

lezione 3 II semestre

array a più dimensioni
puntatori

Array a 1, 2, 3, 4, 5, dimensioni:

```
int A[20];
```

```
int X[5][10];
```

```
int Y[3][4][10];
```

```
int Z[10][10][20][30];
```

e così via

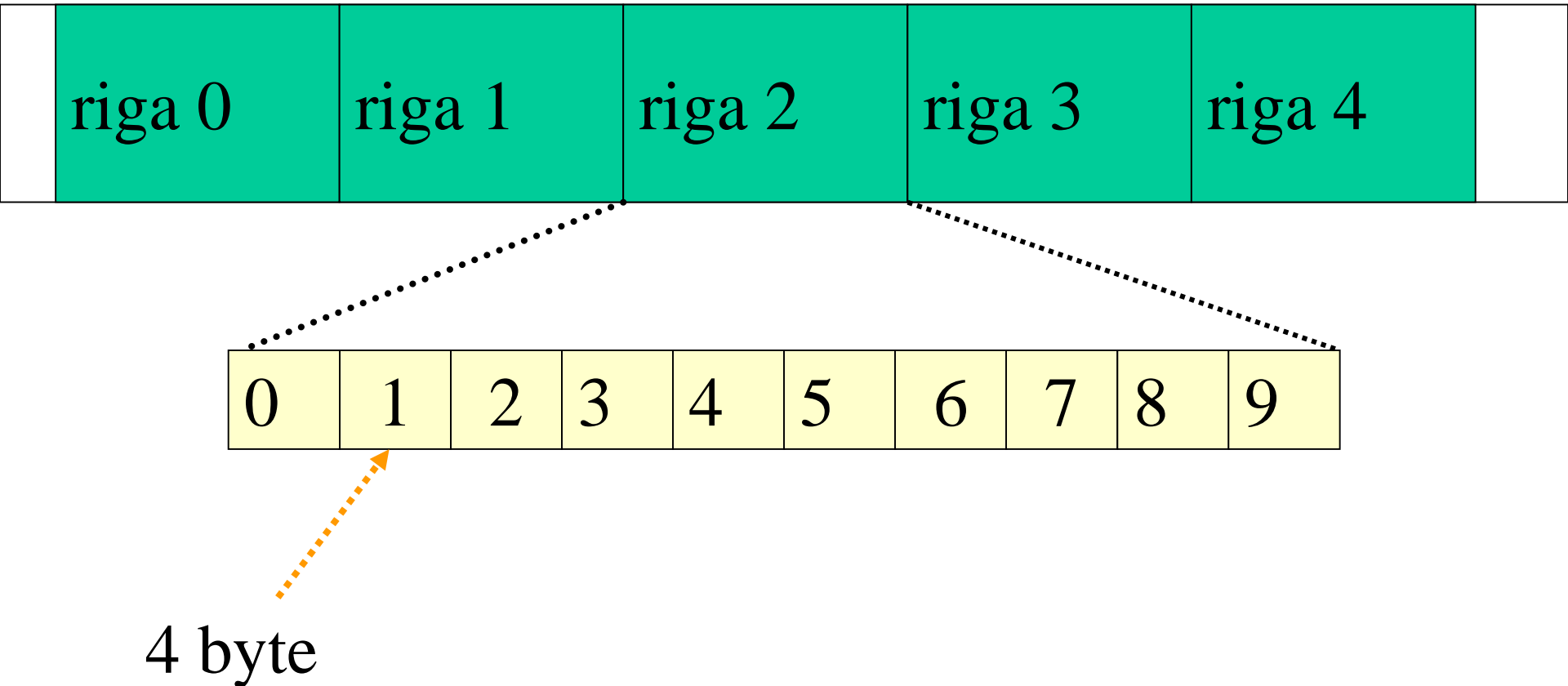
limite della prima
dimensione è 5, della
seconda è 10

elementi: X[0][0] X[4][9] Z[0][0][0][1]

Y[3][0][1] non esiste, gli strati sono 0, 1 e 2

di nuovo gli elementi sono accostati nella
RAM per righe: `int X[5][10]`

RAM



e `int Y[3][4][10];` ?

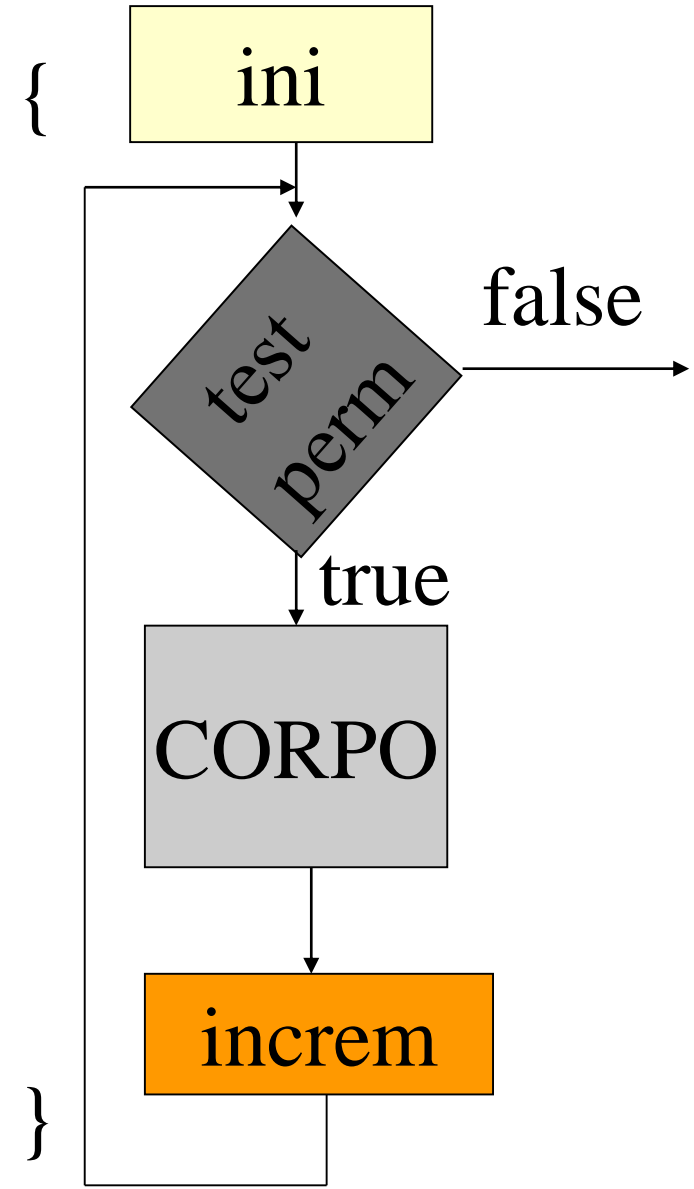


ogni strato è un array `int [4][10]`
immagazzinato in memoria come visto
prima

per scandire gli array è comodo
il comando iterativo for

```
for ( ini ; test-perm ; increm )  
{ CORPO }
```

è equivalente al while
da solo comodità aggiuntiva



//riempimento di un array a 1 dimensione

```
int A[20];
```

```
for(int i=0; i<20; i=i+1)
```

```
    cin >> A[i];
```

//riempimento di array a 2 dimensioni per righe

```
char X[5][10];
```

```
for(int i=0; i<5; i=i+1)
```

```
    for(int j=0; j<10; j=j+1)
```

```
        cin >> X[i][j];
```

e a 3 dimensioni? 3 for annidati e così via

puntatori

testo Sezione 5.1

`int * y;`

dichiara che `y` è di tipo puntatore
ad una variabile intera

`y` è indefinito

così come `x` dopo `int x;`

che R-valore ha un puntatore??

ogni variabile ha un R- ed un L-valore:

```
int x=10;
```

R-valore di x è 10

L-valore di x è l'indirizzo di memoria dove c'è 10 ed è il risultato di &x

```
cout << x << " " << &x << endl;
```

```
int x=10;
```

```
int *p = &x;
```

L-valore di x= 3458



```
cout<< *p;
```

stampa 10 che è l'oggetto puntato da p

ATTENZIONE

il simbolo ***** ha 2 significati diversi a seconda del contesto in cui appare

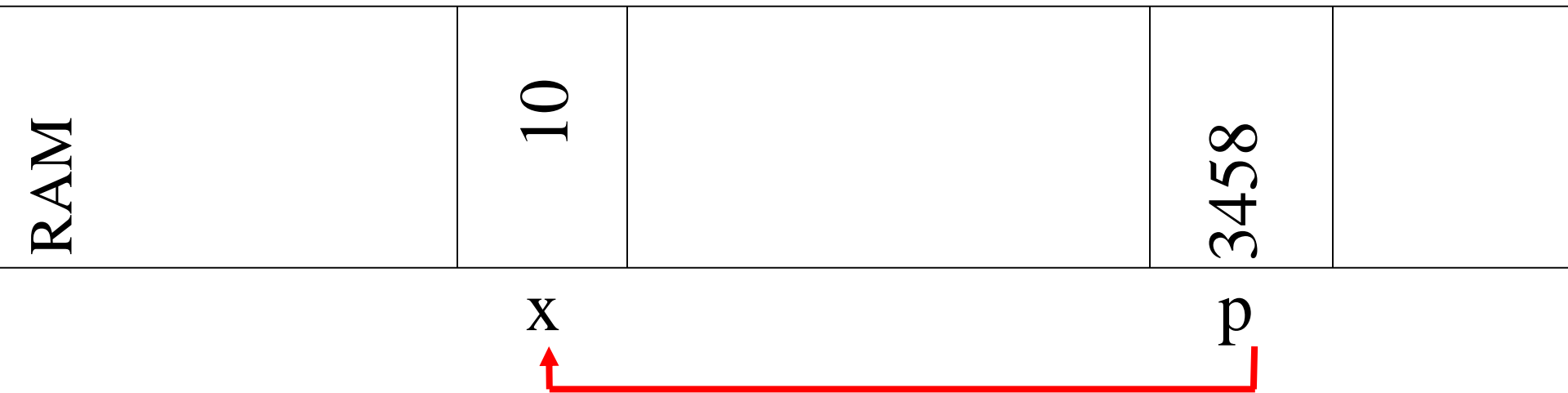
in una DICHIARAZIONE:

char ***** x; puntatore **a char**

in una ESPRESSIONE:

* X=...	}	dereferenziazione
= ... * X...		

L-valore di x= 3458



`int *y= *p; // errore di tipo`

`int *y=p; //OK`

altro esempio:

```
char c = 'h';
```

```
char *p = &c, w = *p;
```

c

'h'

p

w

'h'

,

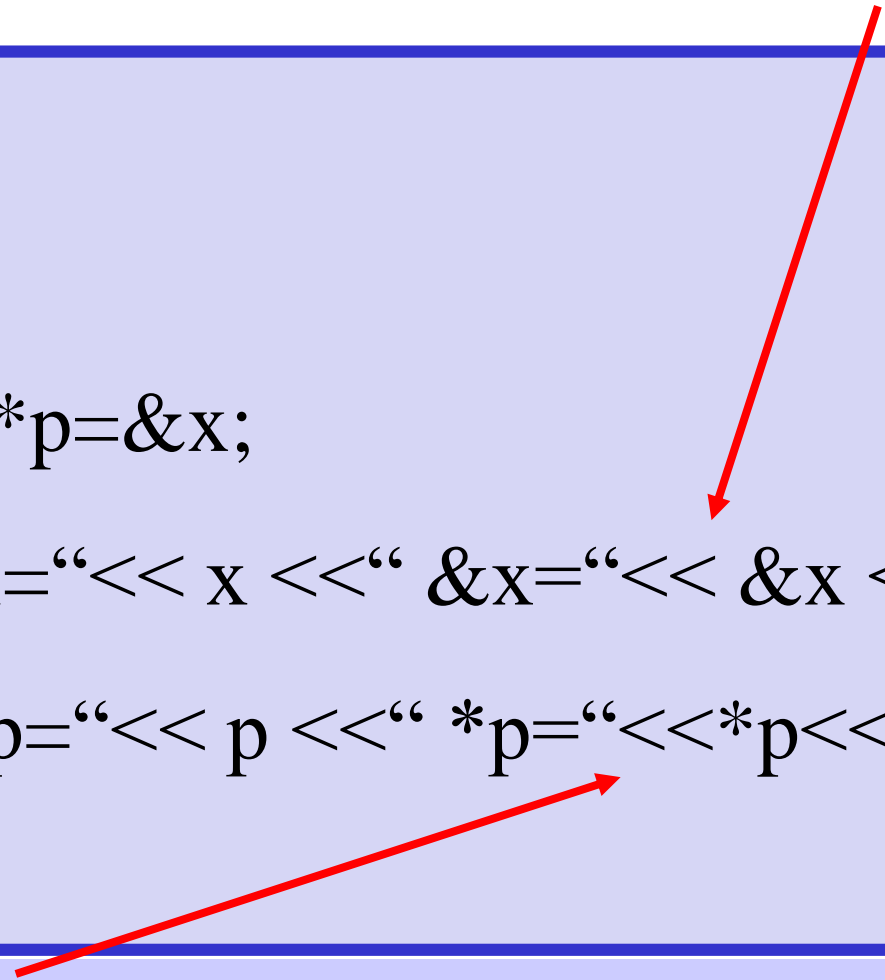
*p indica
l'oggetto puntato
da p cioè c, è
come: char w=c;

w viene inizializzata con valore 'h'

esempio:

stampiamo indirizzi Ram,
in esadecimale

```
main()
{
int x=10, *p=&x;
cout<< "x="<< x <<" &x="<< &x << '\n';
cout<< "p="<< p <<" *p="<<*p<<'\n';
}
```



dereferenziare p, è come avere x

dereferenziare un puntatore significa ottenere
l'oggetto puntato

```
double d=3.14, *pd=&d;
```

```
*pd = *pd + 1.2;
```

The diagram illustrates the L-value and R-value of the variable `d` in the expression `*pd = *pd + 1.2;`. An upward arrow points from the text "L-valore di d" to the first `*pd` (the L-value). A diagonal arrow points from the text "R-valore di d" to the second `*pd` (the R-value).

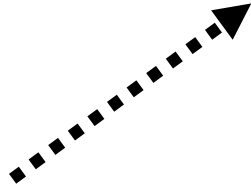
L-valore di d R-valore di d

```
cout<< d; // cosa stampa ??
```

per stampare un puntatore in base 10

```
int x, *p = &x;
```

```
cout << "p=" << (int) p << endl;
```



cast alla C = richiesta di conversione

cast = conversione = PERICOLO

C++ ha cast migliori che vedremo nel Cap. 9

altra possibile insidia

```
int x, *p = &x;
```

```
cout<<"p="<< p << "\n";
```

```
int y=*p;    // Errore !   x è indefinita
```

`int *p;` p ha R-valore indefinito, come
distinguerlo da un indirizzo buono?

BUONA PRATICA: `int *p=0;`

sfruttando che `0 == false` e `(non 0) == true`

`if(p)`

....fai qualcosa con p

`else`

...inizializza p

esempio:

```
int x, *p=&x, *q=p;  
p=0;
```

che succede ?

fare il disegno

errori frequenti:

```
int x, *p=x;    // NO x è int e non int *
```

```
int x, *p= &x;  // OK
```

```
float * f = p;    // ERRORE di TIPO  
                // int * assegnato a float*
```

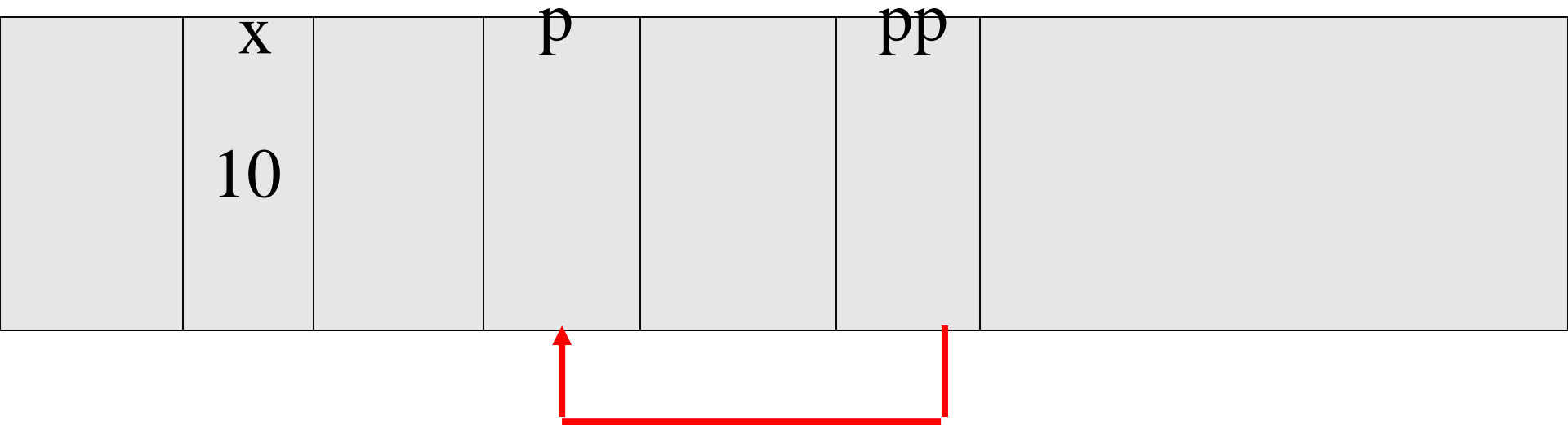
il tipo dell'oggetto puntato è importante

```
int *p;  *p=6; // ERRORE di TIPO
```

puntatori a puntatori a puntatori a....puntatori

```
int x=10, *p, **pp=&p;
```

la situazione è questa:

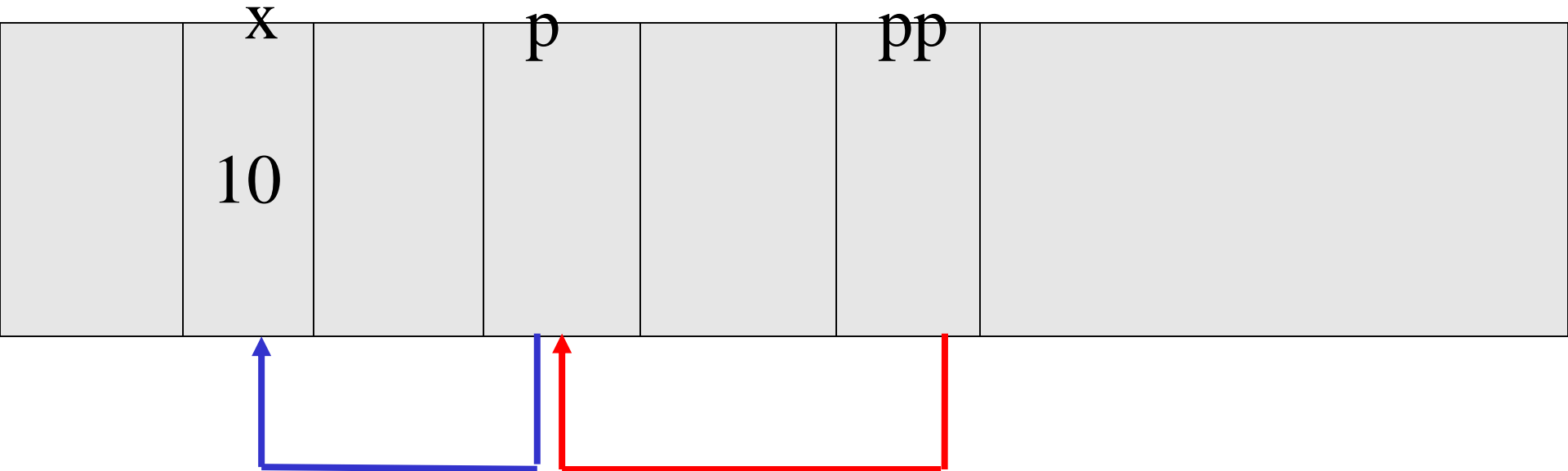


se ora eseguiamo:

```
*pp=&x;
```

se invece scrivessimo

```
p=&x?
```



```
cout<<*p<< **pp ; //stampa ?
```

con lunghe catene di puntatori è facile dimenticare di inizializzare qualche livello

```
int x=10, *p, **p2=&p;
```

```
**p2 = x; // errore *p2 è p che è indefinito  
        // e quindi dereferenziarlo è  
        // ERRORE GRAVE
```

che dire di

```
p2= &&x; ?
```

esercizio

```
int *p, *q, **Q, x=0, y=1;
```

```
p=&y;
```

```
q=&x;
```

```
Q=&q;
```

```
q=p;
```

```
cout<<**Q; ????
```

fare disegno

esercizio

Si consideri il seguente frammento di programma:

```
int x=10, **y;
```

```
*y=&x;
```

```
cout<<**y<<endl;
```

esercizi su letsfeedback.com
login=vzacc

RIFERIMENTI

i riferimenti ci permettono di creare **alias** di variabili

```
int x, &y=x;
```

y è un alias di x

cioè ha lo stesso R- e lo stesso L-valore

i riferimenti non esistono in C

sono introdotti nel C++ per facilitare il passaggio dei parametri alle funzioni

```
int x=2, &y=x;
```

```
cout<< x <<‘ ‘<< &x << ‘ ‘ << y << ‘ ‘ << &y;
```

se I indica l'L-valore di x, stampa:

2 I 2 I

x e y sono variabili con uguale L-valore e quindi
anche uguale R-valore

```
int x, &y=x;
```

come viene realizzato un alias ?

con un puntatore !

in realtà `int &y=x;` definisce un puntatore `int *z = &x;` e ogni volta che scriviamo `y` nel programma il compilatore lo traduce in `*z`

tecnica usata in Java dove tutti puntatori sono nascosti da riferimenti

regole dei riferimenti:

1) va inizializzato subito nella dichiarazione

```
int x, int & y; // NON VA !!!
```

```
y=x;
```

```
int x;
```

```
.....
```

```
int & y=x; // OK
```

2) non si possono definire puntatori a riferimenti:

`int & * x; // NON è C++ e non ha proprio senso`

`int x=1, &y=x; // da questo punto x e y
//sono la stessa`

`x++; y++; cout << x << y; // stampa 3 3`

`int *p=&y; //OK`