

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

Capitolo 1: Il tipo di dato puntatore

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

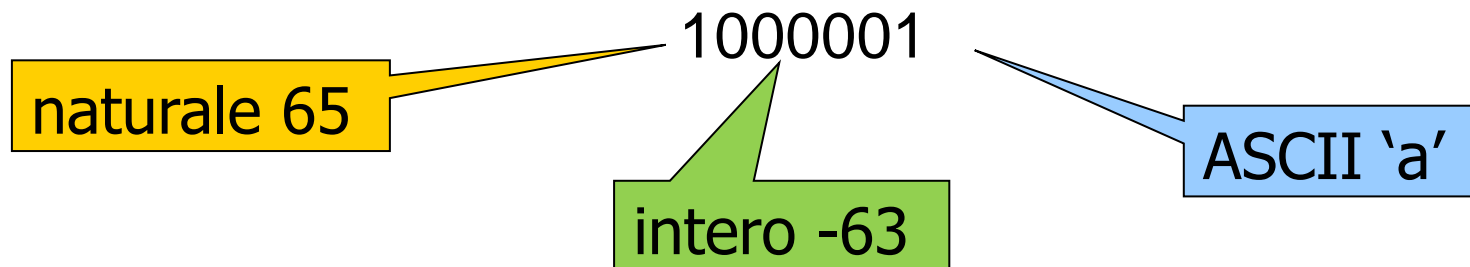


I dati in memoria centrale

Dati memorizzati come sequenze di 1 e 0 che codificano simboli di insiemi finiti:

- naturali, interi, razionali, caratteri

La sequenza ha significato solo se associata alla codifica:





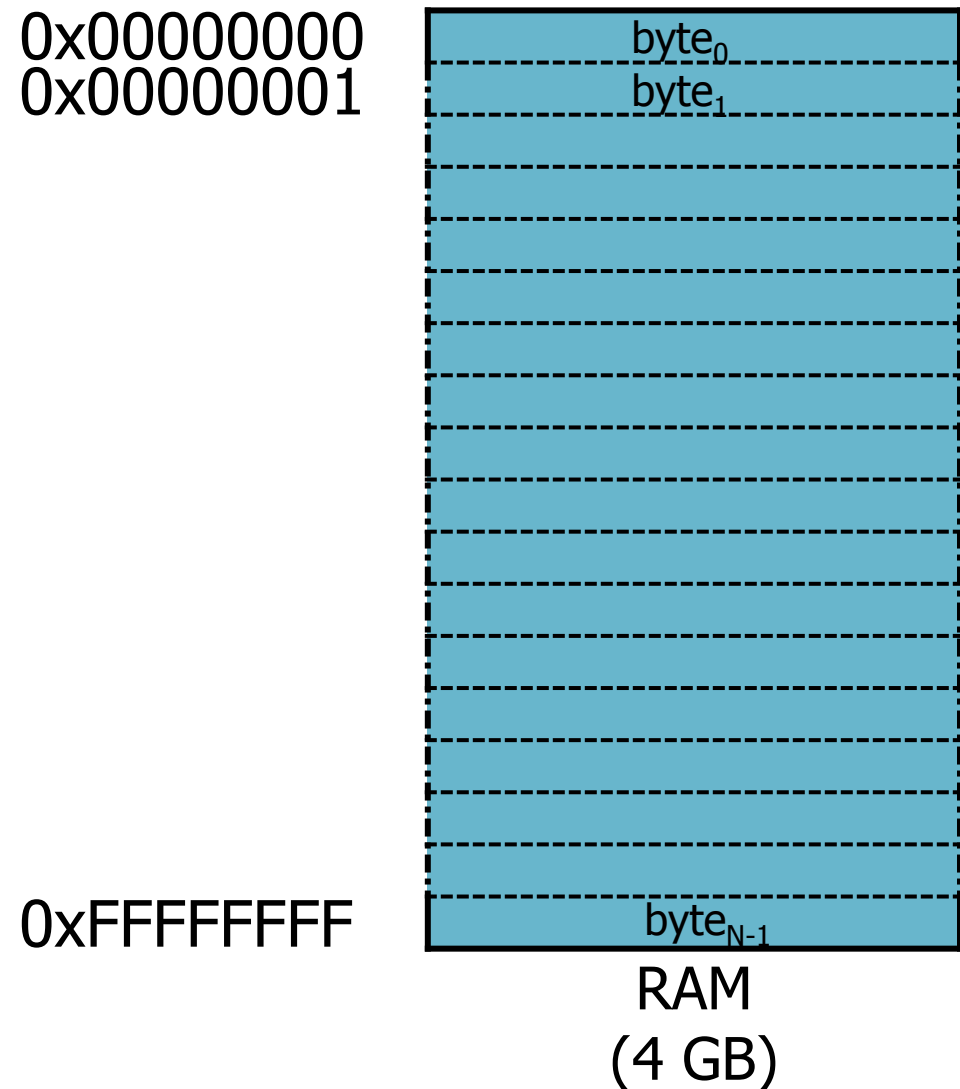
Il modello della memoria

- ❑ **Memoria RAM:** matrice di bit con n righe e m colonne. Esempio: matrice da 128 bit:
 - 32 righe x 4 colonne
 - 16 righe x 8 colonne
 - 8 righe x 16 colonne

In generale:

- n è una potenza di 2
- m è un multiplo di 8 (1 byte = 8 bit)

Esempio di RAM



- ❑ Memoria da 4 GB ($N=2^{32}$ byte)
- ❑ byte-addressable
- ❑ **celle** e **parole** da 1 byte





Cella e parola

□ Cella:

- gruppo di k bit cui si accede unitariamente
- in generale $k = 8 \Rightarrow 1$ byte
- cella da 1 byte \Rightarrow memoria byte-addressable
- identificata da un indirizzo: N celle \Rightarrow indirizzi tra 0 e $N-1$
- Indirizzo: stringa di $\lceil \log_2 N \rceil$ bit

□ Parola:

- raggruppamento di celle
- in generale occupa 4 o 8 byte
- può stare su 1 riga o su più righe successive
- raramente word-addressable.



Big/Little Endian

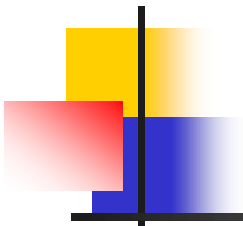
Parole su più celle:

❑ **Big Endian:**

- il Most Significant Byte occupa l'indirizzo di memoria più basso
- il Least Significant Byte occupa l'indirizzo di memoria più alto

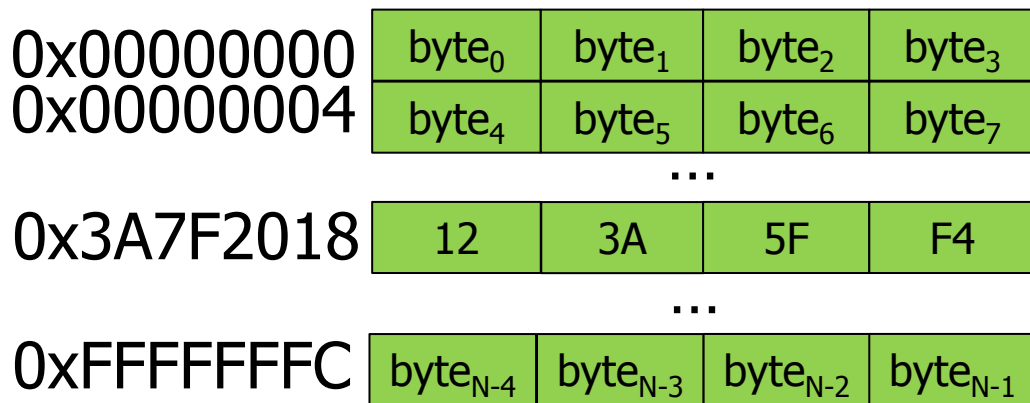
❑ **Little Endian:**

- il Most Significant Byte occupa l'indirizzo di memoria più alto
- il Least Significant Byte occupa l'indirizzo di memoria più basso.

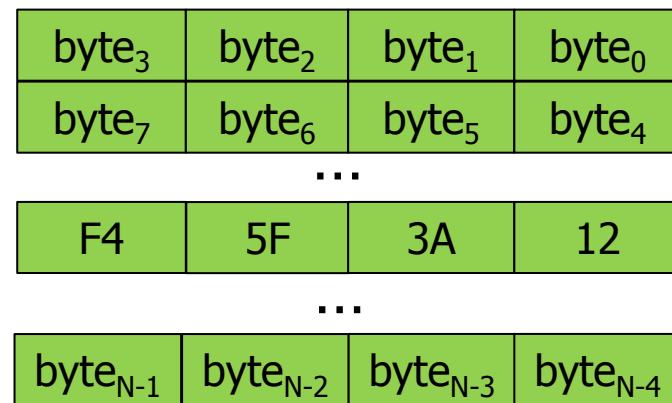


- ❑ Memoria da 4 GB
- ❑ byte-addressable
- ❑ celle da 1 byte

- ❑ parole da 4 byte
- ❑ dato 0x123A5FF4 all'indirizzo 0x3A7F2018



Big Endian



Little Endian



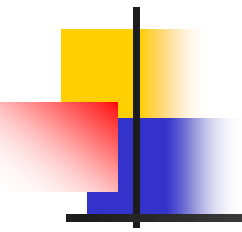
Allineamento

“allineata”:

- ❑ parola di memoria che inizia ad un indirizzo divisibile per il numero di byte che compongono la parola stessa

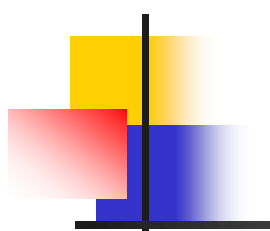
Esempio: memoria con 8 celle da 1 byte e parole da 2 byte con tecnica Big Endian: **allineata**, **non allineata**

0x0	MSB	LSB
0x2		
0x4		MSB
0x6	LSB	

- 
-
- ❑ anche nelle memorie byte-addressable si legge/scrive per parole
 - ❑ i dati di dimensione \geq parole sono allineati

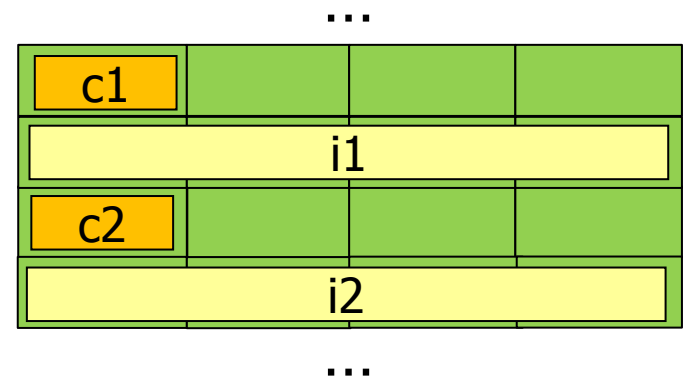
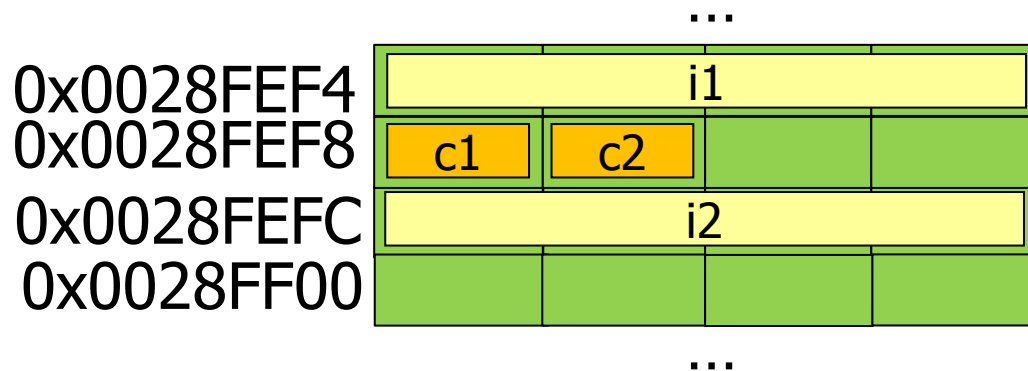
Esempio:

- ❑ memoria da 4 GB byte-addressable con celle da 1 byte e parole da 4 byte
- ❑ 2 struct con gli stessi campi in ordine diverso



```
typedef struct item1_s {  
    int i1;  
    char c1, c2;  
    int i2;  
} Item1;
```

```
typedef struct item2_s {  
    char c1;  
    int i1;  
    char c2;  
    int i2;  
} Item1;
```

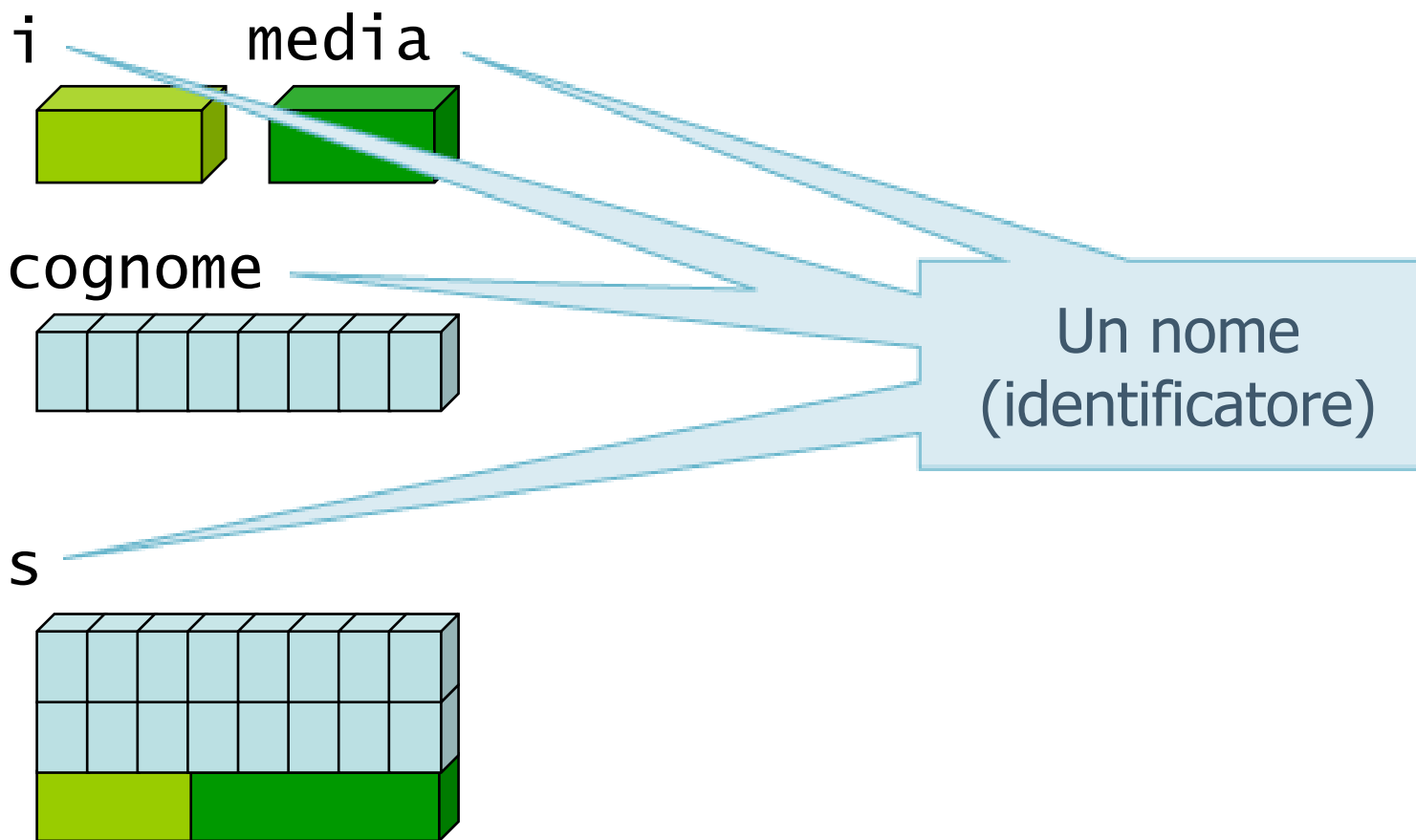




Le variabili

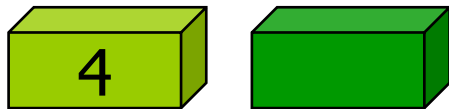
- ❑ I dati in memoria stanno in **contenitori** caratterizzati da:
 - nome (identificatore univoco)
 - tipo
- ❑ se i dati possono variare nel tempo, i contenitori si dicono **variabili**
- ❑ Compilatore/linker (e loader) collocano (**allozano**) le variabili a certi indirizzi di memoria su 1 o più parole e mantengono una tabella di corrispondenza identificatore-indirizzo-tipo.

Come si identifica una variabile?

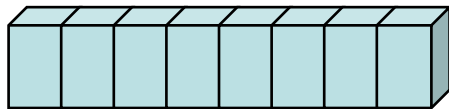




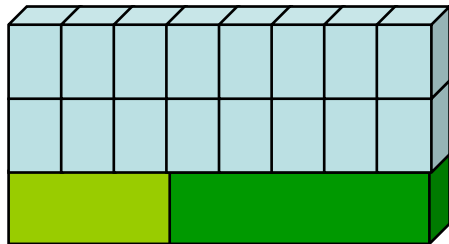
i media



cognome[i]



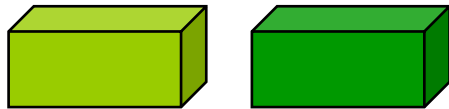
s



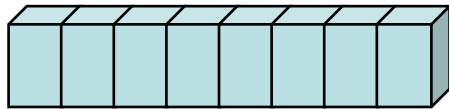
Nome+indice
(array)



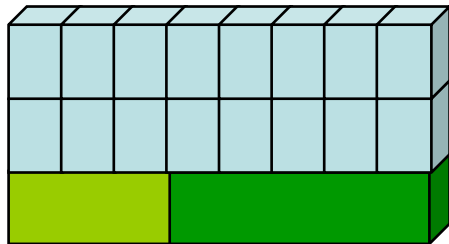
i media



cognome[i]



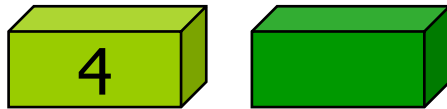
s.cognome



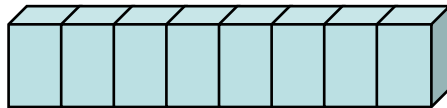
Sequenza di
nomi



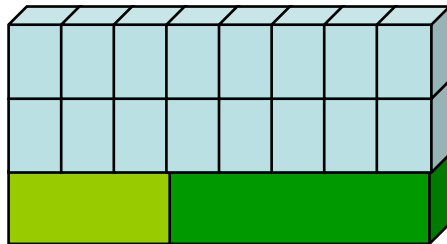
i media



cognome



s.cognome[i]



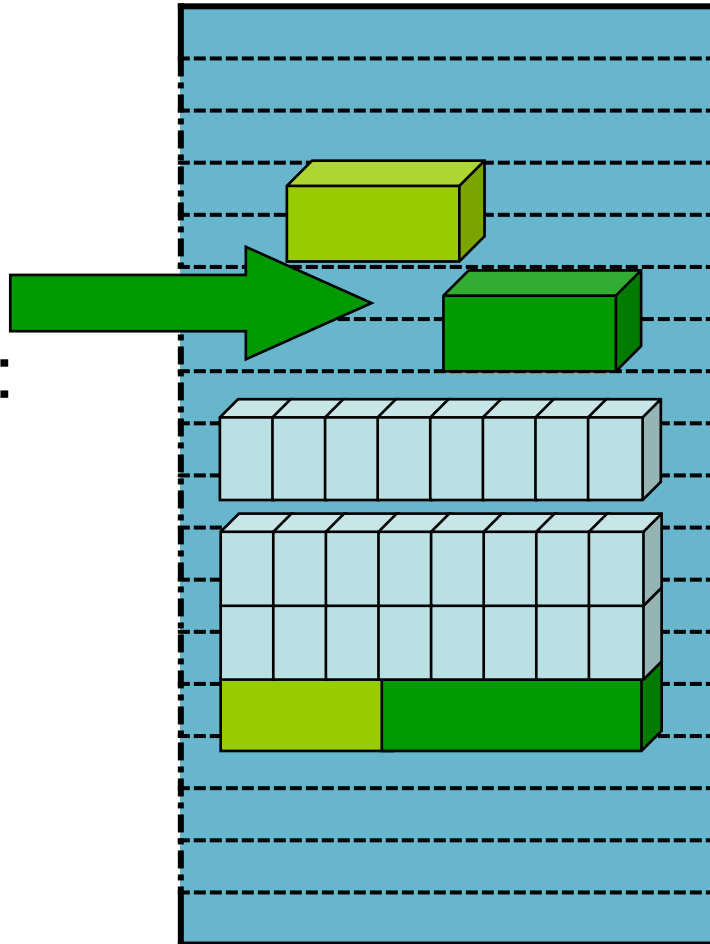
Sequenza di
nomi+indice

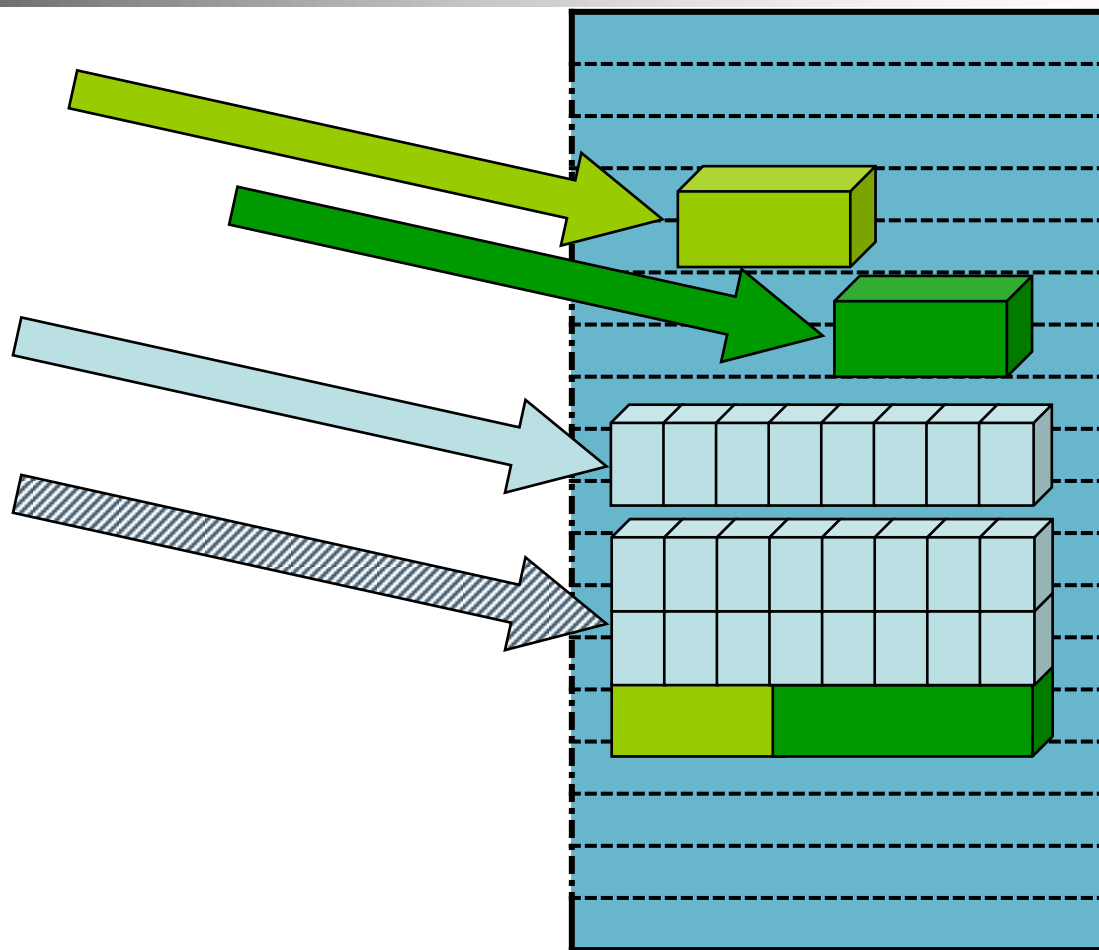
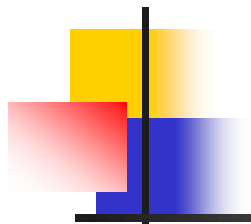
Il puntatore

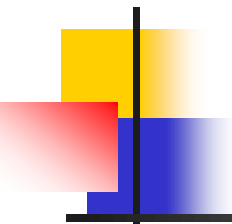
1. Strumento alternativo alle variabili per l'accesso ai dati.

Informazioni necessarie:

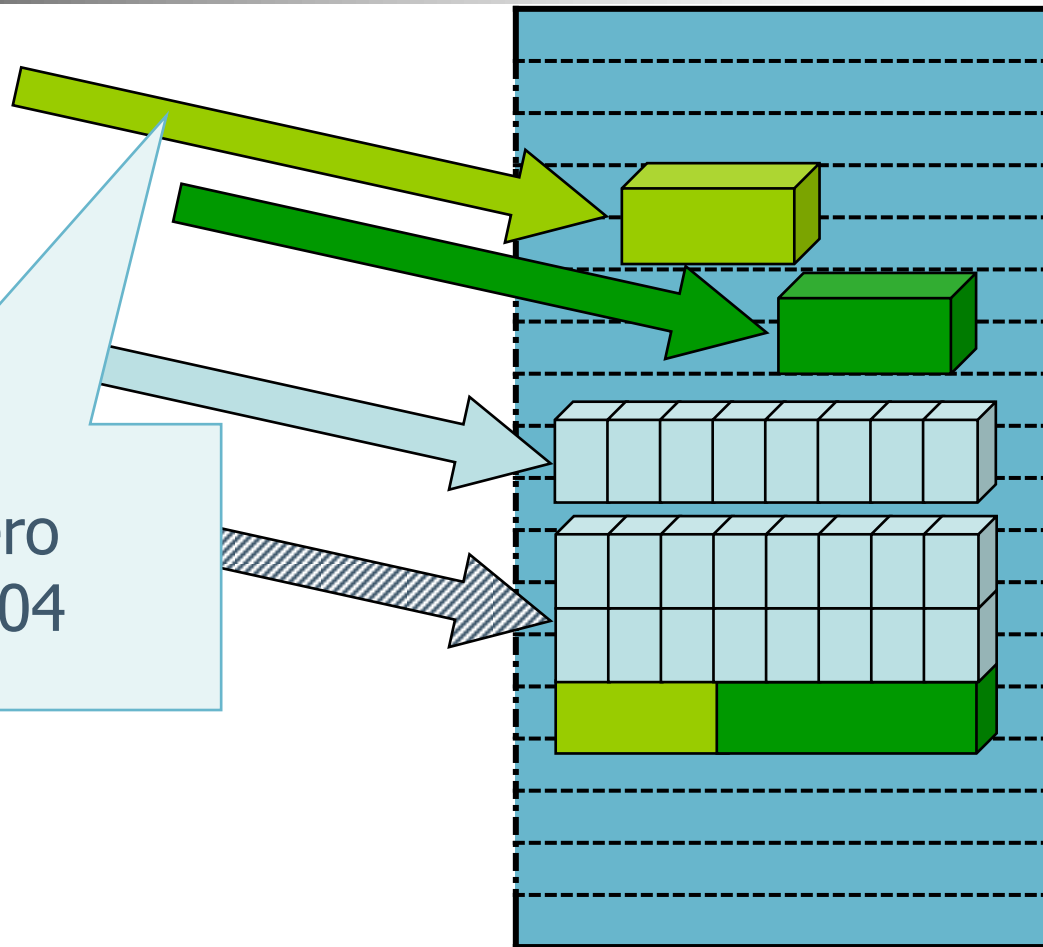
- dove si trova il dato in memoria (indirizzo)
- come è codificato (tipo di dato)

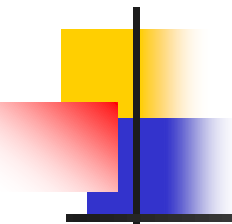




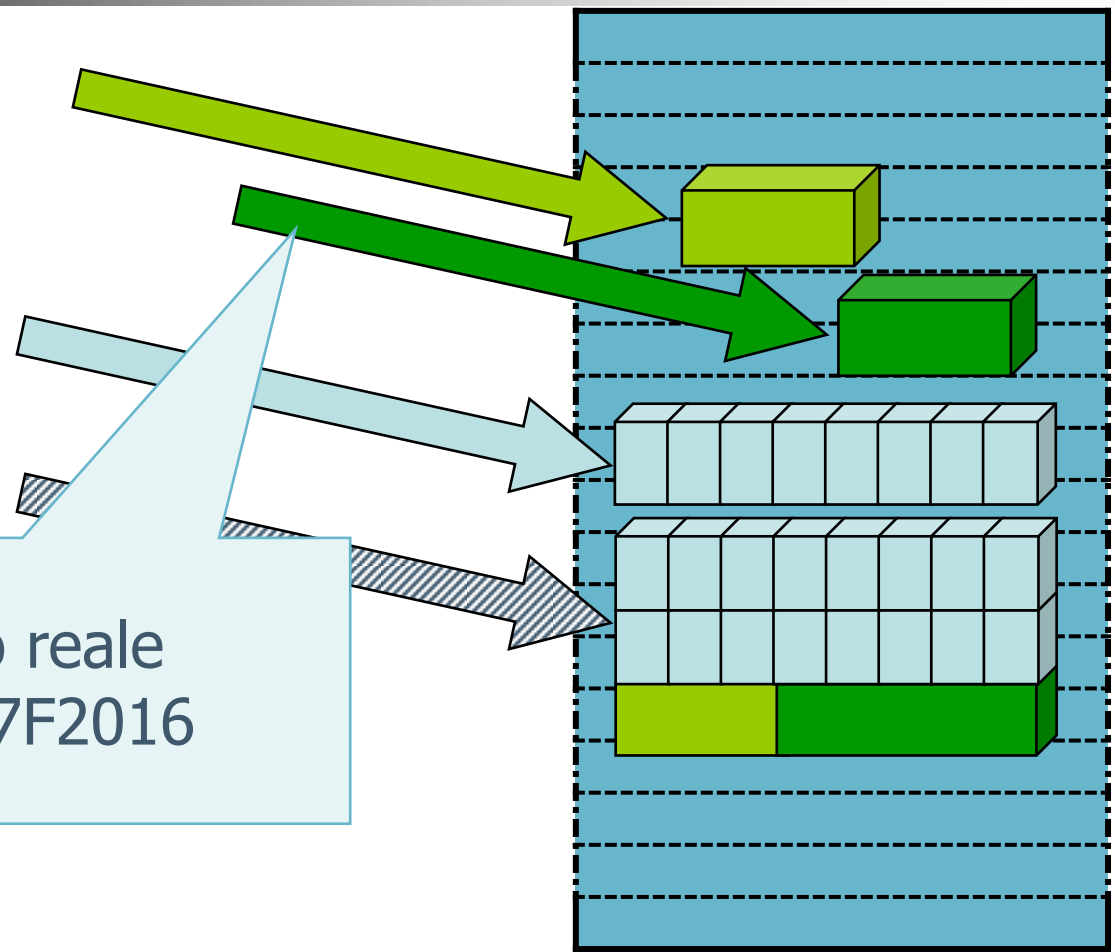


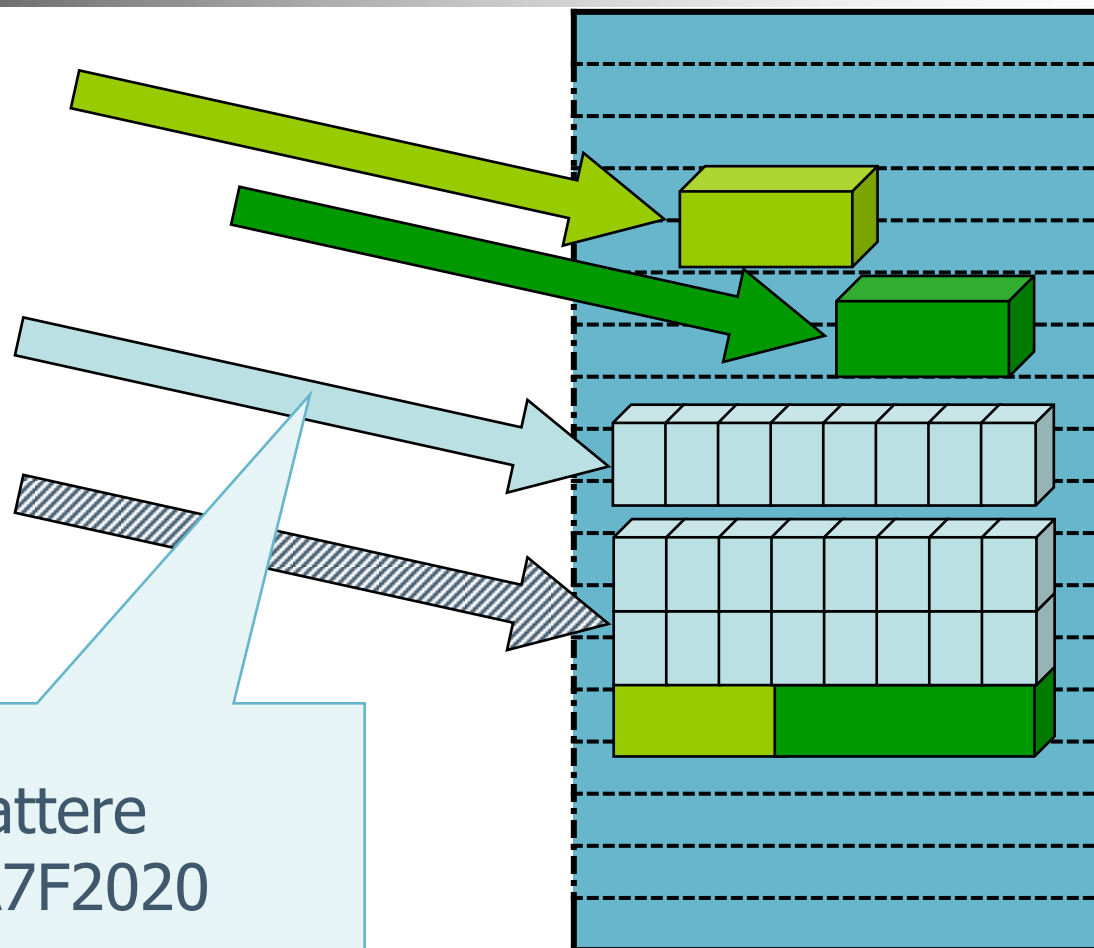
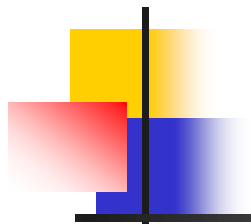
Puntatore dato intero
all'indirizzo 3A7F0304



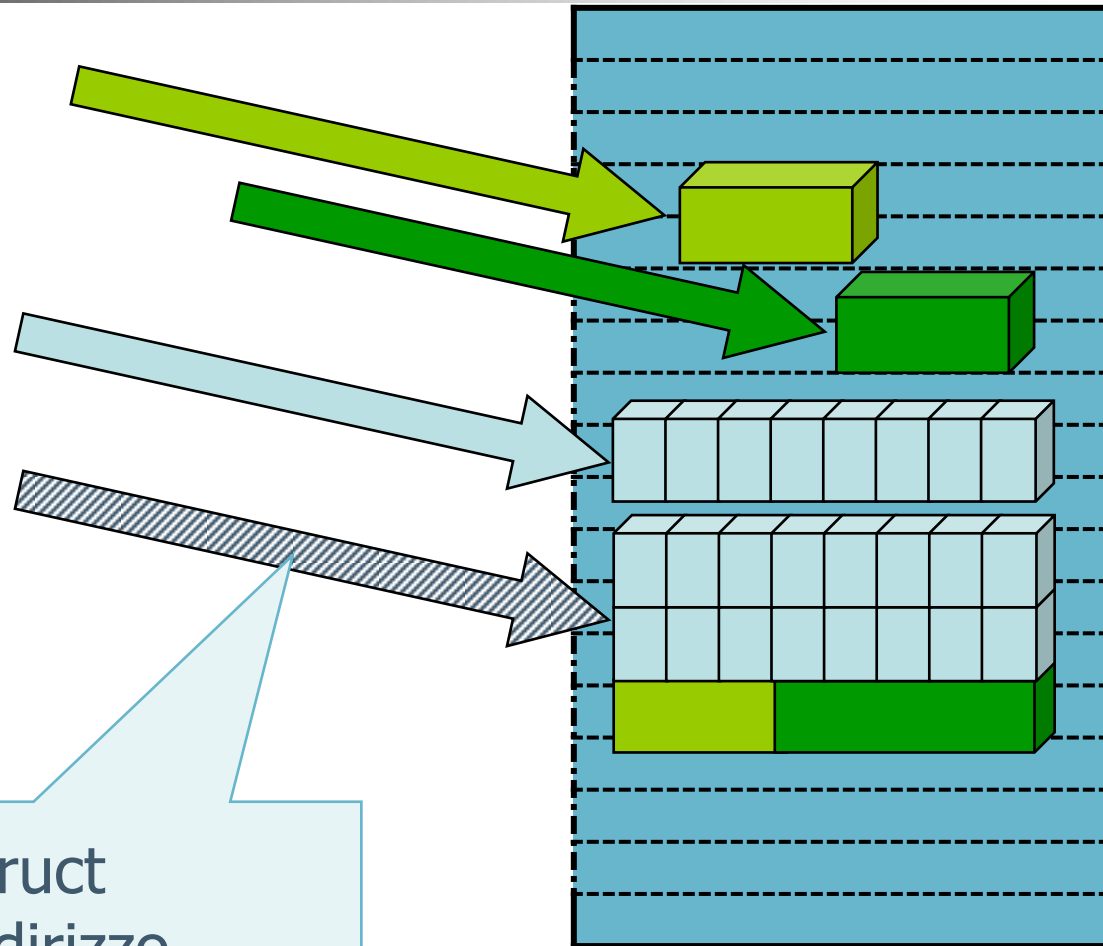
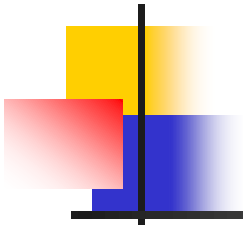


Puntatore dato reale
all'indirizzo 3A7F2016





Puntatore carattere
all'indirizzo 3A7F2020

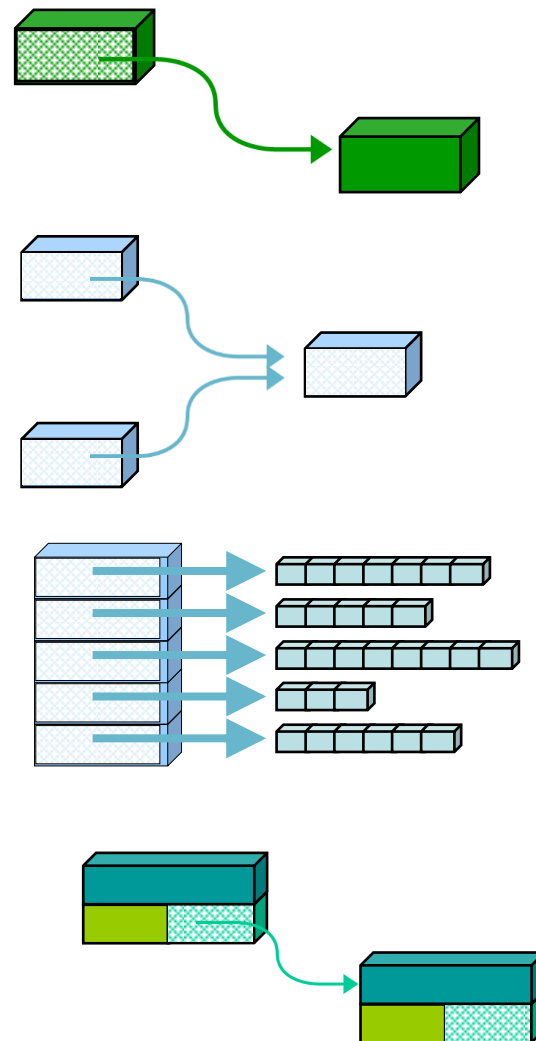


Puntatore a struct
studente all'indirizzo
3A7F2020

Il puntatore

2. Il puntatore è un'informazione manipolabile (si può calcolare, modificare, assegnare), a differenza di un identificatore (che non può essere modificato)

Novità: il puntatore è (anche) un dato (che punta a un dato)!

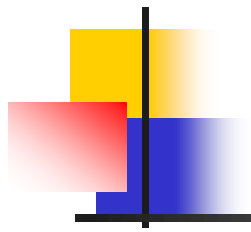




Operatori: riferimento

S





Data S (variabile con nome)

S





Puntatore a S

Data S (variabile con nome)



S



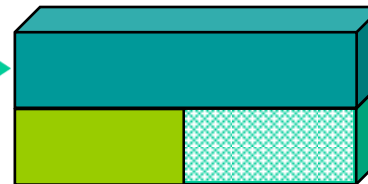


Puntatore a S

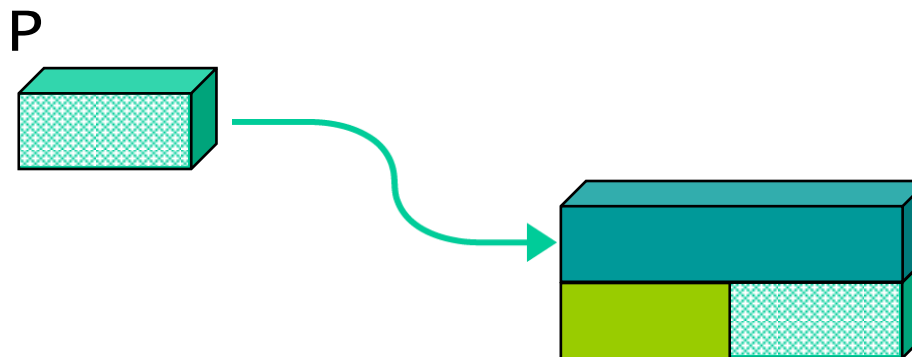


Data S (variabile con nome)

S



Operatori: dereferenziazione





Variable Puntatore

P





Variabile Puntatore

P



Oggetto (senza nome)
puntato da P





Variabile Puntatore

P



*P



Oggetto (senza nome)
puntato da P



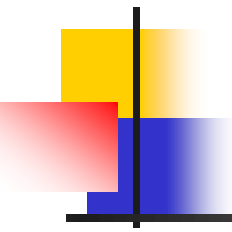
* e &

- I simboli * e & sono utilizzati, in definizioni e uso dei puntatori, per indicare (in forma prefissa)
 - *... : dato puntato da ...
 - &... : puntatore a ...
- Gli operatori dereferenziazione * e riferimento & sono duali.



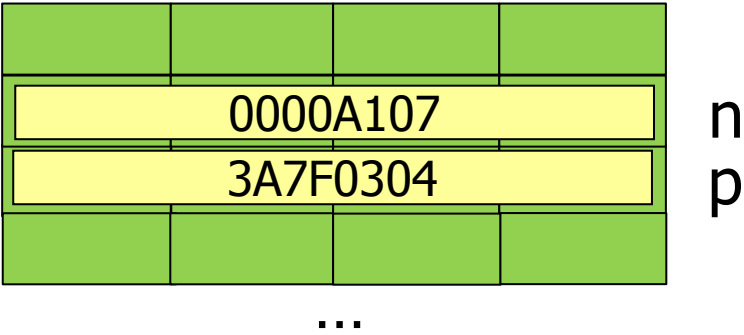
Esempio

- ❑ Variabile intera $n = 41223$ ($=0x0000A107$) all'indirizzo $0x3A7F0304$
- ❑ Variabile puntatore a intero p (già dichiarata) all'indirizzo $0x3A7F0308$
- ❑ Memoria 4 GB, byte-addressable, celle da 1 byte, parole da 4 byte



`p = &n;`

`0x3A7F0304`
`0x3A7F0308`



`printf("n: %d\n", n);`
`printf("n: %d\n", *p);`

sono equivalenti

`scanf("%d", &n);`
`scanf("%d", p);`

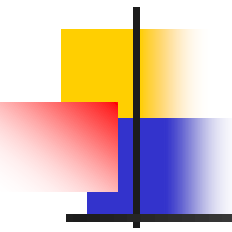
sono equivalenti



Dichiarazione

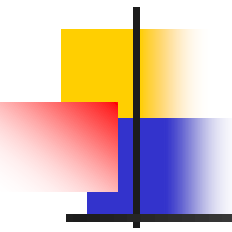
- ❑ La dichiarazione di una variabile puntatore richiede il riferimento a un tipo base (quello del dato puntato)

```
int *px;  
char *p0, *p1;  
struct studente *pstud;  
FILE *fp;
```

- 
- ❑ La dichiarazione di una variabile puntatore richiede il riferimento a un tipo base (quello del dato puntato)

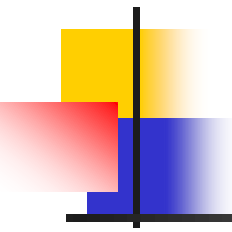
Variabile px di tipo
"puntatore a intero"

```
int *px;  
char *p0, *p1;  
struct studente *pstud;  
FILE *fp;
```

- 
- ❑ La dichiarazione di una variabile puntatore richiede il riferimento a un tipo base (quello del dato puntato)

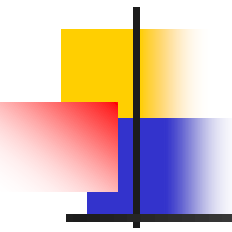
Variabili p0 e p1 di tipo
"puntatore a carattere"

```
int *px;  
char *p0, *p1;  
struct studente *pstud;  
FILE *fp;
```

- 
- ❑ La dichiarazione di una variabile puntatore richiede il riferimento a un tipo base (quello del dato puntato)

Variabile pstud di tipo
"puntatore a struct studente"

```
int *px;  
char *p0, *p1;  
struct studente *pstud;  
FILE *fp;
```

- 
- ❑ La dichiarazione di una variabile puntatore richiede il riferimento a un tipo base (quello del dato puntato)

Variabile fp di tipo
"puntatore a FILE"

```
int *px;  
char *p0, *p1;  
struct studente *pstud;  
FILE *fp;
```



La dichiarazione

```
int *px;
```

può essere letta in due modi:

- a) `*px` (dato puntato da `px`) sarà (!) di tipo intero.
NOTA: la variabile `px`, al momento della definizione, NON contiene ancora un dato (un puntatore). NON esiste ancora un dato puntato, ma ci sarà dopo la prima assegnazione!
- b) `int *` (tipo puntatore a intero) è il tipo della variabile `px`

- 
- La dichiarazione di un puntatore può esser fatta in due modi (con diverso uso degli spazi):

a) `<tipo base> *<identificatore>;`
l'asterisco viene posto accanto
all'identificatore

```
int *px;
```

b) `<tipo base> * <identificatore>;`
spazi tra asterisco e identificatore

```
int * px;
```

oppure

```
int* px;
```




Dichiarazione fattorizzata:

- ❑ la dichiarazione di più variabili puntatore (stesso tipo base) nella stessa istruzione segue la strategia (a):

`<tipo base> *<id_1>, *<id_2>..., *<id_n>;`

- ❑ si scrive una sola volta il tipo base, mentre si premette un asterisco per ogni variabile dichiarata.



Esempio:

puntatori a intero

```
int *px, *py;  
char *s0, *s1, *s2;
```

puntatori a carattere

puntatore a intero

intero

carattere

```
int *p0, p1;  
char *s0, *s1, s2;
```

puntatori a carattere



Dichiarazione con inizializzazione:

- Si può assegnare un valore a una variabile puntatore contestualmente alla dichiarazione

- Esempi:

```
int x=0;  
int *p = &x;  
char *s = NULL;
```



Dichiarazione con inizializzazione:

- Si può assegnare un valore a una variabile puntatore contestualmente alla dichiarazione

- Esempi:

```
int x=0;  
int *p = &x;  
char *s = NULL;
```

oppure (dichiarazioni equivalenti):

```
int x, *p = &x;  
char *s = NULL;
```



La costante NULL

- ❑ Il valore effettivamente assegnato ad una variabile puntatore è un indirizzo in memoria
- ❑ Esiste una costante utilizzabile come “puntatore nullo” (lo “zero” dei tipi puntatori). Tale costante corrisponde al valore intero 0
- ❑ La costante simbolica **NULL** (definita in `<stdio.h>`) può essere utilizzata per rappresentare tale costante



Il tipo `void *`

- Un puntatore generico può essere definito in C facendo riferimento al tipo **`void *`**
- Un puntatore generico (`void *`) può essere convertito (e assegnato) in modo legale da/a un puntatore di altro tipo (es. `int *`)

```
int *px;  
char *s0;  
void *generic;  
...  
generic = px;  
...  
s0 = generic;
```



Assegnazione

- Due tipologie:
 - puntatore come dato: si assegna a una variabile puntatore il risultato di un'espressione che calcola un puntatore (del tipo corretto)
 - puntatore come riferimento: si assegna al dato (variabile) puntato (da un puntatore) un valore compatibile con il tipo di dato



Puntatore come dato: esempi

```
p = &x;  
s = p;  
pname = &(stud.nome);  
p_i = &dati[i];
```




Puntatore come dato: esempi

```
p = &x;  
s = p;  
pnome = &(stud.nome);  
p_i = &dati[i];
```



Puntatore a variabile x



Puntatore come dato: esempi

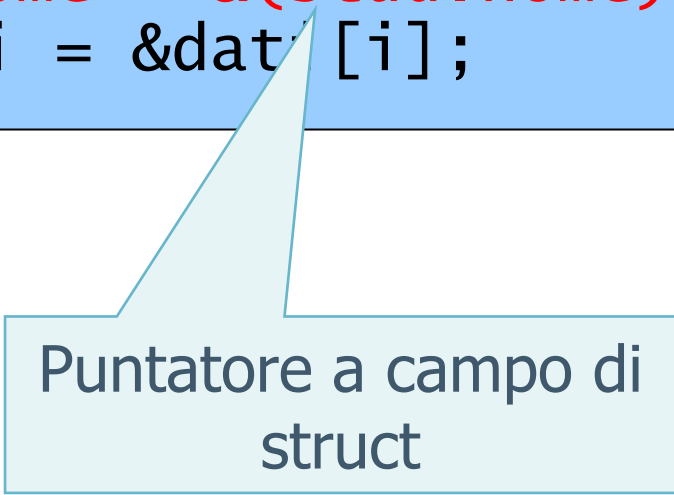
```
p = &x;  
s = p;  
pnome = &(stud.nome);  
p_i = dati[i];
```

Assegnazione tra puntatori



Puntatore come dato: esempi

```
p = &x;  
s = p;  
pnome = &(stud.nome);  
p_i = &data[i];
```



Puntatore a campo di
struct



Puntatore come dato: esempi

```
p = &x;  
s = p;  
pname = &(stud.nome);  
p_i = &dati[i];
```

Puntatore a casella di
vettore



Puntatore come riferimento: esempi

```
*p = 3*(x+2);  
*s = *p;  
*p_i = *p_i+1;
```



Puntatore come riferimento: esempi

```
*p = 3*(x+2);  
*s = *p;  
*p_i = *p - 1;
```

Assegna espressione intera
a dato puntato da p



Puntatore come riferimento: esempi

```
*p = 3*(x+2);  
*s = *p;  
*p_i = *p_i+1;
```

Copia variabile puntata da
p in variabile puntata da s



Puntatore come riferimento: esempi

```
*p = 3*(x+2);  
*s = *p;  
*p_i = *p_i+1;
```

Incrementa variabile
puntata da p_i



Operatori relazionali == e !=

- Un confronto tra due puntatori ritorna valore vero se i due puntatori fanno riferimento allo stesso dato (stesso indirizzo in memoria)
 - `p1==p2`
- Un confronto tra dati puntati ritorna valore vero se (pur con puntatori a dati in locazioni diverse di memoria) i contenuti delle variabili puntate sono uguali
 - `*p1==*p2`

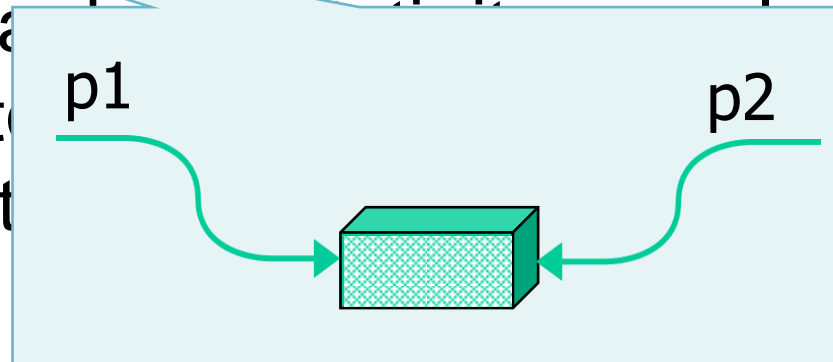
Operatori relazionali == e !=

- Un confronto tra due puntatori ritorna valore vero se i due puntatori fanno riferimento allo stesso dato (stesso indirizzo in memoria)

- `p1==p2`

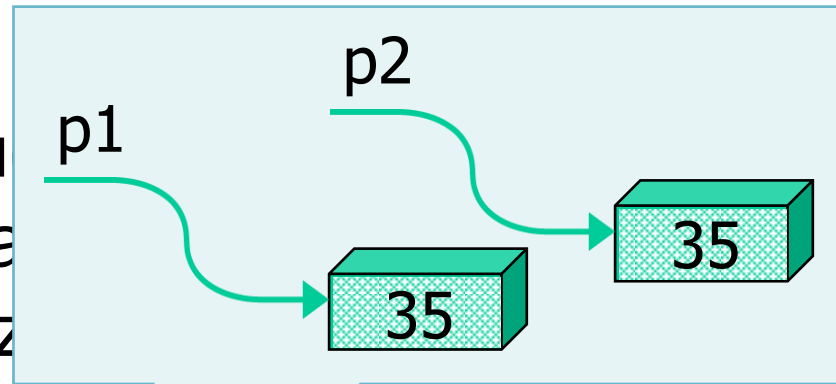
- Un confronto tra due puntatori ritorna valore vero se i due puntatori fanno riferimento allo stesso dato (pur con puntatori diversi) (per esempio, se i due puntatori sono uguali)

- `*p1==*p2`



Operatori relazionali == e !=

- Un confronto tra due puntatori `p1` e `p2` fa ritorno valore vero se i due puntatori fanno riferimento allo stesso dato (stesso indirizzo di memoria)
 - `p1==p2`
- Un confronto tra dati puntati ritorna valore vero se (pur con puntatori a dati in locazioni diverse di memoria) i contenuti delle variabili puntate sono uguali
 - `*p1==*p2`





Aritmetica dei puntatori

- Una variabile puntatore contiene un indirizzo
- l'indirizzo è un intero
- sugli interi sono definite
 - somma e sottrazione + -
 - incremento e decremento di 1 ++ --

Data l'istruzione `p+i` ; o `p++` ; l'effettivo incremento non è `i` o `1`, bensì:

- per `p+i` `i*(sizeof(*p))`
- per `p++` `sizeof(*p)`

`i` e `1` non rappresentano indirizzi contigui, bensì dati del tipo puntato.



Esempio:

```
int a[3]={1,9,2}, *p_a=&a[0];  
char b[5]={'a','e','i','o','u'}, *p_b=&b[0];
```



Esempio:

```
int a[3]={1,9,2}, *p_a=&a[0];  
char b[5]='a','e','i','o','u', *p_b=&b[0];
```

Dichiarazione e
inizializzazione di un
vettore di interi e di uno di
caratteri



Esempio:

```
int a[3]={1,9,2}, *p_a=&a[0];  
char b[5]='a','e','i','o','u', *p_b=&b[0];
```

Dichiarazione e
inizializzazione di 2
puntatori alla prima cella

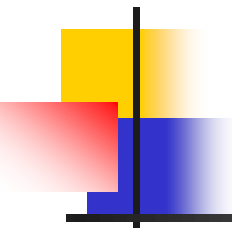


Le istruzioni:

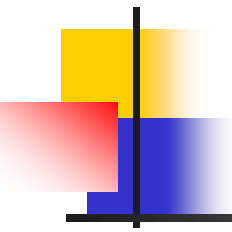
```
printf("a[0]=*p_a=%d,p_a=%p\n",a[0],p_a);  
printf("a[1]=*(p_a+1)=%d,p_a+1=%p\n",a[1],p_a+1);  
printf("b[0]=*p_b  =%c,p_b=%p\n",b[0],p_b);  
printf("b[3]=*(p_b+3)=%c,p_b+3=%p\n",b[3],p_b+3);
```

visualizzeranno:

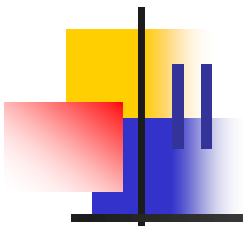
```
a[0]=*p_a=1,p_a=0028FEF8  
a[1]=*(p_a+1)=9,p_a+1=0028FEFC  
b[0]=*p_b=a,p_b=0028FEF3  
b[3]=*(p_b+3)=o,p_b+3=0028FEF6
```


- 
- Incrementare (decrementare) di 1 un puntatore equivale a calcolare il puntatore al dato successivo (precedente) in memoria (supposto contiguo) dello stesso tipo
 - Esempio:

```
int x[100], *p = &x[50], *q, *r;  
...  
q = p+1; /* equivale a q=&x[51] */  
r = p-1; /* equivale a r=&x[49] */  
q++;    /* ora q punta a x[52] */
```

- 
- Sommare (sottrarre) un valore intero i a un puntatore corrisponde a incrementare (decrementare) i volte di 1 il puntatore
 - Esempio:

```
int x[100], *p = &x[50], *q, *r;  
...  
q = p+10; /* equivale a q=&x[60] */  
r = p-10; /* equivale a r=&x[40] */  
r -= 5;   /* ora r punta a x[35] */
```



Il passaggio dei parametri

- Il linguaggio C prevede unicamente passaggio di parametri a funzioni per valore (“by value”)
 - Il valore del parametro **attuale**, calcolato alla chiamata della funzione, viene copiato nel parametro **formale**
- Non è previsto passaggio per riferimento (“by reference”), ma lo si realizza, in pratica, mediante
 - Passaggio per valore di puntatore a dato



Esempio: swap di 2 interi (**ERRATO!**)

```
void swapInt (int x, int y) {  
    int tmp =x;  
    x=y; y=tmp;  
}  
...  
void main (void) {  
    int a, b;  
    ...  
    swapInt(a,b);  
    ...  
}
```

Lo scambio ha
effetto
solo nella
funzione, non
nel main



Esempio: swap di 2 interi (**CORRETTO!**)

```
void swapInt (int *px, int *py) {  
    int tmp = *px;  
    *px=*py; *py=tmp;  
}  
...  
void main (void) {  
    int a, b;  
    ...  
    swapInt(&a,&b);  
    ...  
}
```