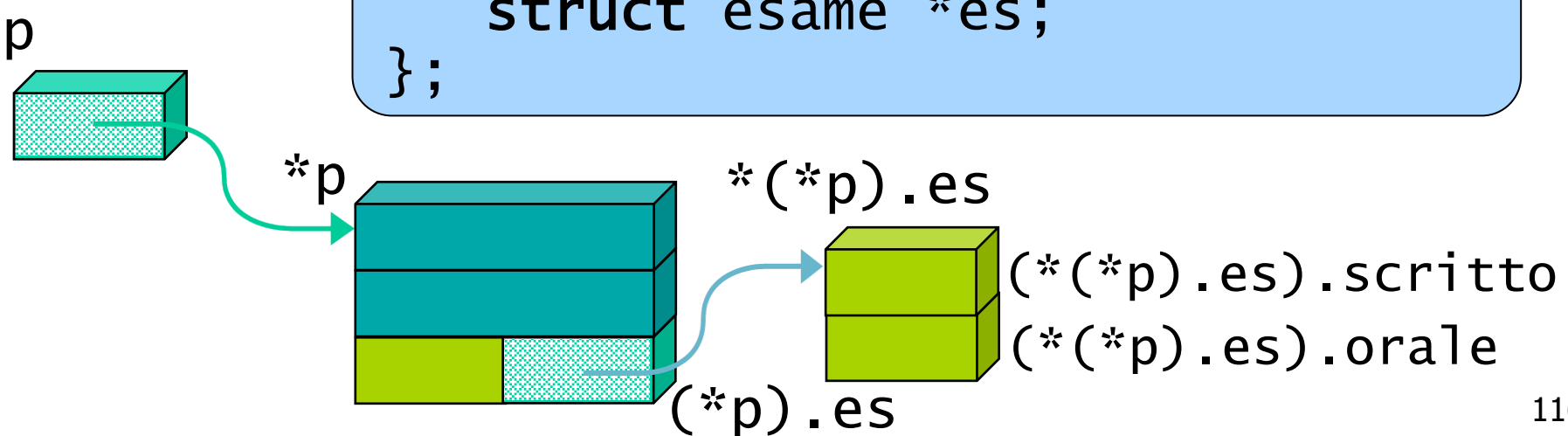


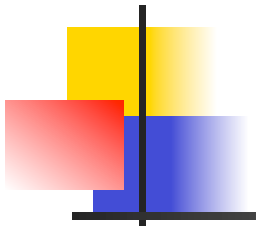
Attenzione: NON è una struttura interna:

```
struct esame es;
```

Puntatore come campo di struttura

```
struct esame {
    int scritto, orale;
};
struct studente {
    char cognome[MAX], nome[MAX];
    int matricola;
    struct esame *es;
};
```





Accesso a struttura puntata:

- Il C dispone di una notazione alternativa (compatta) per rappresentare i campi di una struttura puntata
  - Anzichè

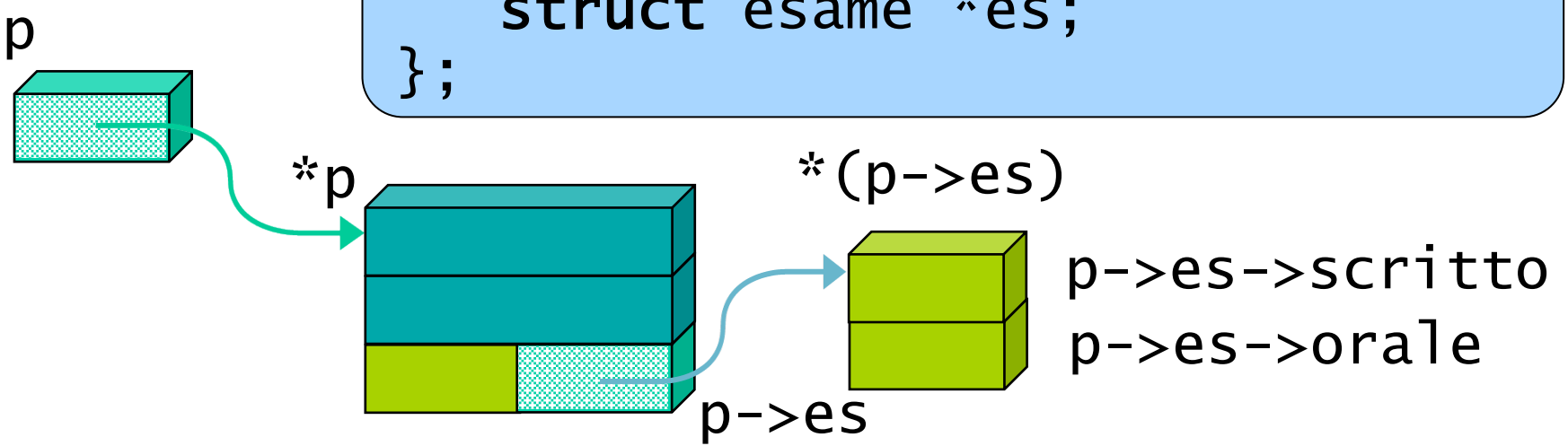
```
(*p).media
(*(*p).esame).scritto
```

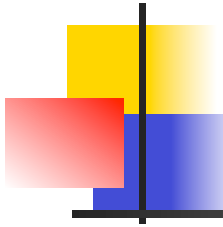
  - Si può scrivere

```
p->media
p->esame->scritto
```

Puntatore come campo di struttura

```
struct esame {
    int scritto, orale;
};
struct studente {
    char cognome[MAX], nome[MAX];
    int matricola;
    struct esame *es;
};
```





Struct ricorsive

- Si dice **ricorsiva** una `struct` che include tra i suoi campi uno o più puntatori a strutture dello stesso tipo
- Servono per realizzare liste, alberi, grafi

Esempio: soluzione 1

```
struct studente {
  char cognome[MAX], nome[MAX];
  int matricola;
  struct studente *link;
};
```

Struct ricorsive

- Si dice **ricorsiva** una `struct` che include tra I suoi campi uno o più puntatori a strutture dello stesso tipo
- Servono per realizzare liste, alberi, grafi

Esempio: soluzione 1

```
struct studente {
  char cognome[MAX], nome[MAX];
  int matricola;
  struct studente *link;
};
```

campo `link` di tipo puntatore a `struct studente`

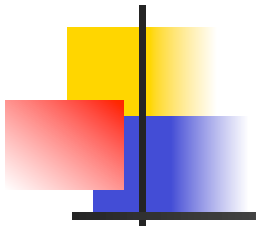
Struct ricorsive

- Si dice **ricorsiva** una `struct` che include tra I suoi campi uno o più puntatori a strutture dello stesso tipo
- Servono per realizzare liste, alberi, grafi

Esempio: soluzione 1

```
struct studente {
  char cognome[MAX], nome[MAX];
  int matricola;
  struct studente *link;
};
```

campo di dimensione nota, essendo un puntatore

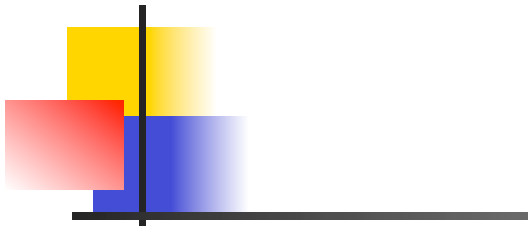


Esempio: soluzione 2

nuovo tipo puntatore a `struct studente` di dimensione nota

```
typedef struct studente *P_stud;
struct studente {
  char cognome[MAX], nome[MAX];
  int matricola;
  P_stud link;
};
```

campo `link` di tipo `P_stud`



Esempio: soluzione 3

nuovo tipo `T_stud` di dimensione ignota

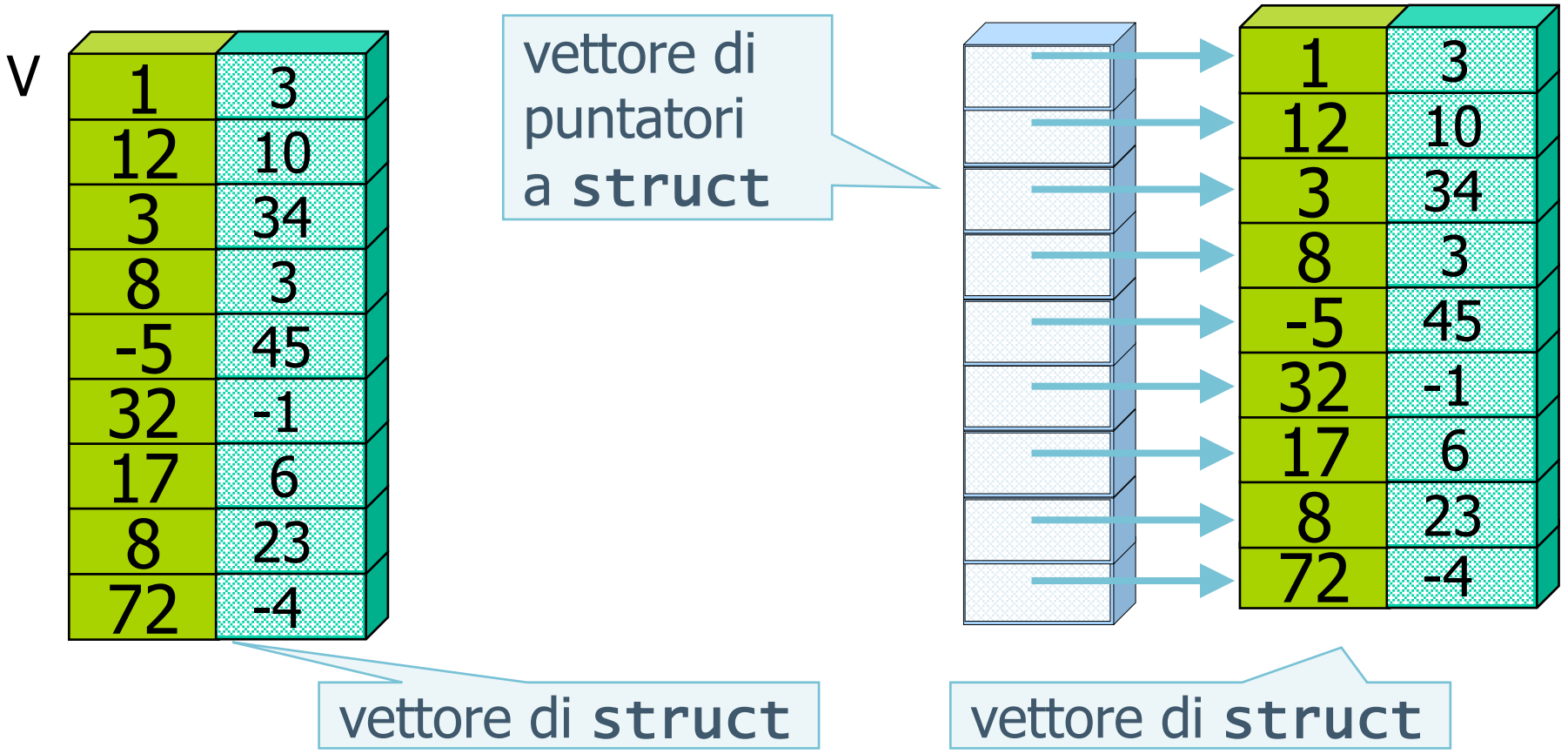
```
typedef struct studente T_stud;
struct studente {
  char cognome
  int matricol
  T_stud *link
};
```

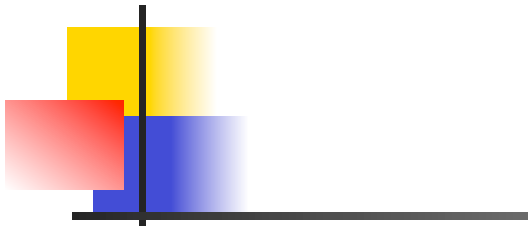
non sono lecite (prima di aver definito struct studente) variabili (o campi) di tipo `T_stud` in quanto di dimensione ignota

campo `link` di tipo puntatore a `T_stud`, quindi di dimensione nota essendo puntatore

Vettori di puntatori a `struct`

- Un vettore di `struct` è diverso da un vettore di puntatori a `struct`

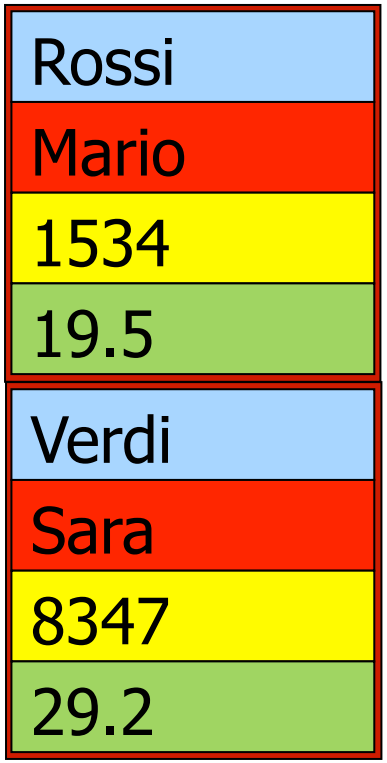




- Esempio:

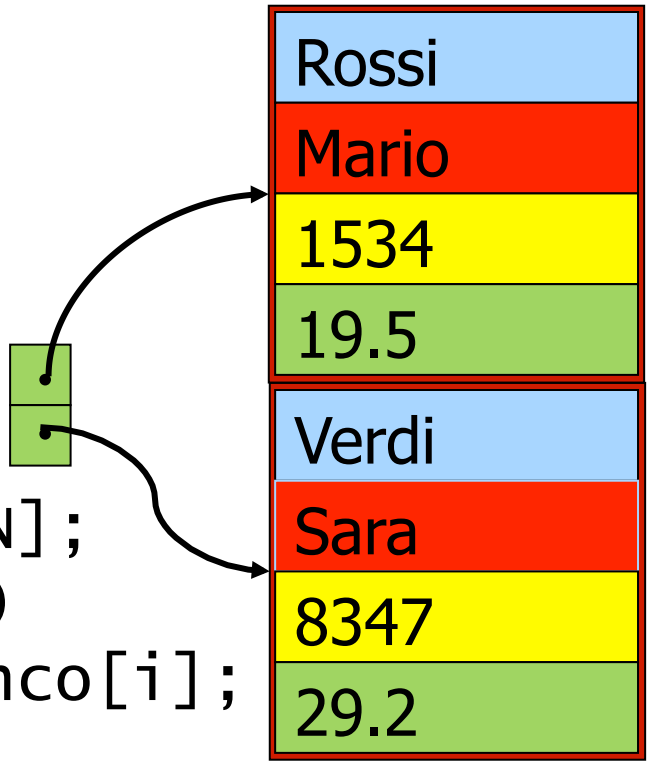
```
typedef struct studente {
  char cognome[MAXS];
  char nome[MAXS];
  int matr;
  float media;
} stud_t;
```

Vettore di `struct`




```
stud_t elenco[MAXN];
```

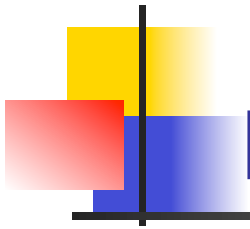
Vettore di puntatori a `struct`




```
stud_t *elencoRif[MAXN];
for (i=0; i<MAXN; i++)
  elencoRif[i] = &elenco[i];
```

Vettore di `struct`

```
stud_t elenco[MAXN];
```



Esempio

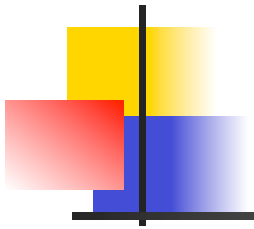
- Scrivere un programma che:
  - acquisisce un elenco di studenti da un file il cui nome è ricevuto come primo argomento al main
  - ordina l'elenco per numeri di matricola crescenti
  - scrive l'elenco su un secondo file, il cui nome è ricevuto come secondo argomento.

- Soluzione 1: vettore di `struc`
- Scambio di valori

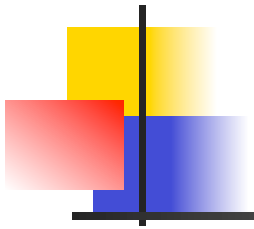
dimensione massima del vettore

```
int main(int argc, char *argv[]) {
  stud_t elenco[MAXN];
  int ns = leggiStud(argv[1],elenco,MAXN);
  ordStudPerMatr(elenco,ns);
  lavoroSuElencoOrd(elenco,ns);
  scriviStud(argv[2],elenco,ns);
  return
}
```

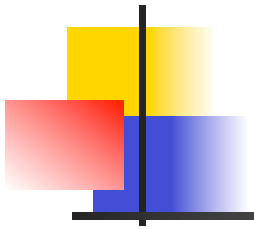
dimensione effettivamente utilizzata del vettore



```
int leggiStud(char *nomeFile, stud_t *el,
              int nmax) {
  int n;
  FILE *fp = fopen(nomeFile,"r");
  for (n=0; n<nmax; n++) {
    if (fscanf(fp,"%s%s%d%f", el[n].cognome,
                  el[n].nome, &el[n].matr,
                  &el[n].media)==EOF) break;
  }
  fclose(fp);
  return n;
}
```



```
void scriviStud(char *nomeFile, stud_t *el,
                int n) {
  int i;
  FILE *fp = fopen(nomeFile,"w");
  for (i=0; i<n; i++) {
    fprintf(fp, "%s %s %d %f\n", el[i].cognome,
                el[i].nome, el[i].matr,
                el[i].media);
 }
 fclose(fp);
}
```



passaggio by value

```
int confrMatr(stud_t s1, stud_t s2) {
  return s1.matricola-s2.matricola;
}
```

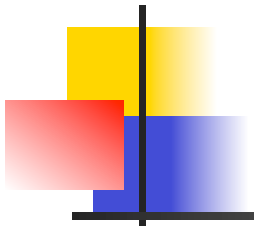
```
int confrMatrByRef(stud_t *ps1, stud_t *ps2) {
  return ps1->matricola-ps2->matricola;
}
```

passaggio by reference

```
void ordStudPerMatr(stud_t *el, int n) {
  stud_t temp;
  int i, j, imin;
  for (i=0; i<n-1;
    imin = i;
    for (j = i+1; j < n; j++)
      if (confrMatr(el[j],el[imin])<0)
        imin = j;
    temp = el[i];
    el[i] = el[imin];
    el[imin] = temp;
  }
}
```

passaggio by value

selection sort



```
void ordStudPerMatr(stud_t *el, int n) {
  stud_t temp;
  int i, j, imin;
  for (i=0; i<n-1; i
    imin = i;
    for (j = i+1; j < n; j++)
      if (confrMatrByRef(&el[j],&el[imin])<0)
        imin = j;
    temp = el[i];
    el[i] = el[imin];
    el[imin] = temp;
  }
}
```

passaggio by reference

selection sort

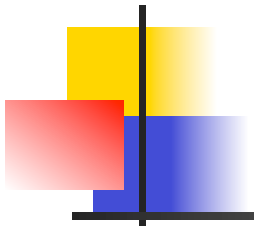
- Soluzione 2: vettore di puntato
- Scambio di puntatori

```
int main(int argc, char *argv[]) {
  stud_t elenco[MAXN], *elencoRif[MAXN];
  int i, ns = leggiStud(argv[1],elenco,MAXN);
  for (i=0; i<ns; i++)
    elencoRif[i]=&elenco[i];
  ordRifStudPer tr(elencoRif,
  lavoroSuElen ifOrd(elencoR
  scriviRif ncoRif,ns);
  return 0;
}
```

dimensione massima del vettore

inizializzazione vettore di puntatori

dimensione effettiva del vettore

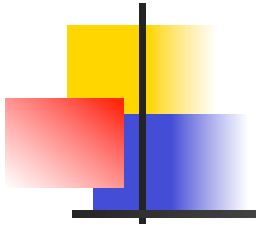


```
int confrMatrByRef(stud_t *s1, stud_t *s2) {
  return s1->matr-s2->matr;
}
```

```
void ordRifStudPerMatr(stud_t **elR, int n) {
  stud_t *temp;
  int i, j, imin;
  for (i=0; i<n-1; i++) {
    imin = i;
    for (j = i+1; j < n; j++)
    if (confrMatrByRef(elR[j],elR[imin])<0)
      imin = j;
    temp = elR[i];
    elR[i] = elR[imin];
    elR[imin] = temp;
   }
 }
```

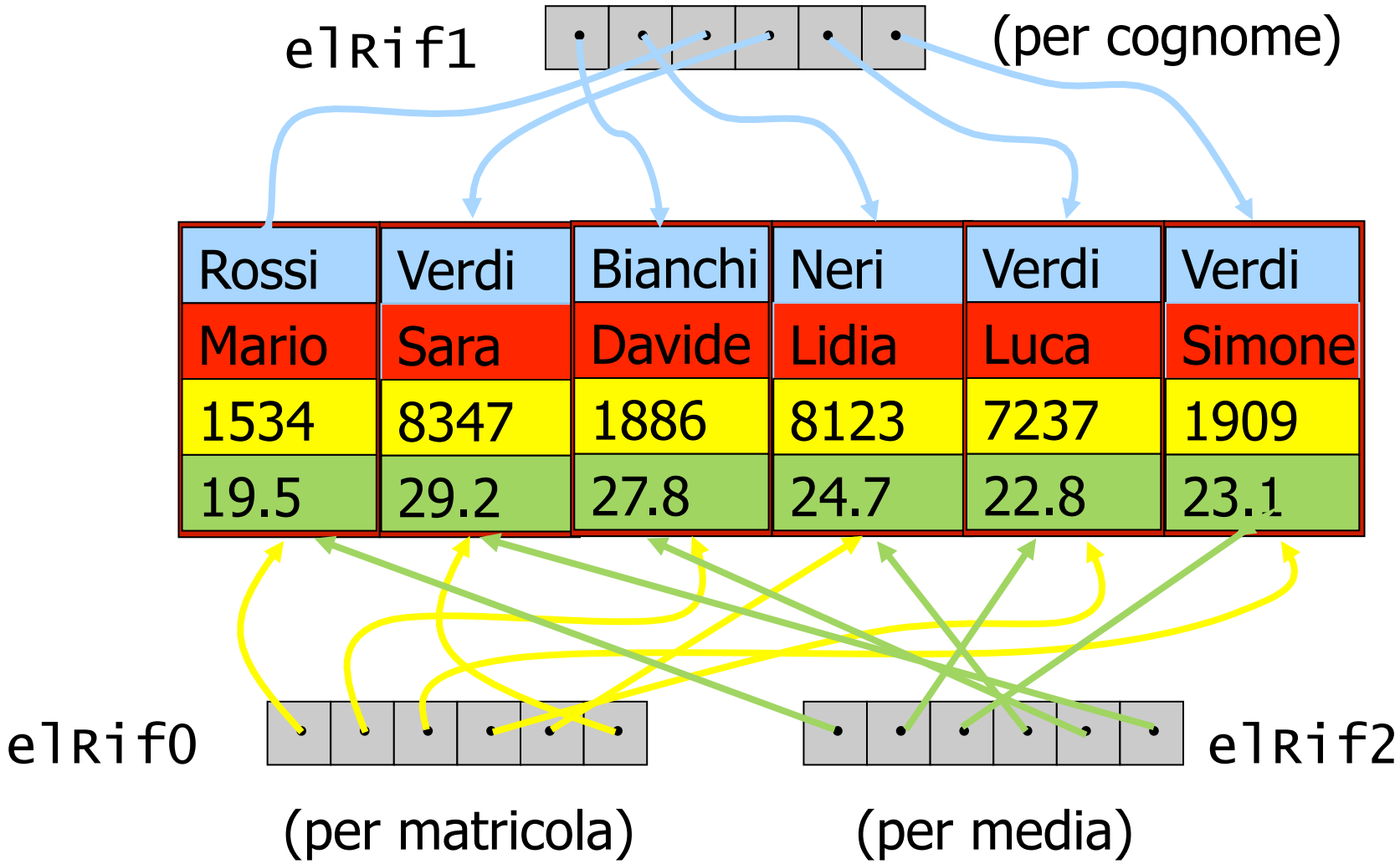
passaggio by reference

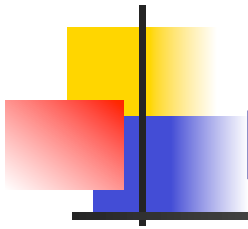
selection sort



- La soluzione con vettore di puntatori a `struct` permette di ordinare secondo più chiavi

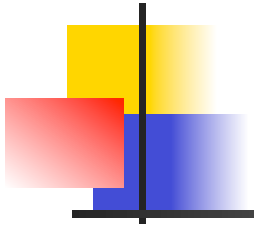
```
...
  stud_t el[MAXN], *elRif0[MAXN],
  *elRif1[MAXN], *elRif2[MAXN];
  int i, ns = leggiStud(argv[1],el,MAXN);
  for (i=0; i<n; i++)
    elRif0[i] = elRif1[i] = elRif2[i] = &el[i];
  ordRifStudPerMatr(elRif0,ns);
  ordRifStudPerCogn(elRif1,ns);
  ordRifStudPerMedia(elRif2,ns);
 // altri lavori a partire dai tre elenchi
 // scritture a partire dai tre elenchi
...
```


elRif1
(per cognome)
Rossi
Verdi
Bianchi
Neri
Verdi
Verdi
Mario
Sara
Davide
Lidia
Luca
Simone
1534
8347
1886
8123
7237
1909
19.5
29.2
27.8
24.7
22.8
23.1
elRif0
elRif2
(per matricola)
(per media)

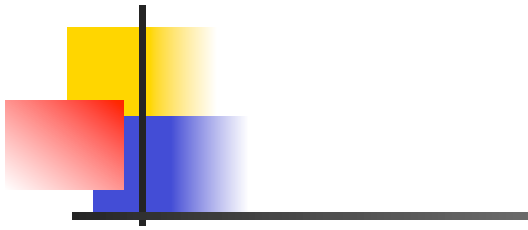


Esempio

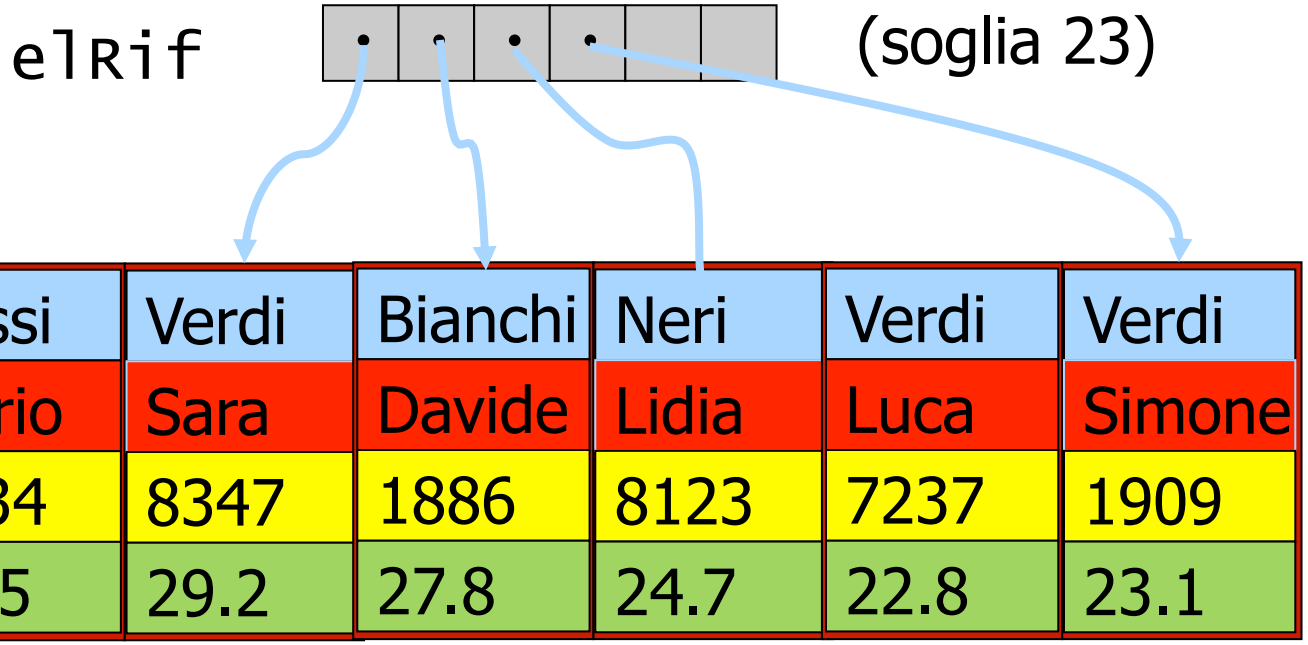
- Scrivere un programma che:
  - acquisisce un elenco di studenti da un file il cui nome è ricevuto come primo argomento al main
  - filtra l'elenco per medie superiori a una soglia
  - scrive l'elenco filtrato su un secondo file, il cui nome è ricevuto come secondo argomento.

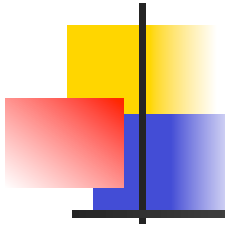


```
int filtraPerMedia(stud_t *el, stud_t **elR,
                   int n, float s);

int main(int argc, char *argv[]) {
  stud_t el[MAXN], *elRif[MAXN];
  int i, ns, ns2;
  float soglia = atof(argv[3];
  ns = leggiStud(argv[1],el,MAXN);
  ns2 = filtraPerMedia(el,elRif,ns,soglia)
  scriviRifStud(argv[2],elRif,ns2);
  return 0;
}
```



```
int filtraPerMedia(stud_t *el, stud_t **elR,
                   int n, float s){
  int i, n2;
  for (i=n2=0; i<n; i++)
    if (el[i].media>=s)
      elR[n2++] = &el[i];
  return n2;
}
```


elRif
(soglia 23)
Rossi
Verdi
Bianchi
Neri
Verdi
Verdi
Mario
Sara
Davide
Lidia
Luca
Simone
1534
8347
1886
8123
7237
1909
19.5
29.2
27.8
24.7
22.8
23.1

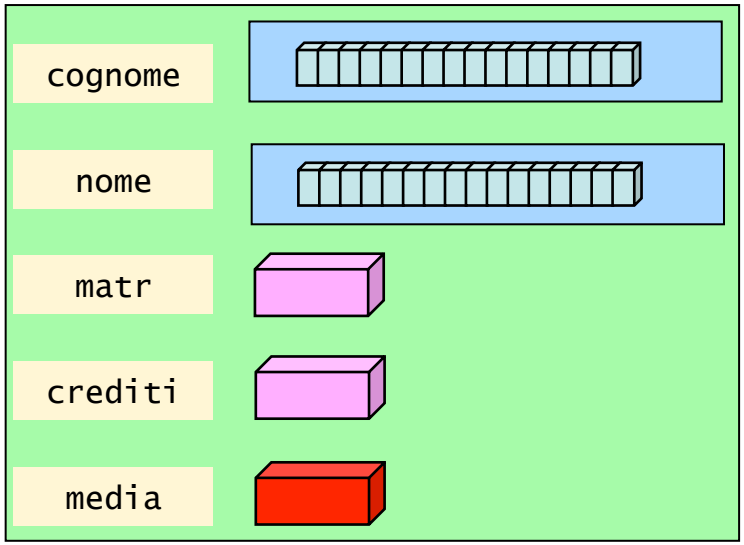


`Struct` come dati

- Una `struct` i cui campi sono tutti interni ad essa è un **dato atomico**
  - se i campi sono tutti scalari, la `struct` è un dato atomico
  - se ci sono campi vettore o `struct`:
    - Se dichiarati all'interno, la `struct` è un dato atomico
    - Se dichiarati come puntatore, allora potrebbe essere un dato esterno alla `struct`
- Essere o meno un dato atomico è rilevante ai fini della modularità e dell'appartenenza (ownership, cap. 5).

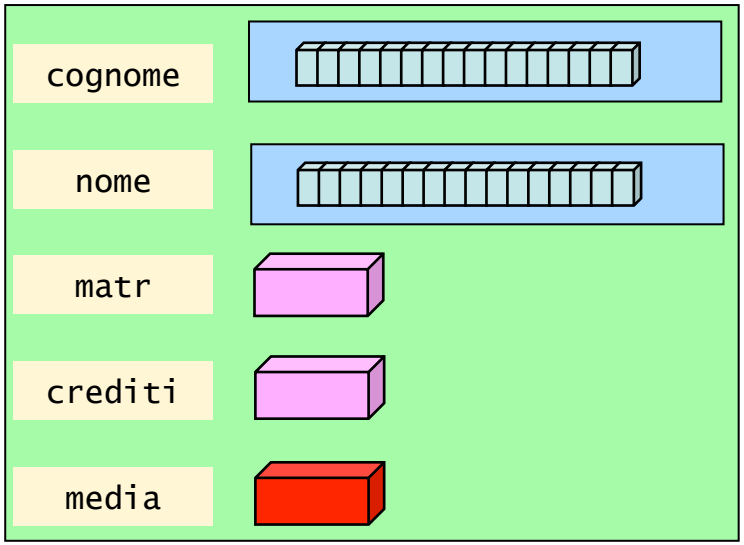
```
struct studente {
   char cognome[MAXS], char nome[MAXS];
   int matr, crediti;
   float media;
};
```

struct studente



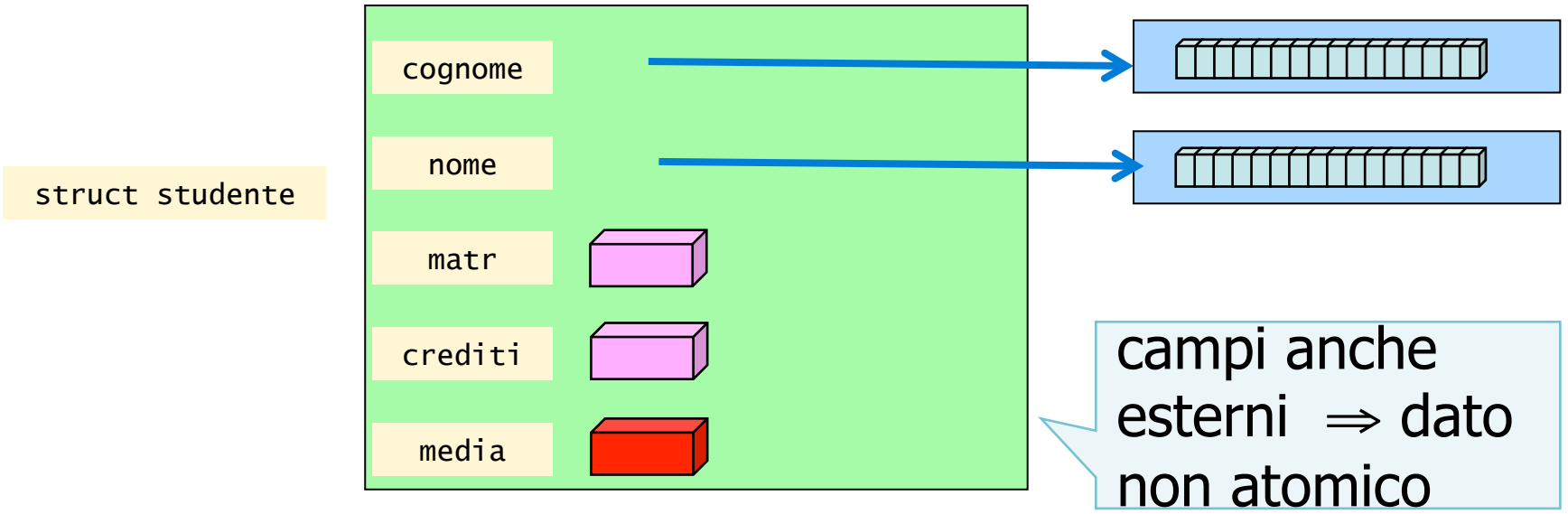
```
struct studente {
   char cognome[MAXS], char nome[MAXS];
   int matr, crediti;
   float media;
};
```

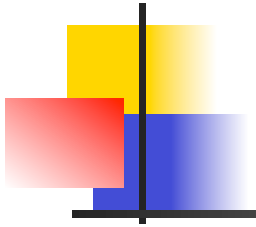
struct studente



campi interni ⇒ dato atomico

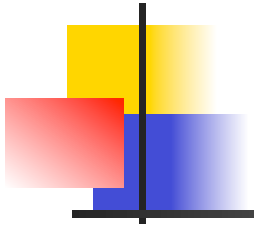
```
struct studente {
   char *cognome, char *nome;
   int matr, crediti;
   float media;
};
```





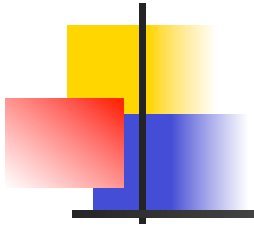
Esistenza di campi esterni ⇒ condivisione dei dati:

- Esempio:
  - nella `struct studente` c'è il campo `relatore`
  - il `relatore` è un aggregato di informazioni
  - l'elenco dei relatori è in una struttura dati esterna (ad esempio vettore)
  - il riferimento al `relatore` si realizza:
    - con puntatore
    - con indice nel vettore dei relatori
    - con una funzione di interfaccia (vedi ADT).

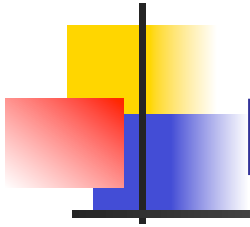


- Una `struct` può fungere da «raccoglitore» di dati (**wrapper**) anche se scarsamente correlati
  - parametri passati ad una funzione
  - valori di ritorno molteplici di una funzione
- Per dati fortemente correlati i costrutti `typedef` e `struct` permettono di creare nuovi tipi di dato:

```
typedef struct studente { float x, y;} punto;
```

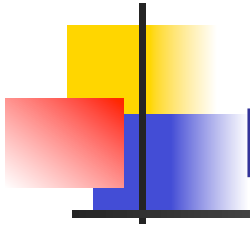


- Dato aggregato generico: `item` di tipo `Item` con campi:
  - scalari (`int`, `float`, `char`)
  - vettoriali interni alla `struct`
  - strutture (`struct` come campo di altra `struct` che la contiene)
  - puntatori a vettori o `struct` esterne.



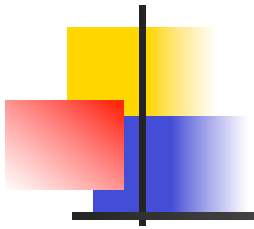
Puntatori e indici

- Ai dati di un vettore si accede mediante il loro **indice**
- L'indice spazio locale di memoria del vettore equivale al puntatore nello spazio globale di memoria
- **Si può emulare in un vettore il comportamento dei puntatori usando gli indici.**

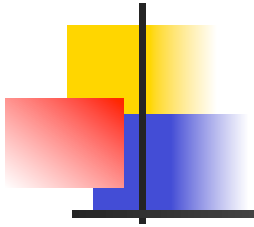


Esempio

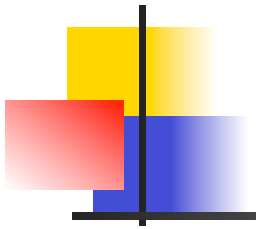
- Ordinamento per matricola di un vettore di riferimenti a strutture studenti
- Vettore `elInd` inizializzato con l'indice corrente
- Funzione `ordIndStudPerMatr` confronta i dati e se necessario scambia gli indici
- Funzione `scriviIndStud` accede al dato da stampare mediante il suo indice contenuto nel vettore degli indici `elInd`.



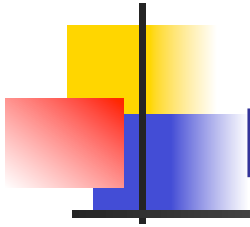
```
int main(int argc, char *argv[]) {
  stud_t el[MAXN];
  int elInd[MAXN];
  int i, ns;

  ns = leggiStud(argv[1],el,MAXN);
  for (i=0; i<ns; i++)
    elInd[i] = i;
  ordIndStudPerMatr(el,elInd,ns);
  scriviIndiStud(argv[2],el,elInd,ns);
  return 0;
}
```



```
void ordIndStudPerMatr(stud_t *el, int *elI,
                       int n) {
  int i, j, imin, temp;
  for (i=0; i<n-1; i++) {
    imin = i;
    for (j = i+1; j < n; j++)
      if (confrMatrPerInd(el,elI[j],
                          elI[imin])<0)
        imin = j;
    temp = elI[i];
    elI[i] = elI[imin];
    elI[imin] = temp;
  }
}
```

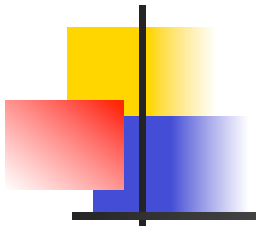


```
int confrMatrPerInd(stud_t *el, int id1,
                    int id2) {
 return el[id1].matr - el[id2].matr; }

void scriviIndStud(char *nomeFile, stud_t *el,
                   int *elI, int n) {
  int i;
  FILE *fp = fopen(nomeFile,"w");
  for (i=0; i<n; i++) {
    fprintf(fp, "%s %s %d %f\n",
          el[elI[i]].cognome, el[elI[i]].nome,
          el[elI[i]].matr, el[elI[i]].media);
  }
  fclose(fp);
}
```

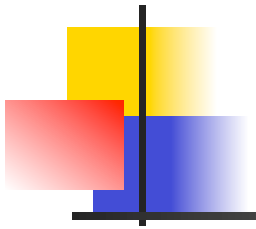


Esempio

- Ordinamento secondo più chiavi
- Più vettori di indici
- Confronto tra i dati e se necessario scambio degli indici
- Accesso al dato da stampare mediante il suo indice contenuto nel vettore degli indici opportuno.



```
int main(int argc, char *argv[]) {
  stud_t el[MAXN]; int i, ns;
  int elInd0[MAXN], elInd1[MAXN], elInd2[MAXN];
  ns = leggiStud(argv[1],el,MAXN);
  for (i=0; i<n; i++)
    elInd0[i] = elInd1[i] = elInd2[i] = i;
  ordIndStudPerMatr(el,elInd0,ns);
  ordIndStudPerCognome(el,elInd1,ns);
  ordIndStudPerMedia(el,elInd2,ns);
  // altri lavori a partire dai tre elenchi
  scriviIndStud(argv[2],el,elInd0,ns);
  scriviIndStud(argv[3],el,elInd1,ns);
  scriviIndStud(argv[4],el,elInd2,ns);
  return 0;
}
```



elInd1

2	3	0	4	1	5

(per cognome)

Rossi	Verdi	Bianchi	Neri	Verdi	Verdi
Mario	Sara	Davide	Lidia	Luca	Simone
1534	8347	1886	8123	7237	1909
19.5	29.2	27.8	24.7	22.8	23.1

elInd0

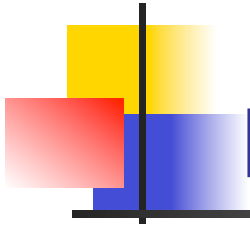
0	2	5	4	3	1

(per matricola)

0	4	5	3	2	1

elInd2

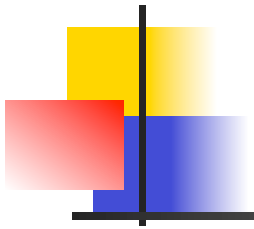
(per media)



Esempio

- Filtro per medie

```
int main(int argc, char *argv[]) {
  stud_t el[MAXN];
  int i, ns, ns2, elInd[MAXN];
  float soglia = atof(argv[3];
  ns = leggiStud(argv[1],el,MAXN);
  ns2 = filtraMediaInd(el,elInd,ns,soglia);
  scriviIndStud(argv[2],el,elInd,ns2);
  return 0;
}
```



```
int filtraMediaInd(stud_t *el, int *elI,
                   int n, float s) {
  int i, n2;
  for (i=n2=0; i<n; i++) {
    if (el[i].media>=s)
      elI[n2++] = i;
  }
  return n2;
}
```

`elInd`

1	2	3	5	-1	-1

(soglia 23)

Rossi	Verdi	Bianchi	Neri	Verdi	Verdi
Mario	Sara	Davide	Lidia	Luca	Simone
1534	8347	1886	8123	7237	1909
19.5	29.2	27.8	24.7	22.8	23.1

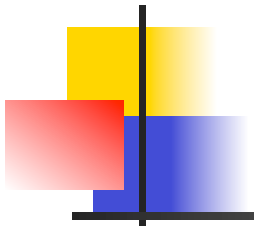
Uso avanzato dei puntatori

Puntatore a funzione:

- A una funzione si può far riferimento mediante un puntatore
- Così a una variabile o parametro formale si può associare a tempo di esecuzione il puntatore alla funzione da chiamare
- Esempio:

Puntatore collocato nel prototipo di funzione

```
void (*sortF)(int *v, int n);
```



- Nel programma, data una funzione:

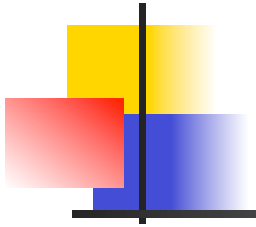
```
void selectionSort(int *vet, int n);
```

- è possibile l'assegnazione:

```
sortF = selectionSort;
```

- e poi la chiamata:

```
sortF(dati, ndati);
```



Il puntatore generico `void *`:

- è un puntatore **opaco**, cioè un semplice indirizzo in memoria senza tipo di dato associato
- in assegnazione è compatibile con qualunque altro puntatore (non è necessario un cast esplicito)
- Serve a:
  - trattare puntatori a tipi di dato ignoti alle funzioni in cui compaiono
  - fare riferimento a un dato che può in esecuzione assumere più tipi.