POLITECNICO
DI TORINO
Dipartimento
di Automatica e Informatica

Capitolo 2: La Dualità Puntatore Vettore

PUNTATORI E STRUTTURE DATI DINAMICHE: ALLOCAZIONE DELLA MEMORIA E MODULARITÀ IN LINGUAGGIO C

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

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

<nome vettore> ⇔ &<nome vettore>[0]

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

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

<nome vettore> ⇔ &<nome vettore>[0]

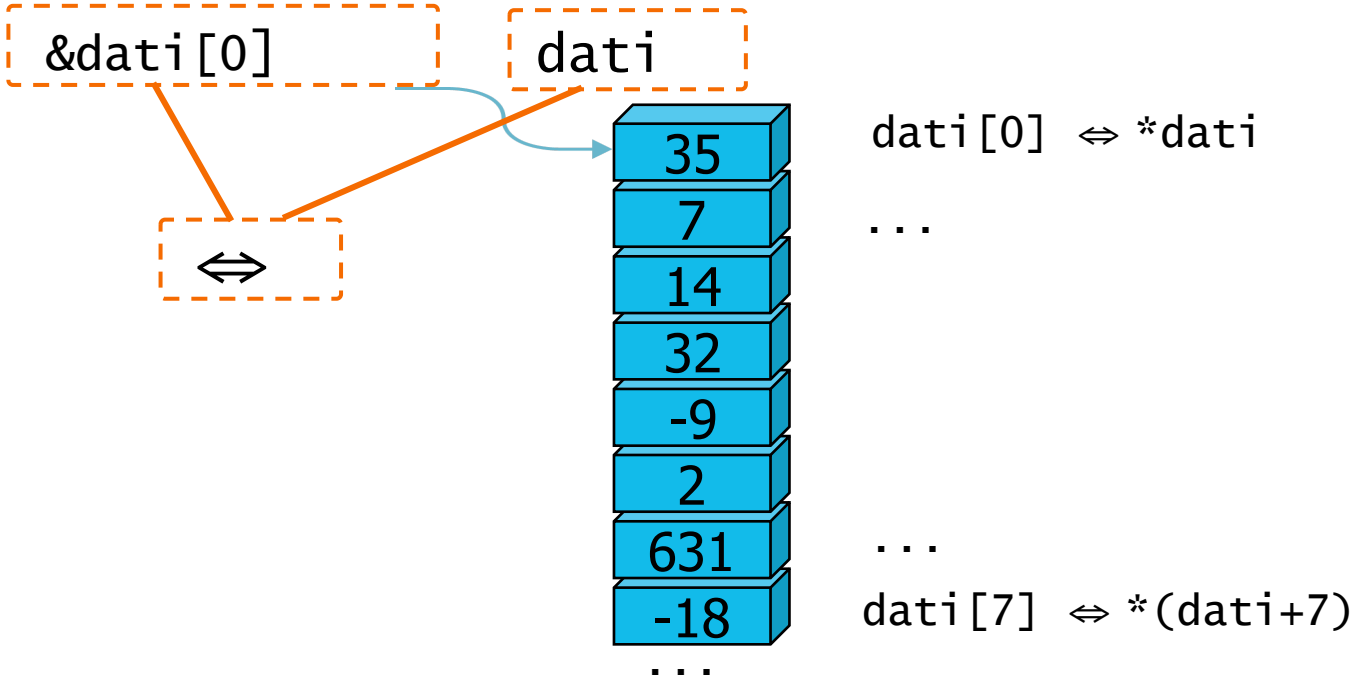
Basterebbe questo. Tutto il resto è una conseguenza!

Vettori e puntatori

Esempio: data una variabile di tipo vettore

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

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

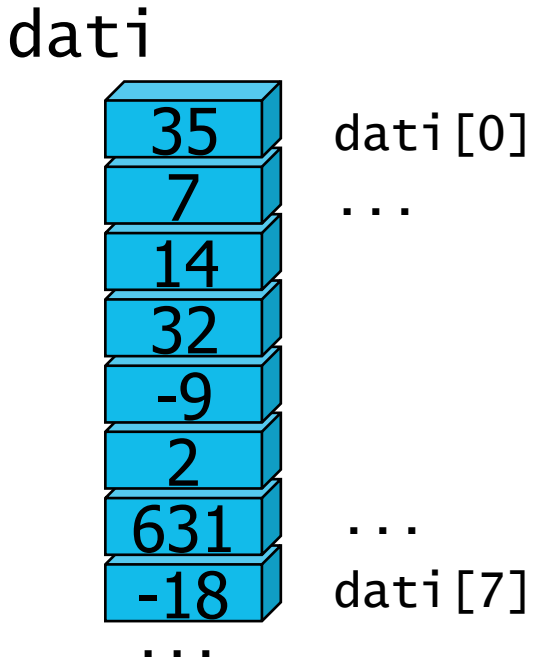

&dati[0]
dati
⇔
35
7
14
32
-9
2
631
-18
...
dati[0] ⇔ *dati
...
...
dati[7] ⇔ *(dati+7)

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]`
   - aritmetica dei puntatori (somma tra puntatore e indice intero)
3. con puntatore a intero `int *p`, inizializzato a `&dati[0]`
   - scansione dei dati direttamente mediante puntatore (aggiornato ad ogni iterazione).

Modo 1:

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

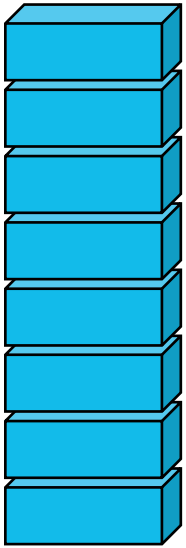


Modo 2:

p

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

dati



dati[0]

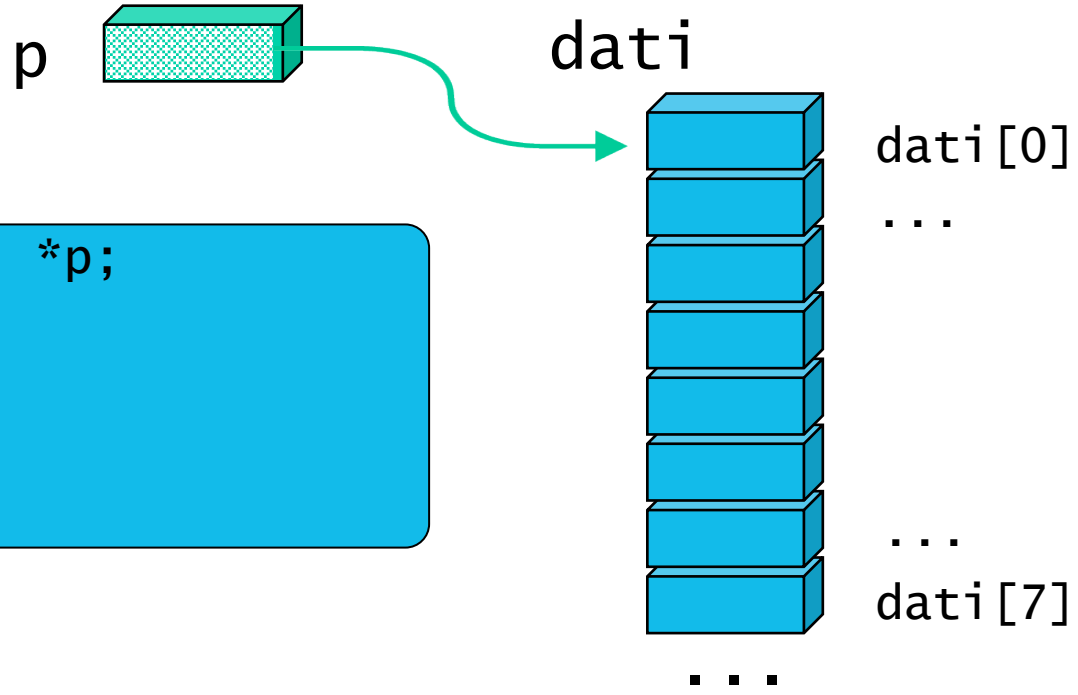
...

...

dati[7]

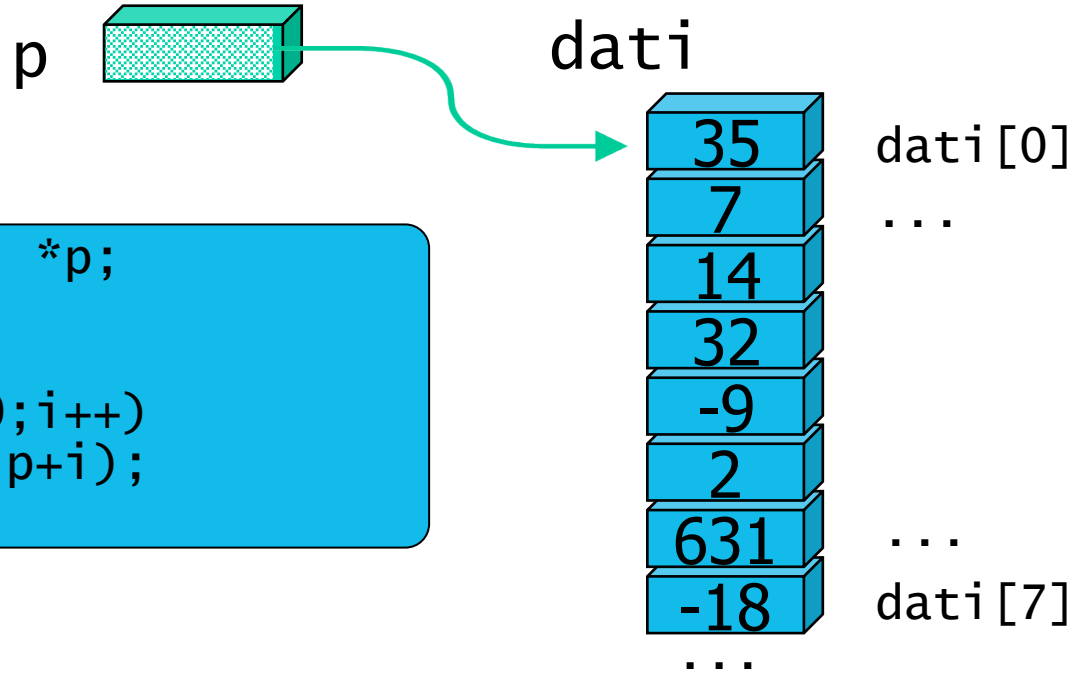
...

Modo 2:




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

Modo 2:

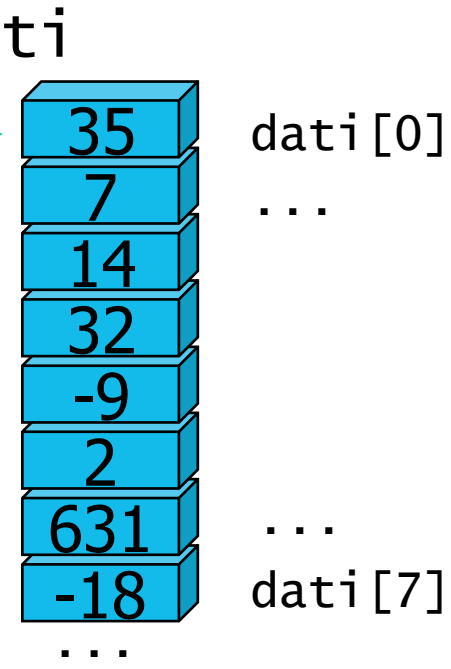



```
int dati[100], *p;
...
p = &dati[0];
for (i=0;i<100;i++)
  scanf("%d", p+i);
```

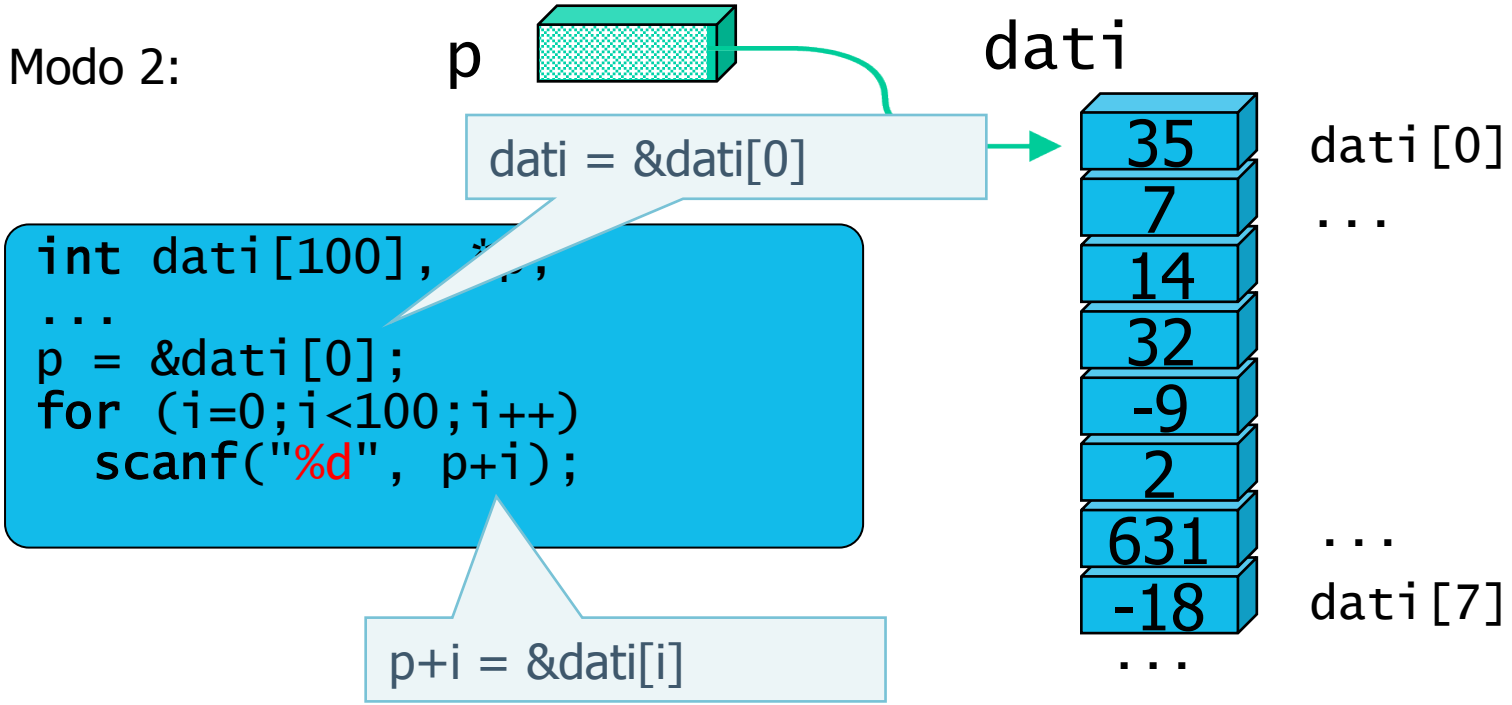
Modo 2:

dati = &dati[0] (sono equivalenti)

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

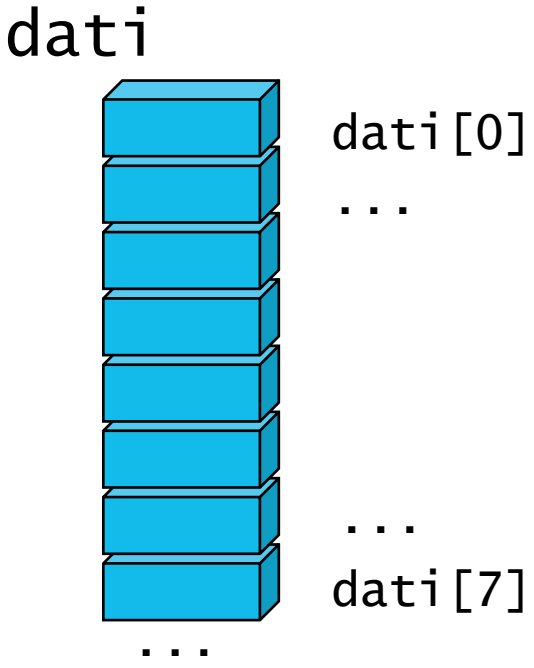


Modo 2:

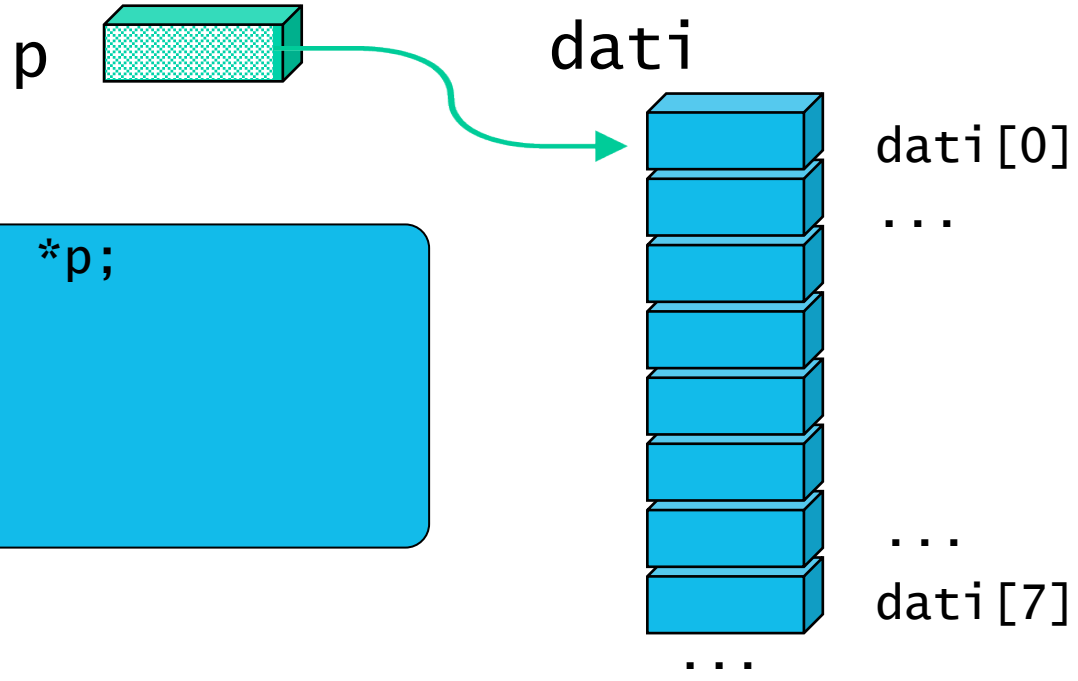

p
dati
dati = &dati[0]
int dati[100], *p;
...
p = &dati[0];
for (i=0;i<100;i++)
scanf("%d", p+i);
p+i = &dati[i]
35
7
14
32
-9
2
631
-18
...
dati[0]
...
...
dati[7]

Modo 3:


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

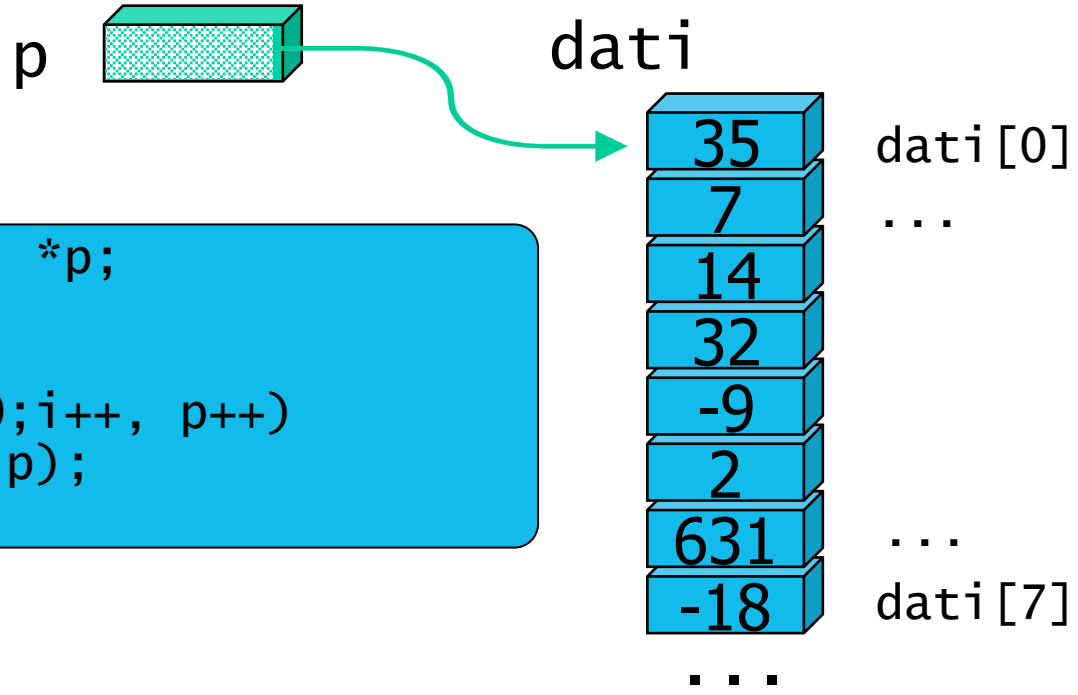


Modo 3:




```
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);
```

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

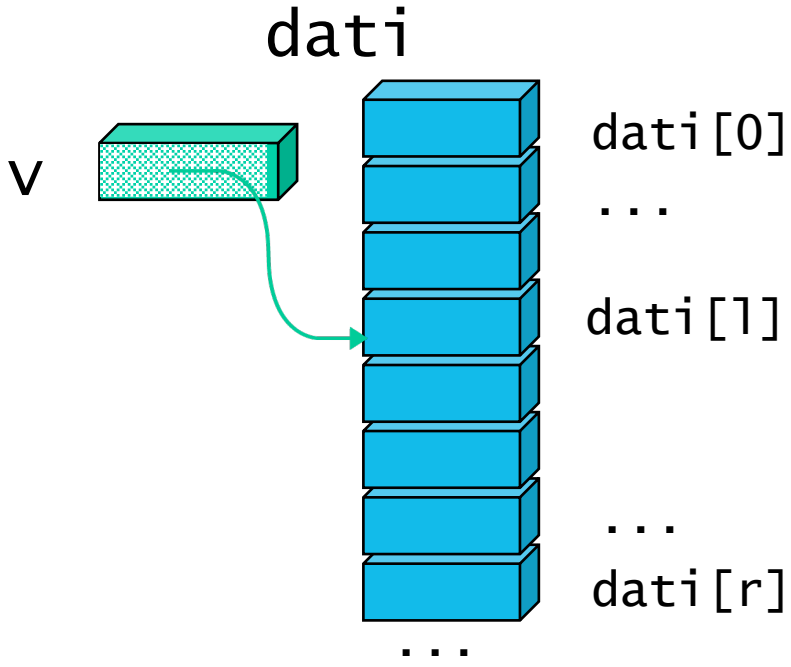
Per identificare un sottovettore compreso tra indici `l` e `r` di un vettore dato:

- 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`:

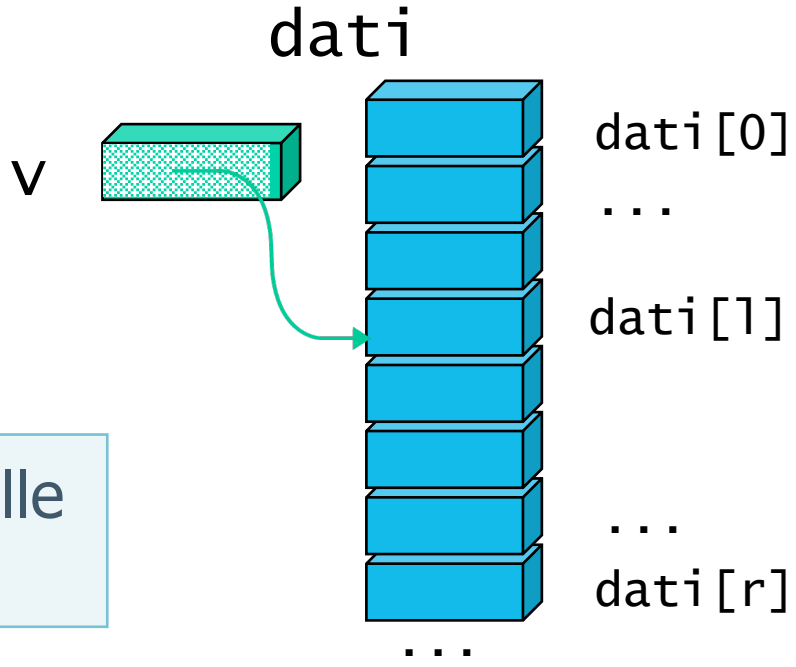
```
int dati[100], *v, i;
v = &dati[l];
n = r - l + 1;
for (i=0; i<n; i++)
  scanf("%d", &v[i]);
for (i=0; i<n; i++)
  printf("%d", v[i]);
```



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]);
```

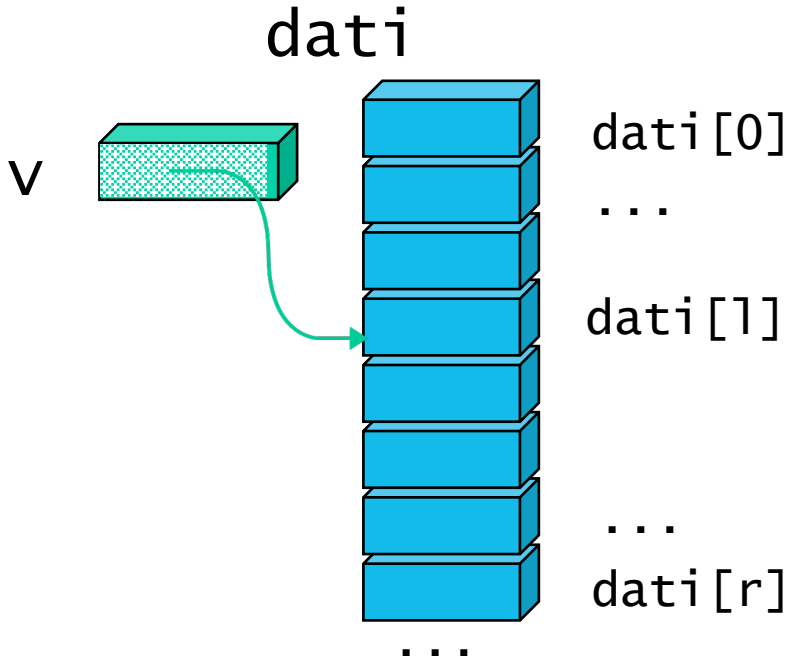


n è il numero di caselle del sottovettore

Modo 1

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

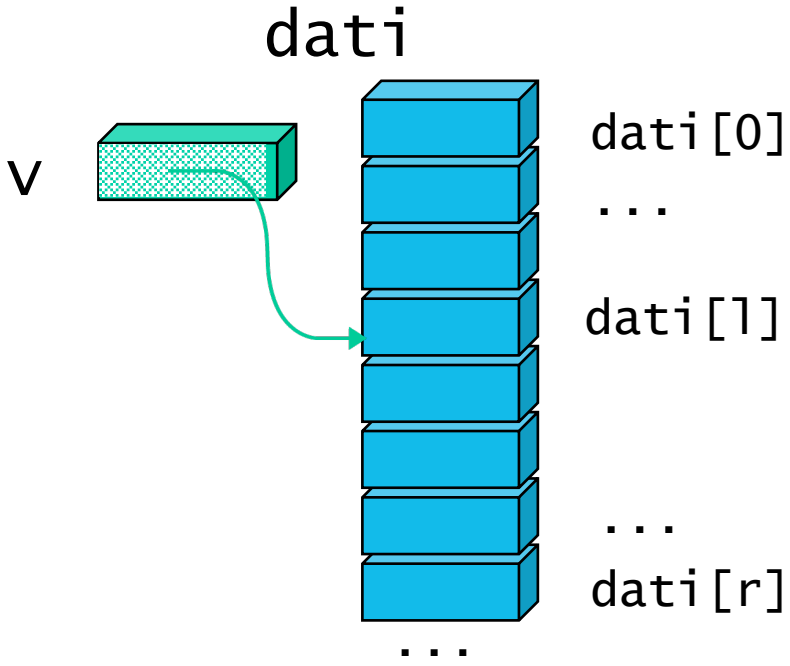
```
int dati[100], *v, i;
v = &dati[l];
n = r - l + 1;
for (i=0; i<n; i++)
  scanf("%d", v++);
v = dati+l;
for (i=0; i<n; i++)
  printf("%d", *v++);
```



Modo 2

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

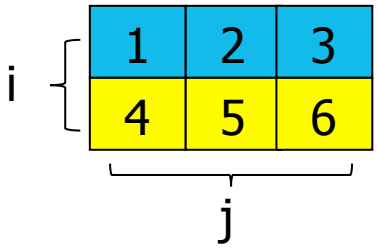
```
int dati[100], *v, i;
v = dati + l;
n = r - l + 1;
for (i=0; i<n; i++)
  scanf("%d", v+i);
for (i=0; i<n; i++)
  printf("%d", *(v+i));
```

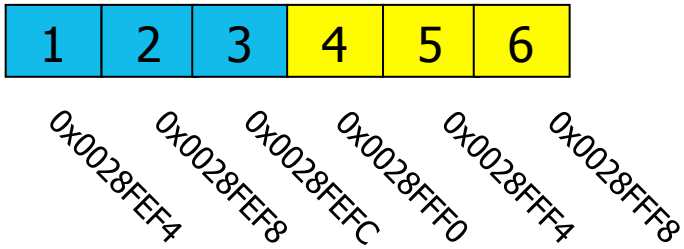


Modo 3

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

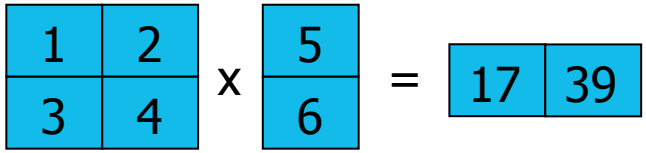




Esempio

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:



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:

- Passando il nome di un vettore (coma parametro attuale) a una funzione, si passa il puntatore al primo elemento del vettore
- si passa solo il puntatore alla prima casella (non la dimensione: se la si vuole, va passata come parametro aggiuntivo e indipendente)

La dualità puntatore ⇔ vettore è:

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

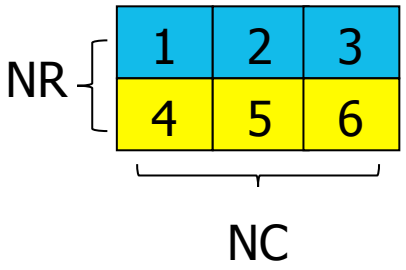
Vettori e matrici come parametri

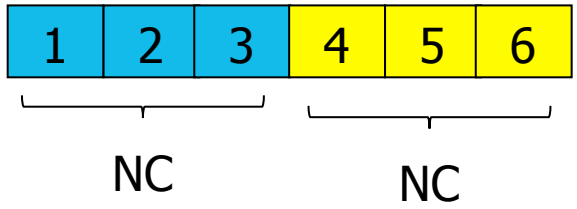
- Par accedere all'i-esimo elemento di un vettore `vett[N]`, una funzione necessita unicamente dell'indirizzo del primo elemento:

  `vett[i]` equivale a `*(vett+i)`

- Per accedere all elemento [i,j] di una matrice `mat[NR][NC]`, a una funzione serve l'indirizzo del primo elemento E il numero di colonne NC della matrice:

  `mat[i][j]` equivale a `*(mat + NC*i + j)`





Vettori e matrici come parametri

Sono ridondanti le seguenti informazioni:

- dimensione di un vettore
- prima dimensione di una matrice

Esempi: prototipi di (compatibili a) 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 di (compatibili a) 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 adimensionato di puntatori

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

- puntatore a puntatore (al primo elemento di unvettore di puntatori)

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

Dimensione (effettiva) come parametro

Sovente si passano come parametri:

- il vettore adimensionato
- la sua dimensione effettiva

per realizzare funzioni che si adattano ad essa (la dimensione):

```
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);
  ...
}
```

```
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

PARAMETRO FORMALE VETTORE - ATTUALE PUNTATORE

PARAMETRO FORMALE PUNTATORE – ATTUALE VETTORE

parametro formale vettore - 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

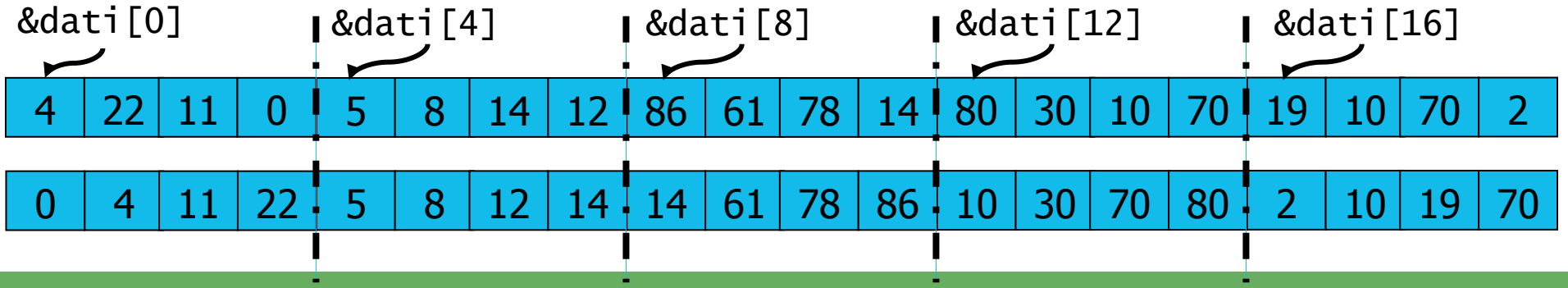
```
void ordinaInt(int v[], int n);
...
int dati[20];
...
for (i=0;i<20;i+=4)
  ordinaInt(&dati[i],4);
```

parametro formale vettore - attuale puntatore

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



parametro formale puntatore – attuale vettore

- Ad un parametro formale puntatore può corrispondere un parametro attuale vettore
  - Il puntatore, a sua volta, può essere trattato internamente come vettore

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

Puntatori e stringhe

LE STRINGHE MANIPOLATE COME VETTORI O MEDIANTE PUNTATORI

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

Strlen (versione 0)

Funzione equivalente a `strlen` con stringa come vettore:

- Conta i caratteri cercando l'indice del '\0'

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

Strlen (versione 0)

Funzione **equivalente** a `strlen` con stringa come vettore:

- Conta i caratteri cercando l'indice del '\0'

```
int strlen0(                ]){
  int cnt=0;
  whi
    c
  ret
}
```

ATTENZIONE: non è la funzione di libreria `strlen`, ma una implementazione equivalente

Strlen (versione 0)

Funzione equivalente a `strlen` con stringa come vettore:

- Conta i caratteri cercando l'indice del '\0'

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

`cnt` funge sia da indice che da contatore

Strlen (versione 1)

Funzione equivalente a `strlen` con stringa come vettore:

- Scorre la stringa mediante un puntatore p
- Conta mediante contatore intero `cnt`

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

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

Strlen (versione 2)

Funzione equivalente a `strlen` con stringa come puntatore:

- Scorre la stringa direttamente mediante il parametro puntatore `s`
- Conta mediante contatore intero `cnt`

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

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

Strlen (versione 3)

Funzione equivalente a `strlen` con stringa come puntatore:

- Scorre la stringa mediante puntatore p
- Non conta ma usa aritmetica dei puntatori (differenza)

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

non serve un contatore esplicito: p scorre la stringa.

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 ASCII dei primi caratteri diversi.
  - strcmp("Hello","Help") -> 'l'-'p' = -4
  - strcmp("Hello","Hello") -> 0

Strcmp (versione 0)

- 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]);
}
```

Strcmp (versione 1)

- 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 (versione 0)

- La funzione si limita ai primi n caratteri delle due stringhe
- 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 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 (la prima volta)

```
char *strstr0(char s[], char c[]){
  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: `strstr`

Date due stringhe, cercare la 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

```
char *strstr0(char s[], char c[]){
  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);
}
```

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

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`).

Esempio: input mediante `sscanf`

- Primo tentativo: SBAGLIATO!!!
  - La `sscanf` "riparte" ogni volta dall'inizio della riga (non va avanti come la fscanf nel file).
  - Quindi legge sempre il primo numero

```
// dichiarazione variabili
...
fgets(riga,MAX,stdin);
while (sscanf(riga, "%d", &x)>0) {
  somma += x;
  cnt++;
}
printf("La media e': %f\n", somma/cnt);
```

Esempio: input mediante `sscanf`

- Primo tentativo: SBAGLIATO!!!
  - La `sscanf` "riparte" ogni volta dall'inizio della riga (non va avanti come la fscanf nel file).
  - Quindi legge sempre il primo numero

```
// dichiarazione variabili
...
fgets(riga,MAX,stdin);
while (sscanf(riga, "%d", &x)>0) {
  somma += x;
  cnt++;
}
printf("La media e': %f\n", somma/cnt);
```

Strategia corretta

- 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 ?
- SOLUZIONE:
  - formato %n (dice quanti caratteri letti)
  - puntatore per identificare sotto-stringa

Codice

```
...
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);
```

Codice

```
...
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);
```

- `s` punta all'inizio del "pezzo" di stringa che interessa
- `i` "dice" (grazie a %n) quandi caratteri si sono letti fin li
- `i` serve ad aggiornare s

Vettori di puntatori

MATRICI REALIZZATE COME VETTORI DI PUNTATORI (A SOTTO-MATRICI)

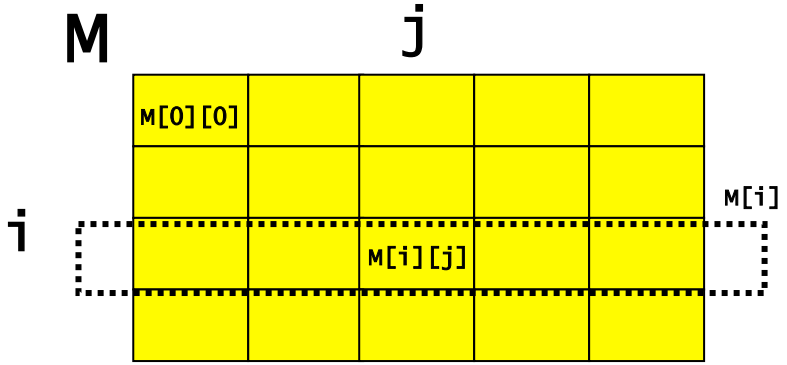
Vettori di puntatori

- 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)
- ATTENZIONE!
  - C'è dualità ma le matrici realizzate come vettori di puntatori sono diverse dalle matrici dichiarate come tali (con più livelli di parentesi quadre)

Matrice come vettore di righe

Esempio: matrice come vettore di righe (NO PUNTATORI!)

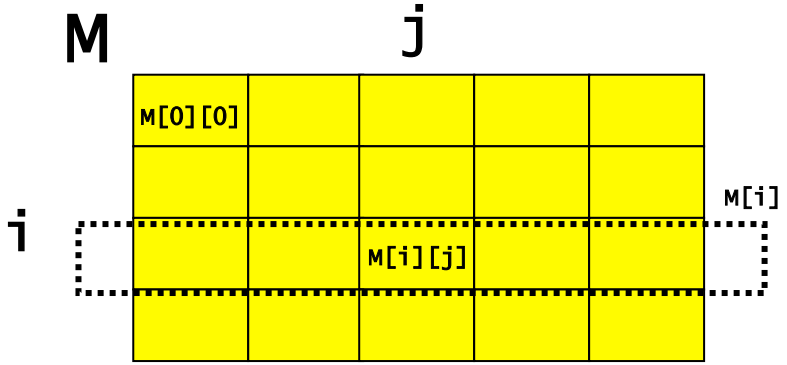
```
#define NR 4
#define NC 5
float M[NR][NC];
...
```



Matrice come vettore di righe

Esempio: matrice come vettore di righe

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



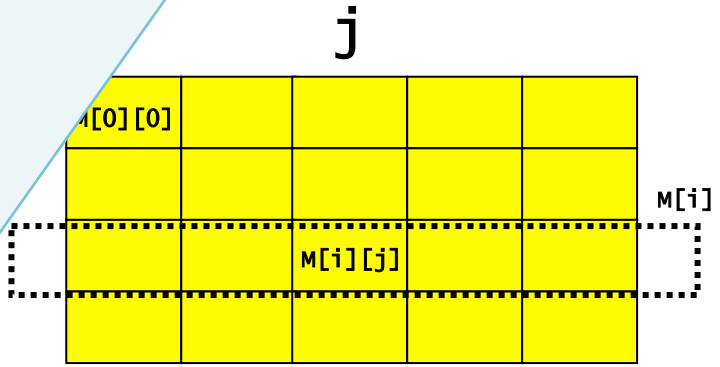

```
printf("dim. matr.: %d\n", sizeof(M));
printf("dim. riga: %d\n", sizeof(M[0]));
printf("dim. elem.: %d\n", sizeof(M[0][0]));
```

Matrice come vettore di righe

Esempi

```
80 = NR x NC x sizeof(float)
```

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



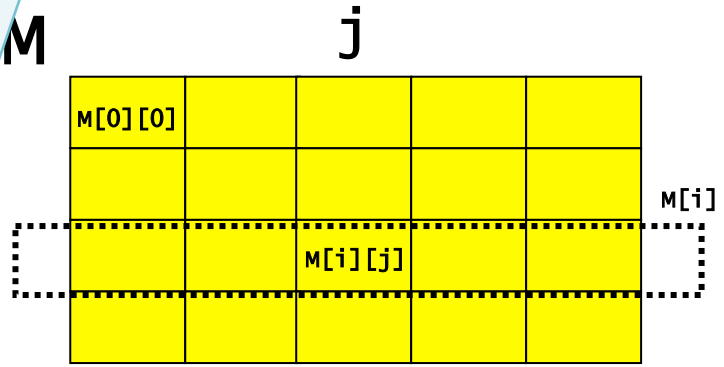

```
printf("dim. matr.: %d\n", sizeof(M));
printf("dim. riga: %d\n", sizeof(M[0]));
printf("dim. elem.: %d\n", sizeof(M[0][0]));
```

Matrice come vettore di righe

Esempio: matrice come

20 = NC x sizeof(float)

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

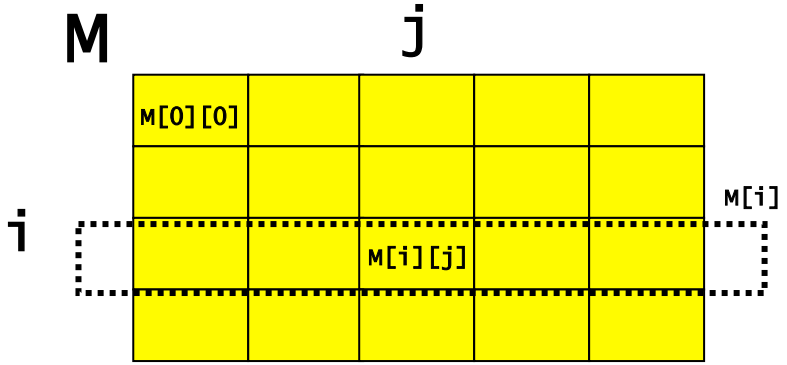



```
printf("dim. matr.: %d\n", sizeof(M));
printf("dim. riga: %d\n", sizeof(M[0]));
printf("dim. elem.: %d\n", sizeof(M[0][0]));
```

Matrice come vettore di righe

Esempio: matrice come `4 = sizeof(float)`

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

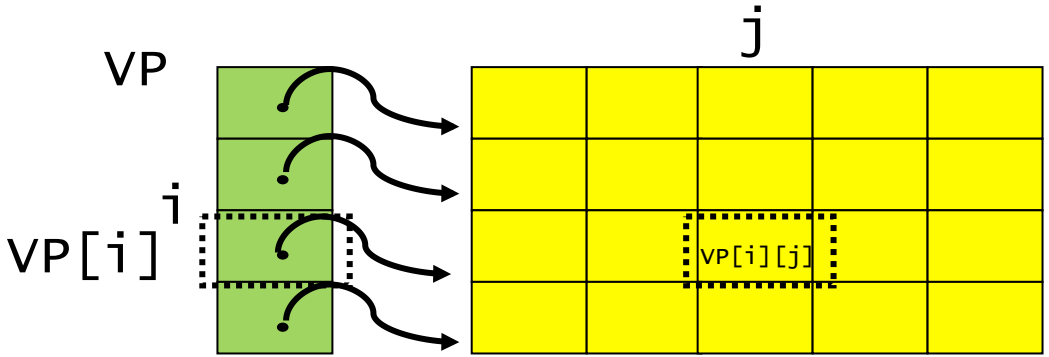



```
printf("dim. matr.: %d\n", sizeof(M));
printf("dim. riga: %d\n", sizeof(M[0]));
printf("dim. elem.: %d\n", sizeof(M[0][0]));
```

Matrice come vettore di puntatori (a righe)

Matrice come vettore di puntatori a (alle caselle iniziali di) vettori

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

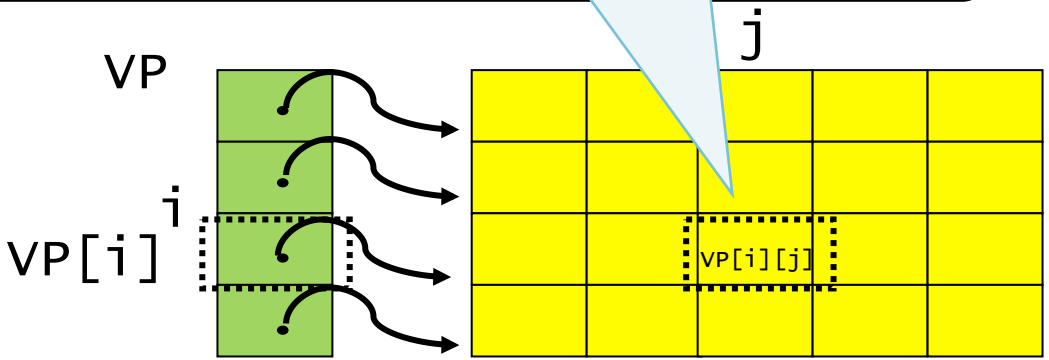


Matrice come vettore di puntatori (a righe)

Matrice come vettore di pu

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

Notazione matriciale per VP[i][j] ⇔(VP[i])[j]
(nonostante VP sia un vettore)

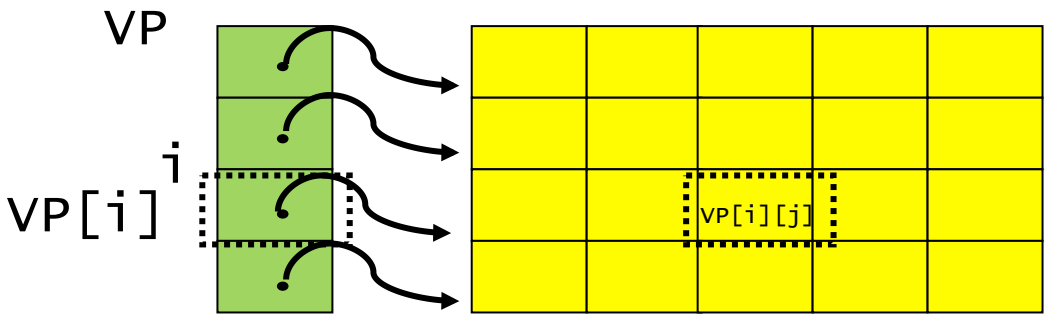


Matrice come vettore di puntatori (a righe)

Dimensioni:

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

Si assume puntatore su 8 byte (processore a 64 bit)



Matrice come vettore di puntatori (a righe)

Dimensioni:

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

Stessa notazione ma risultati diversi: M e VP non sono la stessa cosa

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

Vettori di vettori a dimensione variabile

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

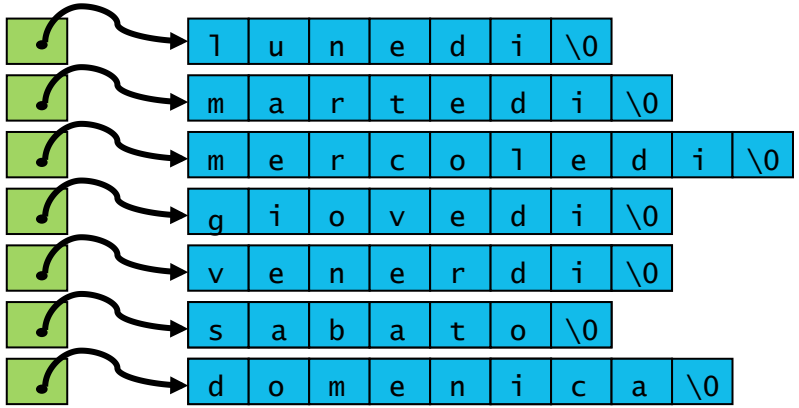
- opportunità sfruttata sovente nei vettori di stringhe, accessibili anche come matrici di caratteri

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

```
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");
}
```

Inizializzazione con costanti stringa

Matrice di caratteri

```
void main (void) {
  int i,g;
  char giorni[7][10]={"lunedì","martedì",
                      "mercoledì","giovedì",
                      "venerdì","sabato","dom
  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");
}
```

Riga generica identificabile con un solo indice

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");
}
```

Singolo carattere identificabile con due indici

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

```
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 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");
}
```

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");
}
```

`giorni[g]`: stringa di ordine g

Vettore di puntatori a stringa

```
void main (void) {
  int i,g;
  char *giorni[7]={"lunedì","martedì",
                   "mercoledì","giovedì",
                   "venerdì",
  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");
}
```

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

Vettore di puntatori a stringa

```
void main (void) {
  int i,g;
  char *giorni[7]={"lunedì","martedì"
                   
                   
  printf("quale car
  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

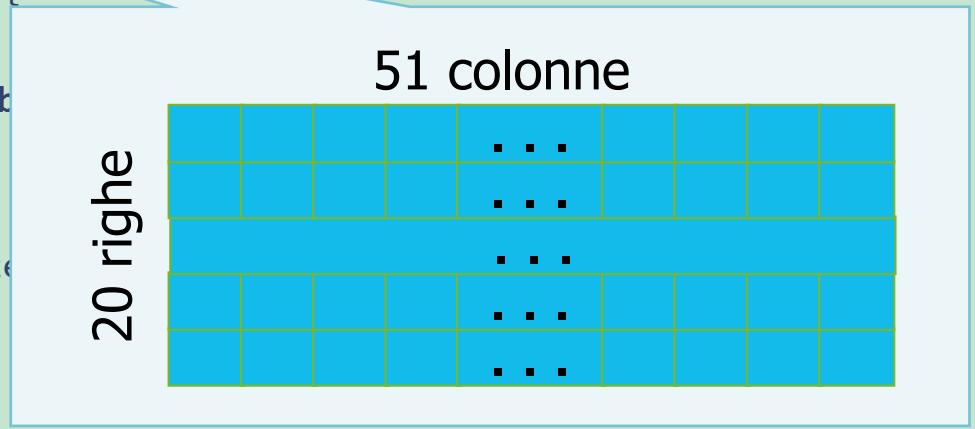
- Leggere da tastiera delle stringhe:
  - al massimo 20
  - ognuna al massimo di 50 caratteri
  - la somma delle lunghezze delle stringhe è <= 500
  - l'input termina con una stringa vuota
- Ordinare le stringhe in ordine crescente (secondo `strcmp`)
- Visualizzarle (secondo l'ordine precedente) su video

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]);
}
```

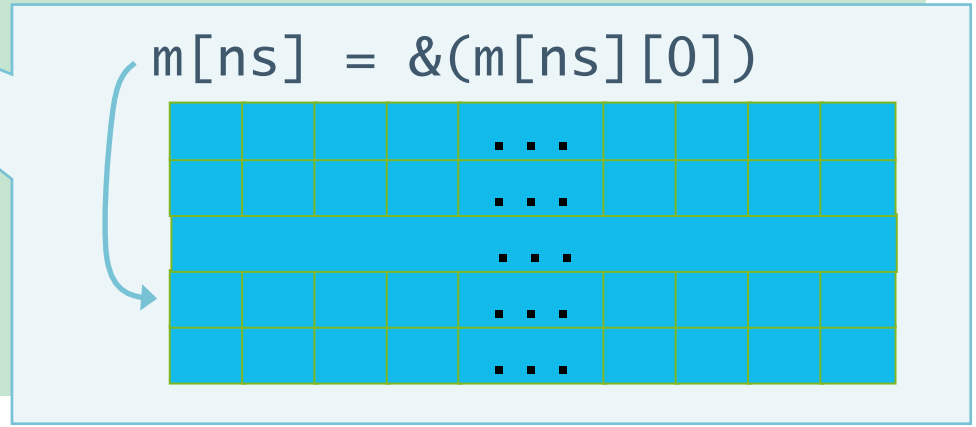
Matrice di caratteri

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



Matrice di caratteri

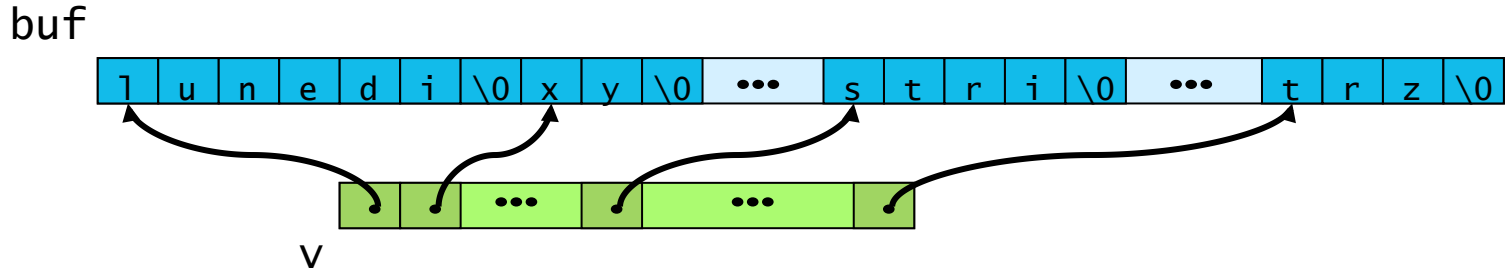
```
void main (void){
  int i,ns;
  char m[20][51]; // 51 colonne per i \0
  printf("scrivi stringhe:\n");
  for (ns=0; ns<20; ns++) {
    gets(m[ns]);
    if (strlen(m[ns])= 0) break;
  }
  ordinaMatrice(m,ns);
  printf("stringhe ordinate:\n"),
  for (i=0; i<ns; i++)
  printf("%s\n", m[i]);
}
```



Vettore di puntatori

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` (o `&buf[0]`), gli altri puntatori si calcolano in base alla lunghezza della stringa

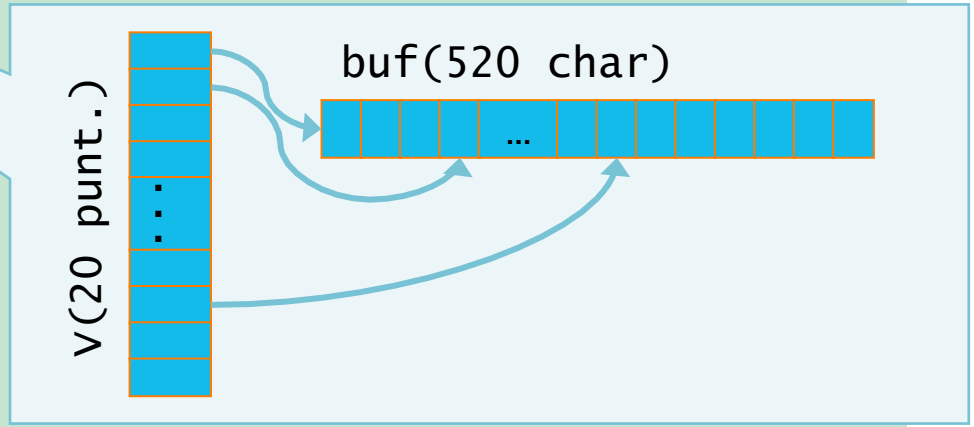


Vettore di puntatori

```
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]);
}
```

Vettore di puntatori

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



Vettore di puntatori

```
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]

Vettore di puntatori

```
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]);
}
```

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 puntatori: dimensione vettore di puntatori
  - 520 caratteri: dimensione di caratteri contenente le stringhe (500 caratteri per le stringhe + 20 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);
  }
}
```

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);
  }
}
```

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

Ordinamento con vettore di puntatori

```
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;
  }
}
```

Ordinamento con vettore di puntatori

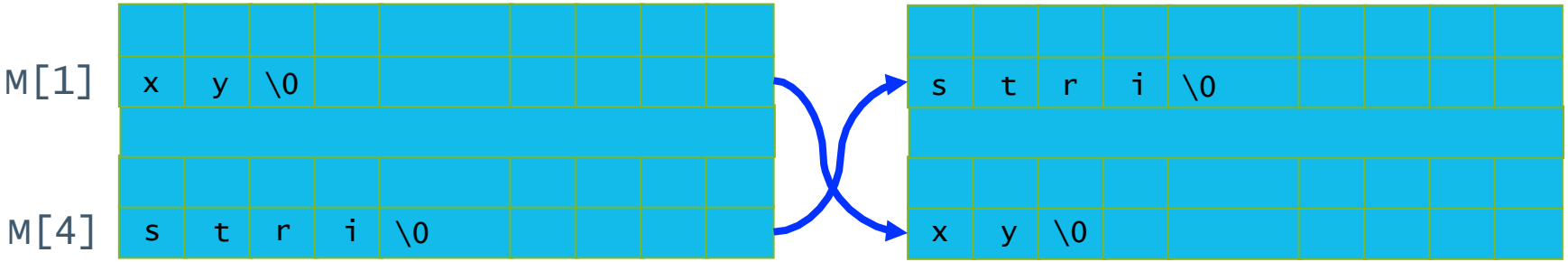
```
void ordinaVettore (char *m[], int n){
  int i, j, min; char *tmp;
  for (i=0; i<n-1; i++) {
    min = i;
    for (j=i+1; j<n; j++)
      if (strcmp(m[min],m[j])>0)
        min = j;
    tmp = m[i];
    m[i] = m[min];
    m[min] = tmp;
  }
}
```

Con il vettore di puntatori gli scambi di stringa sono realizzati mediante scambio di puntatori (NO strcpy)

Esempio

Scambio riga 1 e 4, contenenti "xy" e "stri"

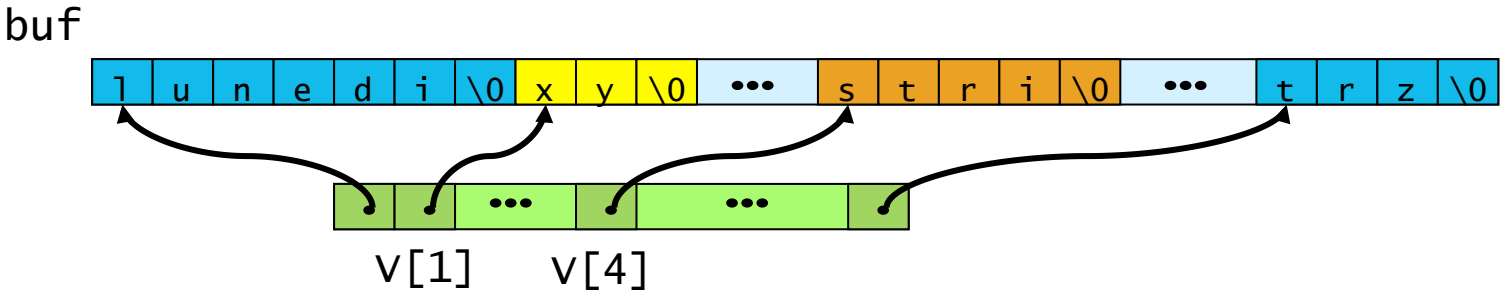
- Matrice



Esempio

Scambio riga 1 e 4, contenenti "xy" e "stri"

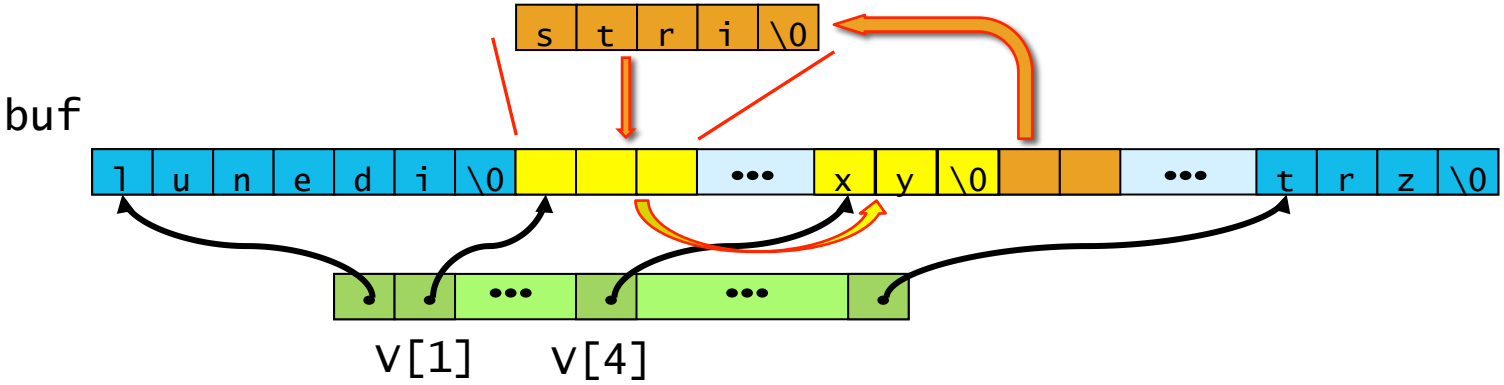
- Vettore di puntatori



Esempio

Scambio riga 1 e 4, contenenti "xy" e "stri"

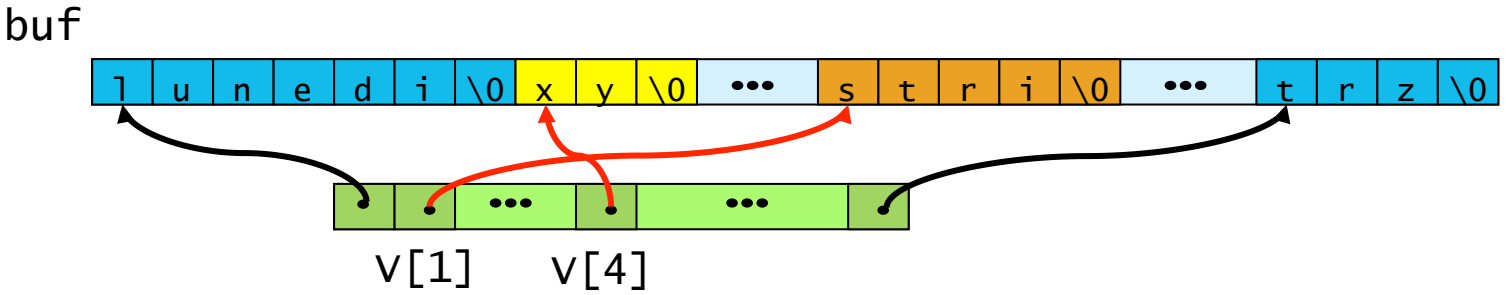
- Vettore di puntatori
- strcpy(v[1],v[4]) non può funzionare!!! NON C'E' SPAZIO!!!



Esempio

Scambio riga 1 e 4, contenenti "xy" e "stri"

- Vettore di puntatori
- Si scambiamo i puntatori:
  tmp=v[1]; v[1]=v[4]; v[4]=tmp;



`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

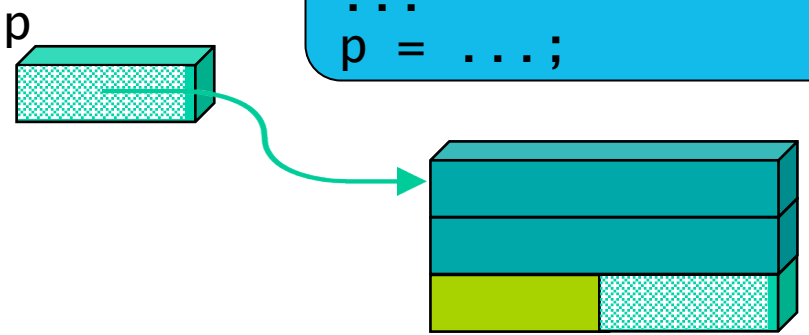
- I campi corrispondono a variabili locali di una `struct`
- Ogni campo è quindi caratterizzato da un tipo (base) e da un identificatore (unico per la struttura)

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)

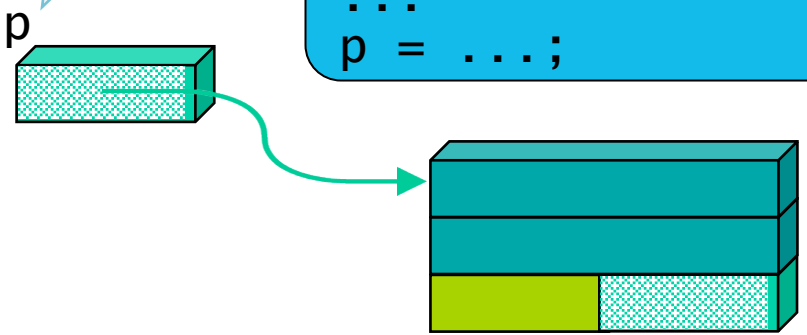
```
struct studente{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
...
struct studente *p;
...
p = ...;
```



Puntatore a struttura (intera)

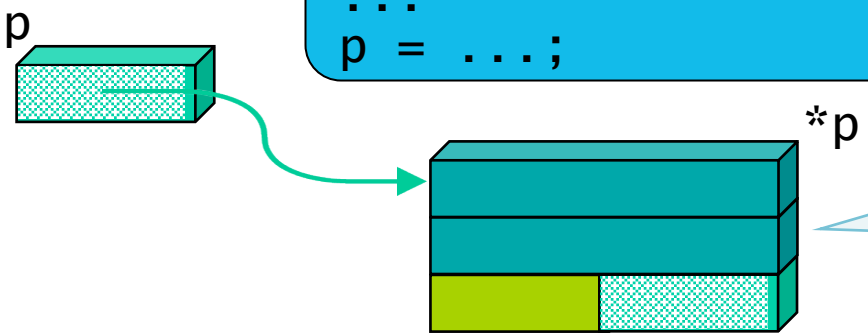
```
struct studente{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
...
struct studente *p;
...
p = ...;
```

Variabile puntatore



Puntatore a struttura (intera)

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



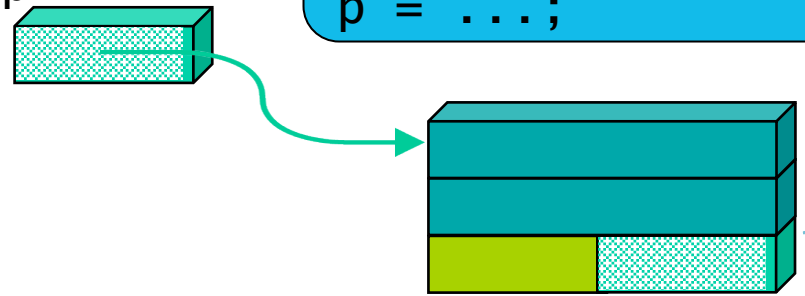
Struttura puntata da p

Puntatore a struttura (intera)

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

Variabile puntatore

p



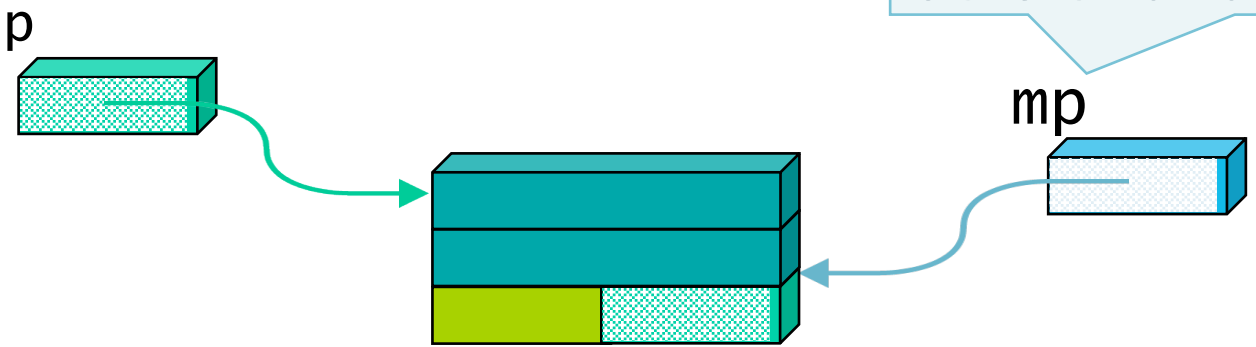
Campo `media` della struttura puntata da p

`(*p).media`

Puntatore a campo di struttura

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

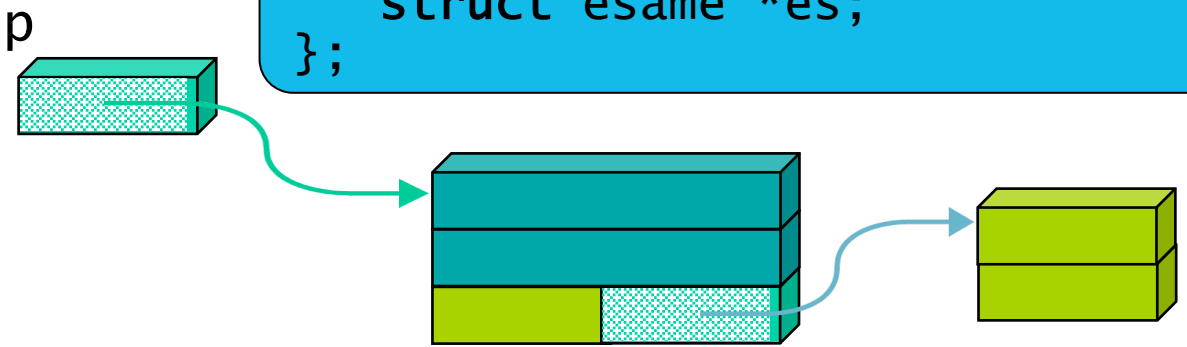
Puntatore a campo `media` della struttura puntata da `p`



Puntatore come campo di struttura

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

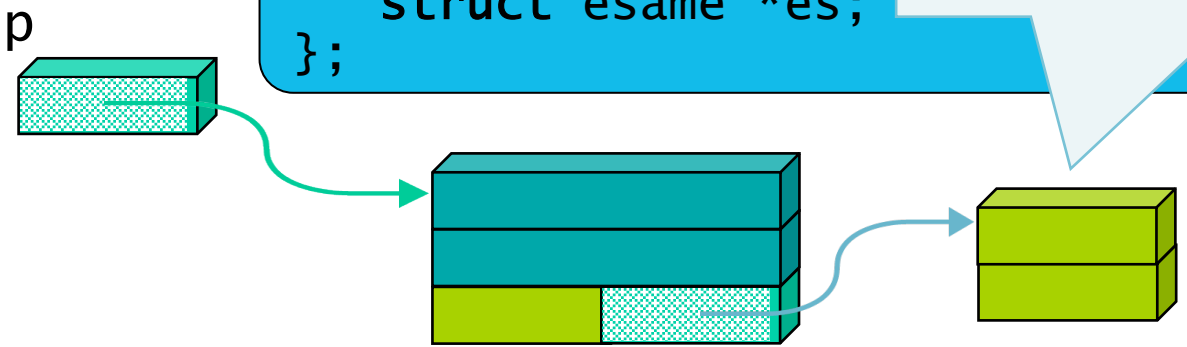
Attenzione: NON è una struttura interna:
`struct esame es;`
ma un puntatore:
`struct esame *es;`



Puntatore come campo di struttura

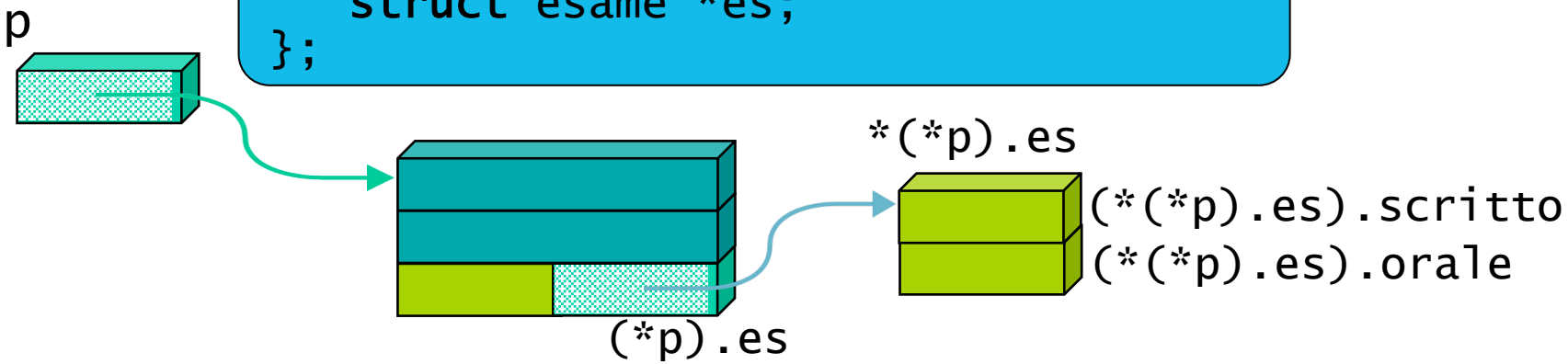
```
struct esame {
   int scritto, orale
};
struct studente {
   char cognome[MAX],
   int matricola;
   struct esame *es;
};
```

La `struct esame` (puntata)
- è FUORI da `struct studente`
- deve esistere autonomamante
- NON viene creata in modo AUTOMATICO



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
```

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
```

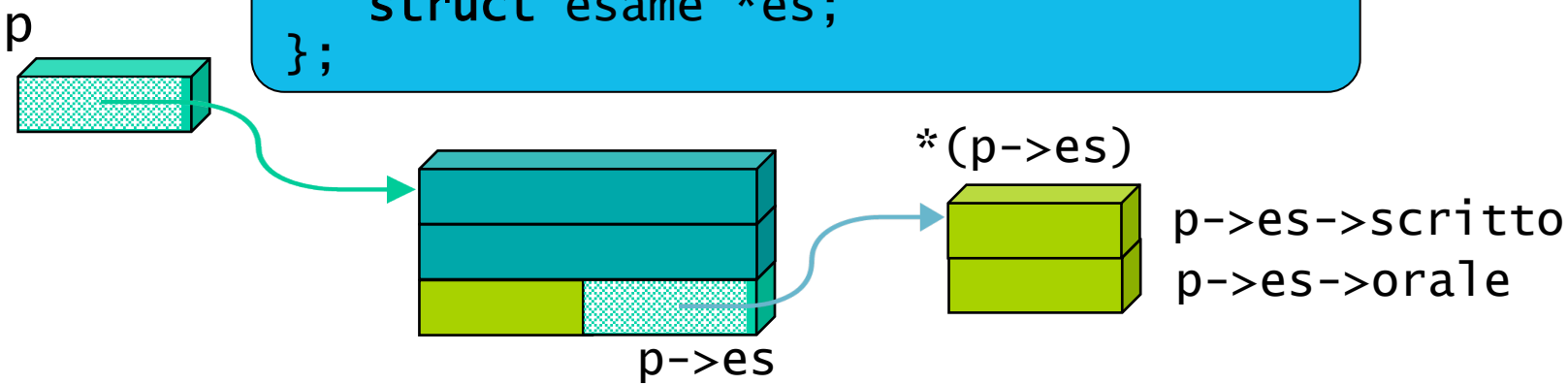
  - Si **può** scrivere

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

Si può va letto **"si deve"**:
- la notazione con le parentesi e l'asterisco è difficile da leggere
- la maggior parte dei programmatori (tutti) usano la notazione ->

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
- ATTENZIONE: definite solo per completezza (non si usano ora)

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
- ATTENZIONE: definite solo per completezza (non si usano ora)

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
- ATTENZIONE: definite solo per completezza (non si usano ora)

Esempio: soluzione 1

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

NON punterà a se stessa ma a un'altra scruct dello stesso tipo

`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
- ATTENZIONE: definite solo

Esempio: soluzione 1

ECCEZIONE: si "usa" `struct studente` (per definire un puntatore) PRIMA di aver definito struct studente (lo è dopo **};**)

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

Struct ricorsive

- Si dice **ricorsiva** una stru
  puntatori a strutture dello
- Servono per realizzare liste
- ATTENZIONE: definite solo

Esempio: soluzione 1

ECCEZIONE: si "usa" `struct studente` (per definire un puntatore) PRIMA di aver definito struct studente (lo è dopo **};**)
Si può fare solo coi puntatori (non con altri tipi) perché la DIMENSIONE è nota (32 o 64 bit, dipende dal processore)

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

Esempio: soluzione 2

nuovo tipo puntatore a `struct studente` (usata ancora prima di iniziarne la definizione!): prchè è 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[MAX], nome[MAX];
  int matricola;
  T_stud *link;
};
```

Esempio: soluzione 3

nuovo tipo T_stud
(di dimensione ignota)

```
typedef struct studente T_stud;
struct studente {
  char cognome[MA
  int matricola;
  T_stud *link;
};
```

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

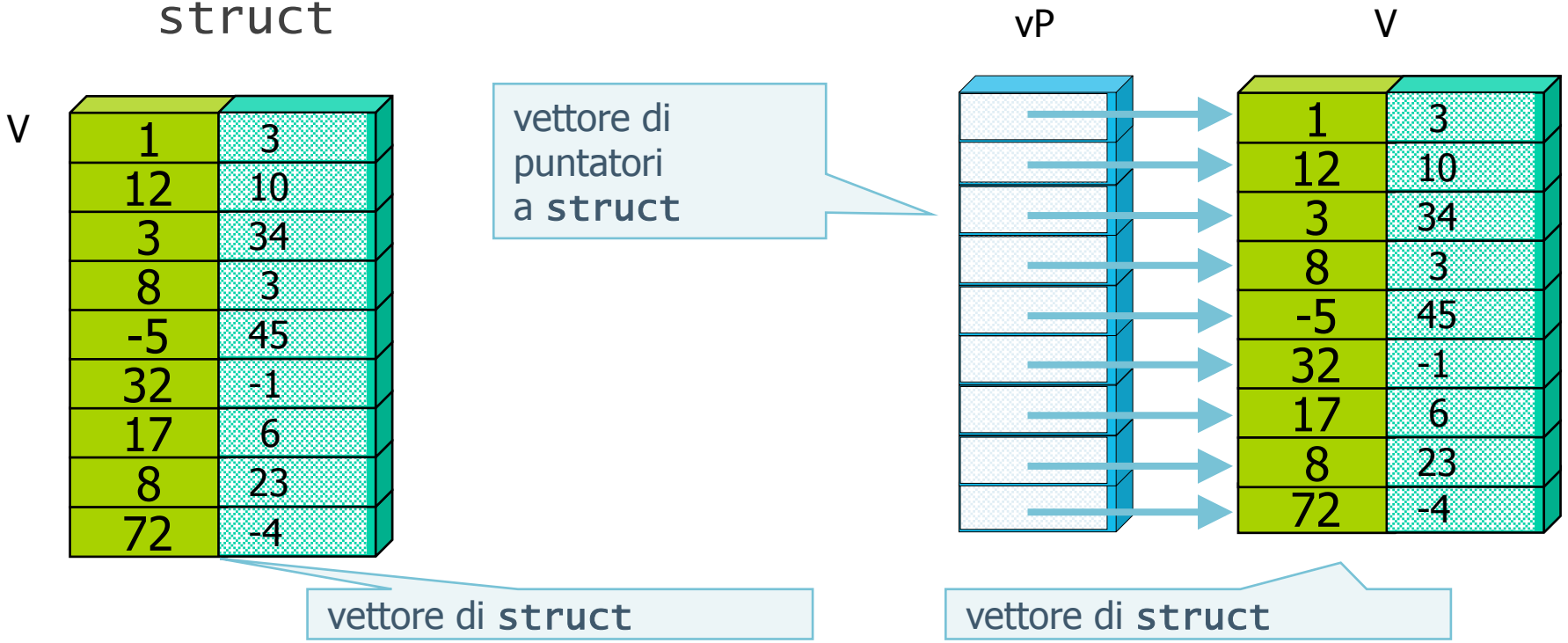
Esempio: soluzione 3

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

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`

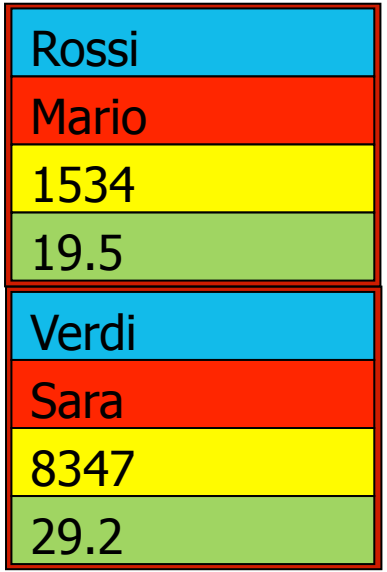


Esempio

- Tipo struct

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

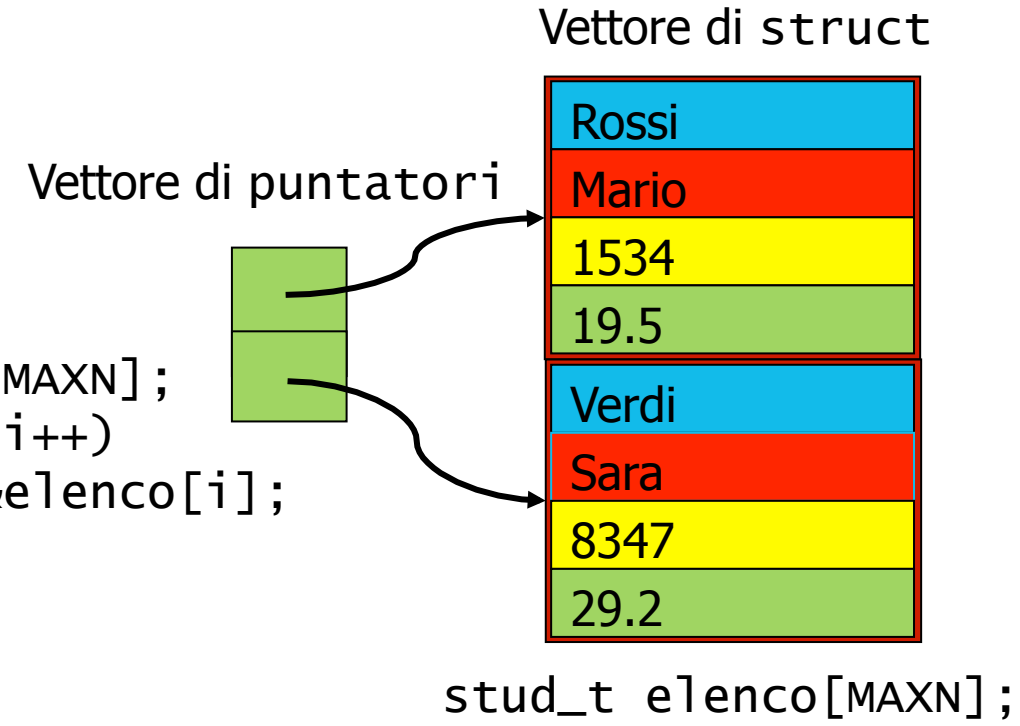
Vettore di `struct`




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

Vettore di puntatori a `struct`

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



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 `struct` (scambio di valori)

```c
/* ... #include e #define */
typedef struct studente {
  char cognome[MAXS]; char nome[MAXS];
  int matr; float media;
} stud_t;
/* ... prototipi */
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 0;
}
```

```c
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;
}
```

Soluzione 1: vettore di `struct` (scambio di valori)

```c
/* ... #include e #define */
typedef struct studente {
  char cognome[MAXS]; char nome[MAXS];
  int matr; float media;
} stud_t;
/* ... prototipi */
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 0;
}
```

```c
int leggiStud(char *nomeFile, stud_t *el, int nmax) {
  ...
    while(fscanf(fp,"%s%s%d%f", el[n].cognome,
                 el[n].nome, &el[n].matr,
                 &el[n].media)==EOF) break;
  }
  fclose(fp);
  return n;
}
```

dimensione massima del vettore

dimensione effettivamente utilizzata del vettore (ricavata dal file)

Soluzione 1: vettore di `struct` (scambio di valori)

```c
/* ... #include e #define */
typedef struct studente {
  char cognome[MAXS]; char nome[MAXS];
  int matr; float media;
} stud_t;
/* ... prototipi */
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 0;
}
```

```c
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;
}
```

Soluzione 1: vettore di `struct` (scambio di valori)

```
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);
}
// confronto per struct ricevute by value
int confrMatr(stud_t s1, stud_t s2) {
  return s1.matricola-s2.matricola;
}
```

```
// confronto per struct ricevute by reference/pointer
int confrMatrByRef(stud_t *ps1, stud_t *ps2) {
  return ps1->matricola-ps2->matricola;
}
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 (confrMatr(el[j],el[imin])<0)
        imin = j;
    temp = el[i]; el[i] = el[imin]; el[imin] = temp;
  }
}
```

Soluzione 1: vettore di `struct` (scambio di valori)

```
void scriviStud(char *nomeFile, stud_t *el,int n) {
  int i;
  FILE *fp = fopen(nomeFile,"w");
  for (i=0; i<n; i++) {
    fprintf(fp
  }
  fclose(fp);
}
// confronto per struct ricevute by value
int confrMatr(stud_t s1, stud_t s2) {
  return s1.matricola-s2.matricola;
}
```

passaggio by value:
la funzione riceve una "copia"
delle struct da confrontare

```
// confronto per struct ricevute by reference/pointer
int confrMatrByRef(stud_t *ps1, stud_t *ps2) {
  return ps1->matricola-ps2->matricola;
}
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 (confrMatr(el[j],el[imin])<0)
        imin = j;
    temp = el[i]; el[i] = el[imin]; el[imin] = temp;
  }
}
```

selection sort

Soluzione 1: vettore di `struct` (scambio di valori)

```
void scriviStud(char *nomeFile, stud_t *el,int n) {
  int i;
  FILE *fp = fopen(nomeFile,"w");
  for (i=0; i<
    fprintf(fp
  }
  fclose(fp);
}
// confronto per struct ricevute by value
int confrMatr(stud_t s1, stud_t s2) {
  return s1.matricola-s2.matricola;
}
```

ALTERNATIVA:
passaggio by reference/pointer:
la funzione riceve i puntatori
alle struct da confrontare

```
// confronto per struct ricevute by reference/pointer
int confrMatrByRef(stud_t *ps1, stud_t *ps2) {
  return ps1->matricola-ps2->matricola;
}
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;
  }
}
```

Soluzione 2: vettore di puntatori a `struct`
(scambio di puntatori)

```
/* ... #include e #define */
/* ... Typedef struct ... */
/* ... prototipi */
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];
  ordRifStudPerMatr(elencoRif,ns);
  lavoroSuElencoRifOrd(elencoRif,ns);
  scriviRifStud(argv[2],elencoRif,ns);
  return 0;
}
/* ... leggiStud e scrivistud non cambiano */
```

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

Soluzione 2: vettore di puntatori a `struct`
(scambio di puntatori)

```
/* ... #include e #define */
/* ... Typedef struct ... */
/* ... prototipi */
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];
  ordRifStudPerMatr(elencoRif,ns);
  lavoroSuElencoRifOrd(elencoRif,ns);
  scriviRifStud(argv[2],elencoRif,ns);
  return 0;
}
/* ... leggiStud e scr
```

```
int confrMatrByRef(stud_t *ps1, stud_t *ps2) {
                                          ;

                                          nt 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;
```

dimensione massima del vettore

dimensione effettivamente utilizzata del vettore (ricavata dal file)

Soluzione 2: vettore di puntatori a `struct`
(scambio di puntatori)

```c
/* ... #include e #define */
/* ... Typedef struct ... */
/* ... prototipi */
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];
  ordRifStudPerMatr(elencoRif,ns);
  lavoroSuElencoRifOrd(elencoRif,ns);
  scriviRifStud(argv[2],elencoRif,ns);
  return 0;
}
/* ... leggiStud e scrivistud non cambiano */
```

Prima carica il vettore di struct

```c
int confrMatrByRef(stud_t *ps1, stud_t *ps2) {
                                          ;

                                       nt 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;
  }
}
```

Soluzione 2: vettore di puntatori a `struct`
(scambio di puntatori)

```c
/* ... #include e #define */
/* ... Typedef struct ... */
/* ... prototipi */
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];
  ordRifStudPerMatr(elencoRif,ns);
  lavoroSuElencoRifOrd(elencoRif,ns);
  scriviRifStud(argv[2],elencoRif,ns);
  return 0;
}
/* ... leggiStud e scrivistud non cambiano */
```

```c
int confrMatrByRef(stud_t *ps1, stud_t *ps2) {
                                          ;

                                       nt 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)

                                       min]=temp;
}
```

Prima carica il vettore di struct

Poi "aggancia" i puntatori alle struct

Soluzione 2: vettore di puntatori a `struct`
(scambio di puntatori)

```
/* ... #inclu
/* ... Typed
/* ... proto
int main(int a
  stud_t elenco[MAXN], *elencoRif[MAXN];
  int i, ns = leggiStud(argv[1],elenco,MAXN);
  for (i=0; i<ns; i++)
    elencoRif[i]=&elenco[i];
  ordRifStudPerMatr(elencoRif,ns);
  lavoroSuElencoRifOrd(elencoRif,ns);
  scriviRifStud(argv[2],elencoRif,ns);
  return 0;
}
/* ... leggiStud e scrivistud non cambiano */
```

La funzione di ordinamento riceve SOLO il vettore di puntatori a struct

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

Soluzione 2: vettore di puntatori a `struct`
(scambio di puntatori)

```c
/* ... #include e #define */
/* ... Typedef struct ... */
/* ... prototipi */
int main(int argc, char *argv[]) {
  ...
  ...N);
  ...
  elencoRif[i]=&elenco[i];
  ordRifStudPerMatr(elencoRif,ns);
  lavoroSuElencoRifOrd(elencoRif,ns);
  scriviRifStud(argv[2],elencoRif,ns);
  return 0;
}
/* ... leggiStud e scrivistud non cambiano */
```

Confronta struct a partire dai puntatori
Scambia puntatori

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

Soluzione 2: vettore di puntatori a `struct`
(scambio di puntatori)

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

Soluzione 2: vettore di puntatori a `struct`
(scambio di puntatori)

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

Come scriviStud, ma parte da vettore di puntatori:
`stud_t **elR`
equivale a
`stud_t *elR[]`

Soluzione 2: vettore di puntatori a `struct`
(scambio di puntatori)

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

… quindi si usano le frecce (->) anziché i punti per accedere ai campi delle struct

vettore di puntatori a `struct:` vantaggi

```
/* ... Parti omesse */
  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
...
```

vettore di puntatori a `struct:` vantaggi

```
/* ... Parti omesse */
  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
...
```

Più ordinamenti possibili insieme

vettore di puntatori a `struct:` vantaggi

```
/* ... Parti omesse */
  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
...
```

Un solo vettore di struct: non REPLICA i dati

vettore di puntatori a `struct:` vantaggi

```
/* ... Parti omesse */
  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
...
```

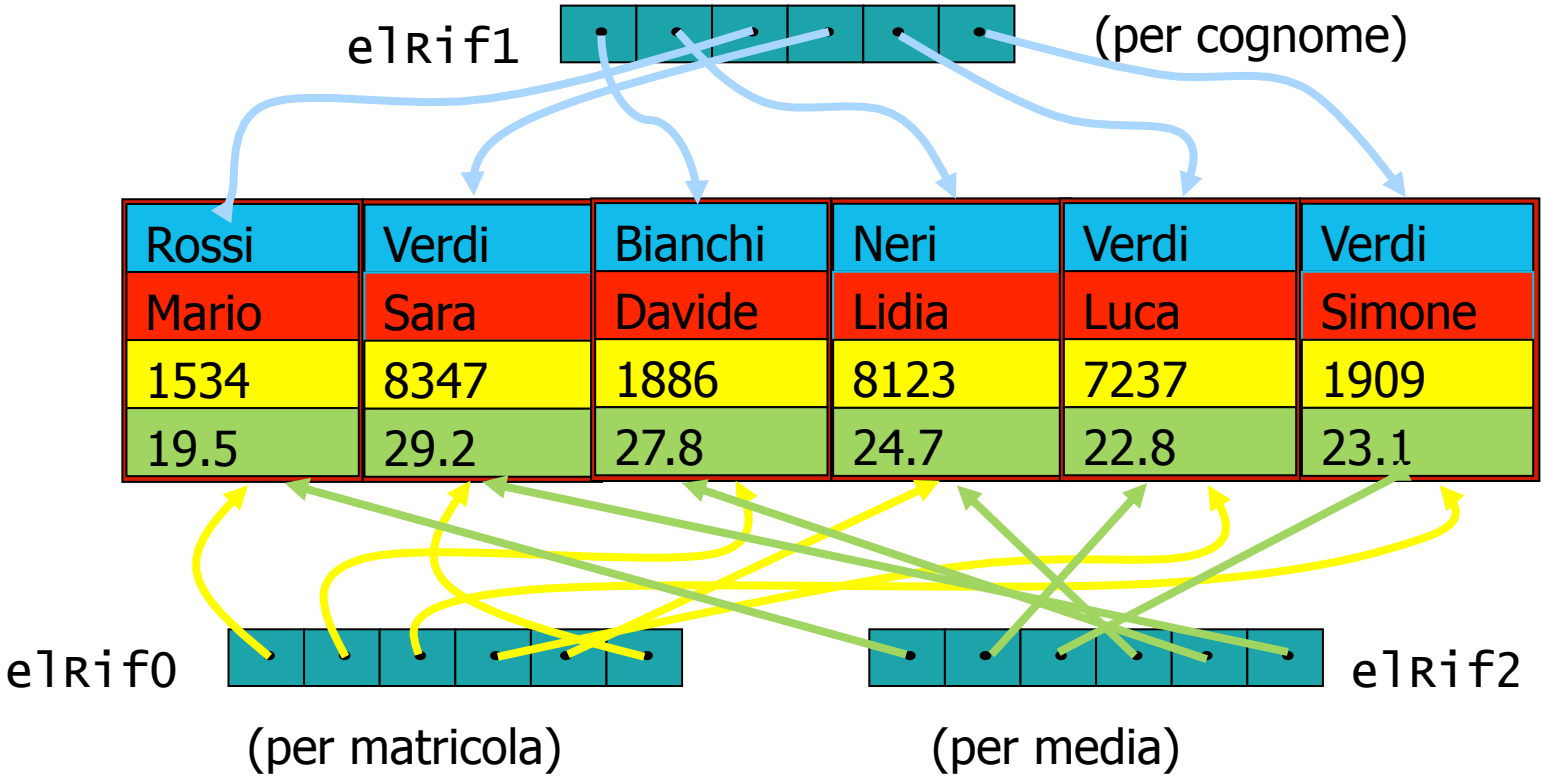
Tre vettori di puntatori:
REPLICA (solo) i puntatori

vettore di puntatori a `struct:` vantaggi

```
/* ... Parti omesse */
  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
...
```

Tre diverse funzioni di ordinamento, applicate ai tre vettori di puntatori

In concreto …


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

Esempio

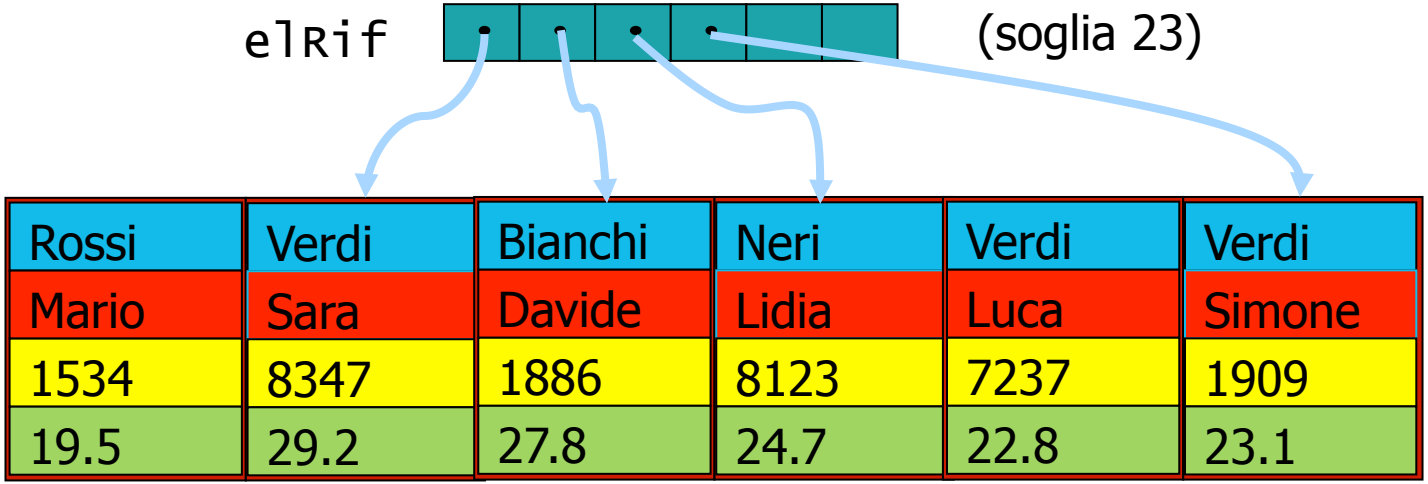
- Scrivere un programma che:
  - acquisisce un elenco di studenti da un file il cui nome è ricevuto come primo argomento al main
  - filtra l'elenco per medie superiori a una soglia (terzo argomento al main)
  - scrive l'elenco filtrato su un secondo file, il cui nome è ricevuto come secondo argomento.
- Soluzione: dato il vettore di TUTTI i dati, si genera come risultato il vettore dei puntatori ai dati selezionati
  - ATTENZIONE: se occorresse solo questo, NON SAREBBE NECESSARIO, basterebbe stampare i dati filtrati, senza il vettore dei puntatori
  - Il vettore (risultato) puo invece essere utile per eventuali elaborazioni successive (NON PREVISTE IN QUESTO ESEMPIO)

```
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;
}
```

In concreto …


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

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.`

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

```c
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;
  }
}
```

```
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.

```c
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;
}
```

In concreto …

`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 media

```
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;
}
```

In concreto …

`elInd`

(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);
```

Puntatore a funzione

- 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.