A.A. 08/09

Fondamenti di Programmazione

(canale E-O)

Docente: Prof.ssa Tiziana Calamoneri
calamo@di.uniroma1.it

Esercitatore: Dott. Roberto Petroccia
petroccia@di.uniroma1.it

Pagina del corso:
http://twiki.di.uniroma1.it/twiki/view/Programmazione1/EO/WebHome

---

Esercitazione del 22/10/08

Indice

1. Passaggio di parametri alle funzioni
2. Call by value and call by reference
3. Accenno sui puntatori
4. Esercizi sulle funzioni
5. Descrizione array
6. Esercizi sugli array

22/10/08 - Roberto Petroccia: petroccia@di.uniroma1.it

Funzioni in C

Ogni funzione è della forma:

Tipo di ritorno Nome funzione Parametri

```
tipo    nomeFunzione    (parametri) {

        CORPO DELLA FUNZIONE

}
```

Funzioni in C

Tipo di ritorno Nome funzione Parametri

```
int    prova    (int x, int y, int z) {

        return x + y + z;

}

int main() {

    int somma = prova (2, 3, 4);

    printf(“La somma vale %d\n”, somma);

}
```

Funzioni in C

Tipo di ritorno Nome funzione Parametri

```
int    prova    (int x, int y, int z) {

        return x + y + z;

}

int main() {

    int somma = prova (2, 3, 4);

    printf("La somma vale %d\n", somma);

}
```

x = 2
y = 3
z = 4

Funzioni in C

Tipo di ritorno Nome funzione Parametri

```
int    prova    (int x, int y, int z) {
    return x + y + z;
}
int main() {
    int z = 2, y = 4, x = 3;
    int somma = prova (z, x, y);
    printf("La somma vale %d\n", somma);
}
```

x = 2
y = 3
z = 4

Call by value

```
void    raddoppia    (int x) {
        x *= 2;
}
int main() {
    int y = 3;
    raddoppia(y);
    printf("La y vale %d\n", y);
}
```

Call by value

```
void    raddoppia    (int x) {
        x *= 2;
}
int main() {
    int y = 3;
    raddoppia(y);
    printf("La y vale %d\n", y);
}
```

y vale sempre 3 dopo
la chiamata della funzione

Call by value

x e' una variabile globale

```
int x = 3;

void    raddoppia    () {
        x *= 2;
}
int main() {
    raddoppia();
    printf("La x vale %d\n", x);
}
```

Call by value

```
int x = 3;

void    raddoppia    () {
        x *= 2;
}
int main() {
    raddoppia();
    printf("La x vale %d\n", x);
}
```

x e' una variabile globale

non posso scegliere quale variabile raddoppiare al momento della chimata

Call by value

```
int x = 3;

void    raddoppia    () {
        x *= 2;
}
int main() {
    raddoppia();
    printf("La x vale %d\n", x);
}
```

x e' una variabile globale

non posso scegliere quale variabile raddoppiare al momento della chimata

ora la x e' stata raddoppiata

Call by value

```
void    raddoppia    (int x) {
        x *= 2;
}
int main() {
    int y = 3;
    raddoppia(y);
    printf("La y vale %d\n", y);
}
```

Call by value

```
void    raddoppia    (int x) {
        x *= 2;
}
int main() {
    int y = 3;
    raddoppia(y);
    printf("La y vale %d\n", y);
}
```

Per ottenere il valore dell'indirizzo di memoria di y scriviamo &y

Call by value

```
void    raddoppia    (int x) {
            x *= 2;
}
int main() {
    int y = 3;
    printf("y = %d, indirizzo = %p\n", y, &y);
    y += 5;
    printf("y = %d, indirizzo = %p\n", y, &y);
}
```

Indirizzo di y si ha con &y

Il valore di y cambia da 3 a 8 ma l'indirizzo in memoria e' sempre lo stesso

Call by value

```
void    raddoppia    (int x) {
        x *= 2;
}
int main() {
    int y = 3;
    raddoppia(y);
    printf("La y vale %d\n", y);
}
```

Call by value

```
void    raddoppia    (int x) {
        x *= 2;
}
int main() {
    int y = 3;
    raddoppia(y);
    printf("La y vale %d\n", y);
}
```

Alla chiamata della funzione viene creata nuova variabile x che contiene lo stesso valore di y

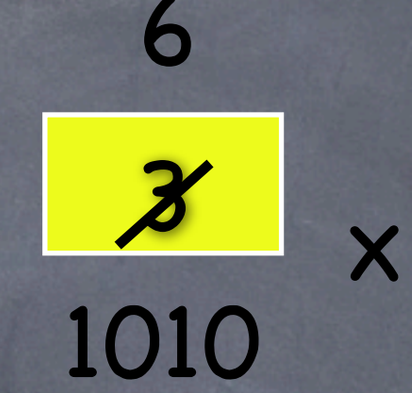
Call by value

```
void    raddoppia    (int x) {
        x *= 2;
}
int main() {
    int y = 3;
    raddoppia(y);
    printf("La y vale %d\n", y);
}
```

Alla chiamata della funzione viene creata nuova variabile x che contiene lo stesso valore di y

Call by value

```
void    raddoppia    (int x) {
        x *= 2;
}
```

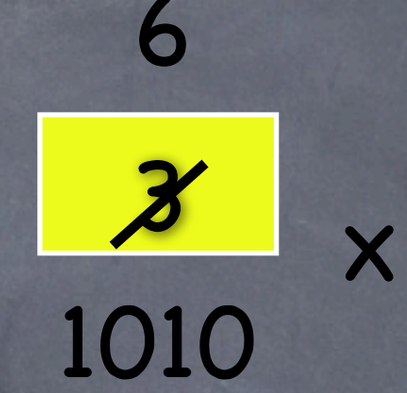


x viene modificata ma non y

```
int main() {
    int y = 3;
    raddoppia(y);
    printf("La y vale %d\n", y);
}
```

Call by value

```
void    raddoppia    (int x) {
        x *= 2;
}
```



x viene modificata ma non y

```
int main() {
    int y = 3;
    raddoppia(y);
    printf("La y vale %d\n", y);
}
```

y vale sempre 3

Accenno ai puntatori

Dichiarazione

```
int count = 7;

int * countPtr;
```

Usiamo la * davanti al nome della variabile

Accenno ai puntatori

Dichiarazione

```
int count = 7;

int * countPtr;
```

Usiamo la * davanti al nome della variabile

I puntatori sono riferimenti in memoria

22/10/08 - Roberto Petroccia: petroccia@di.uniroma1.it

Accenno ai puntatori

Dichiarazione

```
int count = 7;

int * countPtr;
```

Usiamo la * davanti al nome della variabile

I puntatori sono riferimenti in memoria

Inizializziamo il puntatore

```
countPtr = &count;
```

Facciamo puntare countPtr alla variabile count

22/10/08 - Roberto Petroccia: petroccia@di.uniroma1.it

Accenno ai puntatori

Dichiarazione

```
int count = 7;

int * countPtr;
```

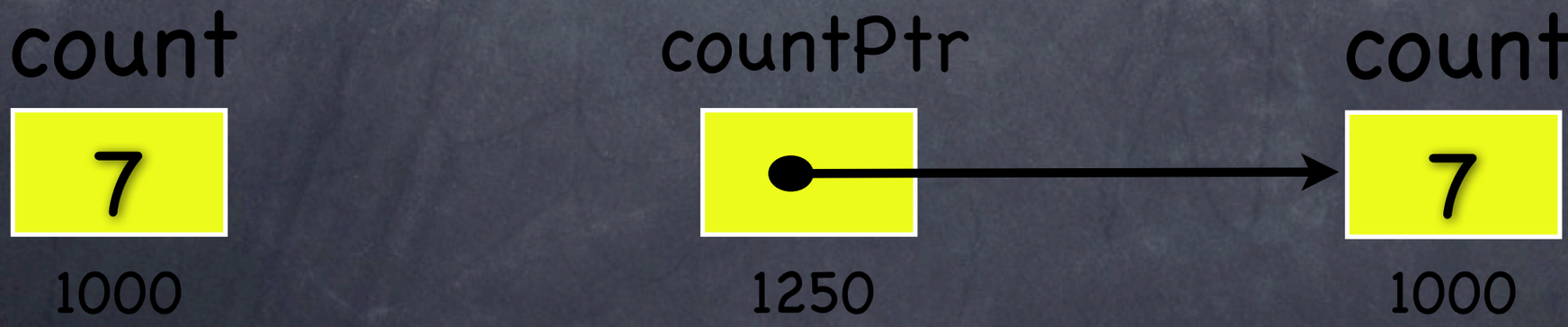
Usiamo la * davanti al nome della variabile

I puntatori sono riferimenti in memoria

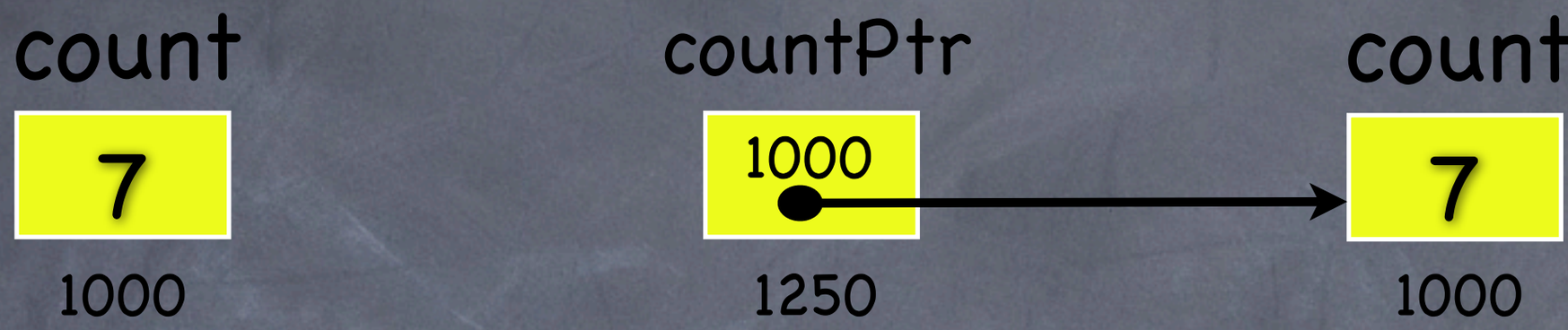
Inizializziamo il puntatore

```
countPtr = &count;
```

Facciamo puntare countPtr alla variabile count

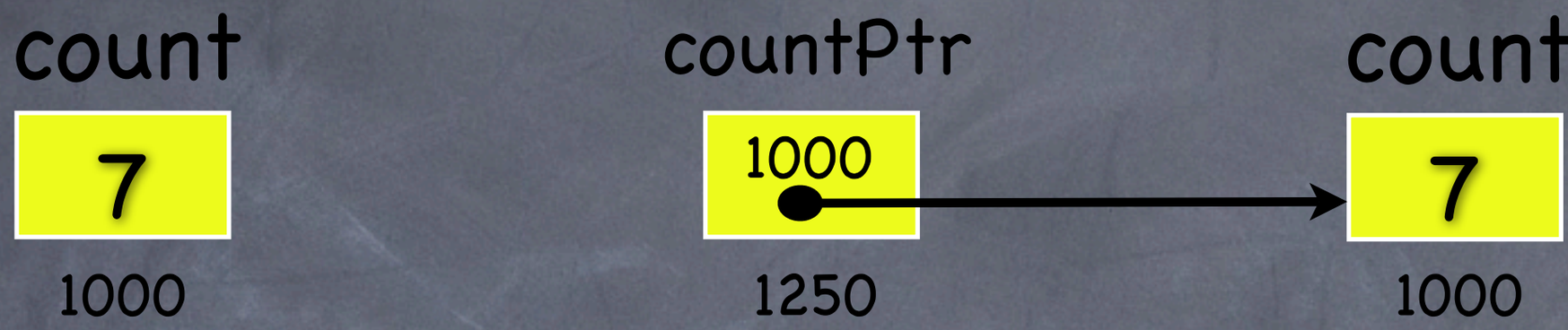


Accenno ai puntatori




```
int count = 7;
int * countPtr = & count;
```

Accenno ai puntatori

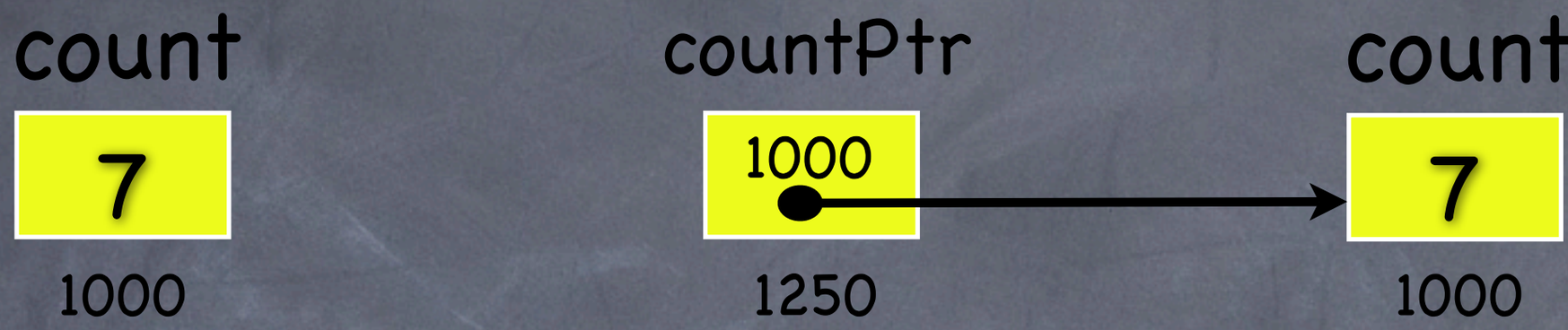


int count = 7;

int * countPtr = & count;

CountPtr contiene l'indirizzo in memoria di count

Accenno ai puntatori



int count = 7;

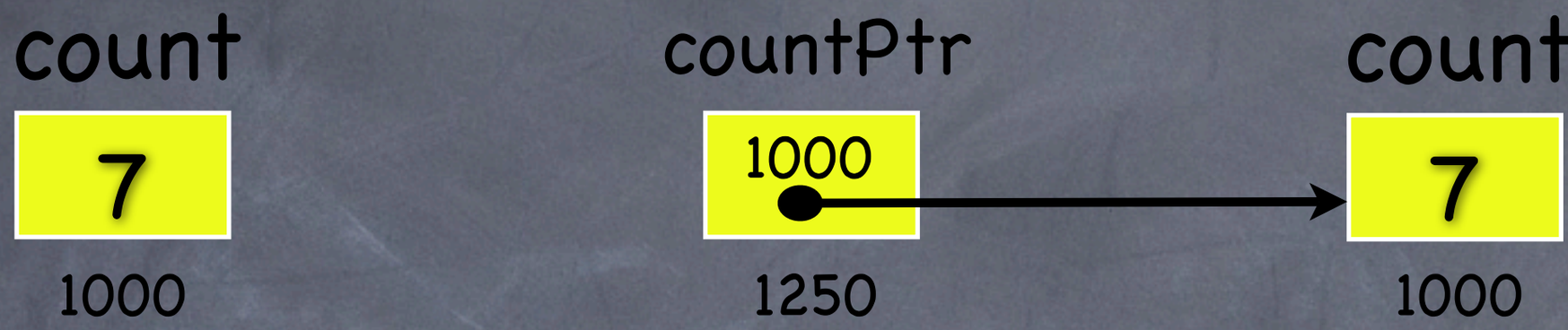
int * countPtr = & count;

CountPtr contiene l'indirizzo in memoria di count

printf("indirizzo count = %p\n", &count);

printf("valore di countPtr = %p\n", countPtr);

Accenno ai puntatori



int count = 7;

int * countPtr = & count;

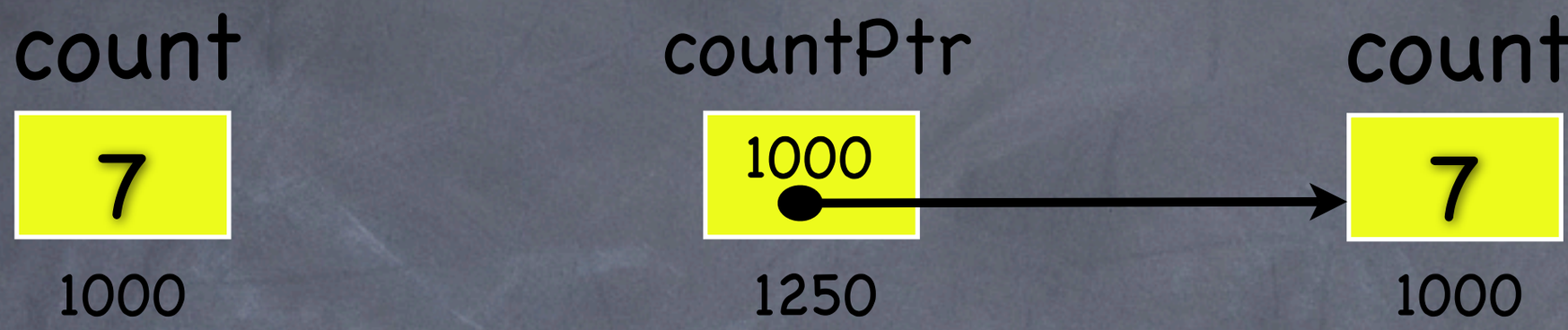
CountPtr contiene l'indirizzo in memoria di count

printf("indirizzo count = %p\n", &count);

printf("valore di countPtr = %p\n", countPtr);

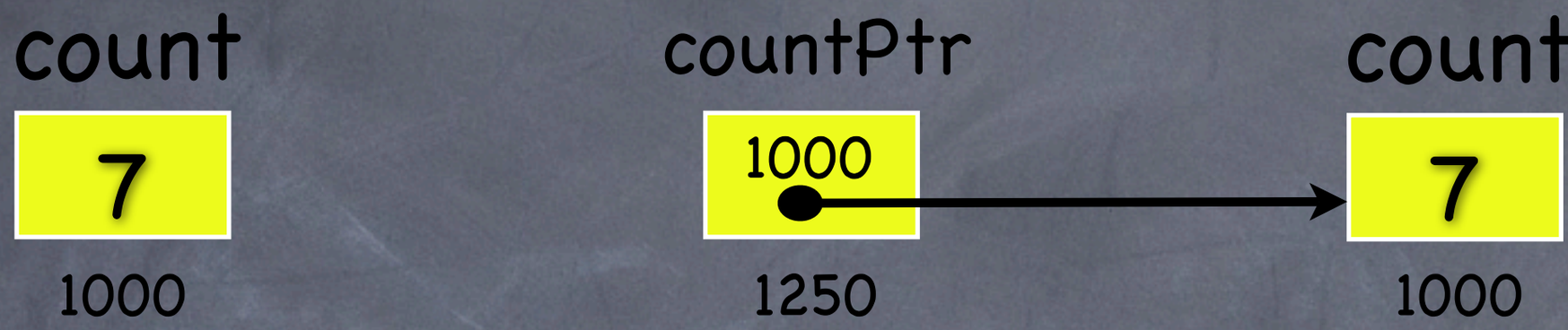
Entrambi stampano la stessa cosa, ossia
indirizzo di count che vale 1000

Accenno ai puntatori




```
int count = 7;
int * countPtr = & count;
```

Accenno ai puntatori

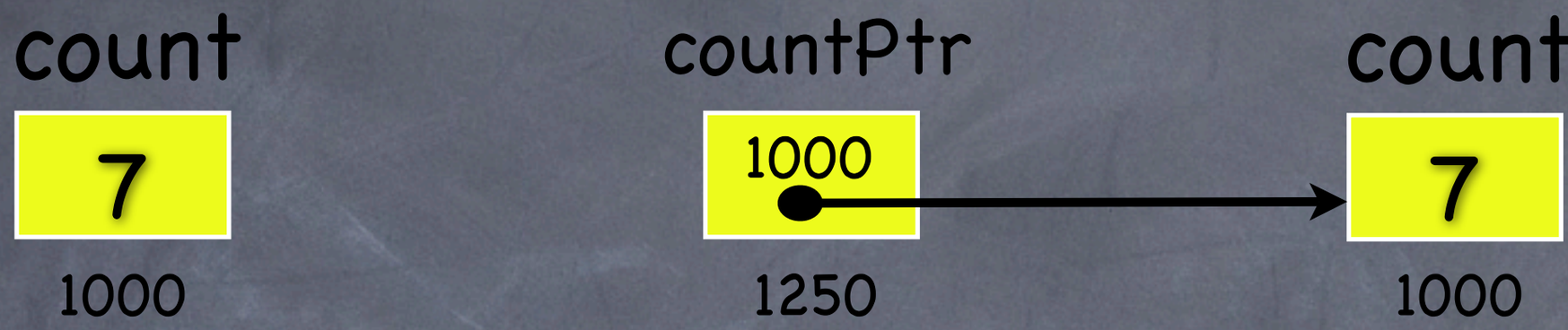


int count = 7;

int * countPtr = & count;

Per ottenere il valore della variabile puntata bisogna dereferenziare il puntatore tramite il simbolo *

Accenno ai puntatori



int count = 7;

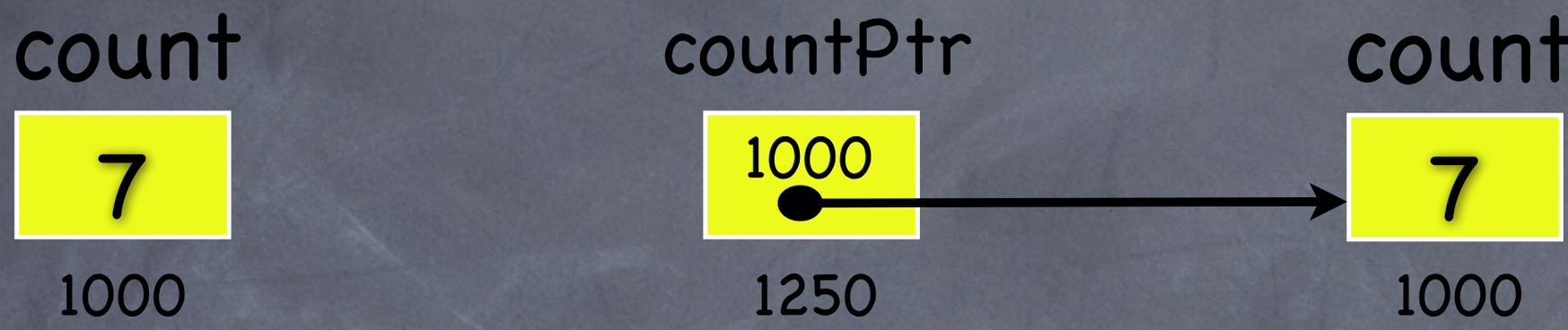
int * countPtr = & count;

Per ottenere il valore della variabile puntata bisogna dereferenziare il puntatore tramite il simbolo *

printf("valore di count = %d\n", count);

printf("valore puntato da countPtr = %d\n", *countPtr);

Accenno ai puntatori



int count = 7;

int * countPtr = & count;

Per ottenere il valore della variabile puntata bisogna dereferenziare il puntatore tramite il simbolo *

printf("valore di count = %d\n", count);

printf("valore puntato da countPtr = %d\n", *countPtr);

Entrambi stampano la stessa cosa,
ossia il valore di count che vale 7

Call by reference

```
void    raddoppia    (int * x) {
        *x *= 2;
}
int main() {
    int y = 3;
    raddoppia(&y);
    printf("La y vale %d\n", y);
}
```

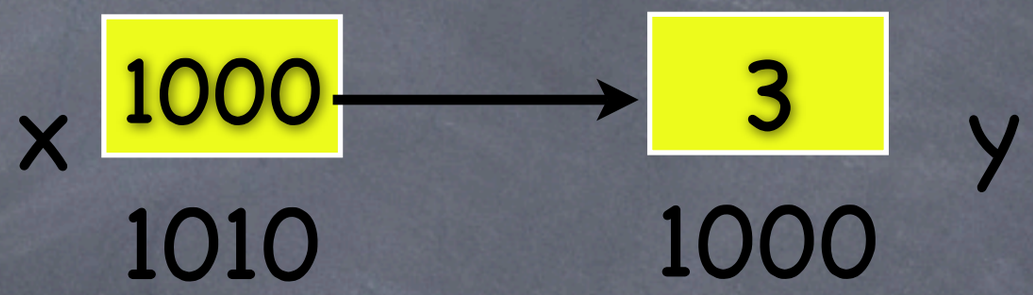
Call by reference

```
void    raddoppia    (int * x) {
        *x *= 2;
}
int main() {
    int y = 3;
    raddoppia(&y);
    printf("La y vale %d\n", y);
}
```

modificata il valore puntato da x, quindi y

Call by reference

```
void    raddoppia    (int * x) {
        *x *= 2;
}
```

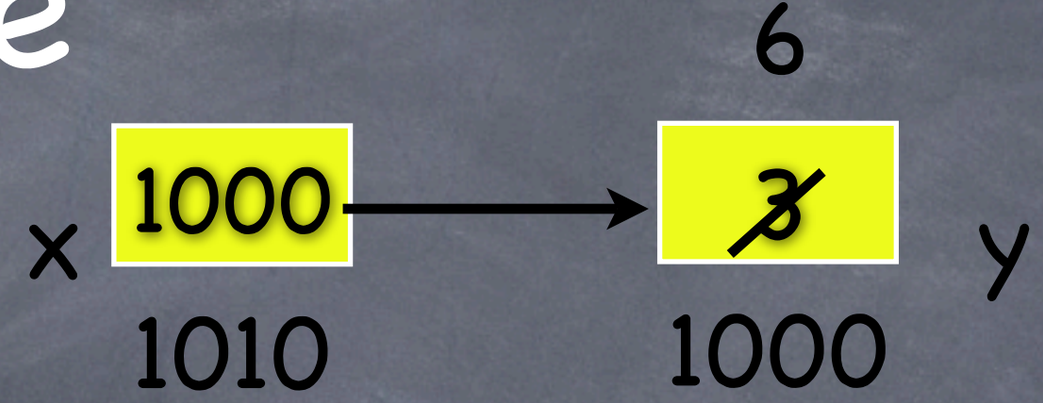


modificata il valore puntato da x, quindi y

```
int main() {
    int y = 3;
    raddoppia(&y);
    printf("La y vale %d\n", y);
}
```

Call by reference

```
void    raddoppia    (int * x) {
        *x *= 2;
}
```

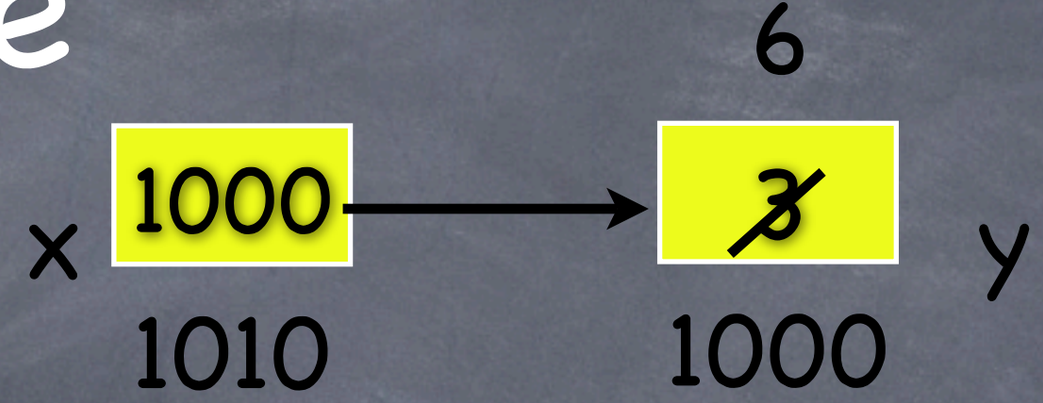


modificata il valore puntato da x, quindi y

```
int main() {
    int y = 3;
    raddoppia(&y);
    printf("La y vale %d\n", y);
}
```

Call by reference

```
void    raddoppia    (int * x) {
        *x *= 2;
}
```



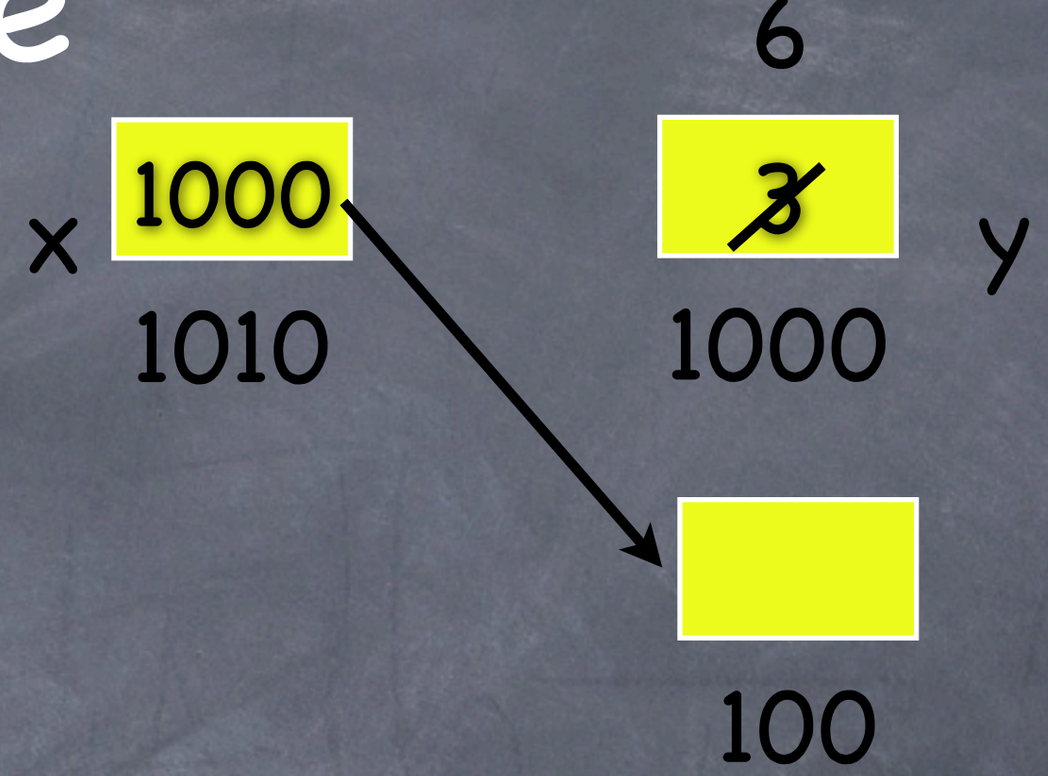
modificata il valore puntato da x, quindi y

```
int main() {
    int y = 3;
    raddoppia(&y);
    printf("La y vale %d\n", y);
}
```

Viene creata una copia che pero' contiene lo stesso indirizzo di memoria, cambiando il valore in memoria cambiamo il valore della variabile puntata

Call by reference

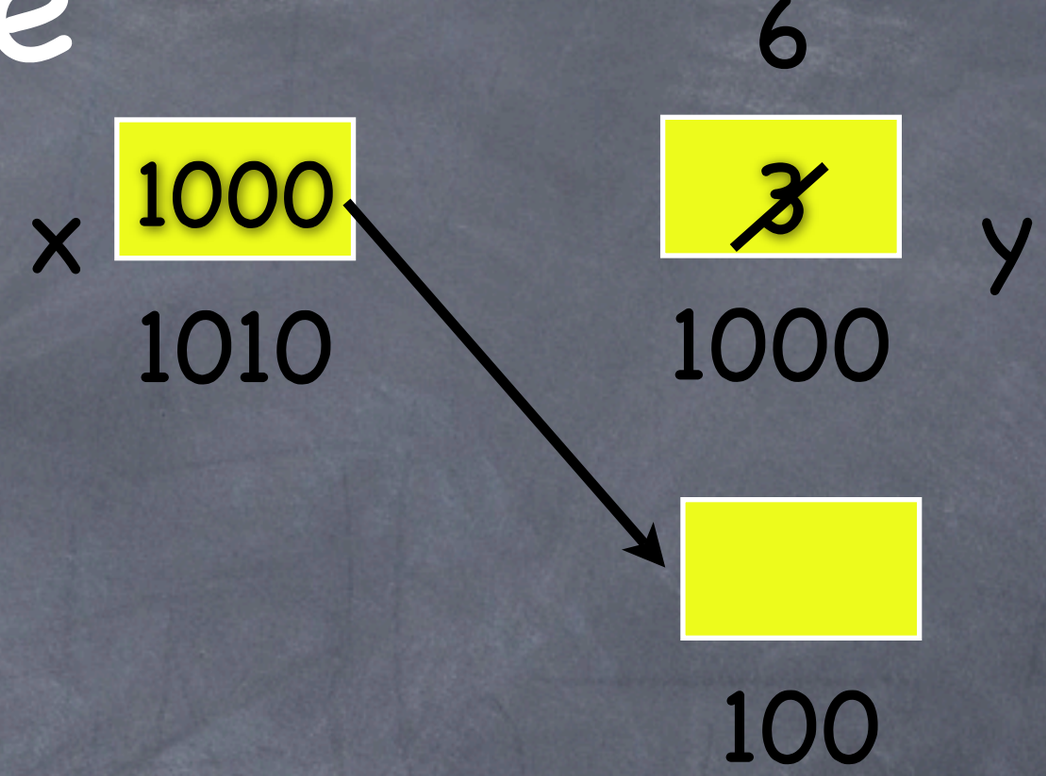
```
void    raddoppia    (int * x) {
        *x *= 2;
        x = 100;
        *x += 1;
}

int main() {
    int y = 3;
    raddoppia(&y);
    printf("La y vale %d\n", y);
}
```



Viene creata una copia che pero' contiene lo stesso indirizzo di memoria, cambiando il valore in memoria cambiamo il valore della variabile puntata

Call by reference

```
void    raddoppia    (int * x) {
          *x *= 2;
          x = 100;
          *x += 1;
}
int main() {
        int y = 3;
        raddoppia(&y);
        printf("La y vale %d\n", y);
}
```

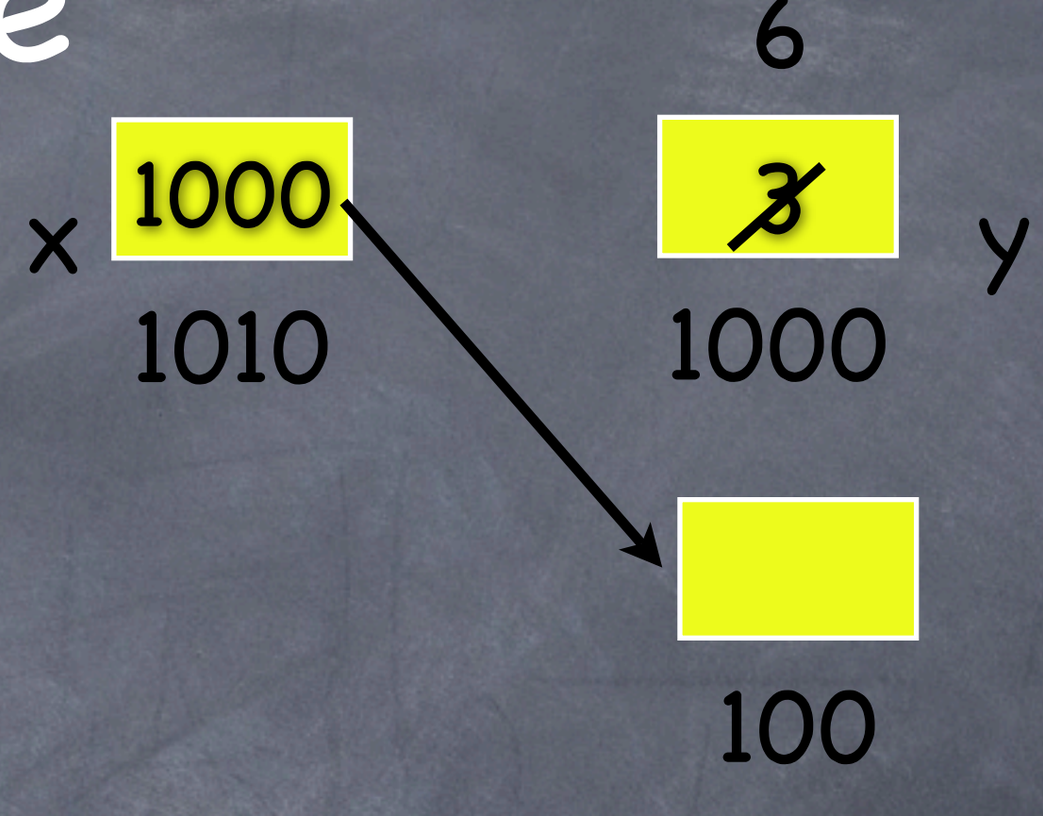


potrebbe essere memoria sporca
o protetta

Viene creata una copia che pero' contiene lo stesso indirizzo di memoria, cambiando il valore in memoria cambiamo il valore della variabile puntata

Call by reference

```
void    raddoppia    (int * x) {
          *x *= 2;
          x = 100;
          *x += 1;
}
int main() {
        int y = 3;
        raddoppia(&y);
        printf("La y vale %d\n", y);
}
```



Non cambia piu' il valore di y, anzi potrebbe mandare il programma in errore

Viene creata una copia che pero' contiene lo stesso indirizzo di memoria, cambiando il valore in memoria cambiamo il valore della variabile puntata

DOMANDE ???

22/10/08 - Roberto Petroccia: petroccia@di.uniroma1.it

Esercizi

1. Scrivere una funzione che letti due interi x ed y assegni ad x il valore di y e viceversa, mantenendo la modifica all'uscita della funzione.

2. Scrivere un funzione che presi in input 5 interi (tre valori più un valore per il min ed il max di output) memorizzi nella variabile min e nella variabile max il minimo ed il massimo dei tre interi presi in input.

Soluzione Ex. 1

```
#include <stdio.h>

void scambia (int* x, int* y);

int main() {
        int x, y;
        scanf (“%d %d”, &x, &y);
        printf(“x = %d - y = %d\n”,x, y);
        scambia(&x, &y);
        printf(“x = %d - y = %d\n”,x, y);
        return 0;
}
```

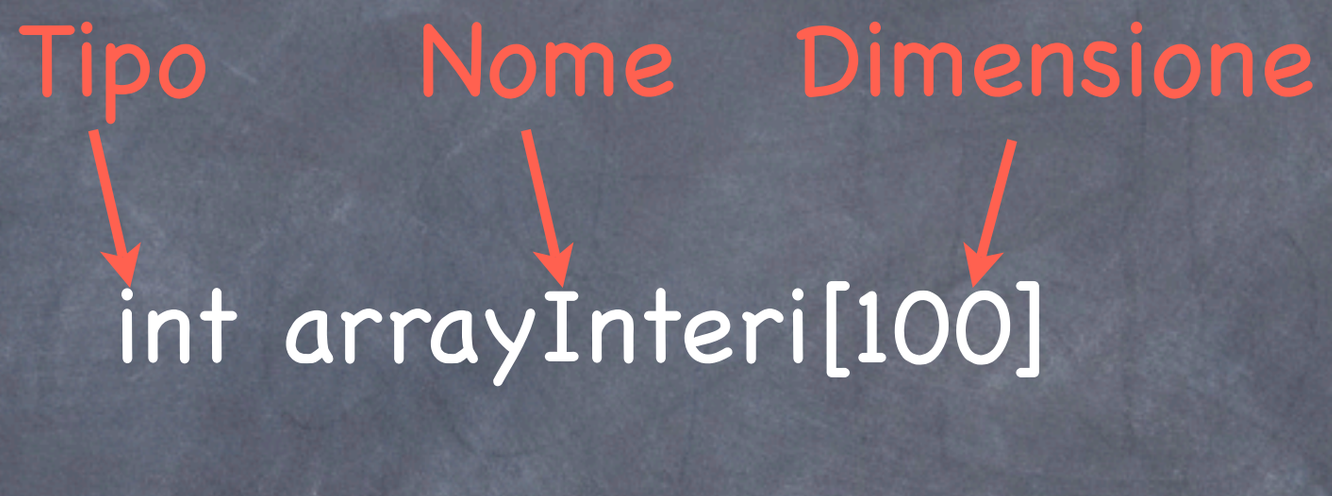
```
void scambia (int * x, int * y) {
        int aux;
        aux = *x;
        *x = *y;
        *y = aux;
    }
```

Soluzione Ex. 2

```
#include <stdio.h>
void trovaValori (int x, int y, int z, int *min, int* max);

int main() {
        int x, y, z;
        int min, max;
        scanf (“%d %d %d”, &x, &y, &z);
        trovaValori(x, y, z, &min, &max);
        printf(“min = %d - max = %d\n”, min, max);
        return 0;
}

void trovaValori (int x, int y, int z, int *min, int* max){
   int a, b;
   a = trovaMinimo(x, y, z);
   b = trovaMassimo(x, y, z);
   *min = a;
   *max = b;
}
```

Array

- L'array è un insieme di locazioni di memoria sequenziali



Array

- L'array è un insieme di locazioni di memoria sequenziali

Tipo Nome Dimensione

```
int arrayInteri[100]
```

int c[8]

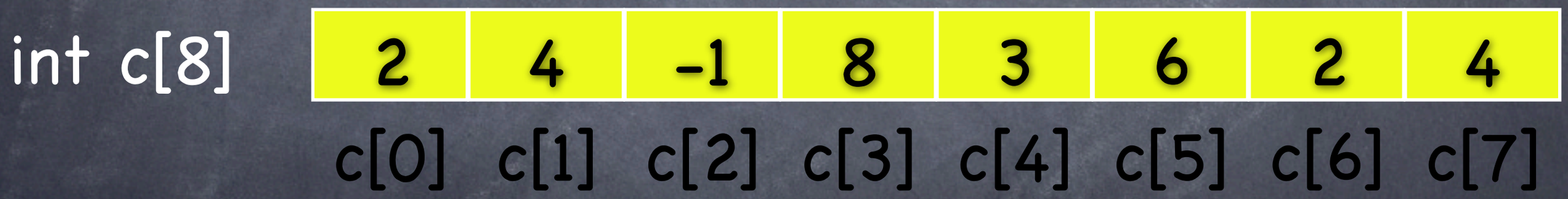
2	4	-1	8	3	6	2	4
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]

Array

- L'array è un insieme di locazioni di memoria sequenziali

Tipo Nome Dimensione

```
int arrayInteri[100]
```



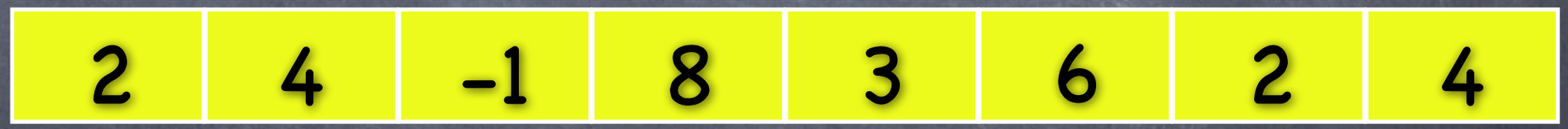
Array

- L'array è un insieme di locazioni di memoria sequenziali

Tipo Nome Dimensione

```
int arrayInteri[100]
```

int c[8]



Tutte stesso nome

c[3] = 8

Array

- L'array è un insieme di locazioni di memoria sequenziali

Tipo Nome Dimensione

```
int arrayInteri[100]
```

int c[8]

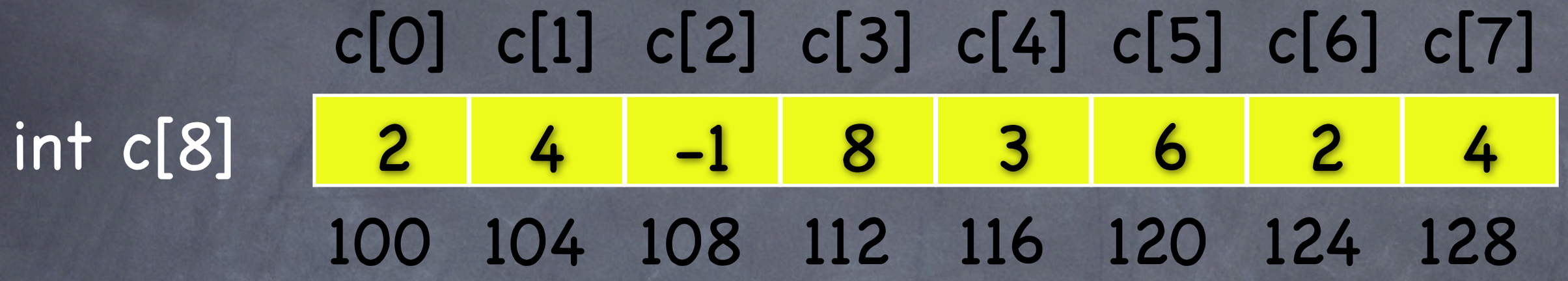
2	4	-1	8	3	6	2	4
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]

Tutte stesso nome

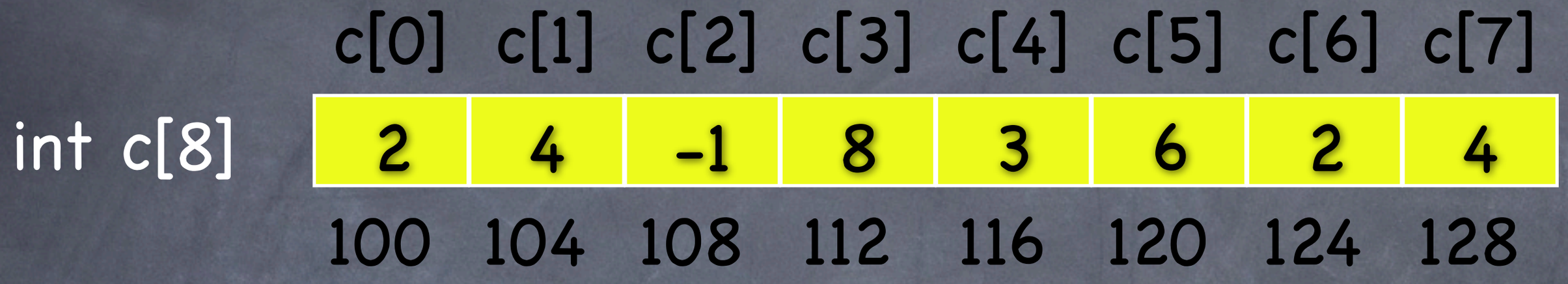
c[3] = 8

```
printf("valore c[3] = %d\n", c[3]);
```

Array

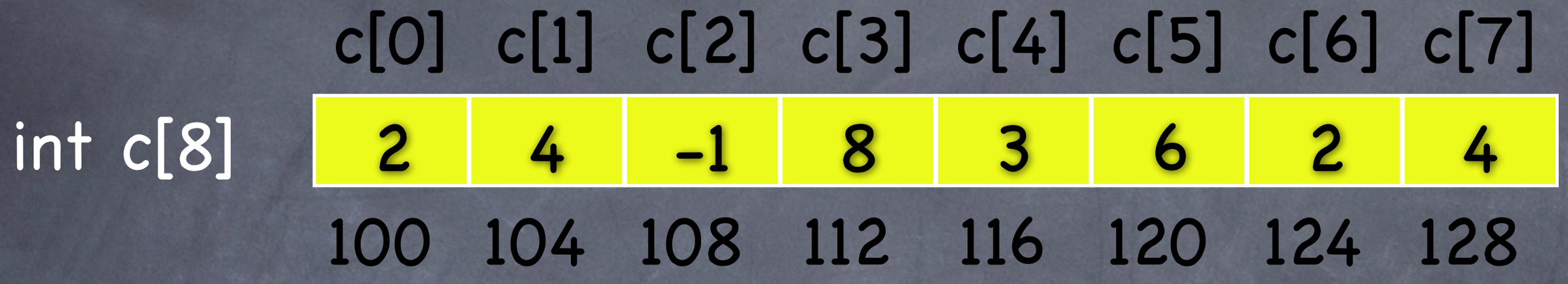

c[0] c[1] c[2] c[3] c[4] c[5] c[6] c[7]
int c[8]
2 4 -1 8 3 6 2 4
100 104 108 112 116 120 124 128

Array



La variabile c contiene l'indirizzo della prima locazione di memoria

Array



La variabile c contiene l'indirizzo della prima locazione di memoria

c ha lo stesso valore di &c[0], in questo caso 100

Array

	c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
int c[8]	2	4	-1	8	3	6	2	4
	100	104	108	112	116	120	124	128

La variabile c contiene l'indirizzo della prima locazione di memoria

c ha lo stesso valore di &c[0], in questo caso 100

```
int c[8];
int d[8];
d = c;
```

Non copia c in d ma solo l'indirizzo, la copia va fatta valore per valore

Copia di un array

Per copiare c in d, d deve poter contenere tutti gli elementi di c

```
int c[8];
int d[8];
int i = 0;
for (i = 0; i < 8; i++) {
  d[i] = c[i];
}
```

Copia di un array

Per copiare c in d, d deve poter contenere tutti gli elementi di c

```
int c[8];
int d[8];
int i = 0;
for (i = 0; i < 8; i++) {
  d[i] = c[i];
}
```

Facciamo la copia elemento per elemento

Stampare un array

```
int c[8];
for (i = 0; i < 8; i++) {
  printf("c[%d] = %d\n", i, c[i]);
}
```

Riempire un array

```
int c[8];
for (i = 0; i < 8; i++) {
  c[i] = i;
}
```

c[0] = 0
c[1] = 1
....
c[7] = 7

Riempire un array

```
int c[8];
for (i = 0; i < 8; i++) {
  c[i] = i;
}
```

c[0] = 0
c[1] = 1
....
c[7] = 7

```
int n;
scanf ("%d", &n);
int c[n];
for (i = 0; i < n; i++) {
  scanf("%d", &c[i]);
}
```

Valori inseriti dall'utente

DOMANDE ???

22/10/08 - Roberto Petroccia: petroccia@di.uniroma1.it

Esercizi

1. Scrivere un programma che letto n da input e letti n interi, memorizzi i valori in un vettore. Poi letto un altro valore k da input stampi quante volte k compare tra gli n valori

2. Scrivere un programma che letto n da input e letti poi n interi stampi tali interi al contrario, dall'ultimo inserito al primo.

Soluzione Ex. 1

```
#include <stdio.h>

int conta (int k, int y[], int n);

int main() {
        int n, y, i, k;
        scanf (“%d”, &n);
        if (n <= 0) {
            return -1;
        }
        int aux[n];
        for (i = 0; i < n; i++) {
            scanf(“%d”, &aux[i]);
        }
        scanf (“%d”, &k);
        printf(“k appare %d volte\n”, conta(k, aux, n));
        return 0;
}
```

```
int conta (int  k, int y[], int n) {
    int i;
    int count = 0;
    for (i = 0; i < n; i++) {
        if (y[i] == k) {
            count++;
        }
    }
    return count;
}
```

Soluzione Ex. 2

```
#include <stdio.h>

void stampa (int y[], int n);

int main() {
        int n, y, i, k;
        scanf (“%d”, &n);
        if (n <= 0) {
            return -1;
        }
        int aux[n];
        for (i = 0; i < n; i++) {
            scanf(“%d”, &aux[i]);
        }
        stampa(aux, n);
        return 0;
}
```

```
void stampa (int y[], int n) {
    int i;
    for (i = n - 1; i >= 0; i--) {
        printf(“%d\n”, y[i]);
    }
}
```