

puntatori, array e funzioni

```
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??

```
int x=10;
```

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

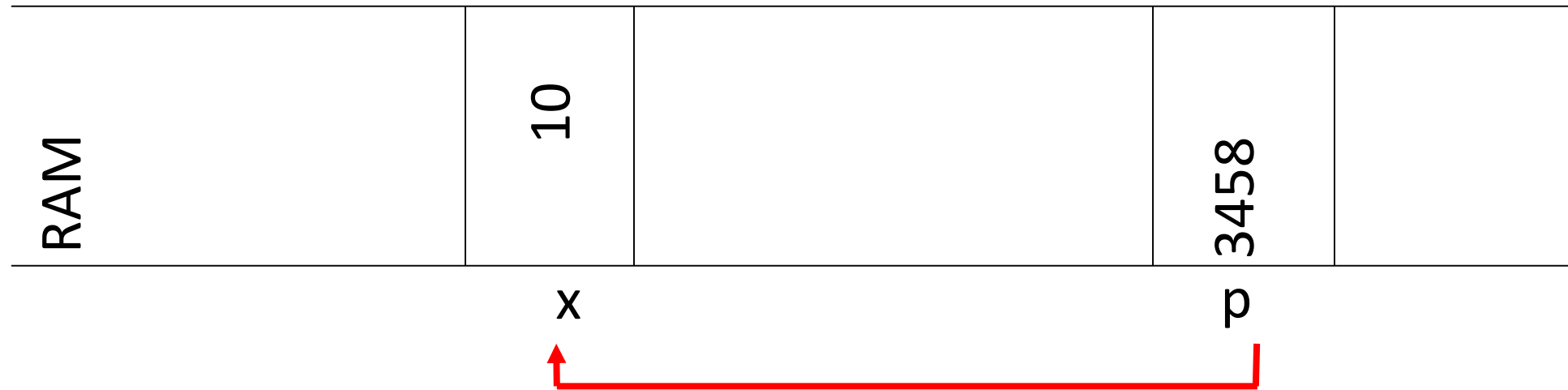
L-valore di x= 3458



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

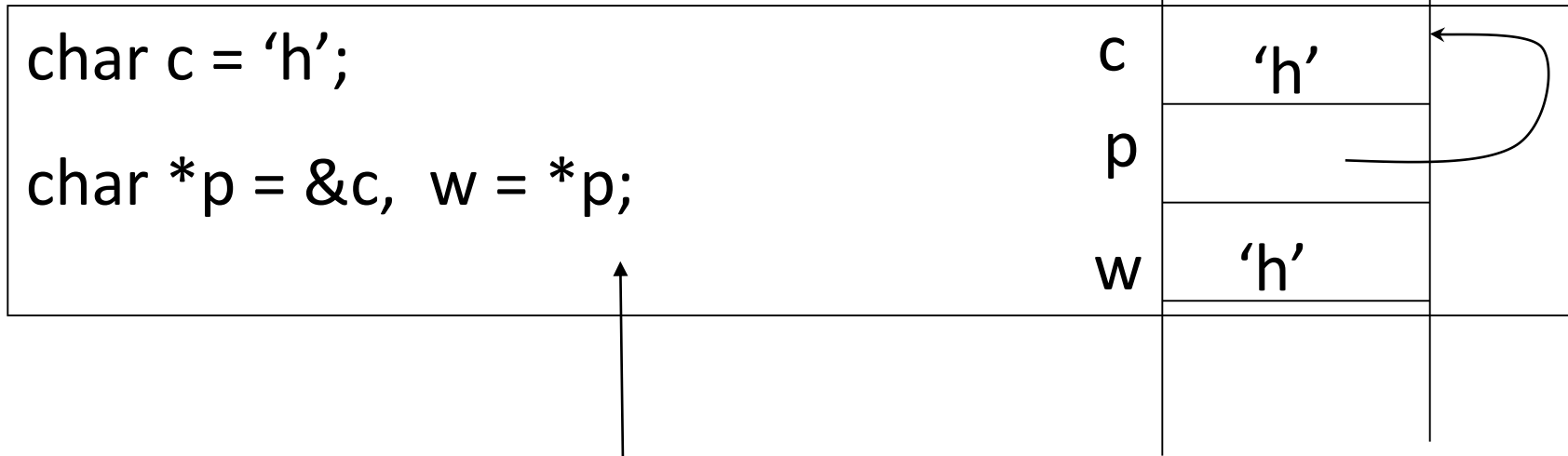
stampa 10 che è l'oggetto puntato da p

L-valore di x= 3458



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

`int *y=p; //OK`

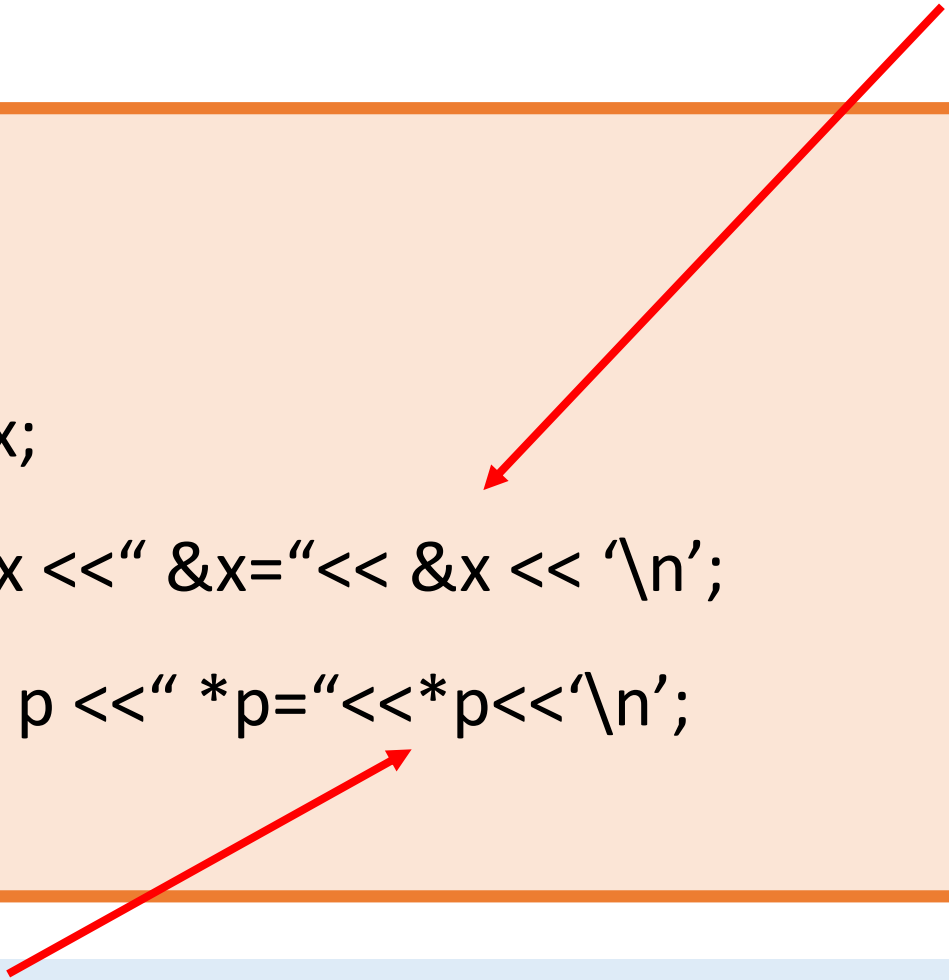


*p indica l'oggetto
puntato da p cioè c,
è come: char w=c;

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; // *pd è d
```



L-valore di d



R-valore di d

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

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

ma abbiamo capito?

```
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

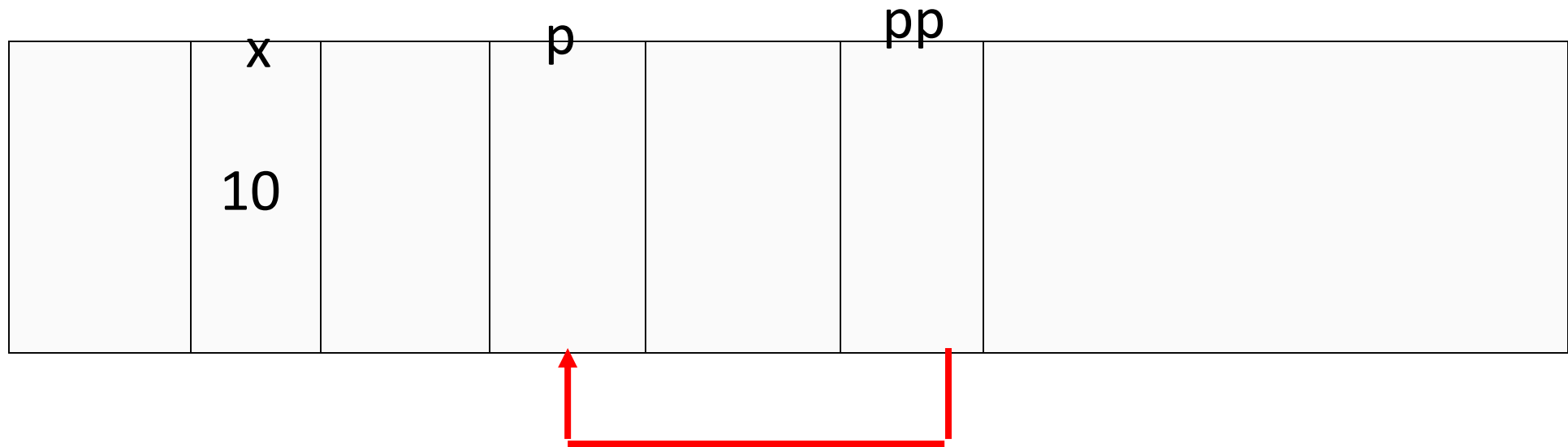
float * f = p; // ERRORE di TIPO
 // int * assegnato a float* il tipo dell'oggetto
 // puntato è importante

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

puntatori a puntatori a puntatori a.....puntatori

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

la situazione è questa:

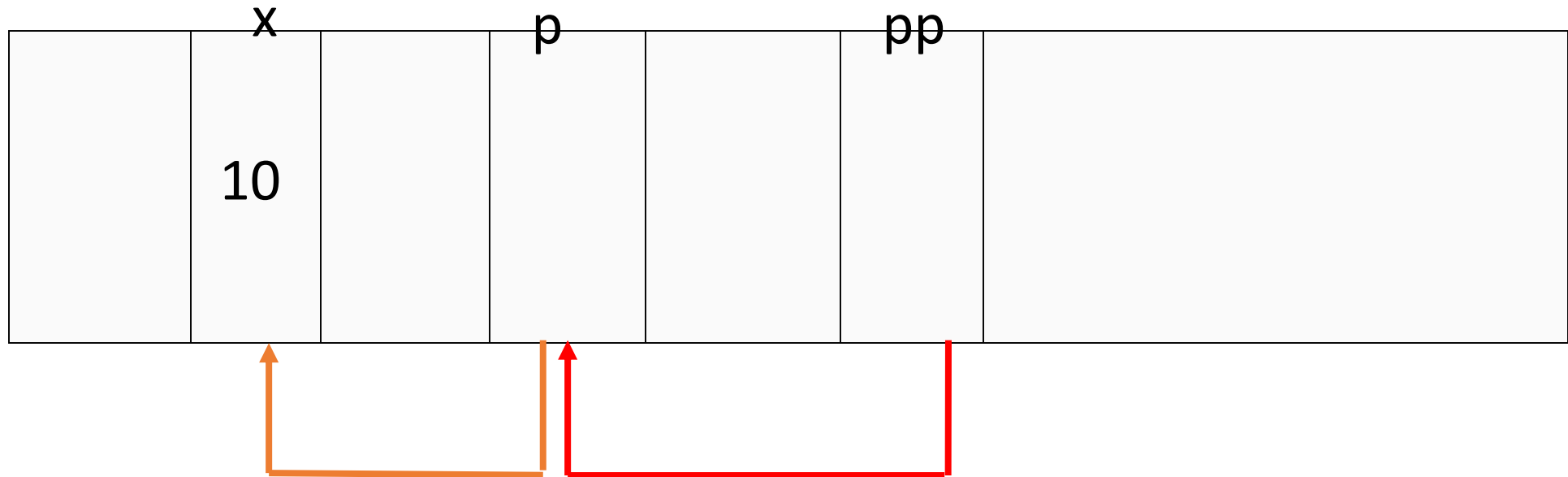


se ora eseguiamo:

```
*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

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


Esercizio

```
int x=0, y=1, *p=&x, **Q;  
*Q=p;  
**Q=x+1;  
cout<<x<<y<<*p<<**Q<<endl;
```

Esercizio

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

il riferimento va inizializzato immediatamente

non va:

```
int & y;
```

.....

```
y=x;
```

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

```
cout<< x <<' '<< &x <<' '<< y <<' '<< &y;
```

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

2 L 2 L

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

i riferimenti non esistono in C

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

```
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

puntatori e array

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

che tipo ha A? char * const ,
insomma è un puntatore costante

```
A=A+1; // ERRORE
```

A punta ad A[0] , quindi *A = A[0]
*(A+1) = A[1]

possiamo dimenticare il const:

```
int* p = A;
```

```
p=p+1; // punta ad A[1]
```

passiamo array alle funzioni:

```
void leggi(int* X, int nelem) // invocaz. leggi(A,20);
```

inutile il const, il passaggio per valore garantisce che A non cambi, ma in leggi, X=X+3; ok, ma A non cambia

quando si passa un array ad una funzione, si passa per valore il puntatore al primo elemento, però la funzione può cambiare il valore degli elementi dell'array

```
void F(int*X){*(X-1)=*(X-2); }
```

```
main(){int A[20]={1,2,3,4}; F(A+3);}
```

le funzioni possono restituire il loro risultato in diversi modi:

- per valore (anche puntatori)
- per riferimento, cioè viene restituita una variabile «intera»

insomma come per i parametri

esempi:

int min (int* A,int dimA) //POST restituisce il valore del min di A

int* minP(int* A, int dimA) // restituisce il puntatore al min di A

int & minR(int* A,int dimA)// restituisce un alias del min di A

int & x = minR(A,30);

1. min per valore

```
int min(int*X, int dimX)
{
    int i=1, min=X[0];
    while(i < dimX)
    {
        if(X[i] < min)
            min=X[i];
        i=i+1;
    }
    return min;
}
```

2. min come puntatore

```
int * min(int*X, int dimX)
{
    int i=1, min=0;
    while(i < dimX)
    {
        if(X[i] < X[min])
            min=i;
        i=i+1;
    }
    return &X[i] ;
}
```

3. min come riferimento

```
int & min(int*X, int dimX)
{
    int i=1, min=0;
    while(i < dimX)
    {
        if(X[i] < X[min])
            min=i;
        i=i+1;
    }
    return X[i] ;
}
```

min come riferimento → ATTENZIONE

```
int & min(int*X, int dimX)
{
    int i=1, min=X[0];
    while(i < dimX)
    {
        if(X[i] < min)
            min=X[i];
        i=i+1;
    }
    return min;
}
```

dangling reference
riferimento
«penzolante»

min viene deallocata
al ritorno della
funzione

puntatore a min → STESSO PERICOLO

```
int * min(int*X, int dimX)
{
    int i=1, min=X[0];
    while(i < dimX)
    {
        if(X[i] < min)
            min=X[i];
        i=i+1;
    }
    return &min;
}
```

dangling POINTER