Indirizzi e puntatori

Andrea Marin

Università Ca' Foscari Venezia
Laurea in Informatica
Corso di Programmazione

a.a. 2012/2013

Richiami sulle variabili

- Una variabile è costituita da una o più locazioni di memoria (a seconda del tipo)
  - `int a;`: 2 Byte (tipicamente)
  - `float b;`: 4 Byte
  - `char str[100];` : 100 Byte
- Nella macchina virtuale ogni locazione di memoria è individuata da un indirizzo, cioè un numero progressivo che la identifica
- Intuitivamente, l'accesso ad una variabile potrebbe avvenire in due modi:
  - Tramite il nome
  - Tramite l'indirizzo

Ottenere l'indirizzo di una variabile

- Per ottenere l'indirizzo di una variabile si usa l'operatore `&`
- Esempio:
  - Consideriamo la dichiarazione `int a`
  - L'indirizzo del primo Byte allocato per la variabile `a` può essere ottenuto mediante l'espressione `&a`
  - Il tipo dell'espressione `&a` è `int*`
- Altro esempio:
  - `float c;`
  - Il tipo dell'espressione `&c` è `float*`
  - `type*` è un tipo che si può leggere come *indirizzo di una variabile di tipo* `type`

Puntatori

Definition (Puntatore)

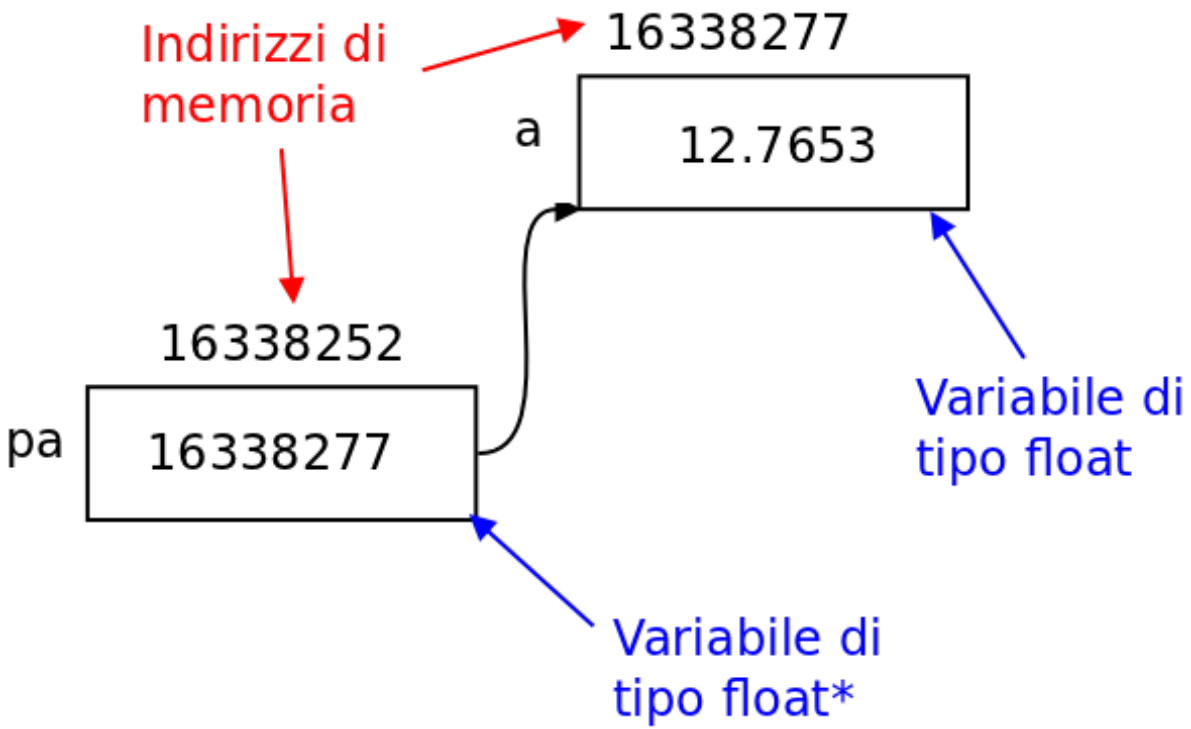
Un puntatore in C è una variabile il cui valore è un indirizzo di memoria.

- La dichiarazione di un puntatore `pa` ad un tipo `type`:
  - `type *pa;`
- Esempio:

```
int *pa;
int a;

a = 10;
pa = &a;
/* Il valore di pa e l'indirizzo del primo */
/* byte riservato per la variabile a*/
```

Esempio: visione grafica



Operazione di dereferenziazione

- Diciamo che un puntatore referenzia una locazione di memoria (*a pointer references a memory location*)
- L'operazione di dereferenziazione consiste nell'accedere alla cella di memoria referenziata
- Nell'esempio precedente la dereferenziazione del puntatore `pa` è la cella in cui è allocata la variabile chiamata `a` perchè il valore della variabile `pa` è l'indirizzo di `a`
- Quando si dereferenzia un puntatore di tipo `type*` si ottiene una variabile di tupo `type`

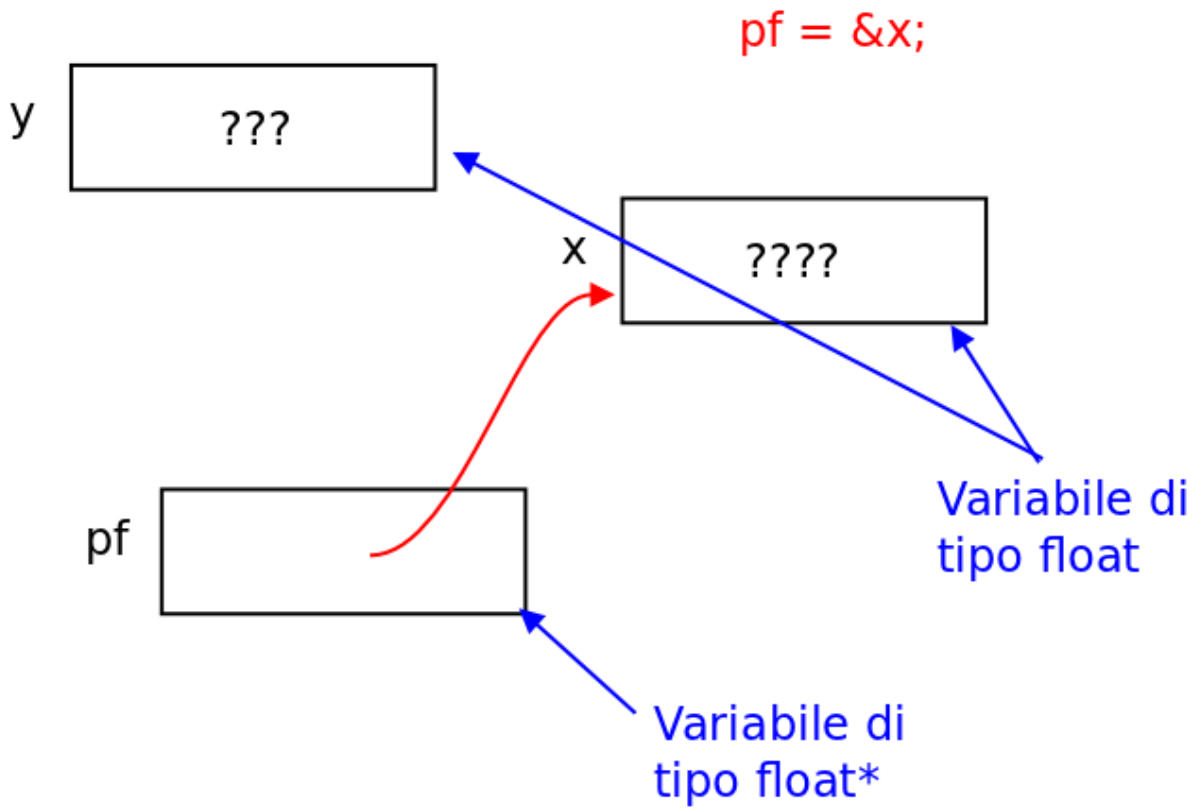
Come si effettua la dereferenziazione

Se pa è di tipo type* la dereferenziazione si ottiene con *pa che indica la locazione di memoria di tipo type referenziata dal puntatore pa

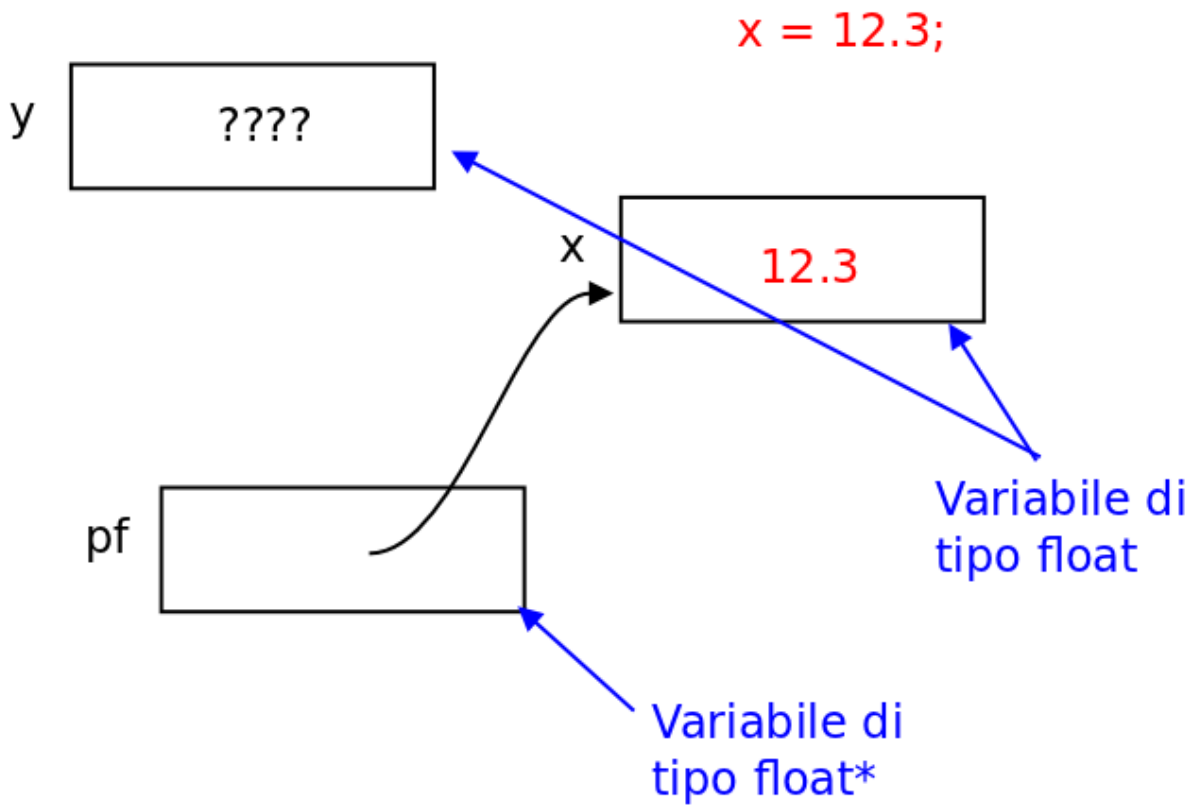
```
float x;
float y;
float *pf;

int main() {
    pf = &x;
    x = 12.3;
    y = (*pf) + 1;
    y = y + 5;
    *pf = (*pf) + 1.2;
    printf(``%f %f %f %p'', x, y, *pf, pf);
    return 0;
}
```

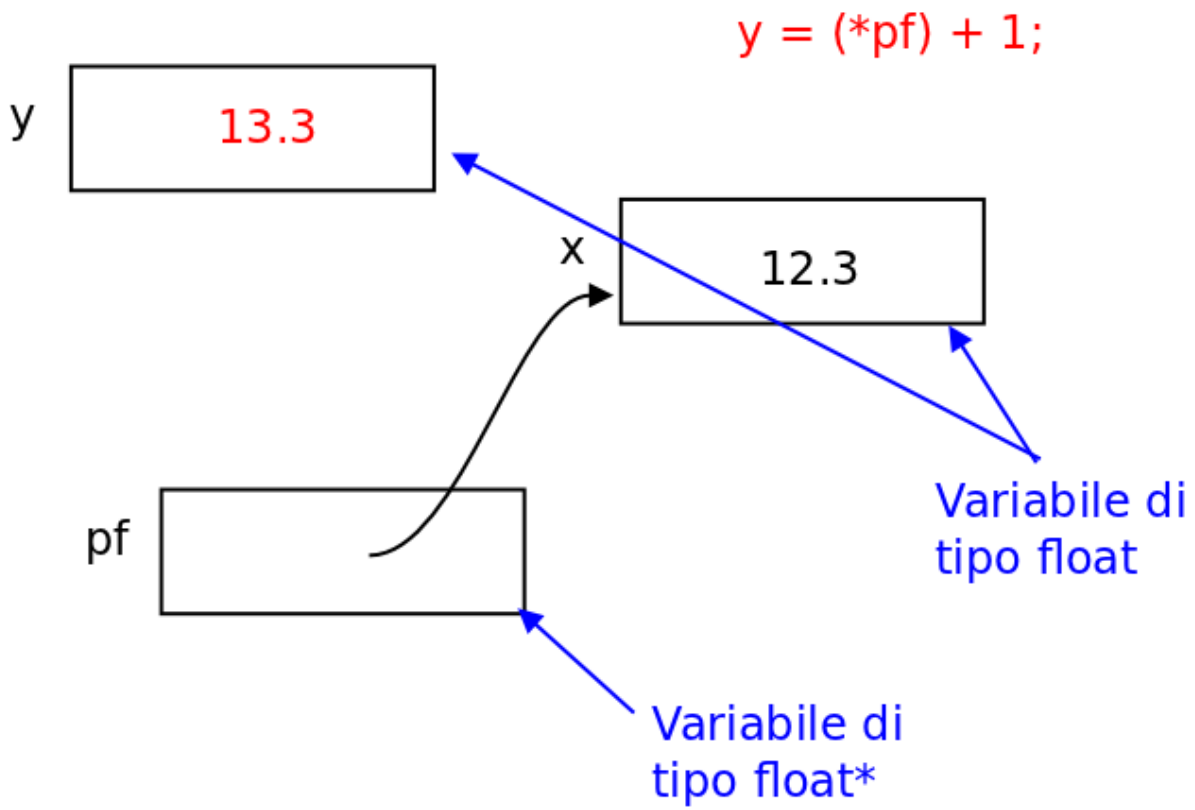
Esempio/1



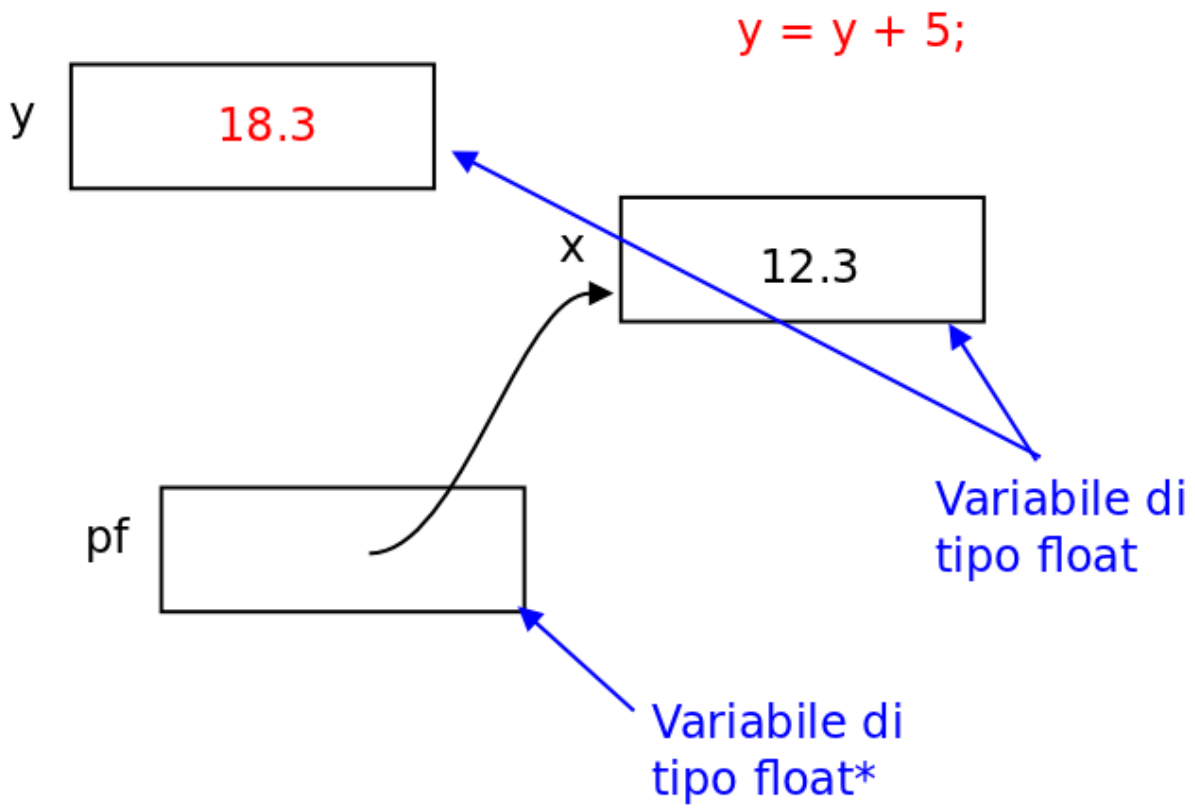
Esempio/2


x = 12.3;
y
????
x
12.3
pf
Variabile di
tipo float
Variabile di
tipo float*

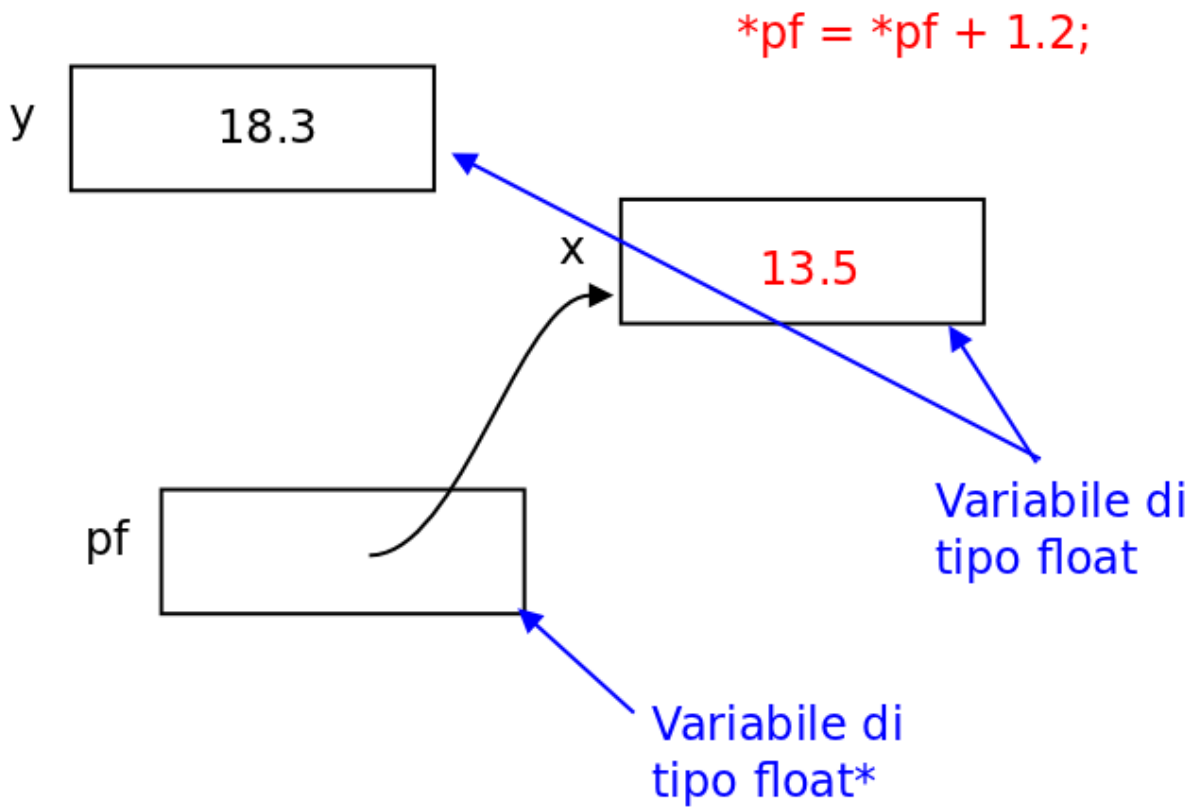
Esempio/3


y = (*pf) + 1;
y
13.3
x
12.3
pf
Variabile di
tipo float
Variabile di
tipo float*

Esempio/4

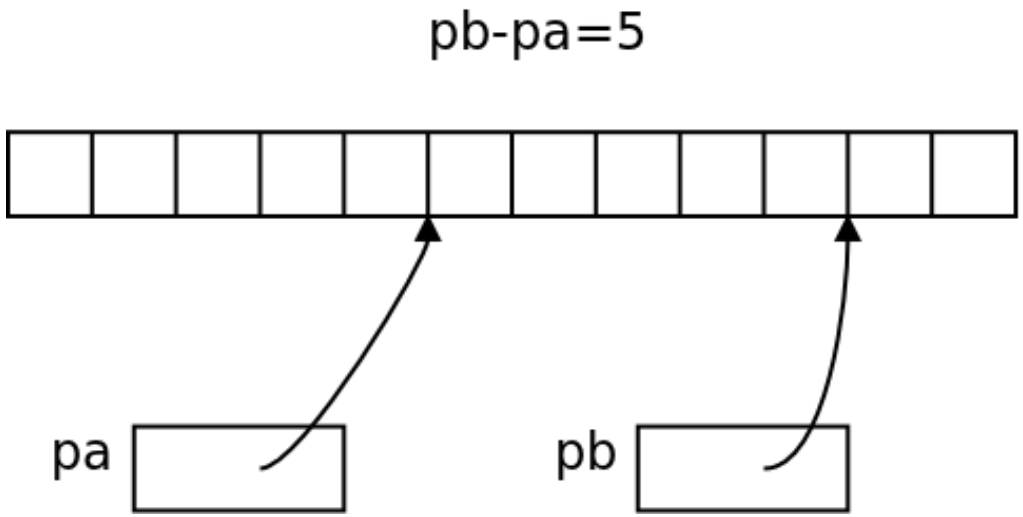

y = y + 5;
y
18.3
x
12.3
pf
Variabile di
tipo float
Variabile di
tipo float*

Esempio/5


*pf = *pf + 1.2;
y
18.3
x
13.5
pf
Variabile di
tipo float
Variabile di
tipo float*

Aritmetica dei puntatori: differenza

- Siano `pa` e `pb` due puntatori di tipo `type*`
- `pb - pa` è un'espressione che indica quanti oggetti di tipo `type` sono possono essere compresi tra l'indirizzo memorizzato in `pa` e quello memorizzato in `pb`
  - Il risultato può essere anche negativo



Aritmetica dei puntatori: confronti

- Siano `pa` e `pb` due puntatori di tipo `type*`
- `pa == pb` è un'espressione che codifica `true` se i puntatori `pa` e `pb` contengono lo stesso indirizzo, `false` altrimenti
- `pa > pb` è un'espressione che codifica `true` se l'indirizzo contenuto in `pa` è maggiore di quello contenuto in `pb`

Attenzione!

```
int a, b, c;
int *pa, *pb, *pc, *pd;
..
a = 7; c = 7;
pa = &a; pc = &c; pd = pc;
```

- `pa == pc` $\Rightarrow$ `false`
- `*pa == *pc` $\Rightarrow$ `true`
- `pd == pc` $\Rightarrow$ `true`

Aritmetica dei puntatori: somme/sottrazioni con `int`

- Sia `pa` un puntatore di tipo `type*`
- Sia `exp` un'espressione di tipo `int`
- Allora `pa + exp` è un'espressione di tipo `type*` il cui valore è l'indirizzo di memoria contenuto in `pa` aumentato di un numero di dimensioni di `type` pari al valore di `exp`
- Il meccanismo è molto utile per la gestione degli array

Esempio d'uso di aritmetica dei puntatori per la gestione degli array

```
float vect[DIM];

int main(){
   /*Leggi vect*/

   float* pc;
/*Pc scorre l'array dal primo elemento il
cui indirizzo e &vect[0] fino all'ultimo il
cui indirizzo e &vect[0]+DIM */

   for (pc = &vect[0]; pc < &vect[0] + DIM; pc = pc +1)
      *pc = (*pc) / 2.0;
   }
..
}
```