

Fondamenti di Informatica

Ingegneria Clinica

Lezione 06-09/12/2010



Raffaele Nicolussi

FUB - Fondazione Ugo Bordoni

Via del Policlinico, 147 – 00161 Roma

I puntatori

Variabili e parametri

- **Parametri formali**
 - Lista di argomenti nella definizione della funzione
 - Scope: Visibili solo nella funzione
 - Ciclo di vita: attivazione della funzione
- **Parametri attuali**
 - Variabili usate per la chiamata della funzione
 - Scope: Visibili nel blocco di definizione
 - Ciclo di vita: attivazione del blocco di definizione
- **Variabili locali**
 - Variabili definite in una funzione o in un blocco
 - Scope: Visibili nel blocco di definizione e in quelli contenuti a meno di mascheramenti
 - Ciclo di vita: attivazione del blocco di definizione

**Le variabili definite in una funzione non sono visibili nelle altre altre funzioni del programma
le funzioni sono blocchi paralleli e non nidificati**

Variabili globali

- Le variabili che si dichiarano **fuori dal main e dalle funzioni del programma** sono dette **variabili globali**
 - sono utilizzabili **da tutte le funzioni definite dopo la loro dichiarazione**
- Le variabili dichiarate all'interno dei blocchi di funzione sono utilizzabili all'interno del blocco stesso
 - sono dette **variabili locali**
- **Attenzione: globali sono anche i nomi delle funzioni**
- Le variabili globali sono visibili in tutto il programma
 - A meno di mascheramenti (ossia ridefinizione in un altro blocco)
 - I nomi di funzione non possono essere utilizzati più volte

```
#include <stdio.h>
```

```
int a=33;
```

```
double duevolte(double);
```

```
int main (void) {
```

```
int b =77;
```

```
printf("a = %d\n", a);
```

```
printf("b = %d\n", b);
```

```
printf("doppio = %lf\n",  
       duevolte(5.0));
```

```
return 0;
```

```
}
```

```
double duevolte(double c) {
```

```
return (2.0 * c* a);
```

```
}
```

```
/* a= 33
```

```
b= 77
```

```
doppio= 330.00 */
```

Variabili e memoria

- Ogni volta che si crea (dichiara) una nuova variabile, **si crea nella memoria un'area in cui il valore della variabile viene immagazzinato.**
 - Una variabile fa riferimento per mezzo di un **identificatore** dichiarato dall'utente ad **un'area di memoria allocata (riservata) per essa**
 - **A basso livello una variabile è una zona di memoria**

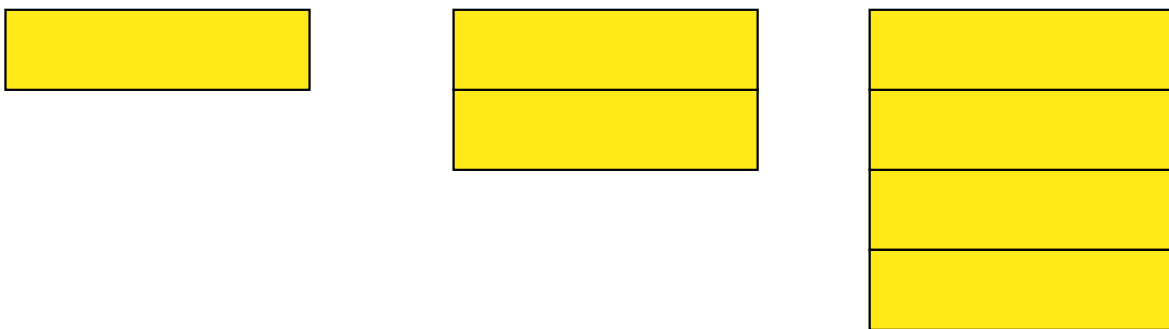
a nome simbolico



I_a indirizzo

Variabili e memoria

- La grandezza (in celle) della zona dipende da **tipo di variabile**
 - Una variabile viene memorizzata in un certo numero di celle (byte) consecutive a partire da una particolare **locazione o indirizzo della memoria della macchina.**
 - **Es.**
 - Le variabili di tipo **char** occupano un solo byte
 - Le variabili di tipo **int** occupano 4 byte



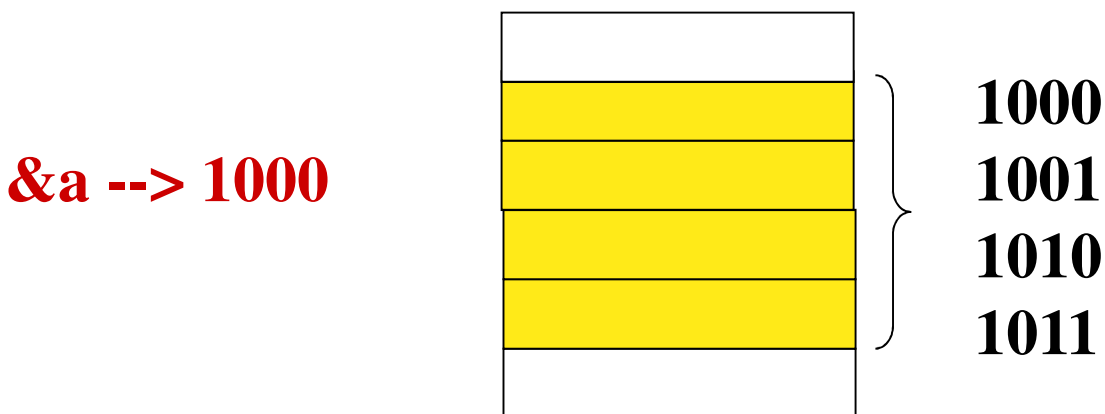
Variabili e memoria

- Una variabile è caratterizzata da:
 - Un **identificatore**: il suo “nome”
 - Un **valore**: quanto vale
 - Un **tipo**: dominio, operatori, funzioni
 - Un **indirizzo**: riferimento area di memoria in cui è memorizzata
 - indirizzo di partenza se occupa più celle
 - Il numero di celle è legato al suo tipo

Indirizzo

- Per sapere l'indirizzo di una variabile si usa l'operatore unario **&**
 - Se **a** è una **variabile**, allora **&a** è il suo **indirizzo** (la prima posizione occupata in memoria dalla variabile)
 - **&a** torna l'indirizzo di **a**
 - valore numerico ma non di tipo numerico
 - ha bisogno di un tipo a parte

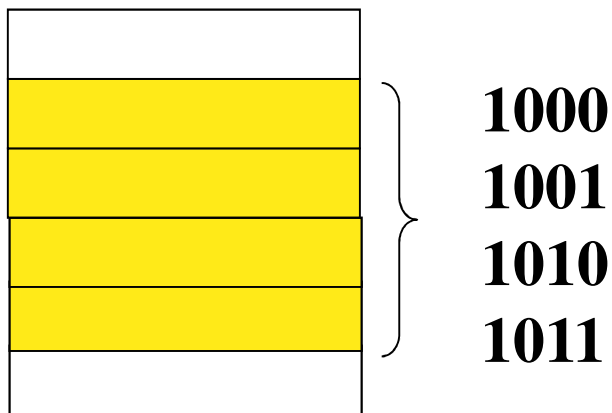
– **&** è detto **operatore di indirizzo**



Dimensione: Numero di byte occupati

- Il numero di byte occupati da una variabile è dato dall'applicazione di **sizeof**.
 - **sizeof(a)** è il numero di byte occupati dalla variabile **a**

&a --> 1000
sizeof(a) --> 4



- **sizeof** può ricevere come argomenti
 - il nome di una variabile
 - un tipo

Indicatori di conversione per la visualizzazione del puntatore

%x %X: intero esadecimale senza segno

%p : valore del puntatore in un formato definito

**/* stampa indirizzo, occupazione di memoria e
valore di alcune variabili */**

```
int main (void) {  
    int a=12;  
    char b='a';  
    float c= 0.1243;  
    printf (“Indirizzo di a e' %x, occupa %d  
            bytes, il suo valore e' %d\n”,  
            &a, sizeof(a), a);  
    printf (“Indirizzo di b e' %x, occupa %d  
            bytes, il suo valore e' %c\n”,  
            &b, sizeof(b), b);  
    printf (“Indirizzo di c e' %x, occupa %d  
            bytes, il suo valore e' %f\n”,  
            &c, sizeof(c), c);  
    return 0;  
}
```

Output

- Indirizzo di **a** è 81cd670,
occupa 4 bytes, il suo valore è
12
- Indirizzo di **b** è 81cd668,
occupa 1 bytes, il suo valore è
a
- Indirizzo di **c** è 81cd66c,
occupa 4 bytes, il suo valore è
0.124300

- **Notare la differenza tra valore e indirizzo di una variabile**
 - **L'indirizzo è dato sottoforma numerica** (ma non di tipo numerico) ed è assegnato dal compilatore
 - **Il valore è assegnato dal programma**
 - **Tutte le variabili dello stesso tipo occupano lo stesso numero di byte**
 - **Hanno la medesima rappresentazione interna**

/* memoria occupata da variabili di un certo tipo */

```
int main (void) {  
    printf (“Il tipo int occupa %d  
        bytes\n”, sizeof(int));  
    printf (“Il tipo char occupa %d  
        bytes\n”, sizeof(char));  
    printf (“Il tipo float occupa %d  
        bytes\n”, sizeof(float));  
    printf (“Il tipo double occupa %d  
        bytes\n”, sizeof(double));  
  
    return 0;  
}
```

Tipo Puntatore: per rappresentare un indirizzo

- L'indirizzo di una **variabile di tipo T** è di tipo **puntatore**

<tipo> *

che viene detto **puntatore a <tipo>**

<tipo> *ptr;

- dove **<tipo>** è il tipo di valore puntato da **ptr**

```
int *ptr;
```

```
int a;
```

```
ptr=&a;
```

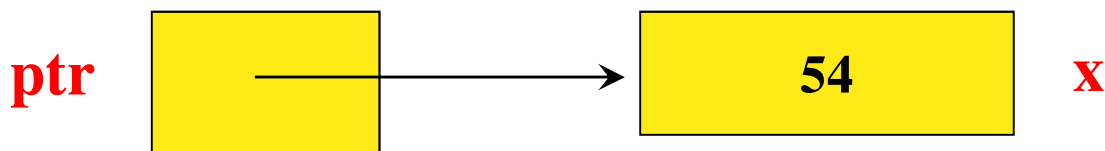
I puntatori

```
int *ptr;  
int x;  
ptr=&x;  
x=54;
```

ptr “puntatore a tipo int” **x** (identificatore)

Indirizzo della cella che
contiene il valore di x

valore di x
(di tipo int)



- La variabile **ptr** PUNTA (fa riferimento) alla variabile il cui identificatore è **x**
 - Ogni identificatore è associato ad una cella di memoria
 - La cella ha un indirizzo, che è la posizione in memoria di quella variabile

I puntatori

- Se **a** è un intero (**int**), il suo indirizzo è di tipo

int *

ossia di tipo **puntatore a int**

- Se **b** è un reale (**float**), il suo indirizzo è di tipo

float *

ossia di tipo **puntatore a float**

int *

float *

sono tipi puntatori

I puntatori

- **int** e **float** indicano il tipo della variabile di cui il puntatore contiene l'indirizzo
- Per **ogni tipo di dato esiste il corrispondente tipo puntatore** (che è un tipo anch'esso)
- Il tipo puntatore è un tipo come gli altri.
- Quindi è possibile dichiarare una **variabile di tipo puntatore a un certo tipo**

Operatore di indirizzo

variabile

indirizzo

cella

j

2489ef

1234

- Al contenuto della cella si può accedere tramite
 - il **nome della variabile** (*indirizzo simbolico*) (j)
 - l'**indirizzo della cella** (*indirizzo fisico*) (**2489ef**), ossia &j
- Il contenuto del puntatore è l'indirizzo della cella
- Si noti che **non** si può assegnare un indirizzo usando l'operatore di indirizzo

&x = 1000; NO! ILLEGALE!

Dichiarazione di puntatori

- Le **variabili puntatore** possono essere dichiarate nei programmi e utilizzate per assumere **indirizzi come valori**.
- Sintassi della dichiarazione di un puntatore

```
<tipo> *<nome>;
```

```
long *ptr;
```

```
int *pi;
```

```
char *pc;
```

Operatore di indirazione *

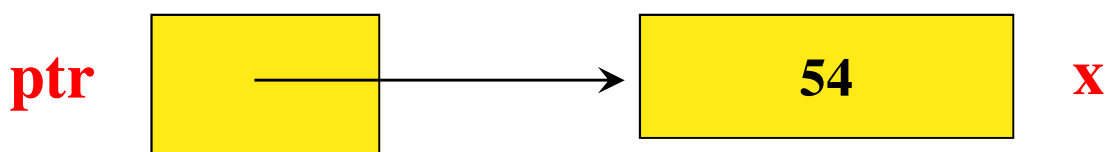
*

(asterisco)

(o op. di **deferimento** o **risoluzione del deferimento**)

Si applica ad un puntatore

- Ritorna il **valore** della cella il cui **indirizzo** è contenuto in una variabile puntatore
- Se **ptr** è una variabile di tipo puntatore
 - ***ptr** restituisce il valore della locazione a cui punta **ptr**



$\&x=ptr=$ indirizzo di x

$*ptr=x=54$

Operatore di indirazione (*)

- Può quindi essere usato in due modi distinti
 - Per **definire** un puntatore a un tipo
 - Per **estrarre** il valore puntato dal puntatore
- Se **p** è una variabile di tipo **puntatore a intero**, allora ***p** può essere visto come una variabile di tipo intero (perché è il valore puntato da p)
 - *Si può **stampare** il valore di ***p***
 - *Si può usare ***p** all'interno di un'espressione, es. (12 + *p - 2)*
 - *Si possono memorizzare valori in questa variabile (***p=34**)*

```
#include <stdio.h>
```

```
int main( void )
```

```
{
```

```
    char *p_ch;
```

```
    char ch1 = 'A';
```

```
    char ch2;
```

```
    p_ch = &ch1;
```

```
    printf( "Il valore memorizzato in p_ch e'  
            %p\n", p_ch );
```

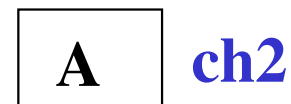
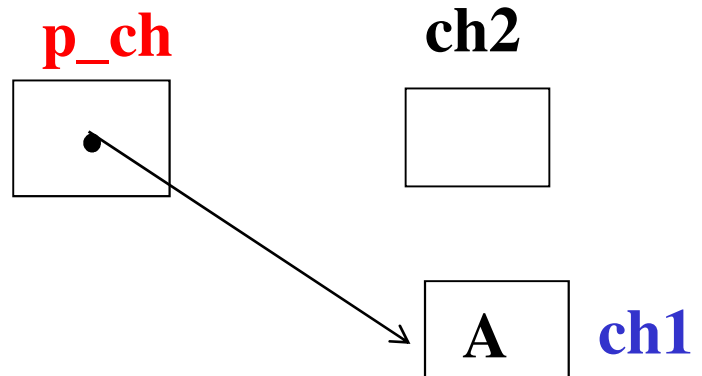
```
    printf( "Il valore di indirezione di p_ch e'  
            %c\n", *p_ch );
```

```
    ch2 = *p_ch;
```

```
    printf( "Il valore di ch2 e' %c\n", ch2 );
```

```
    return 0;
```

```
}
```



Il valore memorizzato in p_ch e' 0x0976fde9

Il valore di indirezione di p_ch e' A

Il valore di ch2 e' A

- **E' un modo contornoso e artificioso di assegnare 'A' a ch1 e ch2.**

char ch1 = 'A';

- **ch1 inizializzata con 'A'**

p_ch = &ch1;

- **viene assegnato a p_ch l'indirizzo di ch1**

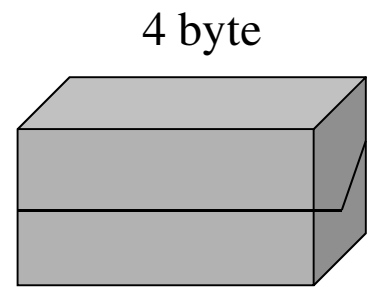
ch2 = *p_ch;

- **Il valore di ch1, ottenuto applicando l'operatore di indirezione al suo puntatore, viene assegnato a ch2**

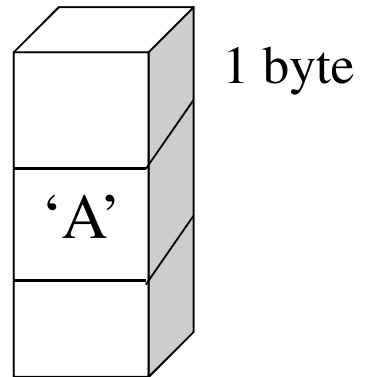
Variabile**Indirizzo****Contenuto**

```
char *p_ch;
char ch1='A', ch2;
```

p_ch 1000
 1004

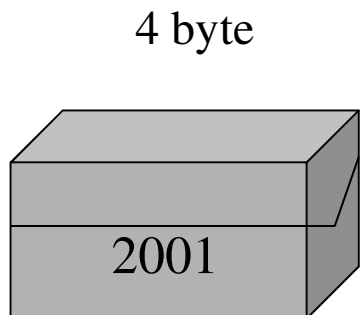


 2000
ch1 2001
ch2 2002

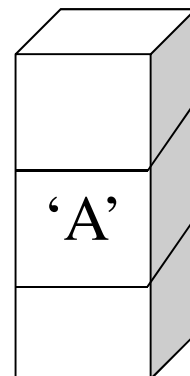


```
p_ch = &ch1;
```

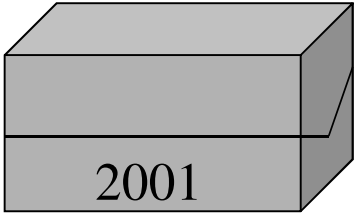
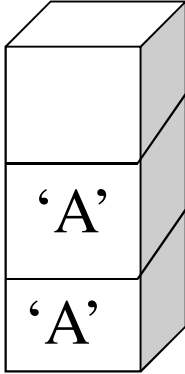
p_ch 1000
 1004



 2000
ch1 2001
ch2 2002



*ch1 e ch2 occupano 1 byte
p_ch occupa 4 byte*

Variabile	Indirizzo	Contenuto
ch2 = *p_ch;	1000	4 byte 
p_ch	1004	
	2000	 1 byte
ch1	2001	
ch2	2002	

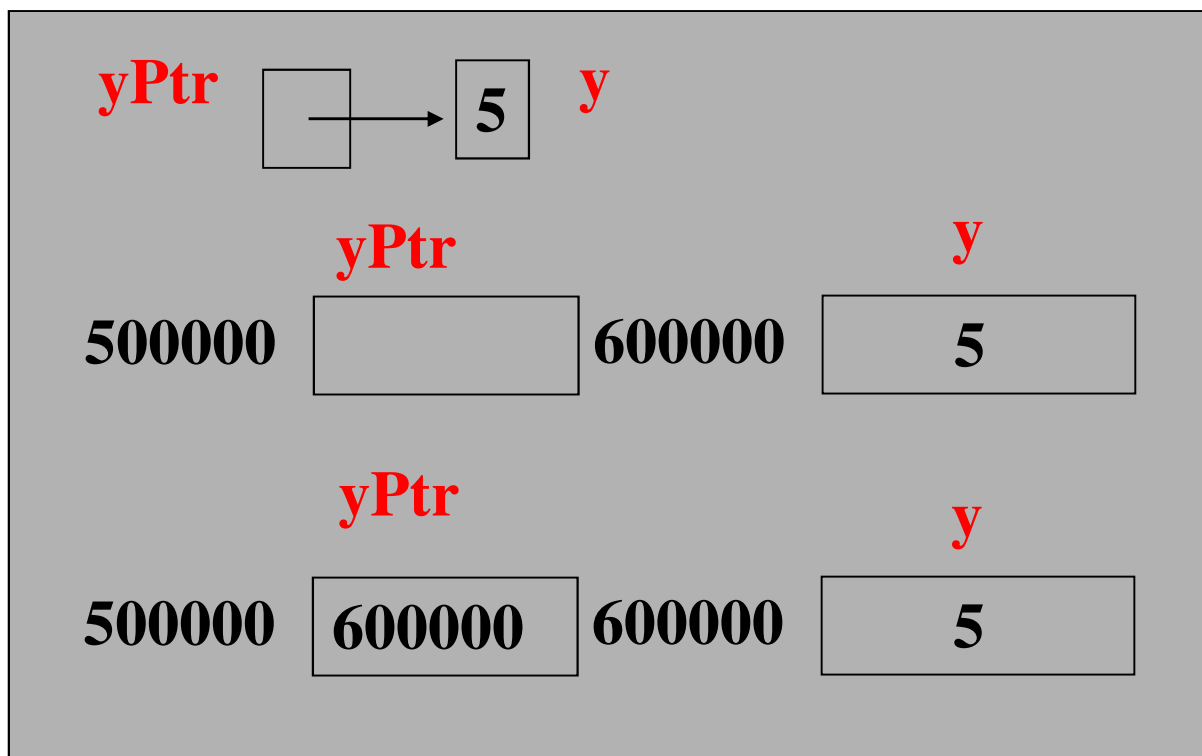
- Dato un puntatore `ptr`, `*ptr` rappresenta il valore della variabile il cui indirizzo è `ptr`.
- Il valore “diretto” di `ptr` è una locazione di memoria
- Il valore “indiretto” di `ptr`, `*ptr`, è il valore della locazione di memoria il cui indirizzo è `ptr`.
- In un certo senso `*` è l’operatore inverso di `&`

```
#include <stdio.h>
int main( void )
{
    int i=7, *p;
    p=&i;
    printf( "Valore di i: %d\n
            Indirizzo di i: %p\n",
            *p, p);
    return 0;
}
```

```
Valore di i:      7
Indirizzo di i:  0x093b1498
```

```
int y = 5;  
int *yPtr;  
yPtr = &y;
```

- assegna al puntatore **yPtr** l'indirizzo di **y**
- Sia **yPtr** che **y** hanno un indirizzo
- L'indirizzo di **y** è proprio **yPtr**



```
int main (void)
{ int a;
  int *aPtr;
  a = 7;
  aPtr = &a;
  printf("Indirizzo di a: %p \nValore di aPtr:
         %p \n\n", &a, aPtr);
  printf("Valore di a: %d \nValore di *aPtr:
         %d \n\n", a, *aPtr);
  printf("Prova che * e & sono
         complementari\n");
  printf("&*aPtr = %p \n*&aPtr = %p\n",
         &*aPtr,*&aPtr);
  return 0;
}
```

Indirizzo di a: 0x0956a3d8

Valore di aPtr: 0x0956a3d8

Valore di a: 7

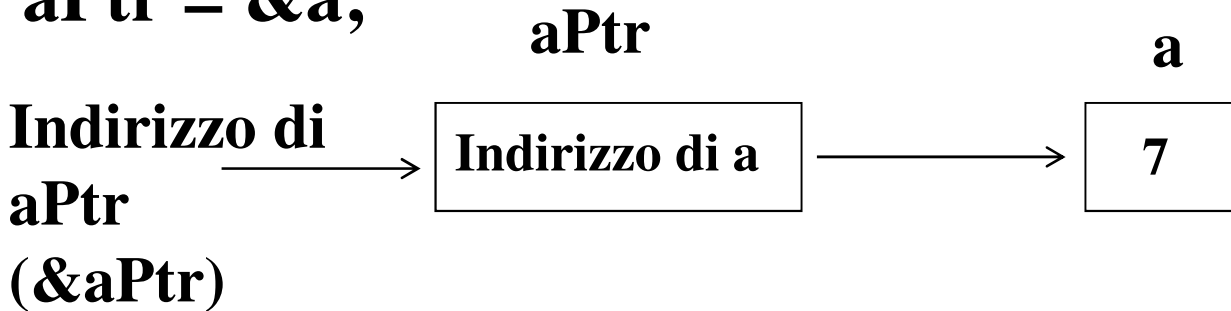
Valore di *aPtr: 7

Prova che * e & sono complementari

&*aPtr = 0x0956a3d8

***&aPtr = 0x0956a3d8**

```
int a;  
int *aPtr;  
a = 7;  
aPtr = &a;
```



***aPtr**: valore di indirezione, ossia 7

&*aPtr: indirizzo del valore di indirezione, ossia quello che è scritto in aPtr

&aPtr: indirizzo di aPtr

***&aPtr**: valore di indirezione di &aPtr, ossia quello che è scritto dentro aPtr

Indirizzo di a: 0x0956a3d8

Valore di aPtr: 0x0956a3d8

Valore di a: 7

Valore di *aPtr: 7

Prova che * e & sono complementari

&*aPtr = 0x0956a3d8

***&aPtr = 0x0956a3d8**

Puntatore nullo

- Puntatore che **non punta ad alcun oggetto valido**
- Gli è stato assegnato o il valore **0** o il valore **NULL**
- **NULL** è una costante simbolica in genere definita in `<stdio.h>`

char *p;

p = 0; oppure p = NULL;

- **Il puntatore nullo è l'unico puntatore che viene valutato FALSO (0)**
 - **In tutti gli altri casi, un puntatore viene sempre valutato vero**
- **Il controllo su puntatore nullo può essere usato nella terminazione di un ciclo**


```
char *p;
```

```
.....
```

```
while (p)
```

```
{ .....
```

```
    /* ripete il ciclo fino a quando p  
    diventa un puntatore nullo
```

```
    */
```

```
.....
```

```
}
```

Puntatore nullo

- Il valore 0 è l'unico valore intero che può essere assegnato direttamente ad una variabile di tipo puntatore
 - I puntatori possono essere inizializzati come una qualsiasi variabile (con valori che corrispondono al suo tipo).
 - il valore di inizializzazione deve essere uno tra i seguenti:
 - un indirizzo
 - 0
 - NULL
- ```
int *p;
p = 0;
p = NULL;
p = &i;
p = (int *) 1776;
```

**int j;**

**int \*ptr\_to\_j = &j;      /ok/**

**int \*ptr\_to\_j = &j;**

**int j;      /errata/**

- **Non è possibile fare riferimento ad una variabile prima che sia dichiarata**

**&3      /errata/**

**&(k+99)      /errata/**

- **no ad una costante**
- **no ad espressioni**

- **E' possibile usare void per definire un tipo da puntare:**

**void \*v;**

- può essere pensato come **puntatore a un tipo generico**
- può rappresentare qualsiasi tipo di puntatore

- **Un puntatore può essere assegnato ad un altro puntatore (se dello stesso tipo o di tipo void)**

▪ **int \*p, \*q; .....**

▪ **p=q;**

- **se non sono dello stesso tipo, si usa l'oper. cast per la conversione**

✓ **int \*pInt;**

✓ **float \*pFloat;**

✓ **pInt = (int \*)pFloat;**

**a = 7;**

**pInt = &a;**

**Indirizzo di a: 0x0956a3e4**

**Valore di pInt (indirizzo di a): 0x0956a3e4**

**Valore di a: 7**

**Indirizzo di pInt: 0x0956a3dc**

**f = 6.78;**

**pFloat = &f;**

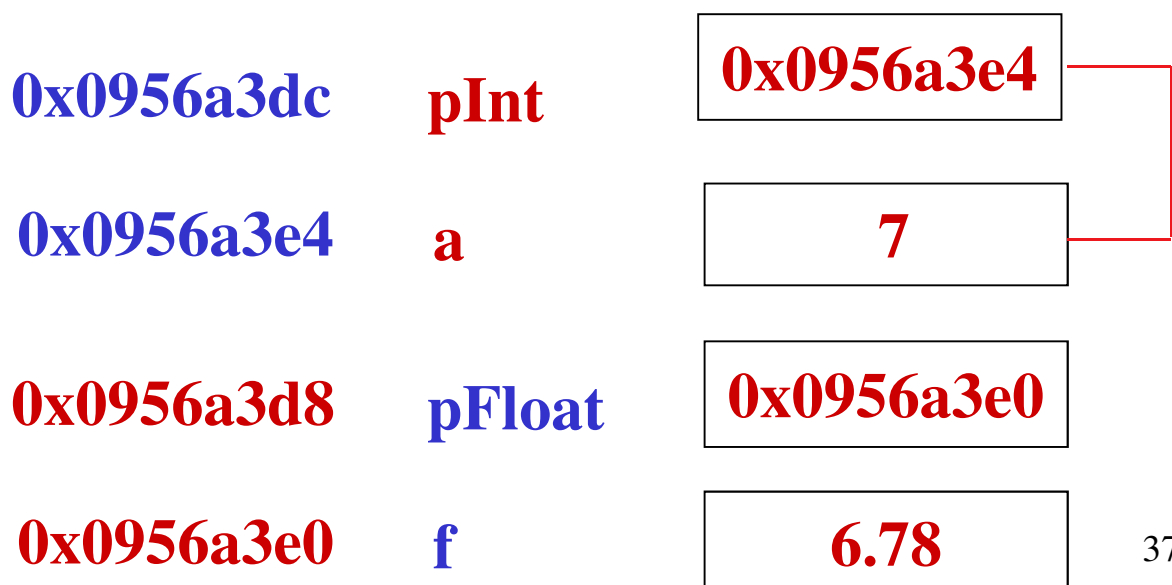
**Indirizzo di f: 0x0956a3e0**

**Valore di pFloat (indirizzo di f):**

**0x0956a3e0**

**Valore di f: 6.780000**

**Indirizzo di pFloat: 0x0956a3d8**

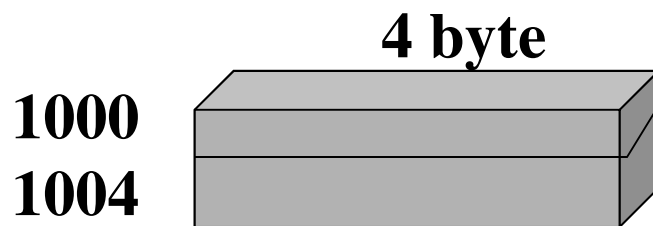


## Aritmetica dei puntatori e dimensione degli elementi

- Si possono svolgere operazioni di addizione (+, +=, ++) e sottrazione (-, -=, --) sui puntatori
- L'assegnamento è valido solo tra puntatori dello stesso tipo
- Se  $p$  è un puntatore  $p + 3$ 
  - indica il terzo oggetto che segue quello puntato da  $p$
  - il risultato è il valore di un indirizzo
- **Attenzione: bisogna tenere conto delle dimensioni dell'oggetto memorizzato**
- Non semplice somma ma
$$p + 3 * n$$
dove  $n$  è il numero di byte usati per memorizzare l'oggetto puntato.

$$p + 3 * 4 = p + 12$$

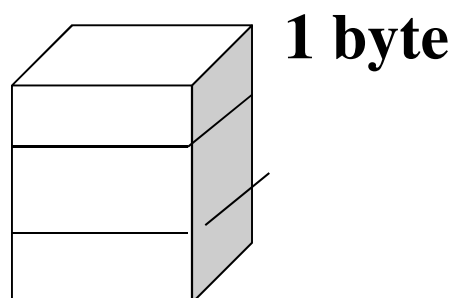
$$p = 1000 \text{ ---> } 1012$$



- Se p fosse dichiarato di tipo char, allora

$$p = 1000$$

$$p = p + 3 \text{ ---> } 1003$$



- **Non si possono sommare puntatori tra di loro**
- **Se i puntatori si riferiscono allo stesso tipo di oggetto, è possibile sottrarre il valore di un puntatore da un altro**
  - **se il primo puntatore è relativo ad un indirizzo minore del secondo, allora il risultato sarà negativo**
    - **risultato** è un valore intero che rappresenta il numero di celle tra i due puntatori
- **È possibile sottrarre un intero da un puntatore**
  - **risultato** è il valore di un puntatore



- **Es.**

**long \*p1, \*p2;**

**int j;**

**char \*p3;**

**p2=p1 + 4;      corretta**

**j = p2-p1;      corretta (j=4);**

**j = p1-p2;      corretta (j=-4)**

**p1=p2-2;      corretta, i tipi dei  
punt. Sono  
compatibili**

**p3 = p1-1;      errata, i tipi sono  
diversi**

**j = p1 - p3;      errata, i tipi sono  
diversi**