

puntatori

testo Sezione 5.1

PUNTANO AD UN
INDIRIZZO IN CUI C'E' UN
VALORE DI UN TIPO

`int *`

è il tipo `puntatore ad un intero`

`int * y;`

`y` è indefinito

così come `x` dopo `int x;`

Come inizializziamo un puntatore??

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

```
int x=10;
```

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



R-valore = 10

L-valore = indirizzo di
memoria dove c'è 10

```
int x=10;
```

```
int *p = &x; //L-valore di x
```

L-valore di x= 3458



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

stampa 10 l'oggetto puntato

ATTENZIONE

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

DICHIARAZIONE:

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

ESPRESSIONE:

$\left. \begin{array}{l} *x = \dots \\ = \dots *x \dots \end{array} \right\} \text{dereferenziazione}$

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'

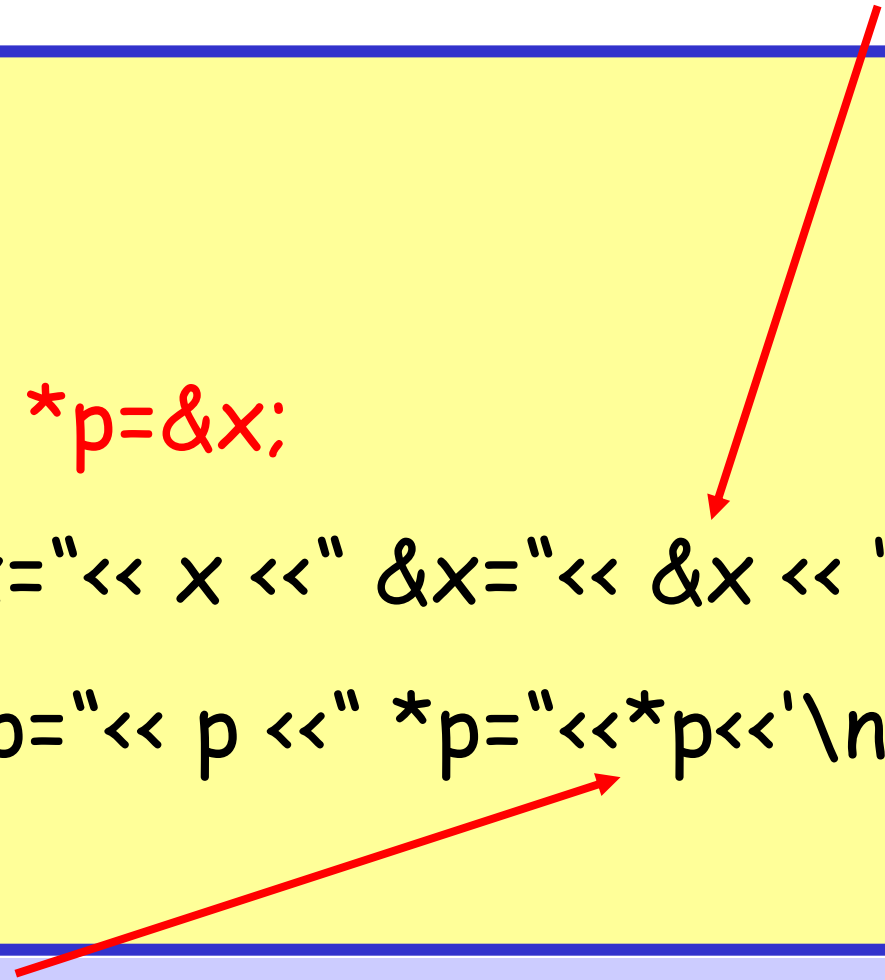
*p indica
l'oggetto
puntato da p
cioè c è come:
char w=c;

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;
```

↑
L-valore di d ← R-valore di d

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

stampare un puntatore in base 10

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

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



cast alla C = richiesta di conversione

cast = 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*`

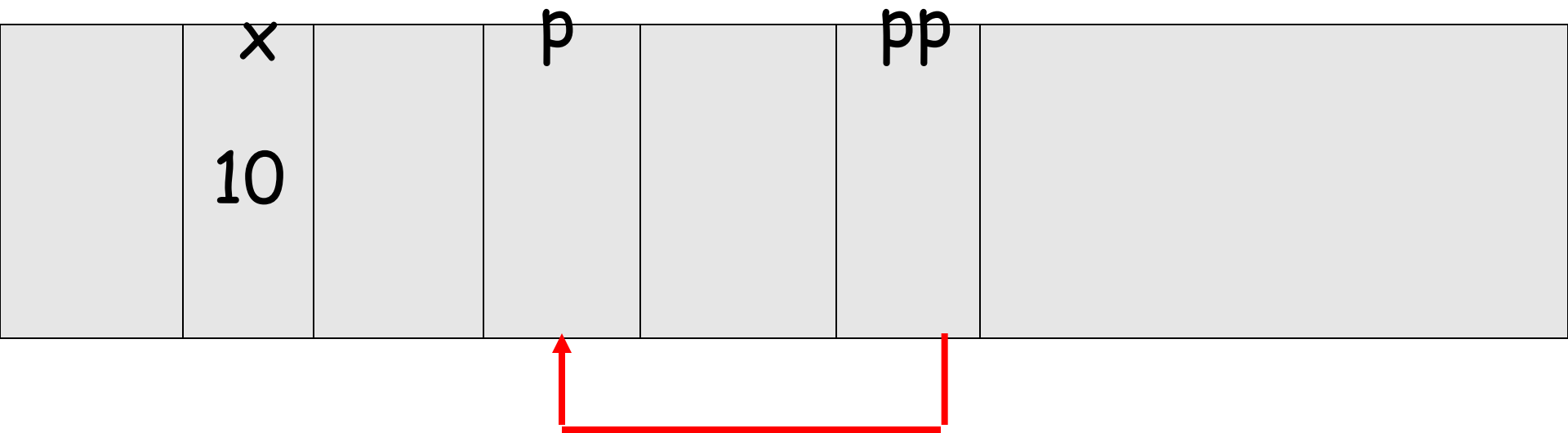
**il tipo dell'oggetto puntato è
importante**

`int *p; *p=6; // p è indefinito non punta a
 // niente`

puntatori a puntatori a puntatori a....puntatori

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

la situazione è questa:

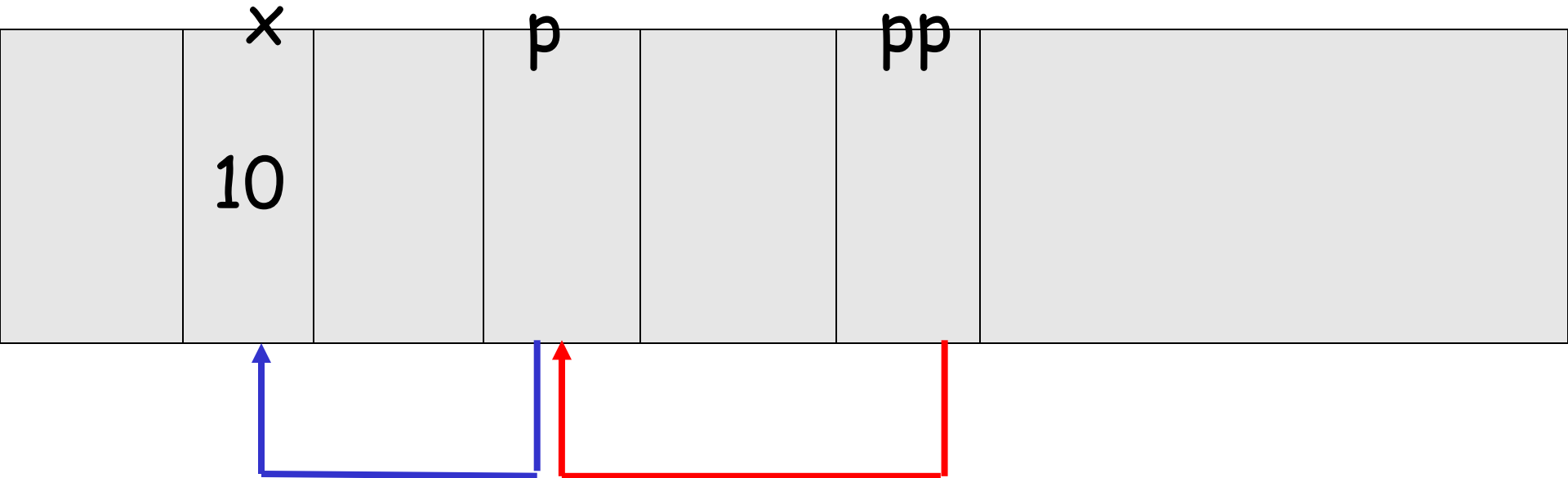


se ora eseguiamo:

```
*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;
```

considerate:

`**Q=x;` sbagliato

`*Q=q;` sbagliato

`Q=&q`

`p=&x;`

`q=p;`

`*Q=&y;`

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

Q	
p	
q	
x	0
y	1

ram

esercizio

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

p=&y;

q=&x;

Q=&q;

q=p;

cout<<**Q; ????

fare disegno

esercizio

Cosa è vero per il seguente frammento di programma?

```
int x=2, y=3, z=4;
```

```
int *p, *q, **P, **Q;
```

```
P=&p ;
```

```
Q=P ;
```

```
*P=q ;
```

```
p=&x ;
```

```
q=&y ;
```

```
**P=z ;
```

```
cout<< *p<<*q<<**P<<**Q ;
```

fare disegno

esercizio

Si consideri il seguente frammento di programma:

```
int x=10, **y;  
*y=&x;  
cout<<**y<<endl;
```

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++
```

```
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
```