Tecniche della Programmazione, lez.13

- Previously on TdP
  - richiami …
  - … che facciamo nella seconda parte
- puntatori
- array e puntatori …
- Richiamo su RDA (**perche' servono molto da qui in poi**)
- Varie applicazioni dell'uso di puntatori
  - Scansione e ricerca in array
  - Riuso di una funzione su array …
- Uso di puntatori per gestire parametri di output

Programma (2/3)

1) Architettura di base del Calcolatore (Memoria, CPU, IO, …)
Funzionamento del Calcolatore,
- da programma in linguaggio ad alto livello (… il C)
→ Compilazione → linking → …
a programma in linguaggio macchina
→ caricamento dell'eseguibile in memoria →
→ esecuzione

2) Processo di sviluppo del software

Scopo: gestire (far gestire dal calcolatore …) le
informazioni che caratterizzano il problema
in modo da produrre
informazioni che caratterizzano la soluzione del problema

dato = quantita` associata ad una informazione

struttura dati = modo formale di definire come sono rappresentati i dati (cioe` le informazioni) dentro al programma (cioe` dentro al calcolatore)

Programma (3/3)

3) Tecniche di programmazione e algoritmi interessanti

3.1) "toolbox":
- strutture di controllo (if, for, while …)
- strutture dati di base (array, file …)
- conoscenze da riusare per risolvere nuovi problemi e per apprendere altre tecniche ed altri algoritmi

3.2) altre Tecniche e Metodologie
- **ricorsione (T)**
- **TEST (M)**
- **Ancora Sviluppo Programma (M)**
- **Gestione strutture dati "non banali" (M)(T)**
  - **puntatori**
  - **struct**

Strutture dati? Modo per rappresentare nella memoria del calcolatore gli oggetti del mondo reale coinvolti nel problema

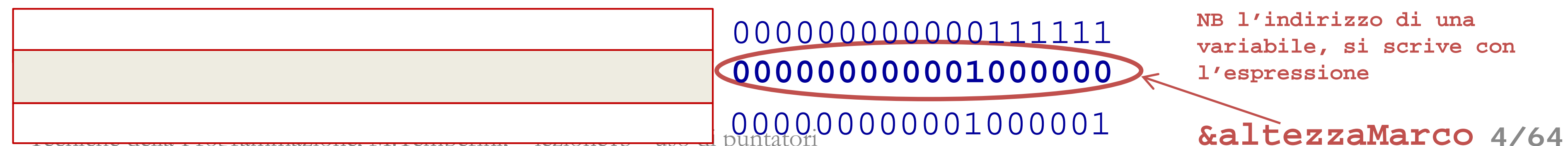
Abbiamo visto	Strutture dati STATICHE (VARIABILE, ARRAY statico)
vedremo	strutture dati DINAMICHE (ARRAY dinamico, … … …)

Concetto di VARIABILE

Una variabile, in programma, e` contemporaneamente

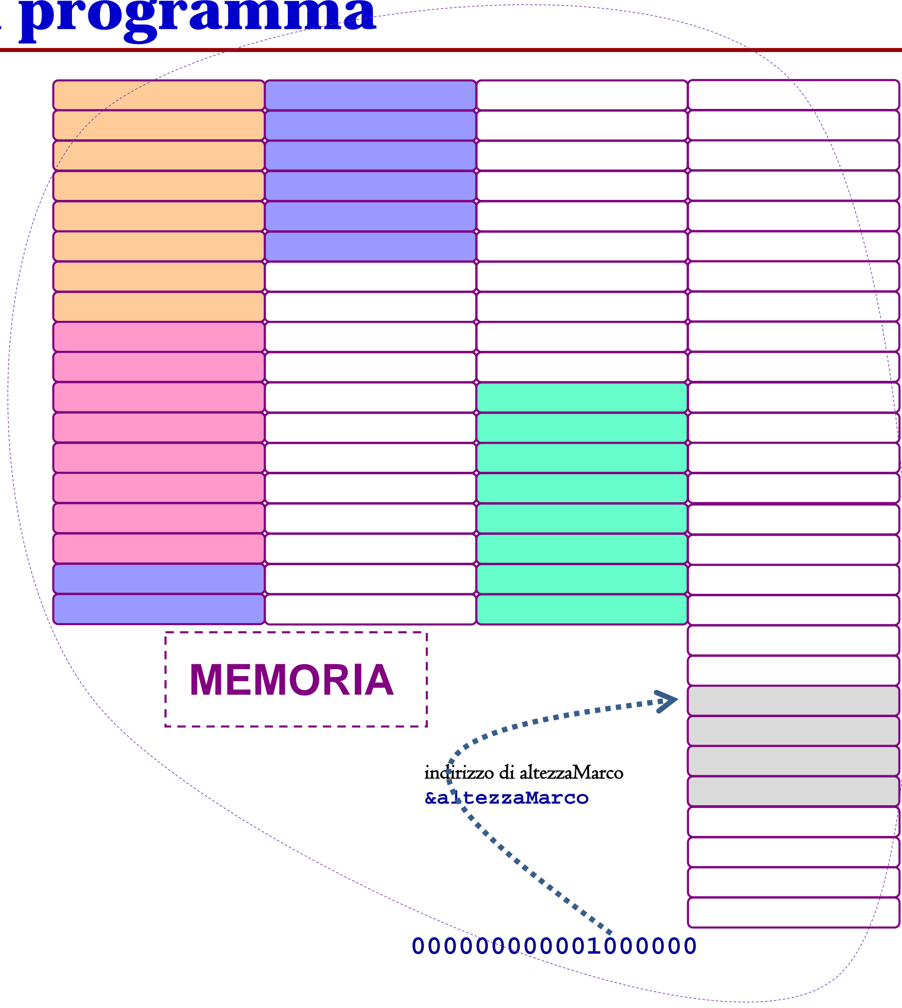
- Un **IDENTIFICATORE** cioe` il nome della variabile, usato nel programma per … usarla)
- Una **LOCAZIONE** di memoria cioe` l'area della RAM, riservata per quella variabile, in cui si memorizzano / accedono i valori associati alla variabile (i valori contenuti nella variabile). Questa e` contraddistinta da un INDIRIZZO!
- Un **VALORE** il valore contenuto nella locazione associata alla variabile

```
int altezzaMarco;              /* dichiarazione di una variabile chiamata
altezzaMarco; al momento opportuno, verra` riservata in memoria una LOCAZIONE, capace
di contenere un intero rappresentato in forma binaria (complemento a 2); questa
locazione avra` un certo INDIRIZZO. Quando si vuole memorizzare il valore 187 nella
variabile (assegnazione), si memorizza 187 nella locazione. Quando si vuole accedere
il valore contenuto, ad esempio per stamparlo in OUTPUT, si accede alla locazione. */
```



PROGRAMMAZIO

```
#include<stdio
int main() {
  double b, h;
  double area;
  double prod;
 scanf("%lf %lf", &b, &h);
 prod = b*h;
 area = prod/2;
 printf("il valore dell'area di un
        triangolo avente base = %g e
        altezza = %g e` %g\n", b, h,
        area);
return 0;
}
```



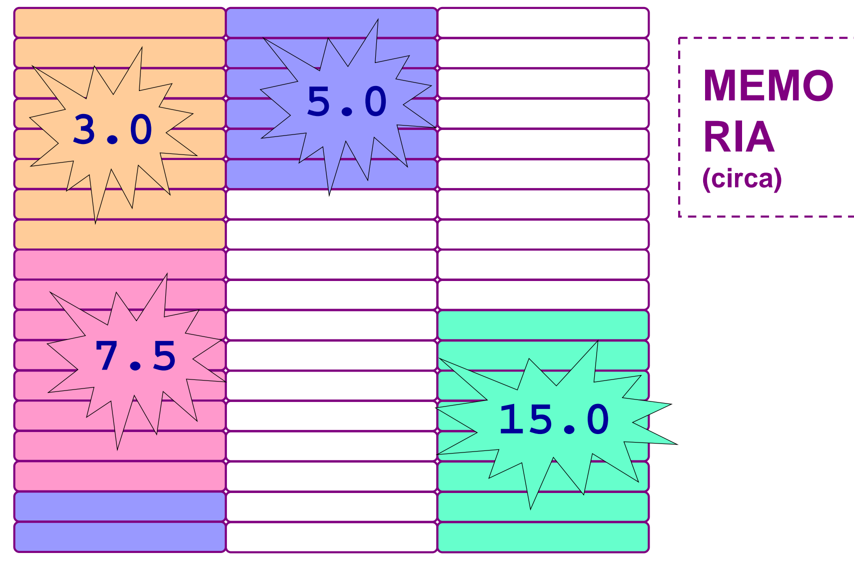
NB altezza Marco qui e` una variabile definita in un altro programma. Anche lei ha la sua locazione di memoria.

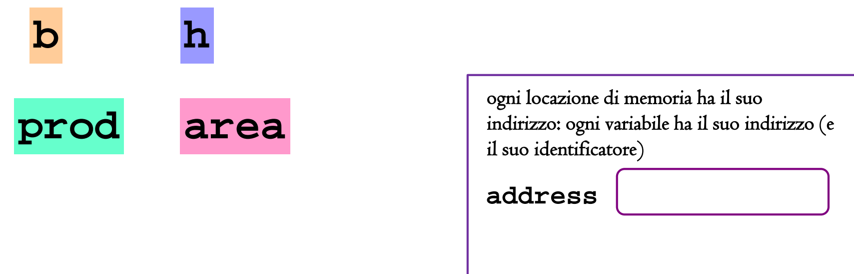
Il programma qui sopra ovviamente non ne sa nulla. Quando ha richiesto l'allocazione delle sue variabili b, h, etc...) lo spazio per tali variabili e` stato riservato (dal sistema di programmazione, e in ultimo dal sistema operativo) in modo da non toccare la memoria gia` riservata ad altezzaMarco.

altezzaMarco

PROGRAMMAZIONE … =

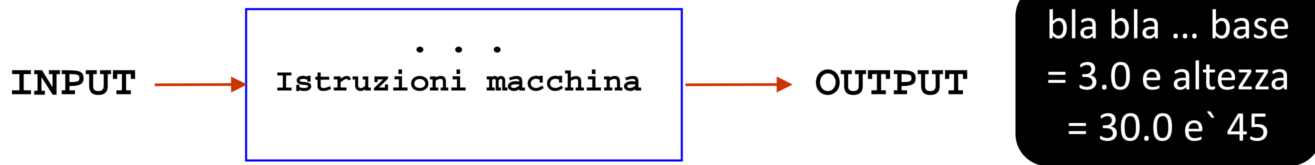
```
#include<stdio.h>
int main() {
  double b, h;
  double area;
  double prod;
 scanf("%lf %lf", &b, &h);
 prod = b*h;
 area = prod/2;
 printf("il valore dell'area di un
        triangolo avente base = %g e
        altezza = %g e` %g\n", b, h,
        area);
return 0;
}
```





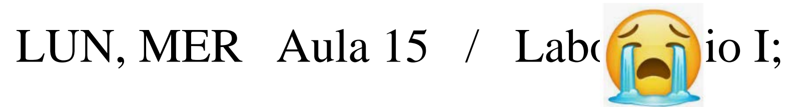
"variabile"

identificatore, indirizzo, contenuto



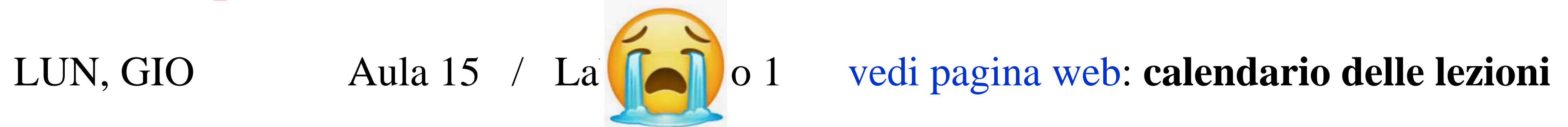
Organizzazione (1/2)

Prima parte: nel primo semestre



Google Classroom

Seconda parte: nel secondo semestre



Come seguire una lezione

- ...

Organizzazione (1/2)

Come seguire una lezione

- esserci il piu` possibile
- prendere appunti
  - su carta, tablet, quaderno, con la penna e la carta, su fogli A4 non rilegati, con la penna, sulle slide disponibili prima della lezione
    - segnando il numero della slide e mettendo osservazioni
    - ah, l'indirizzo indirizza (ah ah ...) un pezzetto di memoria
    - ... ah, una variabile int corrisponde ad un pezzetto di memora di 32 bit ...
    - ah, il prof chama il pezzetto di memoria "locazione" che boomer ...
    - ah, forse potrei fare uno schizzo del disegno che c'e` sulle slide, cosi` forse collego meglio le cose a casa; anche se non sono un boomer io ...
- a casa, ripetere la lezione e sperimentare il codice presentato
  - usando le slide come guida ... una dopo l'altra
  - consultando i complementi didattici quando ci sono
- partecipare alle Esercitazioni Guidate
  - provare qualche esercizio in anticipo
  - farsi aiutare
- NON rimandare a dopo ...

Organizzazione (2/2)

Prova Intermedia = … vedi descrizione e valore sulla pag. web

- l'esame intermedio e` relativo ad una parte del programma
- ci sono date per esami intermedi anche successivamente a febbraio, **MA** e` meglio seguire le lezioni e fare subito l'esame intermedio …

Durante la seconda parte

- 3 HOMEWORK (esercizi) da sottomettere attraverso classroom (vedi pagina web del corso TdP e classroom).
- si tratta di una testimonianza di aver fatto pratica, ed anche di un modo per capire se siete pronti per l'esame
- fateli personalmente; se avete problemi chiedete aiuto ma alla fine fateli personalmente;
- fateli per tempo … non riducetevi alle ultime due o tre settimane prima dell'esame

ESAME FINALE (esempi sulla pagina web)

- si accede *"dopo aver passato la Prova Intermedia"* **AND** *"aver avuto i 3 Homework (classroom) accettati"*

Variabile "di tipo T"

una variabile di tipo T

corrisponde

nel programma

ad un IDENTIFICATORE

nella memoria

ad una LOCAZIONE

avente un certo INDIRIZZO

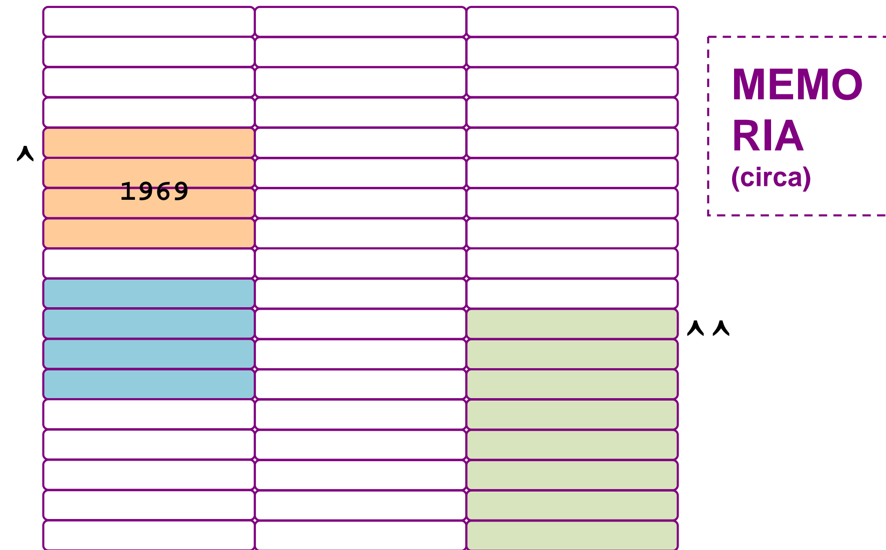
la LOCAZIONE e` capace di contenere un VALORE di tipo T

```
int num = 1969;
```

variabile di tipo int; locazione di 4 byte (per contenere un valore intero); il suo indirizzo e` ^

```
double deltaQ;
```

variabile di tipo double; locazione di 8 byte (per contenere un valore double); il suo indirizzo e` ^^



num

deltaQ

ogni locazione di memoria ha il suo indirizzo: ogni variabile ha il suo indirizzo (e il suo identificatore)

address

Variabile "Puntatore a tipo T"

una variabile puntatore

("puntatore a locazioni di tipo T")

e` una variabile …

corrisponde,

- nel programma

ad un IDENTIFICATORE

- nella memoria

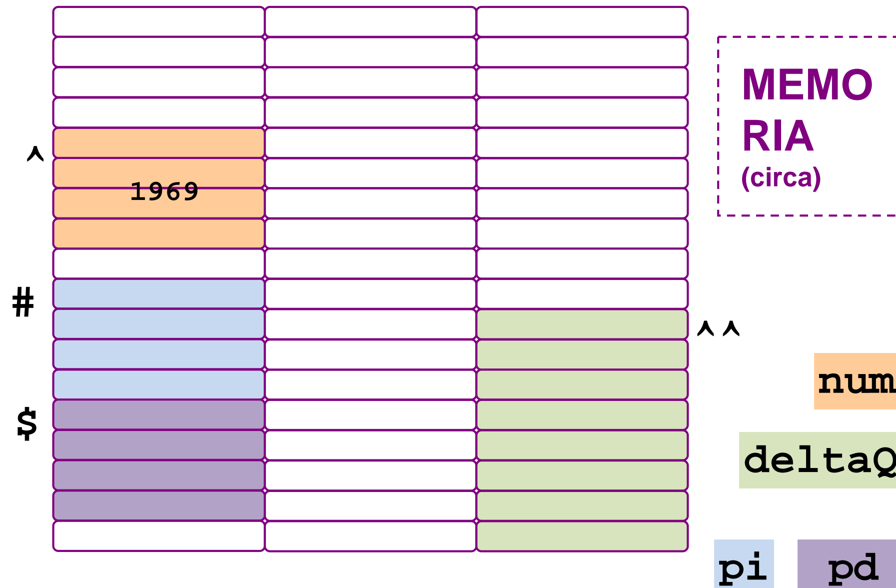
ad una LOCAZIONE avente un certo INDIRIZZO

IL SUO VALORE PERO` e` l'indirizzo di una variabile di tipo T

cioe` un valore di tipo

"indirizzo di locazione di memoria capace di contenere valori di tipo T"

```
int *pi;

double *pd;
```



Variabile di tipo `int *` (*puntatore a intero*)

e` una locazione capace di contenere indirizzi di locazioni intere

ad esempio, l'indirizzo di num: `&num`

anche `pi` e` una variabile …

e il suo indirizzo (`&pi`) e` #

Variabile "Puntatore a tipo T"

una variabile puntatore

("puntatore a locazioni di tipo T")

e` una variabile …

corrisponde,

- nel programma

  ad un IDENTIFICATORE

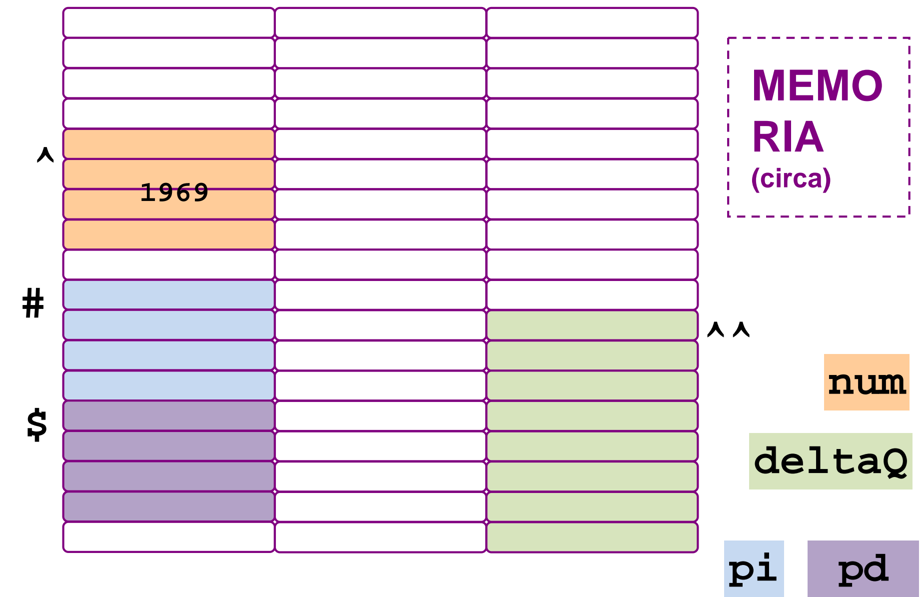
- nella memoria

  ad una LOCAZIONE avente un certo INDIRIZZO

IL SUO VALORE PERO` e` l'indirizzo di una variabile di tipo T

cioe` un valore di tipo

"indirizzo di locazione di memoria capace di contenere valori di tipo T"



Variabile di tipo `int *` (*puntatore a intero*)

e` una locazione capace di contenere indirizzi di locazioni intere, ad esempio, l'indirizzo di num: `&num`

anche `p` e` una variabile ... il suo indirizzo `(&pi)` e` #

```
int *pi;
double *pd;
```

Variabile di tipo `double *` (*puntatore a double*)

locazione capace di contenere indirizzi di locazioni di tipo double: es. `&deltaQ`

anche `pd` e` una variabile ... e il suo indirizzo `(&pd)` e` $

che ci facciamo con un puntatore? Dichiarazione/Definizione

DICHIARAZIONE/DEFINIZIONE

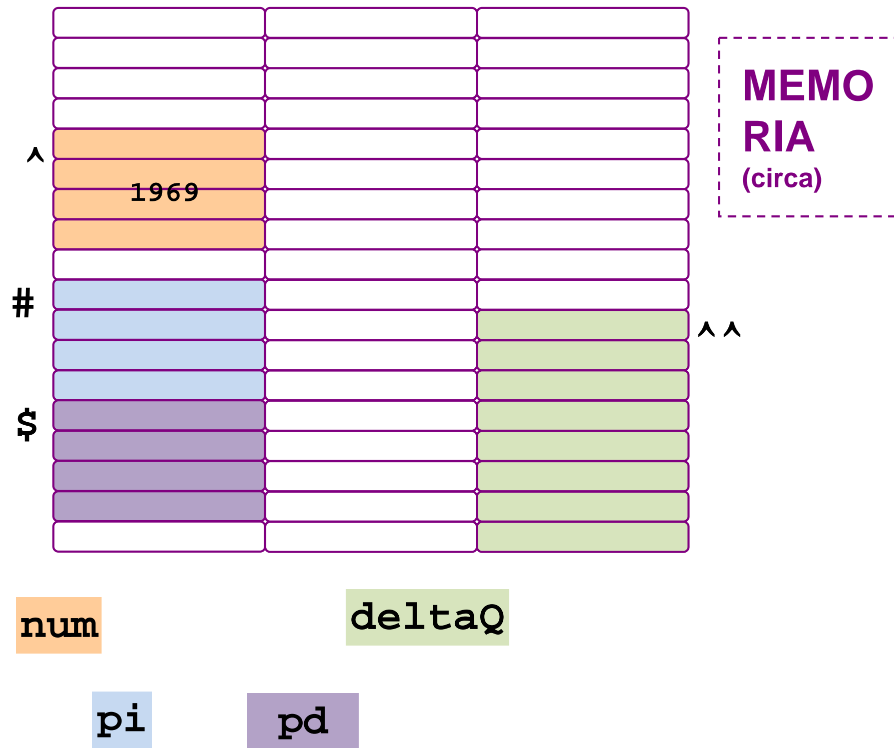
```
int num, *pi;
double *pd, deltaQ;
```

num e` di tipo int

deltaQ e` di tipo double

pi e` di tipo int *

pd e` di tipo double *




```
int *pi1, *pi2;
/* dichiarazione di due puntatori a interi, equivalente a
int *pi1;
int *pi2; */
```

MA ATTENZIONE

```
int * pi1, pi2;  /* dichiara un puntatore pi1 e una var intera pi2 … */
```

che ci facciamo con un puntatore? Assegnazione

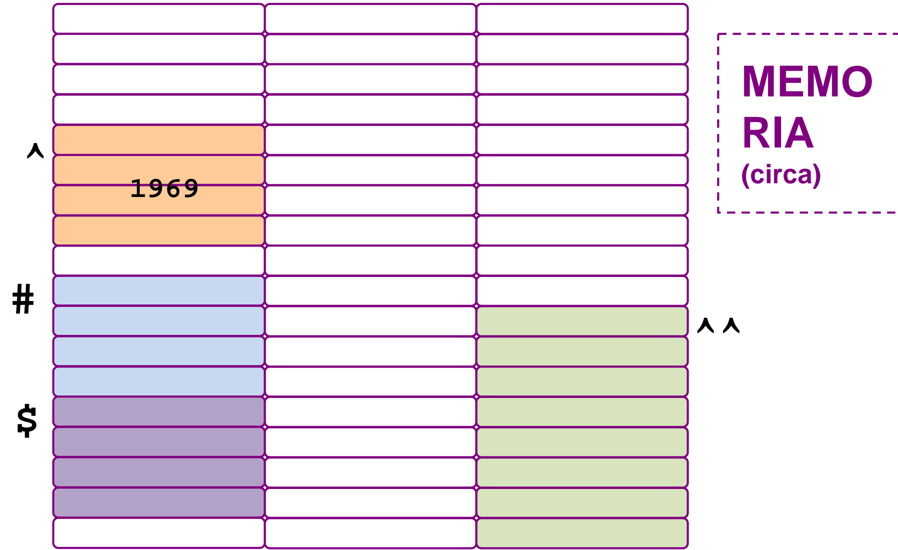
ASSEGNAZIONE

```
int num, *pi;

double *pd, deltaQ
```

istruzione di assegnazione (come per qualsiasi variabile)

```
pi = &num;
```

☺



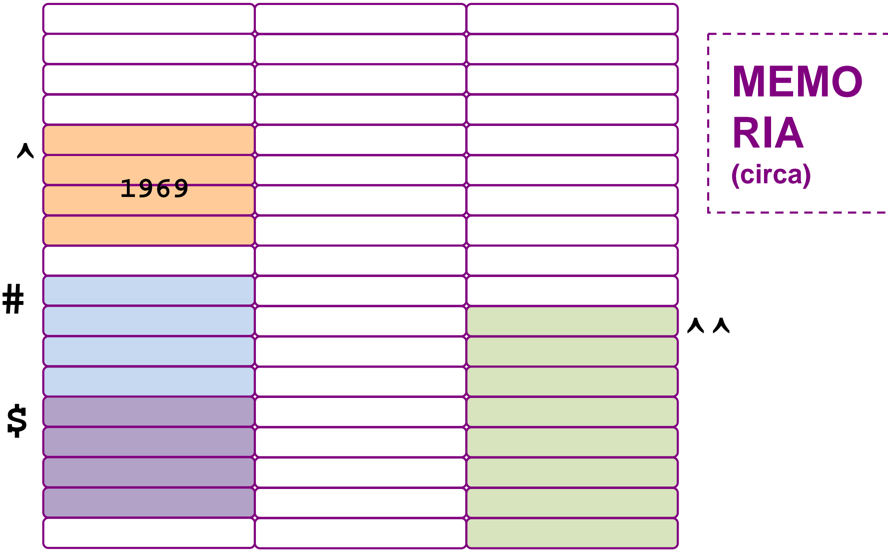
che ci facciamo con un puntatore? Assegnazione

ASSEGNAZIONE

```
int num, *pi;
double *pd, deltaQ
```

istruzione di assegnazione (come per qualsiasi variabile)

```
pi = &num;
```



indirizzo di num ... ☺

indirizzo di pi ... ☺

cosa viene messo in pi? ☺

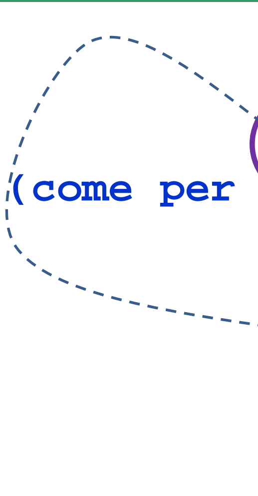
che ci facciamo con un puntatore? Assegnazione

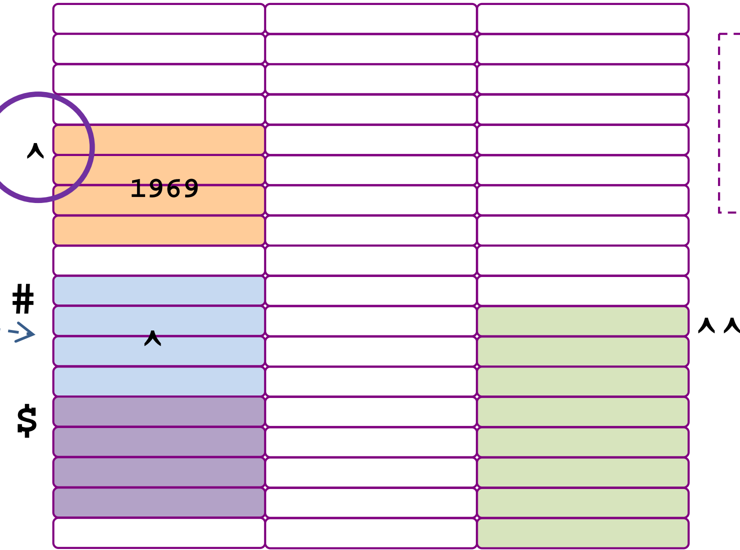
ASSEGNAZIONE

```
int num, *pi;
double *pd, deltaQ
```

istruzione di assegnazione (come per qualsiasi variabile)

```
pi = &num;
```





```
indirizzo di num ... ^
indirizzo di pi ... #
cosa viene messo in pi? L'indirizzo di num
```

che ci facciamo con un puntatore? Assegnazione

ASSEGNAZIONE

```
int num, *pi;
double *pd, deltaQ
```

istruzione di assegnazione (come per qualsiasi variabile)

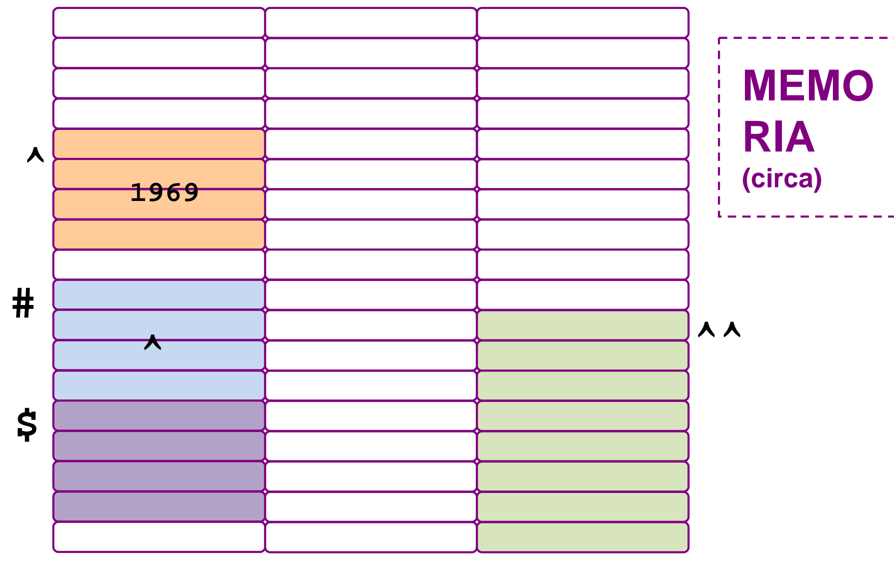
```
pi = &num;
```

si puo` fare "assegnazione in definizione", come per qualsiasi variabile …

```
int *pi = &num;
```

ma in ogni caso i tipi devono corrispondere!

```
pi = &deltaQ;   /* NO!! */
```



che ci facciamo con un puntatore? Assegnazione

ASSEGNAZIONE

```
int num, *pi;
double *pd, deltaQ
```

istruzione di assegnazione (come per qualsiasi variabile)

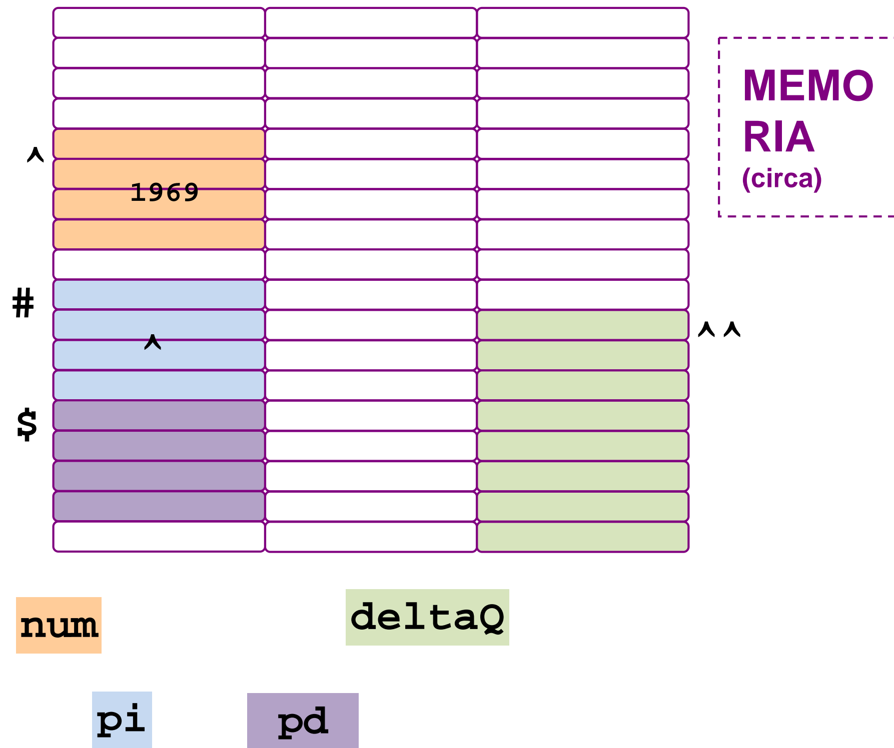
```
pi = &num;
```

si puo` fare "assegnazione in definizione", come per qualsiasi variabile …

```
int *pi = &num;
```

ma in ogni caso i tipi devono corrispondere!

```
pi = &deltaQ;   /* NO!! */
```



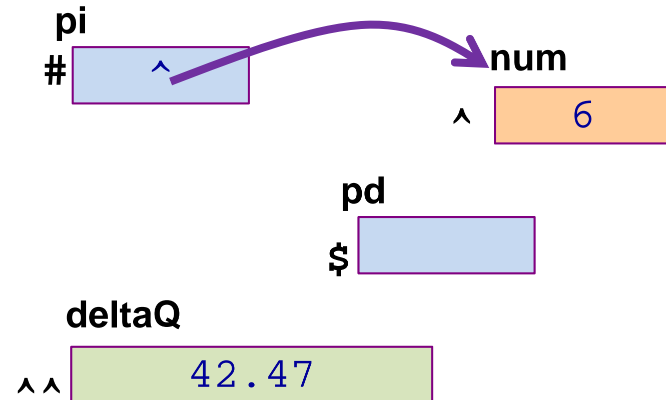
dopo l'assegnazione vista sopra si dice pi PUNTA A num

e da ora ricominciamo a disegnare le variabili in modo piu` semplice

che ci facciamo con un puntatore? Assegnazione

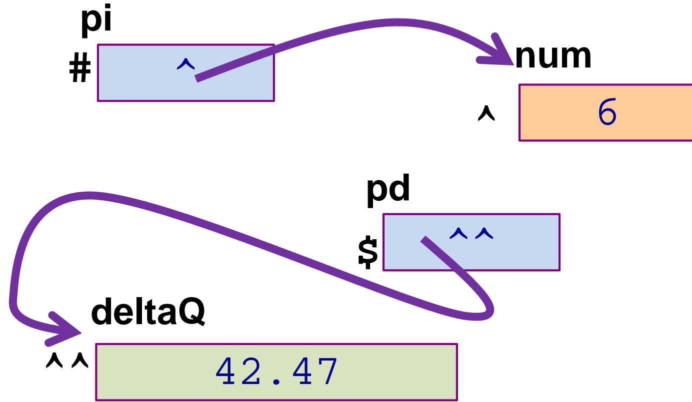
```
int num;

int *pi;

double *pd, deltaQ;


pi = &num;      /*ok o KO? ☺ */

pi = &deltaQ;   /*ok o KO? ☺ */

pd = &deltaQ;   /*ok o KO? ☺ */

pd = &num;      /*ok o KO? ☺ */

pd = pi;        /*ok o KO? ☺ */

pd = &pi;       /*ok o KO? ☺ */
```

per le istruzioni OK completare il disegno con i contenuti delle variabili e le frecce



che ci facciamo con un puntatore? Assegnazione

```
int num;

int *pi;

double *pd, deltaQ;
```



`pi = #` ok

`pi = &deltaQ;` KO

tipo di pi: int *

tipo di &deltaQ: double *

`pd = &deltaQ;` ok

`pd = #` KO tipo di pd: double *

tipo di &num: int *

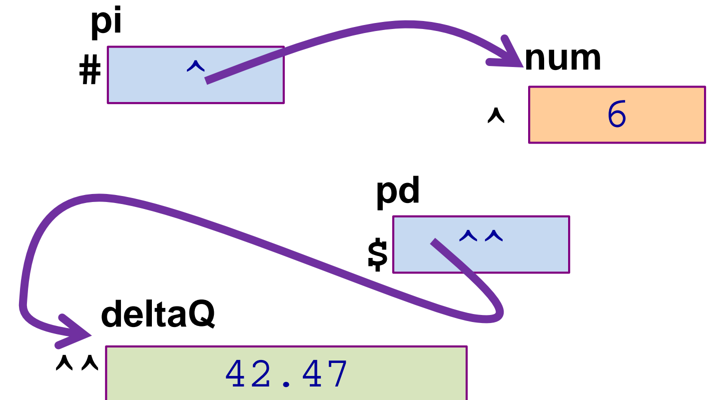
`pd = pi;` KO tipo di pd: double *

tipo di pi: int *

`pd = π` KO *(&pi e` l'indirizzo di una locazione di tipo "indirizzo di intero" ... cioe` puntatore a punttore ad int ... mischiamo mele, fagioli, self-sealing steambolt ... no)*

che ci facciamo con un puntatore? Assegnazione

```
int num;

int *pi;

double *pd, deltaQ;
```



`pi = #` ok

`pi = &deltaQ;` KO

tipo di pi: int *

tipo di &deltaQ: double *

`pd = &deltaQ;` ok

`pd = #` KO — tipo di pd: double *

tipo di &num int *

`pd = pi;` KO — tipo di pd: double *

tipo di pi: int *

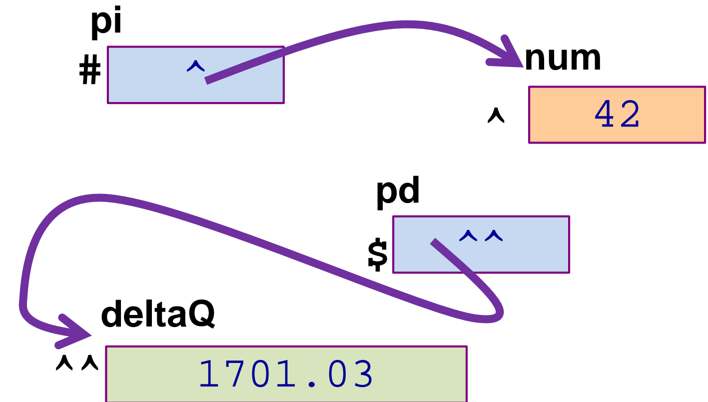
`pd = π` KO — tipo di &pi: int **

vedi commento precedente

che ci facciamo con un puntatore? Lettura dati

MEMORIA (circa)

```
int num;

int *pi;

double *pd, deltaQ;


pi = &num;      ok
```



ora le prossime due istruzioni sono equivalenti

```
scanf("%d", &num);

scanf("%d", pi);
```

funzionano bene per leggere num … il 42 dato in input viene memorizzato nella locazione di indirizzo ^

```
pd = &deltaQ;   ok


scanf("%lf", pd);
```

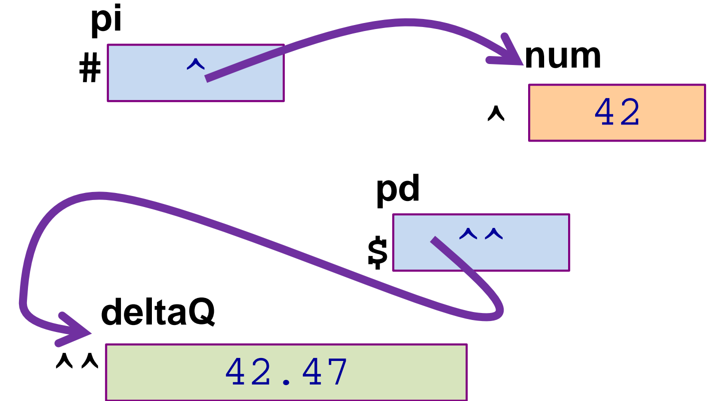
legge il 1701.03 dato in input e lo memorizza nella locazione di indirizzo ^^

che ci facciamo con un puntatore? *Dereferenziazione*

Oltre all'operatore "indirizzo" (&)

c'e` l'operatore di dereferenziazione '*'

che si usa su variabili puntatore



`*pi` e`

il contenuto della locazione

puntata da pi

queste due diverse printf

```
printf("%d", num);
printf("%d", *pi);
```

sono equivalenti:

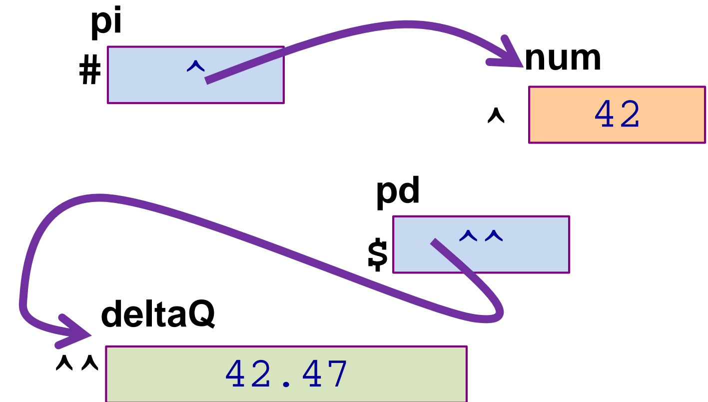
che effetto hanno in output le due printf?

che ci facciamo con un puntatore? *Dereferenziazione*

Oltre all'operatore "indirizzo" (&)

c'e' l'operatore di dereferenziazione '*'

che si usa su variabili puntatore



`*pi` e`

il contenuto della locazione

puntata da pi

queste due diverse printf sono equivalenti:

```
printf("%d", num);
```

/* viene stampato il 6 contenuto in num */

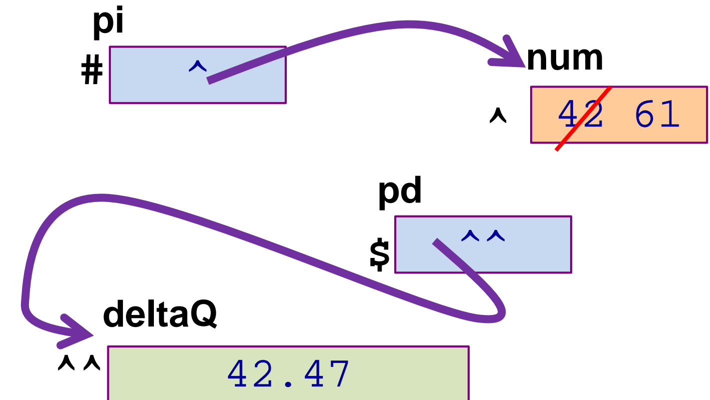
```
printf("%d", *pi);
```

/* viene stampato il 42 contenuto nella locazione di indirizzo ^
cioe` il 42 contenuto nella locazione puntata da pi */

che ci facciamo con un puntatore? *Dereferenziazione*

In effetti

MEMORIA (circa)



se pi punta a num

`*pi` e` un modo alternativo

di scrivere num,

```
num = 61;

*pi = 61;
```

sono assegnazioni equivalenti:

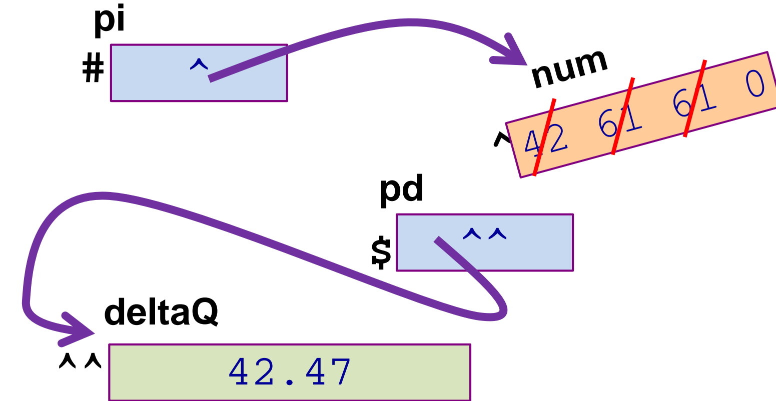
che effetto hanno in memoria le due assegnazioni?

e che succede se si esegue l'istruzione

`*pi = 0;` ?

che ci facciamo con un puntatore? *Dereferenziazione*

In effetti



se pi punta a num

`*pi` e` un modo alternativo

di scrivere num,

```
num = 61;
*pi = 61;
```

la prima memorizza 61 nella locazione associata a num, sovrascrivendo il 42 di prima

la seconda memorizza 61 nella locazione puntata da pi, che e` quella di indirizzo ^
cioe` quella di num (e il vecchio 61 viene sovrascritto ma non ce ne accorgiamo);

`*pi = 0;` memorizza zero in num

esercizio

```
int main() {
 dichiarare variabili per un intero i, un
puntatore a interi pi, un puntatore a double
pd ed un double d
(dichiarare pd e d separatamente; dichiarare
i e pi in un'unica dichiarazione).
```

```
disegnare le variabili definite come fatto
prima …
indicare i loro indirizzi a matita ;)
```

```
disegnare anche le frecce che indicano,
eventualmente, come i puntatori puntano alle
variabili

                    :>
```

MEMORIA (circa)

#

esercizio

```
int main() {
 int i, *pi;
 double d;
 double *pd;


disegnare le variabili definite come fatto prima …
indicare i loro indirizzi a matita ;)


disegnare anche le frecce che indicano,
eventualmente, come i puntatori puntano alle
variabili
```

MEMORIA (circa)

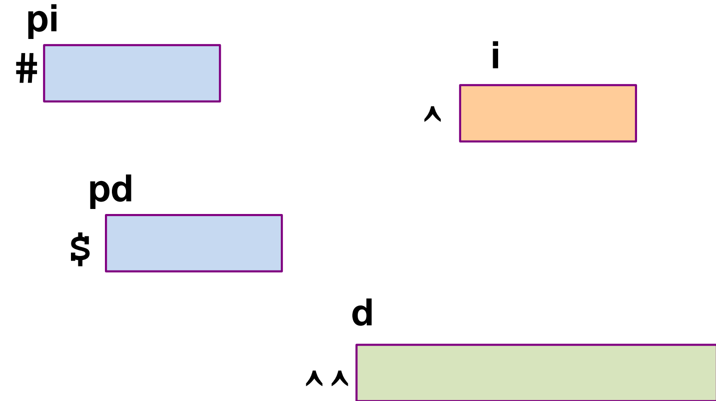
esercizio

```
int main() {
 int i, *pi;
 double d;
 double *pd;
```

```
disegnare anche le frecce che indicano,
eventualmente, come i puntatori puntano alle
variabili
```

```
??? quali frecce? In questo momento le variabili non
hanno valore significativo (non sono inizializzate
ne' sono state assegnate)
```

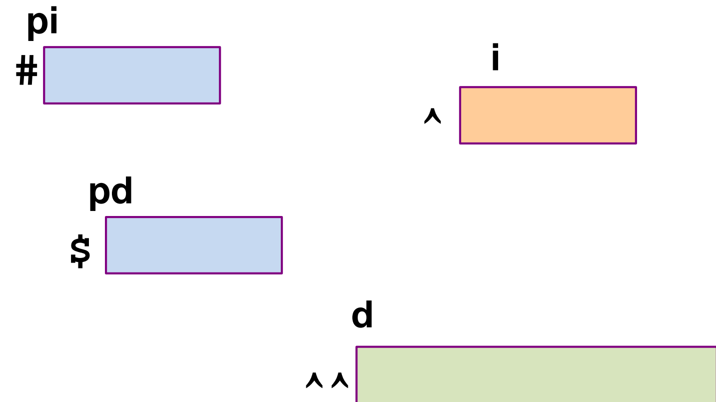
MEMORIA (circa)



esercizio

```
int main() {
 int i, *pi;
 double d;
 double *pd;
mettere un OK vicino alle istruzioni che hanno un
senso; KO a quelle scorrette …
non eseguire le istruzioni per ora

d=7.9;
i=3;
i=&d;
pi=&i;
pd=&d;
scanf("%d", &i);    /* input 7 */
scanf("%d", pi);    /* input 70 */
scanf("%lf", pi);   /* input 5.4 */
printf("%g", *pd);
d=&d;
d=&i;
pd=&i;
*pi=0;
```

continua

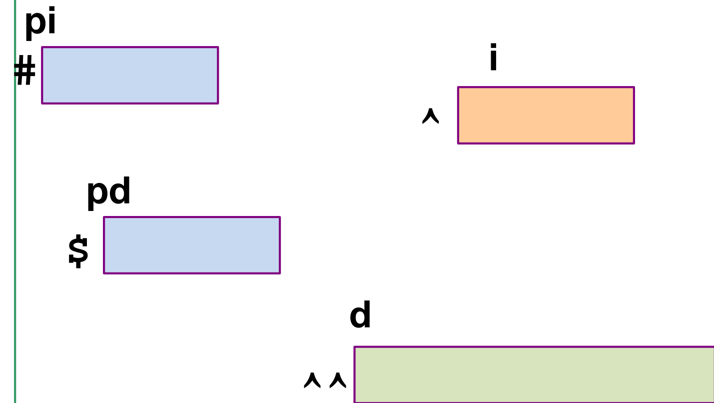



```
*pd=13.1;
printf("%g\n", d);
printf("%d", *pi+6);
return 0;
}
```

esercizio

```
int main() {
 int i, *pi;
 double d;
 double *pd;

d=7.9;     OK
i=3;       OK
i=&d;      KO       indirizzo di double a intero?
pi=&i;     OK
pd=&d;     OK
scanf("%d", &i);    /* input 7 */        OK
scanf("%d", pi);    /* input 70 */       OK
scanf("%lf", pi);   /* input 5.4 */      KO
printf("%g", *pd);  OK
d=&d;      KO    tipo di d == double; tipo di &d == double*
d=&i;      KO
pd=&i;     KO
*pi=0;     OK
*pd=13.1;           OK
printf("%g\n", d);  OK
printf("%d", *pi+6);          OK
return 0;                               }
```

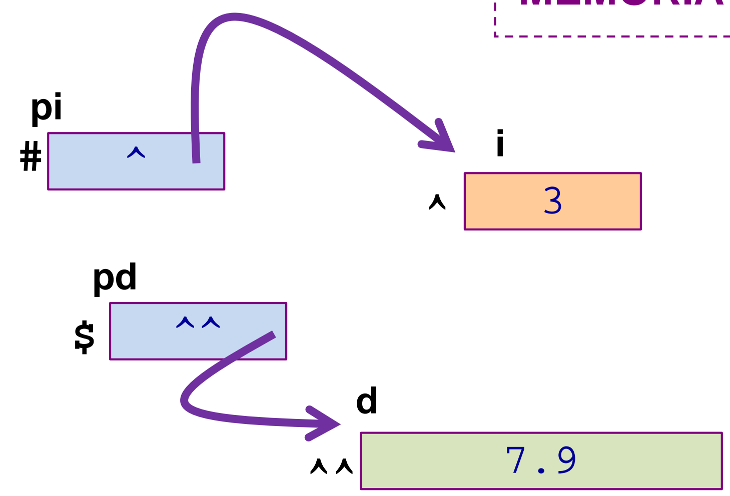
MEMORIA (circa)



ok, adesso eseguire le istruzioni e annotare il disegno con gli effetti delle istruzioni (frecce comprese)

esercizio

```
int main() {
 int i, *pi;
 double d;
 double *pd;
d=7.9;  OK
i=3;    OK
pi=&i;  OK
pd=&d;  OK
scanf("%d", &i);       /* input 7 */   OK
scanf("%d", pi);       /* input 70 */  OK
printf("%g", *pd);     OK
*pi=0;  OK
*pd=13.1;       OK
printf("%g\n", d);     OK
printf("%d", *pi+6);   OK
return 0;                              }
```

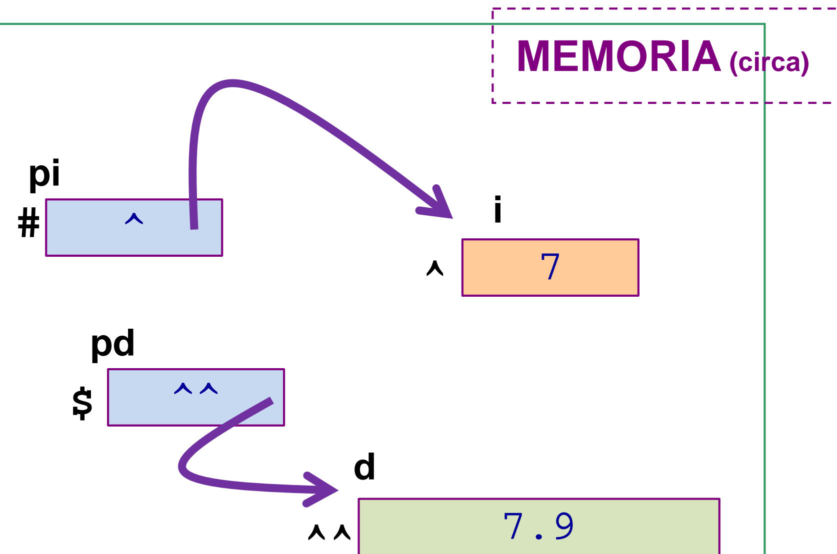


era
"eseguire le istruzioni e annotare il disegno con gli effetti delle istruzioni (frecce comprese)"

pi punta a i ... freccia da pi a i
pd punta a d ...

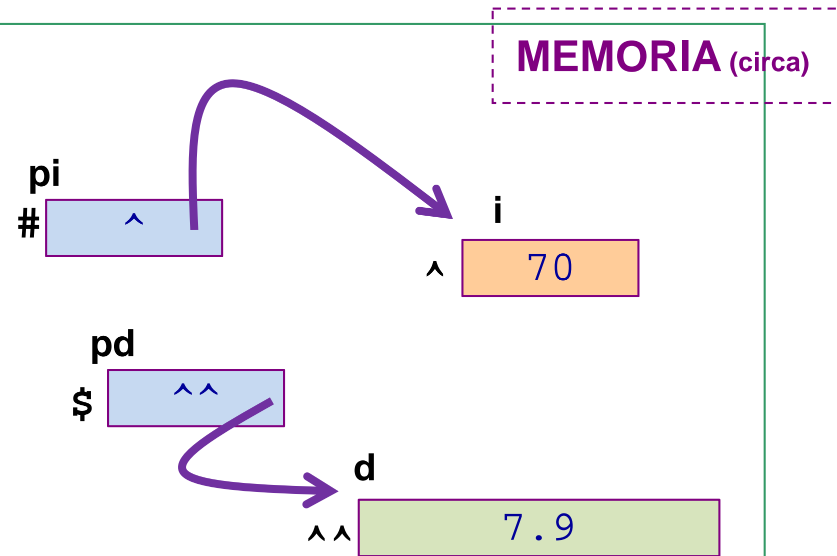
esercizio

```
int main() {
 int i, *pi;
 double d;
 double *pd;
d=7.9;  OK
i=3;    OK
pi=&i;  OK
pd=&d;  OK
scanf("%d", &i);       /* input 7 */   OK
scanf("%d", pi);       /* input 70 */  OK
printf("%g", *pd);     OK
*pi=0;  OK
*pd=13.1;      OK
printf("%g\n", d);     OK
printf("%d", *pi+6);   OK
return 0;                              }
```



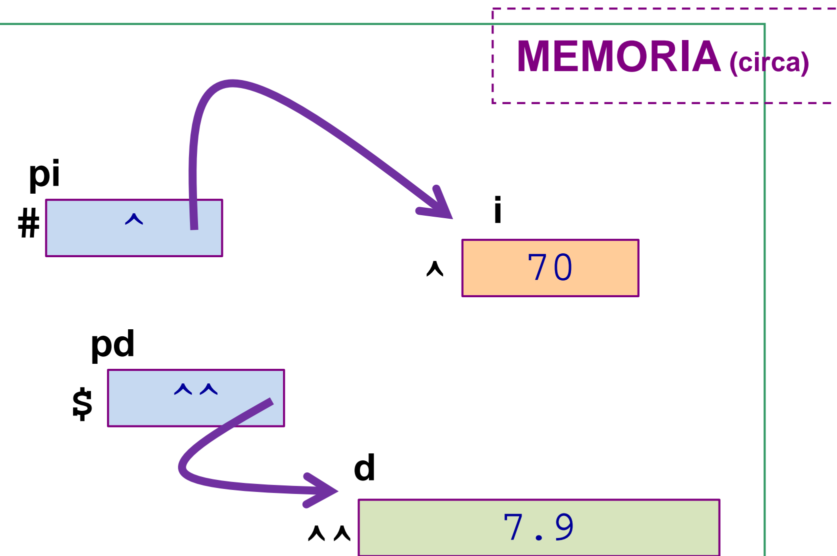
esercizio

```
int main() {
 int i, *pi;
 double d;
 double *pd;
d=7.9;  OK
i=3;    OK
pi=&i;  OK
pd=&d;  OK
scanf("%d", &i);      /* input 7 */   OK
scanf("%d", pi);      /* input 70 */ OK
printf("%g", *pd);    OK
*pi=0;  OK
*pd=13.1;      OK
printf("%g\n", d);    OK
printf("%d", *pi+6);  OK
return 0;                               }
```



esercizio

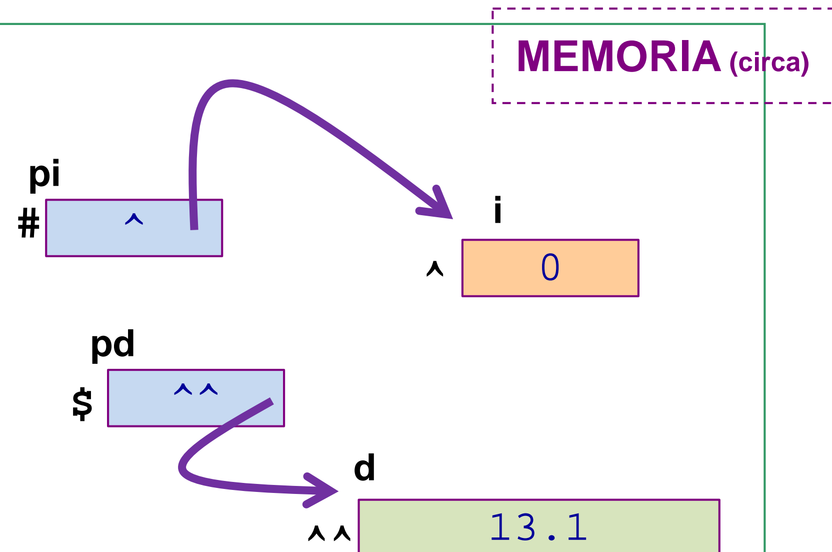
```
int main() {
 int i, *pi;
 double d;
 double *pd;
d=7.9;  OK
i=3;    OK
pi=&i;  OK
pd=&d;  OK
scanf("%d", &i);        /* input 7 */   OK
scanf("%d", pi);        /* input 70 */  OK
printf("%g", *pd);      OK
*pi=0;  OK
*pd=13.1;       OK
printf("%g\n", d);      OK
printf("%d", *pi+6);    OK
return 0;                               }
```



7.9

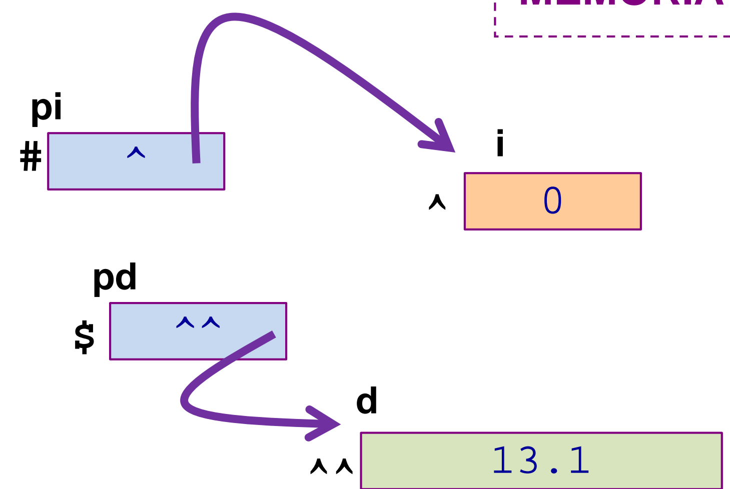
esercizio

```
int main() {
 int i, *pi;
 double d;
 double *pd;
d=7.9;  OK
i=3;    OK
pi=&i;  OK
pd=&d;  OK
scanf("%d", &i);        /* input 7 */   OK
scanf("%d", pi);        /* input 70 */  OK
printf("%g", *pd);      OK
*pi=0;  OK     <-----------
*pd=13.1;       OK    <-----------
printf("%g\n", d);      OK
printf("%d", *pi+6);    OK
return 0;                                }
```



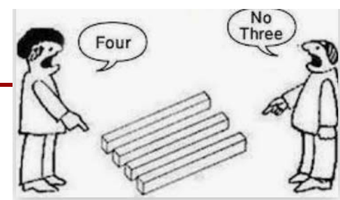
esercizio

```
int main() {
 int i, *pi;
 double d;
 double *pd;
d=7.9;  OK
i=3;    OK
pi=&i;  OK
pd=&d;  OK
scanf("%d", &i);        /* input 7 */   OK
scanf("%d", pi);        /* input 70 */  OK
printf("%g", *pd);      OK
*pi=0;  OK
*pd=13.1;       OK
printf("%g\n", d);      OK <-----------
printf("%d", *pi+6);    OK <-----------
return 0;                                }
```



13.1
6

Aritmetica dei puntatori



L'aritmetica serve a fare i conti …

che succede se sommo 1 ad un puntatore?

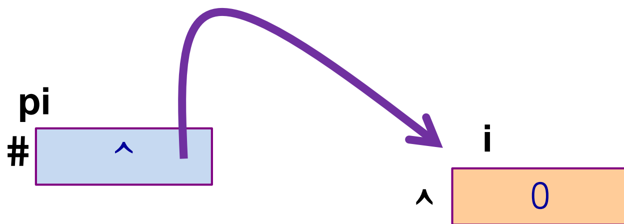
cioe` , se sommo 1 al contenuto di un puntatore? E se sommo 3?

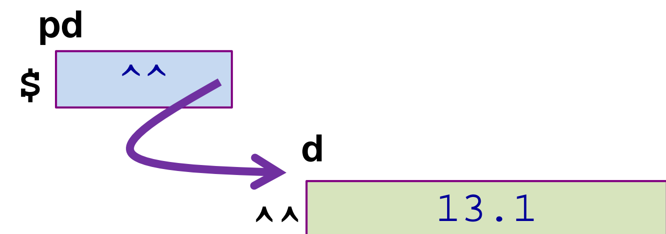
pi
267431

pi+1 e` 267432 ?      pi+3 e` 267434 ?

… non e` detto …

Questi conti si fanno tenendo conto della dimensione della locazione che il puntatore e` capace di puntare.





sizeof(TIPO_T) e` il numero di byte che costituiscono una locazione di tipo TIPO_T

sizeof(int) … 4

sizeof(double) … 8

Aritmetica dei puntatori

in generale

se p e` una variabile puntatore a tipo T

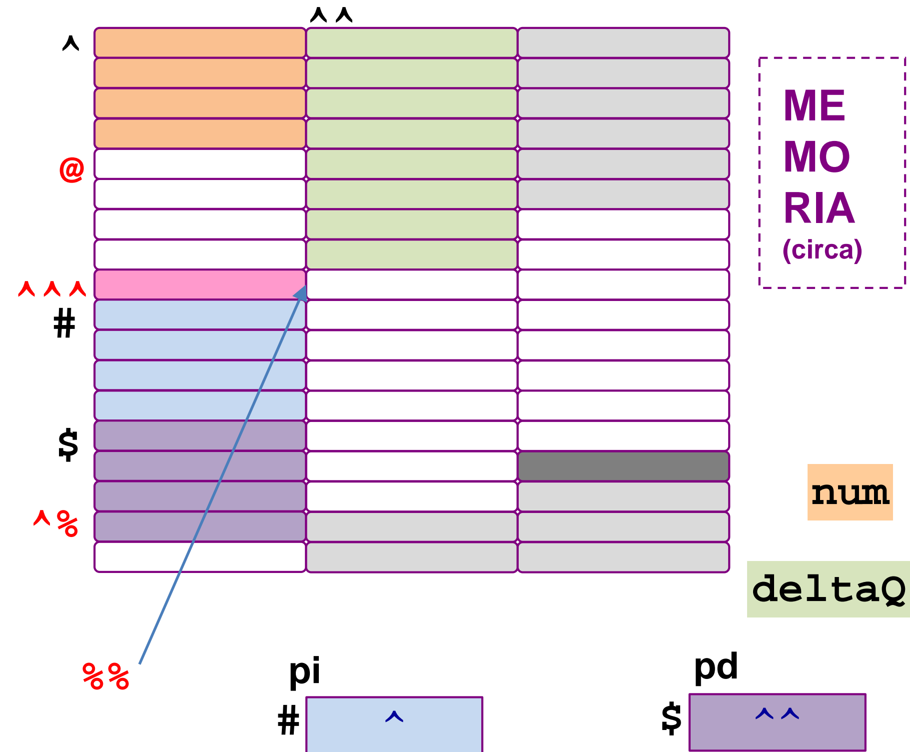
ed n e` un intero

p + n = (valore dell'indirizzo contenuto in p)

+

n * sizeof(T)

```
int num, *pi;
double *pd, deltaQ;
 pi = &num;
 pd = &deltaQ
```



pi + 1 e` @ cioe` ^ + 4byte

pi + 2 e` ^^^ cioe` ^ + 2 volte 4byte … o anche [(pi+1)+1]

pi + 4 e` ^%

pd + 0 e` ^^

pd + 1 e` %% cioe` ^^ + 8byte

pd + 4 e` l'ind. di

Aritmetica dei puntatori ... e array

sull'aritmetica dei puntatori si basa la gestione degli array.

La natura di un array e` in effetti proprio quella di un puntatore ... anche se non propriamente variabile ...

```
int arr[6];
```

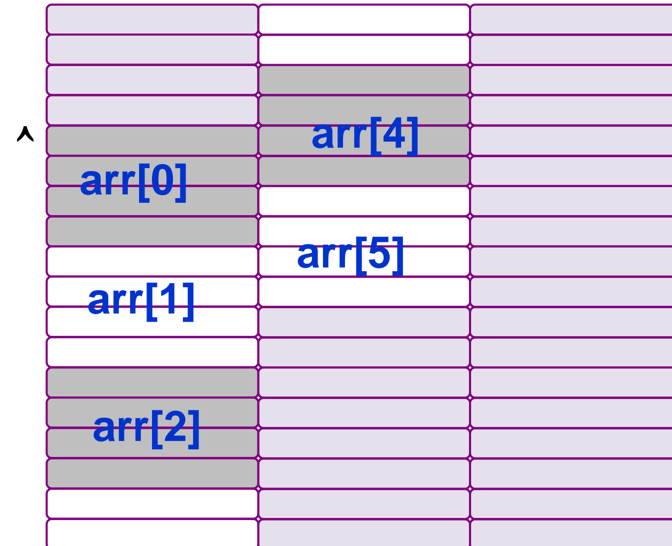
possiamo immaginare `arr` come una variabile,

associata ad una sequenza di interi;

`arr e`` memorizzata in RAM, e contiene

l'indirizzo del 1° elemento della

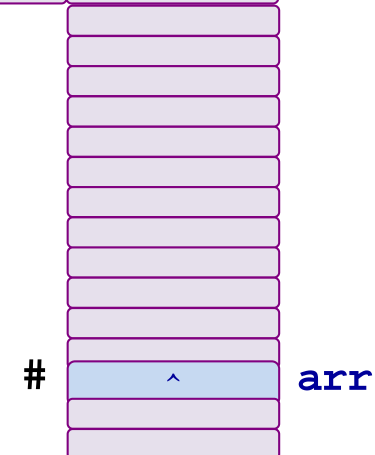
sequenza di 6 interi (cioe` ^)



ME
MO
RIA
(circa)

arr

3259	116	5008	5618	47	42
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]

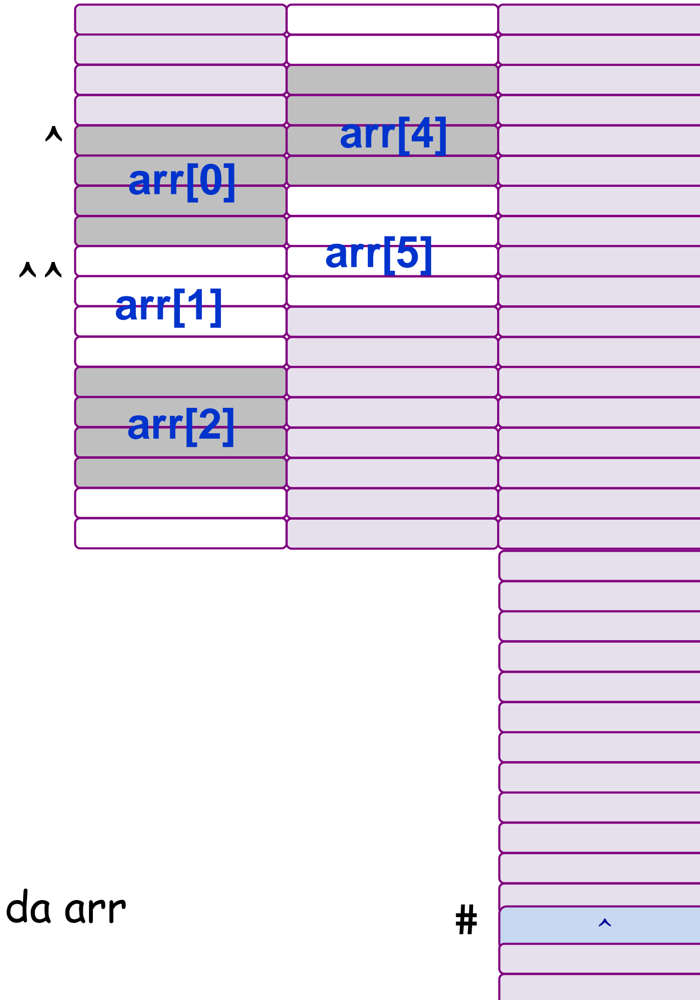


`arr` corrisponde ad un array statico (dichiarato staticamente, cioe` nel programma). La variabile `arr` non cambia mai valore durante il programma e il suo contenuto permette di conoscere sempre gli indirizzi di tutti gli elementi dell'array

Aritmetica dei puntatori ... e array

arr

3259	116	5008	5618	47	42
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]



Osservazione 1

arr e` un indirizzo; l'indirizzo del primo degli elementi della sequenza

`arr == &arr[0]`

Osservazione 2

arr + 1 e` l'indirizzo, ottenuto con l'aritmetica dei puntatori,

^ + 1*sizeof(int) cioe` ^^

cioe` `arr + 1 == &arr[1]`

Osservazione 3

*arr esprime il contenuto della locazione puntata da arr

cioe` `*arr == arr[0]`

Aritmetica dei puntatori ... e array: regole base

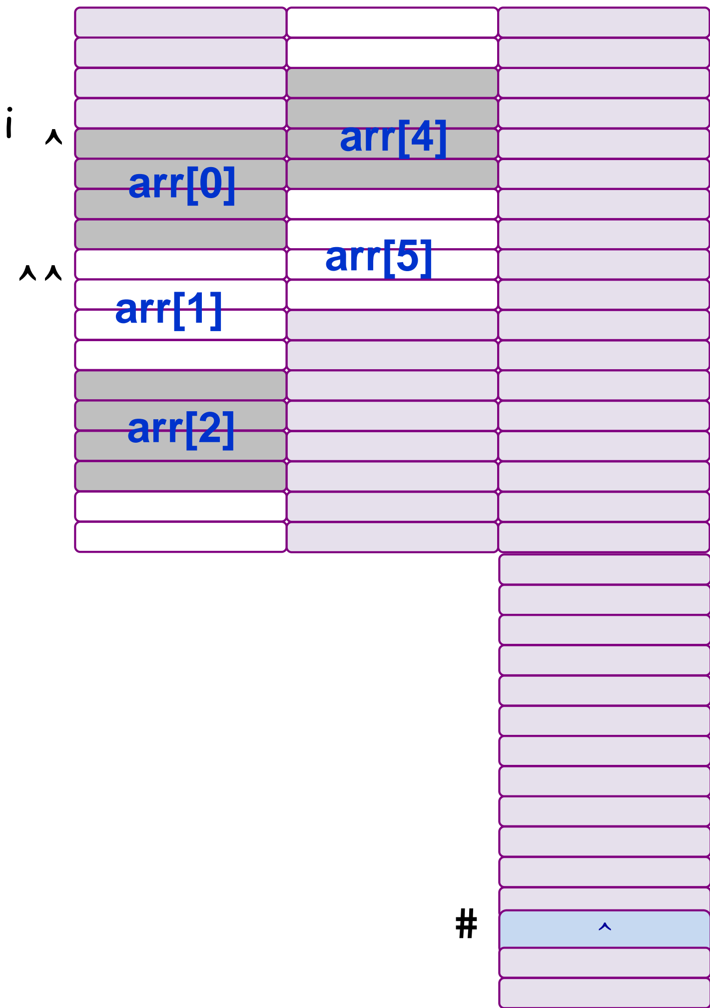
`arr`

3259	116	5008	5618	47	42
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]

Viste le osservazioni precedenti, le regole generali per gestire gli elementi di una array arr sono le due seguenti

con j indice degli elementi dell'array arr

```
arr + j == &arr[j]

*(arr+j) == arr[j]
```



ME
MO
RIA
(circa)

nel seguito ci sono esercizi … prova a svolgerli tutti, consultando le slide precedenti per trovare aiuto.

Tipicamente dopo l'enunciato di un esercizio c'e` lo svolgimento … prima fai l'esercizio per conto tuo … poi guarda la soluzione.

Se non riesci ad andare avanti

- consulta le slide precedenti per trovare esempi e ispirazione
- guarda piu` avanti per poco tempo … cercando di prendere spunto

Leggi le parti corrispondenti a quel che abbiamo visto qui, sul libro (i capitoli/paragrafi sono indicati nel programma del corso).

Aritmetica dei puntatori ... e array: due esercizi (1/6)

arr

3259	116	5008	5618	47	42
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]

`*arr = 7;`

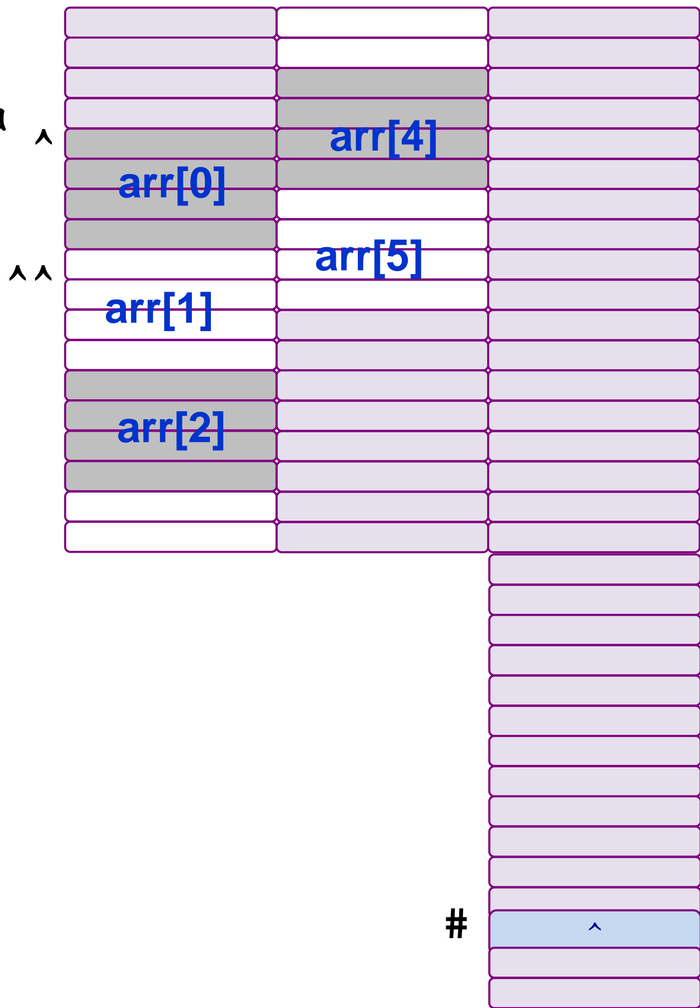
mette 7 nel primo elemento, cioe` e` equivalente a

`arr[0]=7;`

`*(arr+2) = 7;` e` equivalente a

se vogliamo mettere 7 in tutti gli elementi

se vogliamo leggere l'array



Aritmetica dei puntatori ... e array: due esercizi (2/6)

arr

3259	116	5008	5618	47	42
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]

`*arr = 7;`

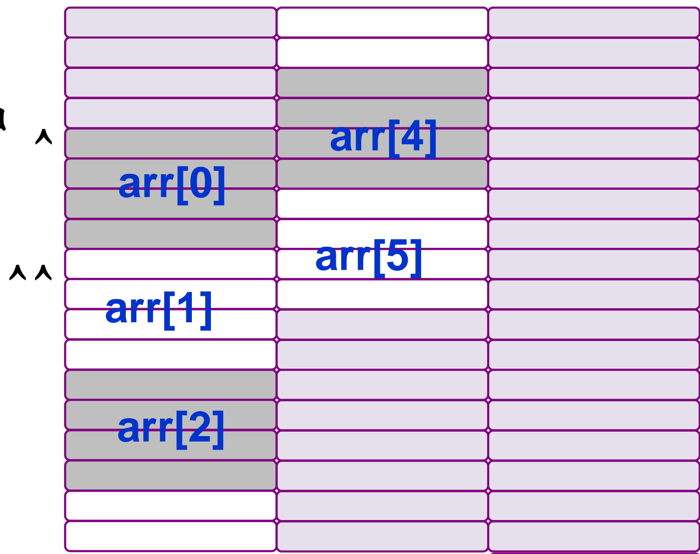
mette 7 nel primo elemento, cioe` e` equivalente a ^

`arr[0]=7;`

^^

`*(arr+2) = 7;` e` equivalente a

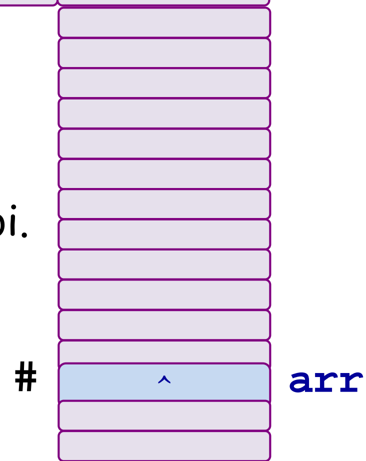
`arr[2] = 7;`



ME
MO
RIA
(circa)

Esercizio: mettere 7 in tutti gli elementi

- Prima scrivere il codice, in cui si usano le [] per accedere agli elementi; scrivi la sequenza di assegnazioni esplicitamente. Poi usa un ciclo se vuoi.
- Poi scrivere il codice in cui si usano (invece di []) le espressioni che usano l'operatore di dereferenziazione per accedere agli elementi.



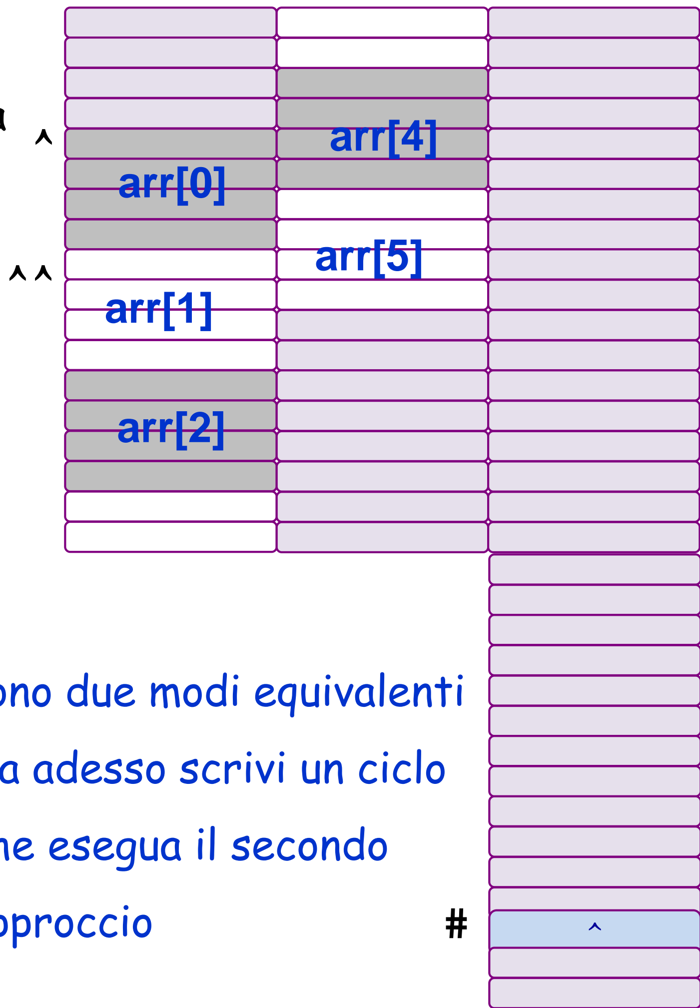
arr

Esercizio: leggere l'array ... *da fare dopo quello qui sopra*

Aritmetica dei puntatori … e array: due esercizi (3/6)

arr

3259	116	5008	5618	47	42
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]



`*arr = 7;`

mette 7 nel primo elemento, cioe` e` equivalente a

`arr[0]=7;`

`*(arr+2) = 7;` e` equivalente a

`arr[2]=7;`

se vogliamo mettere 7 in tutti gli elementi

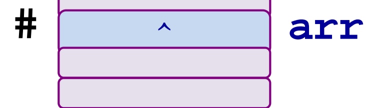
```
arr[0]=7          *arr = 7;
arr[1]=7          *(arr + 1) = 7;
arr[2]=7          *(arr + 2) = 7;
arr[3]=7          *(arr + 3) = 7;
arr[4]=7          *(arr + 4) = 7;
arr[5]=7          *(arr + 5) = 7;
```

sono due modi equivalenti

ma adesso scrivi un ciclo

che esegua il secondo

approccio



Aritmetica dei puntatori ... e array: due esercizi (4/6)

`arr`

3259	116	5008	5618	47	42
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]

```
*arr = 7;
```

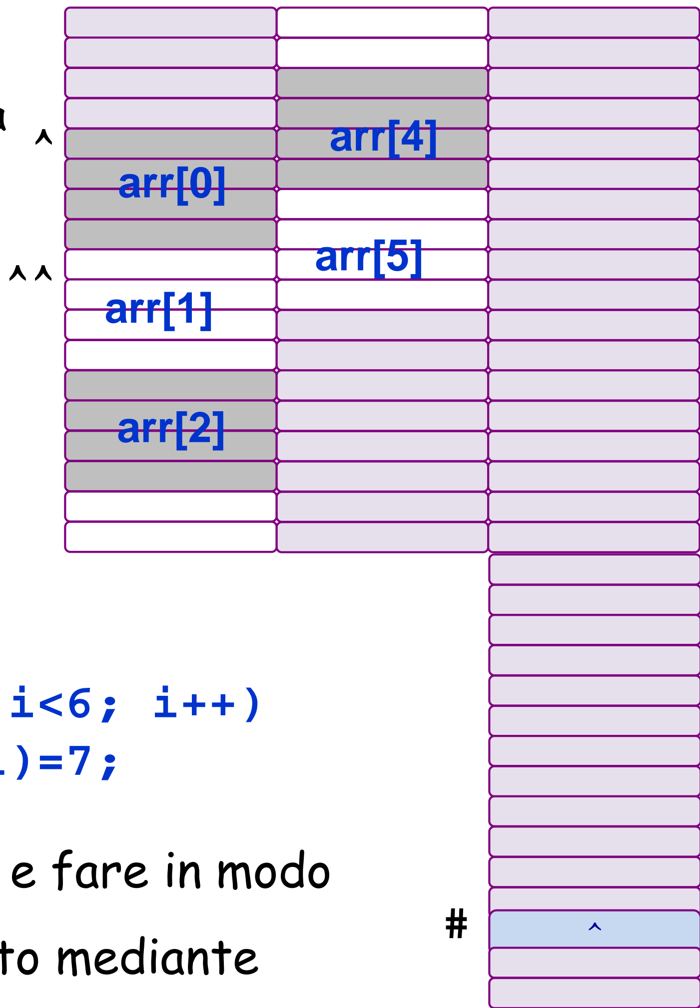
mette 7 nel primo elemento, cioe` e` equivalente a

```
arr[0]=7;
```

```
*(arr+2) = 7;
```

e` equivalente a

```
arr[2]=7;
```



ME
MO
RIA
(circa)

arr

se vogliamo mettere 7 in tutti gli elementi

```
for (i=0; i<6; i++)
    *(arr+i)=7;
```

Esercizio: leggere l'array - scrivere un ciclo, e fare in modo che l'accesso agli elementi dell'array sia fatto mediante puntatori, non usando le [] ...

Aritmetica dei puntatori ... e array: due esercizi (5/6)

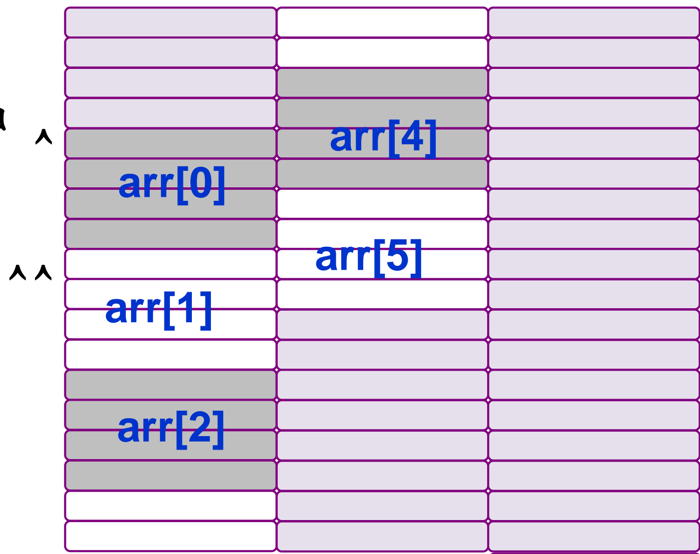
arr

3259	116	5008	5618	47	42
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]

```
*arr = 7;
```

mette 7 nel primo elemento, cioe` e` equivalente a

```
arr[0]=7;
```



ME
MO
RIA
(circa)

```
*(arr+2) = 7;
```

e` equivalente a

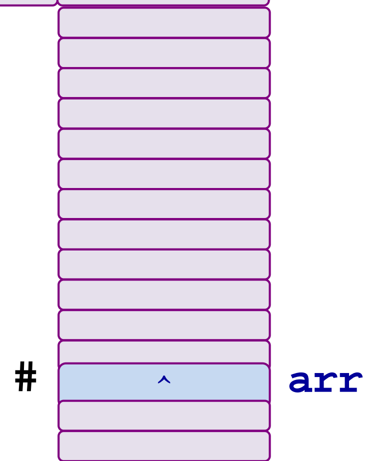
```
arr[2]=7;
```

se vogliamo mettere 7 in tutti gli elementi

```
for (i=0; i<6; i++)
    *(arr+i)=7;
```

se vogliamo leggere l'array

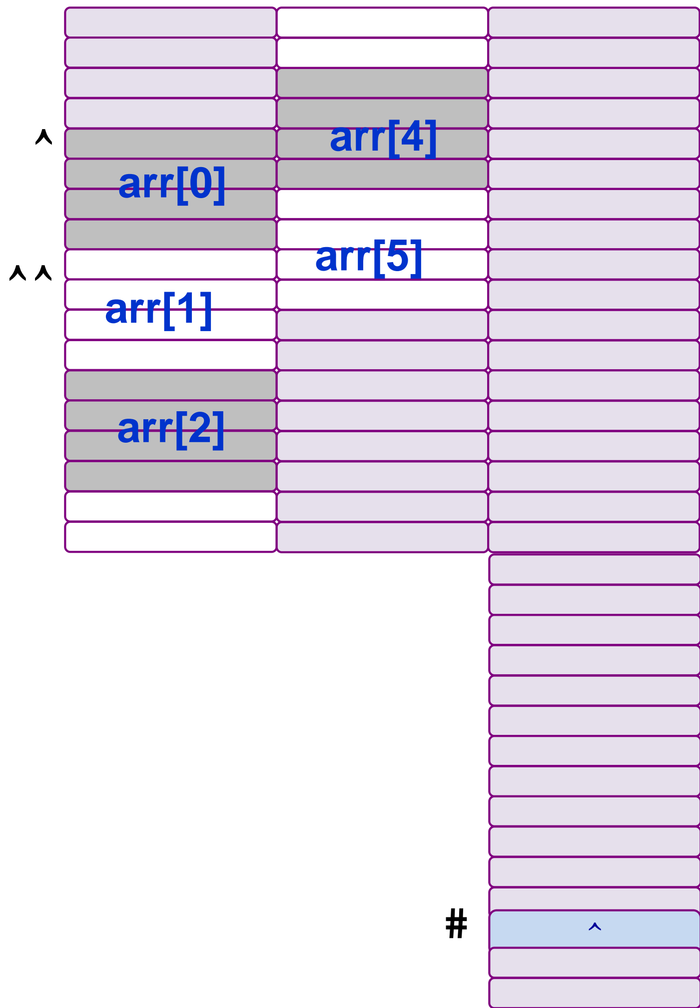
```
for (i=0; i<6; i++)
    scanf("%d", arr+i);
```



Aritmetica dei puntatori ... e array: due esercizi (6/6)

arr

3259	116	5008	5618	47	42
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]



ME
MO
RIA
(circa)

```
dopo aver sperimentato il codice precedente,
realizzare una funzione che
-  riceva un array di N interi e un valore init
   intero (come il 7 usato prima) e
-  inizializzi l'array inserendo in ogni suo
   elemento il valore init.
```

```
dopo aver sperimentato il codice precedente,
realizzare una funzione che
-  riceva un array di N interi e
-  legga gli elementi dell'array
```

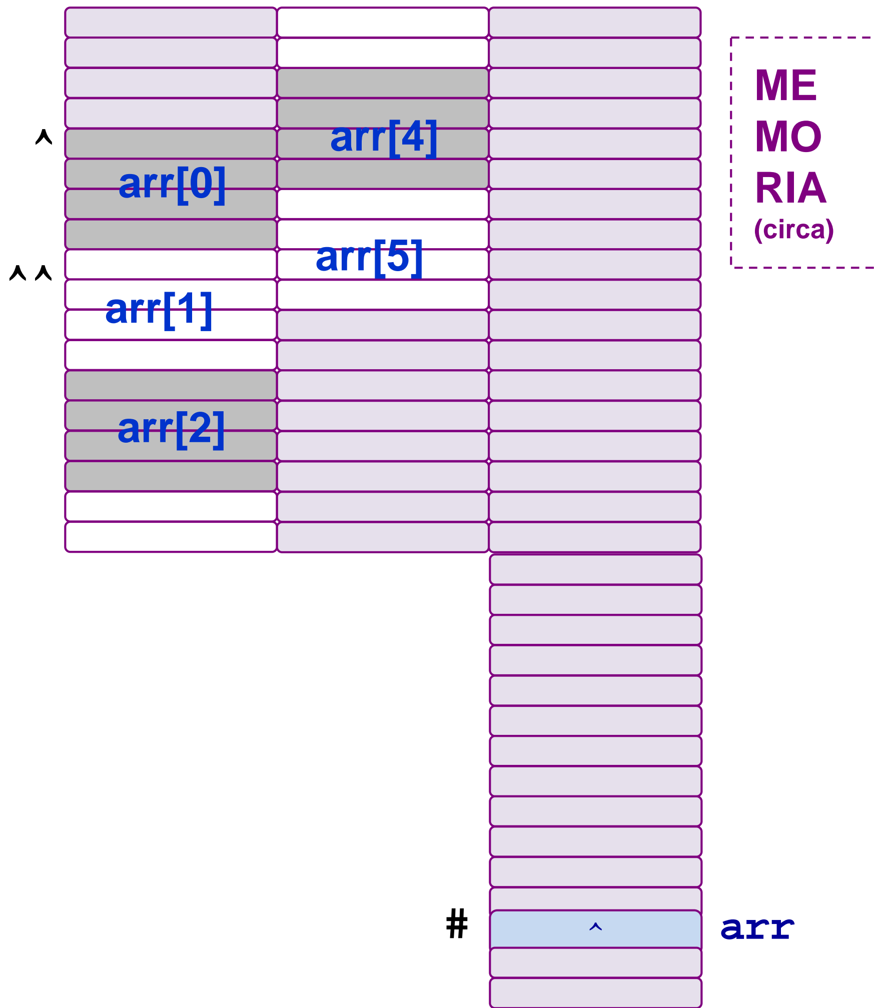
```
sperimentare le due funzioni precedenti in uno o
due programmi ad hoc;
```

Aritmetica dei puntatori ... e array: esercizio 10+i (1/3)

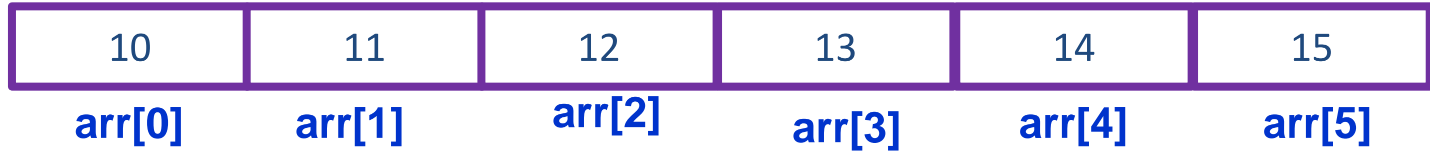
arr

3259	116	5008	5618	47	42
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]

scrivere un pezzetto di codice che riempia l'array, assegnando ad ogni elemento di indice i, il valore 10+i.

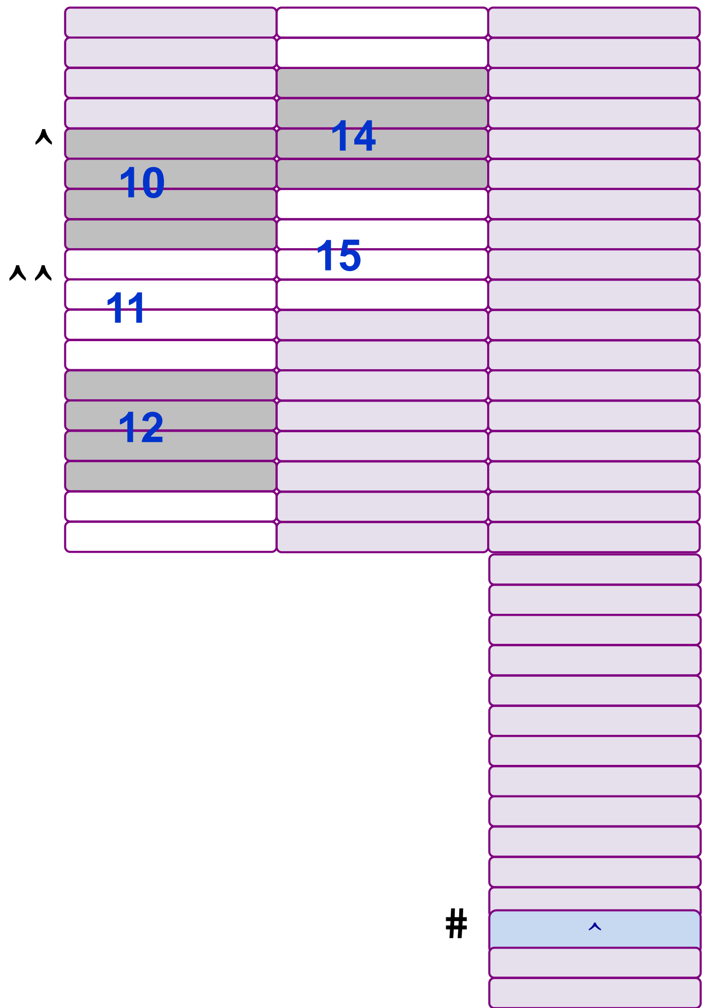


Aritmetica dei puntatori … e array: esercizio 10+i (2/3)

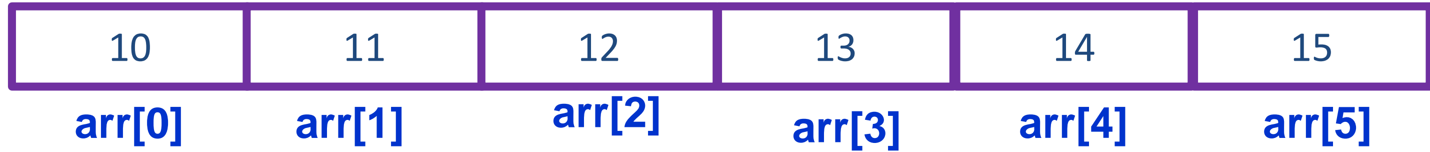


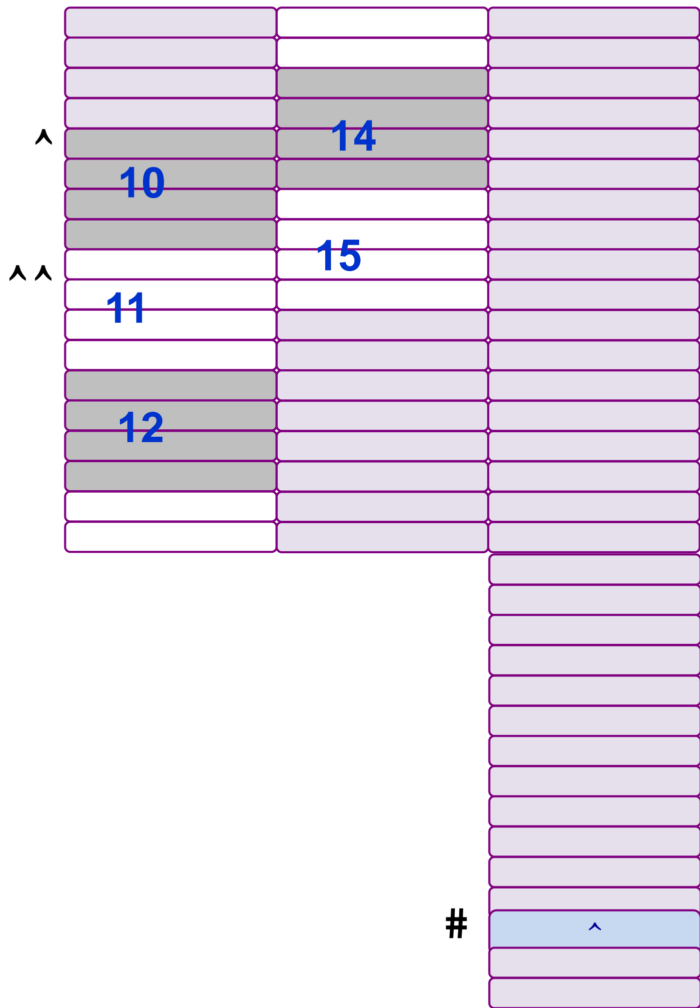
scrivere un pezzetto di codice che riempia l'array, assegnando ad ogni elemento di indice i, il valore 10+i.

```
for (i=0; i<6; i++)
    *(arr+i)= 10+i;
```



Aritmetica dei puntatori … e array: esercizio 10+i (3/3)



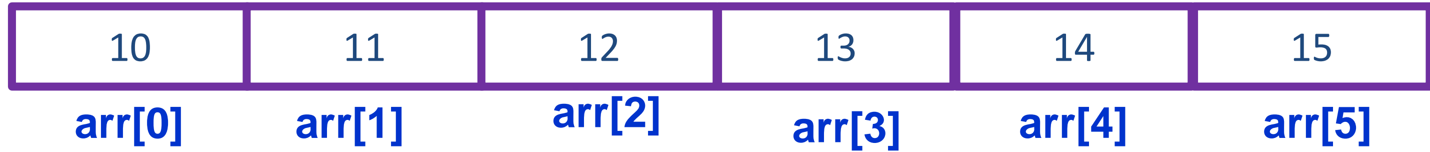


scrivere un pezzetto di codice che riempia l'array, assegnando ad ogni elemento di indice i, il valore 10+i.

```
for (i=0; i<6; i++)
    *(arr+i)= 10+i;
```

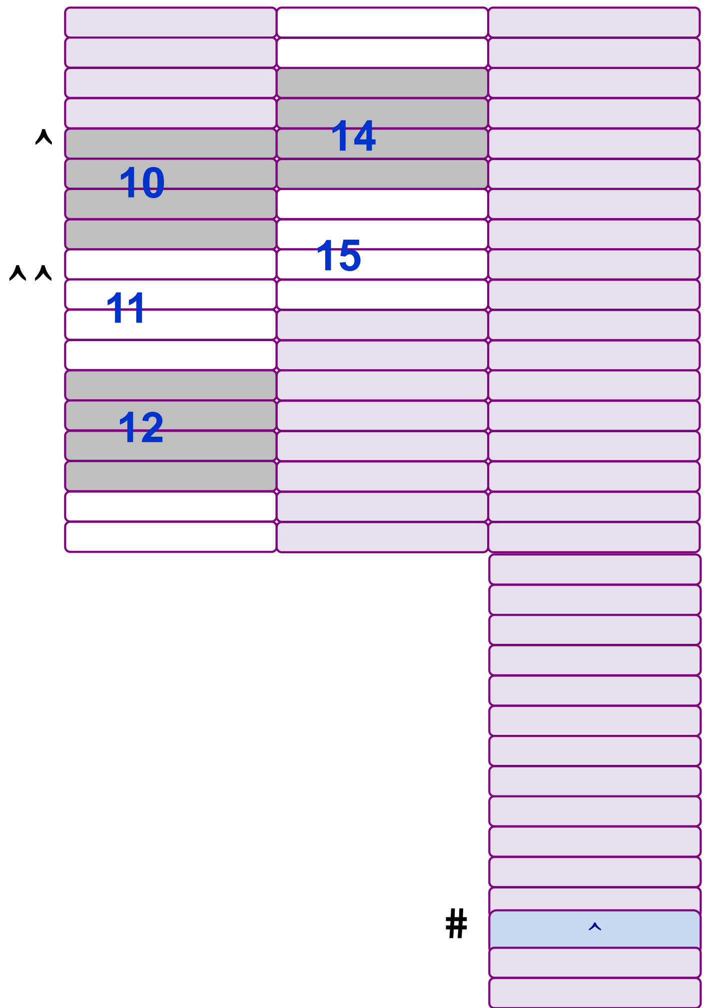
```
dopo aver sperimentato il codice precedente,
realizzare una funzione che
-  riceva un array di N interi
-  inizializzi l'array inserendo in ogni suo
   elemento il valore (10+i) dove i e` l'indice
   dell'elemento.

sperimentare la funzione in un programma ad hoc;
```

Aritmetica dei puntatori … e array: che fa?

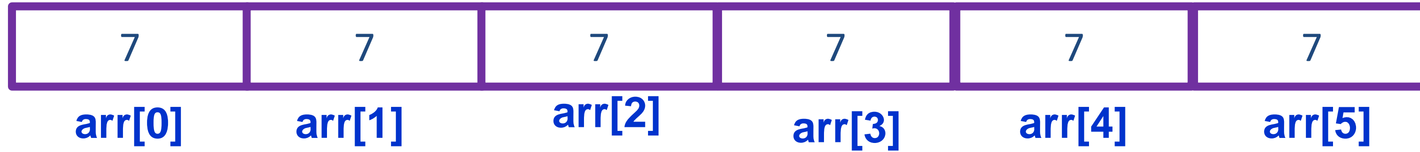


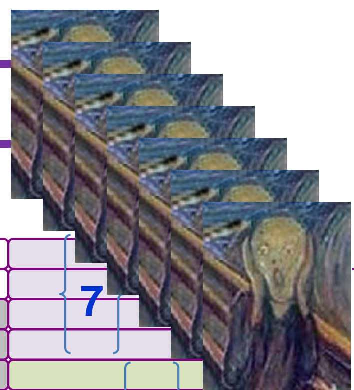
Che cosa fa il seguente codice?

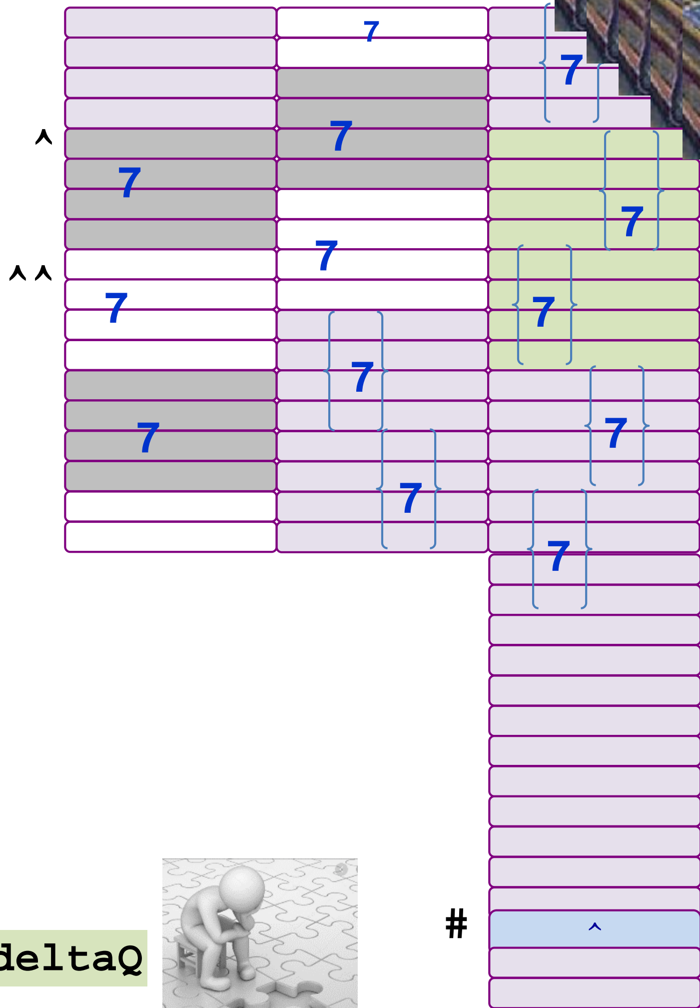
```
for (i=0; i<14; i++)
    *(arr+i)=7;
```



Aritmetica dei puntatori … e array: che fa? Umghch… bleah

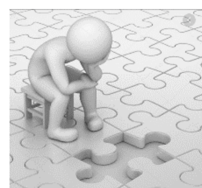


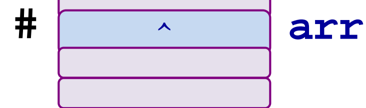




Che cosa fa il seguente codice?

```
for (i=0; i<14; i++)
    *(arr+i)=7;
```

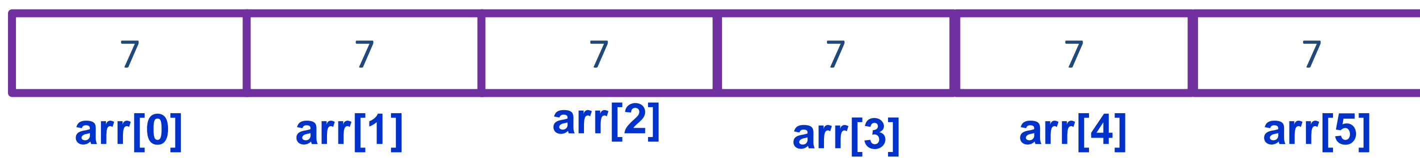


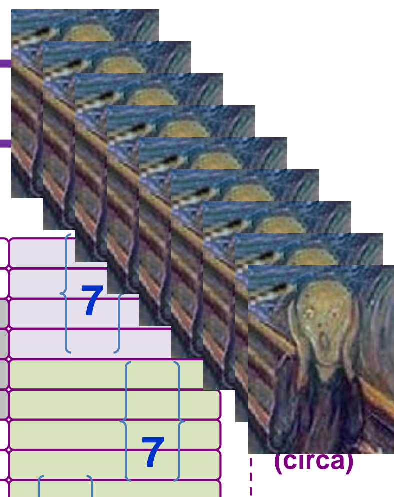


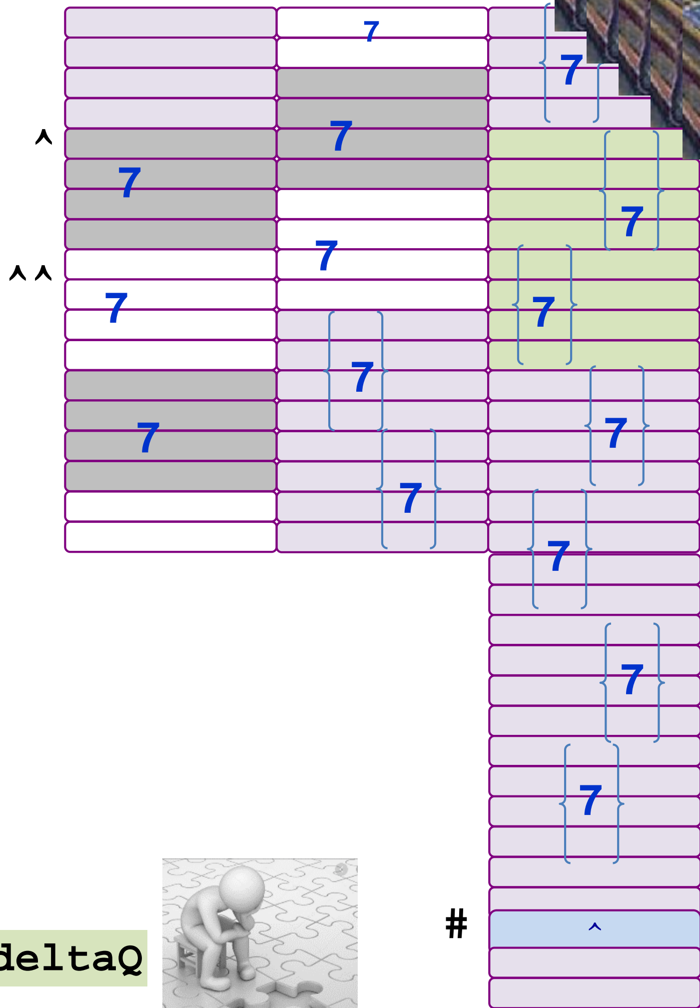
deltaQ

☺

Aritmetica dei puntatori … e array: che fa? (povero deltaQ)







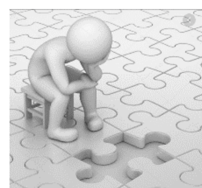
(deltaQ e` triste perche' il valore Floating point su 64 bit che conteneva e` stato scardinato: ora nei primi 32 bit c'e` la rappresentazione in complemento a 2 di 7, e nei secondi 32 bit idem.

si` , mancava qualcosa nella slide precede nte …

Che cosa fa il seguente codice?

```
for (i=0; i<14; i++)
    *(arr+i)=7;
```

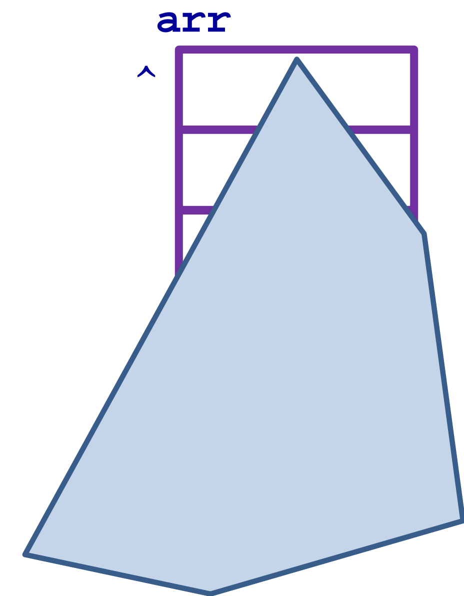
deltaQ



Aritmetica dei puntatori ... e array: che fa questo codice? (1/5)

```
int main() {

int *pi;
double *pd;
int arr[5] = {6,4,14,8,3};

pi = arr;
scanf("%d", pi); /* input 47 */
printf("%d\n", *pi);
printf("%d", *arr);
scanf("%d", pi+2);         /* input 42 */
pi = pi+1;

scanf("%d", pi+2);         /* input 1701 */
printf("%d", pd+2);
printf("%d", *(pd+4));
printf("%d", *(arr+5));

return 0;                                    }
```

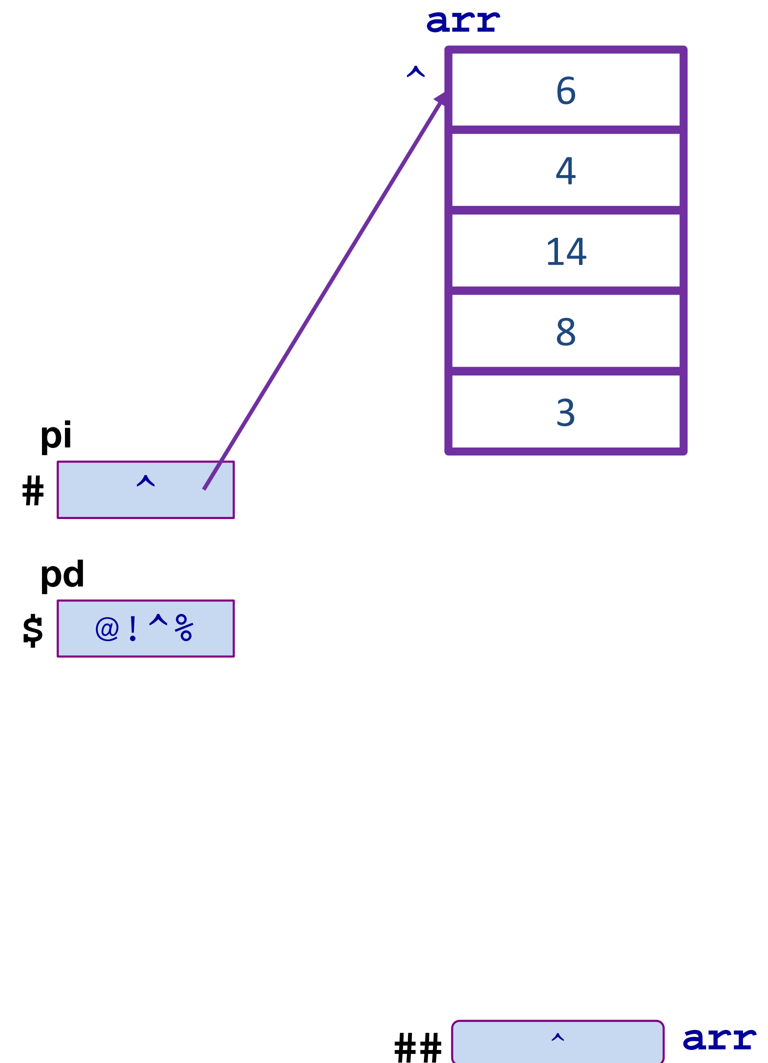
MEMORIA (circa)



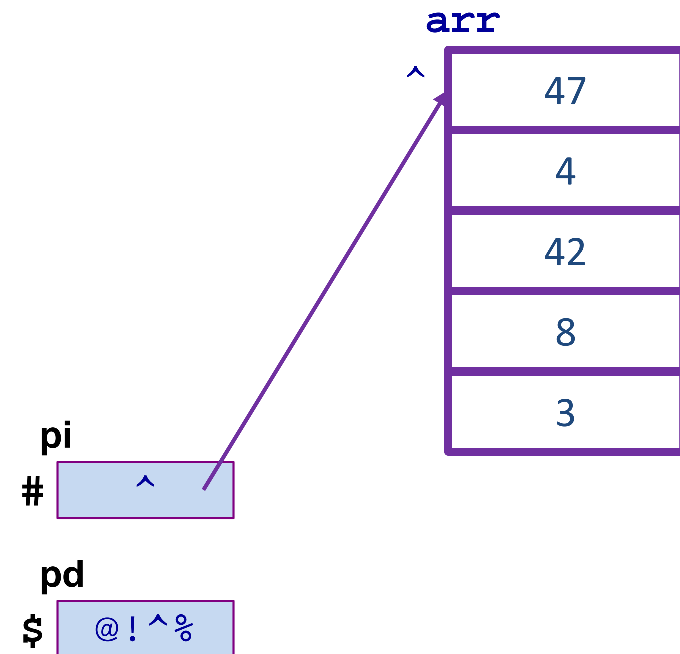
disegnare le variabili,
poi vedere le istruzioni e
decidere quali sono scorrette;
poi eseguire quelle corrette e
tracciare sul disegno i
cambiamenti in memoria

Aritmetica dei puntatori … e array: che fa questo codice? (2/5)

```
int main() {

int *pi;
double *pd;
int arr[5] = {6,4,14,8,3};

pi = arr;
scanf("%d", pi); /* input 47 */
printf("%d\n", *pi);
printf("%d", *arr);
scanf("%d", pi+2);         /* input 42 */
pi = pi+1;

scanf("%d", pi+2);         /* input 1701 */
printf("%d", pd+2);        KO
printf("%d", *(pd+4));     KO
printf("%d", *(arr+5));    KO

return 0;                                    }
```



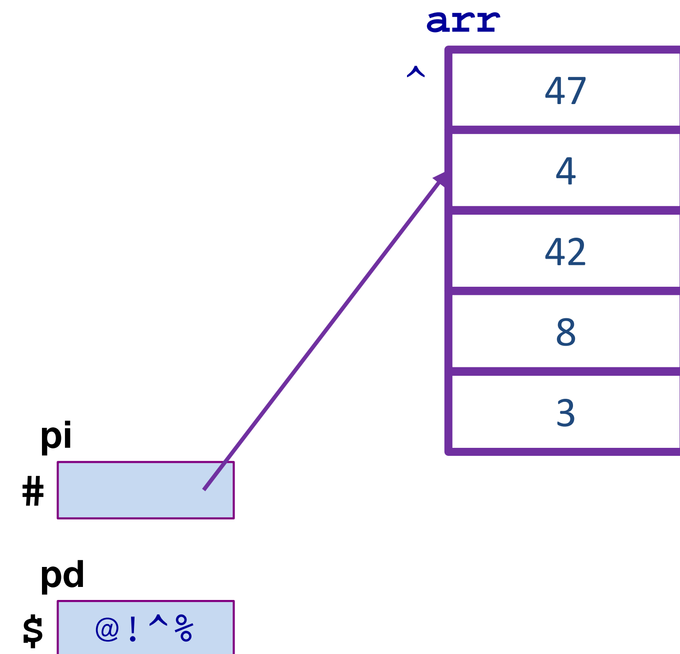
```
int main() {

int *pi;
double *pd;
int arr[5] = {6,4,14,8,3};

pi = arr;
scanf("%d", pi); /* input 47 */
printf("%d\n", *pi);
printf("%d", *arr);
scanf("%d", pi+2);          /* input 42 */
pi = pi+1;

scanf("%d", pi+2);          /* input 1701 */
printf("%d", pd+2);         KO
printf("%d", *(pd+4));      KO
printf("%d", *(arr+5));     KO

return 0;                                    }
```



47
47

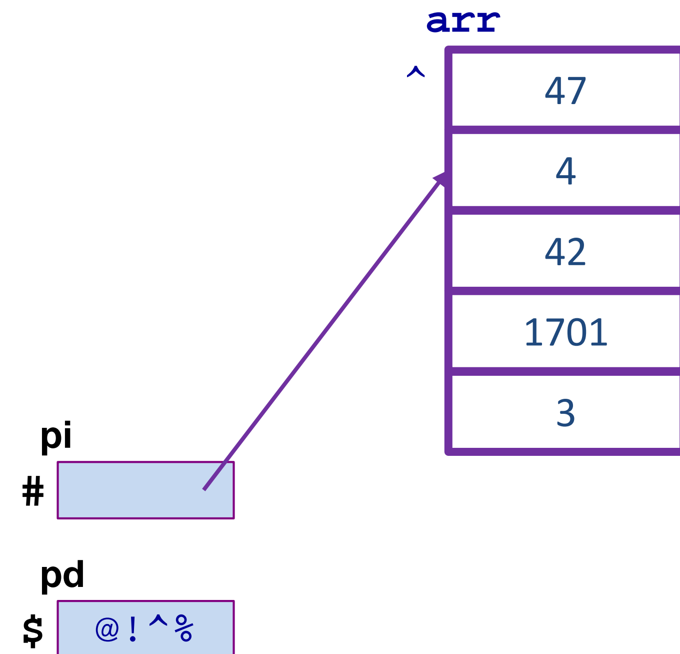
Aritmetica dei puntatori … e array: che fa questo codice? (4/5)

```
int main() {

int *pi;
double *pd;
int arr[5] = {6,4,14,8,3};

pi = arr;
scanf("%d", pi); /* input 47 */
printf("%d\n", *pi);
printf("%d", *arr);
scanf("%d", pi+2);          /* input 42 */
pi = pi+1;    <-----------

scanf("%d", pi+2);          /* input 1701 */
printf("%d", pd+2);         KO
printf("%d", *(pd+4));      KO
printf("%d", *(arr+5));     KO

return 0;                                    }
```



47
47

```
int main() {

int *pi;
double *pd;
int arr[5] = {6,4,14,8,3};

pi = arr;
scanf("%d", pi); /* input 47 */
printf("%d\n", *pi);
printf("%d", *arr);
scanf("%d", pi+2);         /* input 42 */
pi = pi+1;

scanf("%d", pi+2);         /* input 1701 */
printf("%d", pd+2);        KO
printf("%d", *(pd+4));     KO
printf("%d", *(arr+5));    KO

return 0;                                   }
```




```
47
47
```

Aritmetica dei puntatori ... e array: approfondimenti (1/3)

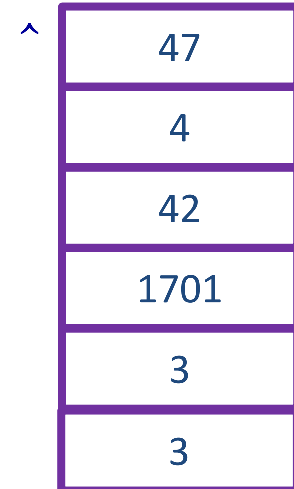
definire per esercizio funzioni che stampano array, anche sfruttando la natura di puntatore che, come abbiamo visto, una variabile array ha

```
#define N 7
...
void stampaArray (int v[N]) {

      riceve un array di N elementi e ne
           stampa gli elementi usando una
           printf("componente %d = %d\n", i, …);
```

farne diverse versioni, che usino l'accesso agli elementi dell'array, anche mediante indirizzo

provare a fare una versione in cui il contatore non è usato direttamente … ma si usa invece un puntatore che progressivamente punta sugli elementi dell'array

MEMORIA (circa)



Aritmetica dei puntatori ... e array: approfondimenti (2/3)

definire per esercizio funzioni che stampano array, anche sfruttando la natura di puntatore che, come abbiamo visto, una variabile array ha

```
#define N 7
...
void stampaArray (int v[N]) {

    riceve un array di N elementi e ne
        stampa gli elementi usando una
        printf("componente %d = %d\n", i, ...);
```

MEMORIA (circa)

^
47
4
42
1701
3
3

Infine riflettere su

"di che tipo e` arr"

int arr[6]

?

E` un intero?

NO, in realta` contiene un valore che e` un indirizzo di una locazione intera …

quindi di che tipo e`?

vedi, la rappresentazione di arr in memoria, qualche slide fa ...

dopo aver provato a rispondere, vedi anche la slide successiva

Aritmetica dei puntatori ... e array: approfondimenti (3/3)

definire per esercizio funzioni che stampano array, anche sfruttando la natura di puntatore che, come abbiamo visto, una variabile array ha

```
#define N 7
...
void stampaArray (int v[N]) {

    riceve un array di N elementi e ne
        stampa gli elementi usando una
        printf("componente %d = %d\n", i, …);
```

MEMORIA (circa)

^
47
4
42
1701
3
3

Infine riflettere su

"di che tipo e` arr"

int arr[6]

?

E` un intero?

NO, in realta` contiene un valore che e` un indirizzo di una locazione intera …

quindi di che tipo e`?

il tipo dell'array e` int [], continuando a seguire la notazione con le []

ma data la natura di arr come puntatore,

il tipo di arr e`, alla fine, int * cioe` puntatore a intero (o "indirizzo di locazione intera")

Aritmetica dei puntatori … e array: approfondimenti (4/3 …)

definire per esercizio funzioni che stampano array, anche sfruttando la natura di puntatore che, come abbiamo visto, una variabile array ha

```
#define N 7
...
void stampaArray (int v[N]) {

      riceve un array di N elementi e ne
            stampa gli elementi usando una
            printf("componente %d = %d\n", i, …);
```

farne diverse versioni, che usino l'accesso agli elementi dell'array, anche mediante indirizzo

provare a fare una versione in cui il contatore non e` usato direttamente … ma si usa invece un puntatore che progressivamente punta sugli elementi dell'array

MEMORIA (circa)

^

47
4
42
1701
3
3

Infine riflettere su

"di che tipo e` arr"

int arr[6]

?

int []

int * cioe` puntatore a intero (o "indirizzo di locazione intera")