

# Laboratorio di Informatica

## Puntatori

(Parte 1)

**docente: Cataldo Musto**

[cataldo.musto@uniba.it](mailto:cataldo.musto@uniba.it)

# Variabili

- Cosa sono?

# Variabili

- Cosa sono?
  - Si definisce «variabile» una particolare **porzione di memoria** destinata a contenere dei valori **acquisiti, elaborati o prodotti da un algoritmo**
- Come descrive una variabile?

# Variabili

- Cosa sono?
  - Si definisce «variabile» una particolare **porzione di memoria** destinata a contenere dei valori **acquisiti, elaborati o prodotti da un algoritmo**
- Come descrive una variabile?
  - **Nome**
  - **Tipo**
  - **Indirizzo di memoria**
  - Nelle variabili tradizionali, i primi due sono **esplicitamente** dichiarati, il terzo è «nascosto»
    - **Es.** `int i = 0;`
    - **Es.** `float a = 8.0;`

# Variabili

Come facciamo a risalire  
all'indirizzo di memoria di una  
variabile?

- Cosa sono?
  - Si definisce «variabile» una particolare **porzione di memoria** destinata a contenere dei valori **acquisiti, elaborati o prodotti da un algoritmo**
- Come descrive una variabile?
  - **Nome**
  - **Tipo**
  - **Indirizzo di memoria**
  - Nelle variabili tradizionali, i primi due sono **esplicitamente** dichiarati, il terzo è «nascosto»
    - **Es.** `int i = 0;`
    - **Es.** `float a = 8.0;`

# Variabili (cont.)

- Come facciamo a risalire **all'indirizzo di memoria** di una variabile?

# Variabili (cont.)

- Come facciamo a risalire **all'indirizzo di memoria** di una variabile?
- Il C mette a disposizione **l'operatore «&» (operatore indirizzo)**
  - Si usa **a fianco del nome di una variabile**
  - **Restituisce l'indirizzo di memoria** di quella variabile

# Variabili (cont.)

- Come facciamo a risalire **all'indirizzo di memoria** di una variabile?
- Il C mette a disposizione **l'operatore «&» (operatore indirizzo)**
  - Si usa **a fianco del nome di una variabile**
  - **Restituisce l'indirizzo di memoria** di quella variabile

```
1  #include <stdio.h>
2
3  int main() {
4      int a = 5; // variabili intere
5      int b = 3;
6
7      // stampro valore e indirizzo
8      printf("a=%d \t\t b=%d \n", a, b);
9      printf("&a=%X \t &b=%X \n", &a, &b);
10 }
```



# Variabili (cont.)

- Come facciamo a risalire **all'indirizzo di memoria** di una variabile?
- Il C mette a disposizione **l'operatore «&» (operatore indirizzo)**
  - Si usa **a fianco del nome di una variabile**
  - **Restituisce l'indirizzo di memoria** di quella variabile

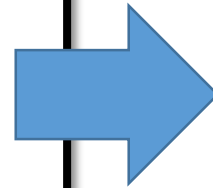
```
1  #include <stdio.h>
2
3  int main() {
4      int a = 5; // variabili intere
5      int b = 3;
6
7      // stampro valore e indirizzo
8      printf("a=%d \t\t b=%d \n", a, b);
9      printf("&a=%X \t &b=%X \n", &a, &b);
10 }
```

# Variabili (cont.)

```
#include <stdio.h>

int main() {
    int a = 5; // variabili intere
    int b = 3;

    // stampo valore e indirizzo
    printf("a=%d \t\t b=%d \n", a, b);
    printf("&a=%X \t &b=%X \n", &a, &b);
}
```

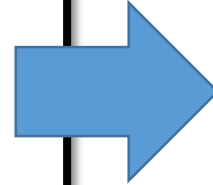


# Variabili (cont.)

```
#include <stdio.h>

int main() {
    int a = 5; // variabili intere
    int b = 3;

    // stampo valore e indirizzo
    printf("a=%d \t\t b=%d \n", a, b);
    printf("&a=%X \t &b=%X \n", &a, &b);
}
```



```
gcc version 4.6.3
a=5          b=3
&a=657D6F6C  &b=657D6F68
```

# Variabili (cont.)

- Quando abbiamo già visto **l'operatore indirizzo**?

# Variabili (cont.)

- Quando abbiamo già visto l'operatore indirizzo?
- Nei parametri della funzione `scanf()`
  - Es. `scanf("%d %d", &n, &m);`
  - In questo caso la **funzione risale all'indirizzo di memoria della variabile e lo usa per memorizzarne il nuovo valore**



# Variabili (cont.)

- Quando abbiamo già visto **l'operatore indirizzo**?
- Nei parametri della funzione `scanf()`
  - Es. `scanf("%d %d", &n, &m);`
  - In questo caso la **funzione risale all'indirizzo di memoria della variabile e lo usa per memorizzarne il nuovo valore**
  - **Il nome della variabile è un «alias» (o un nickname)** per riferirci a un indirizzo di memoria dell'area dati del programma

# Variabili (cont.)

- Quando abbiamo già visto l'operatore indirizzo?
- Nei parametri della funzione `scanf()`
  - Es. `scanf("%d %d", &n, &m);`
  - In questo caso la **funzione risale all'indirizzo di memoria della variabile e lo usa per memorizzarne il nuovo valore**
  - **Il nome della variabile è un «alias» (o un nickname) per riferirci a un indirizzo di memoria dell'area dati del programma**
  - **L'operatore indirizzo (insieme all'operatore di indirizzamento, che sarà introdotto successivamente) è di fondamentale importanza per l'uso dei puntatori**

# Puntatori

- Cosa sono?

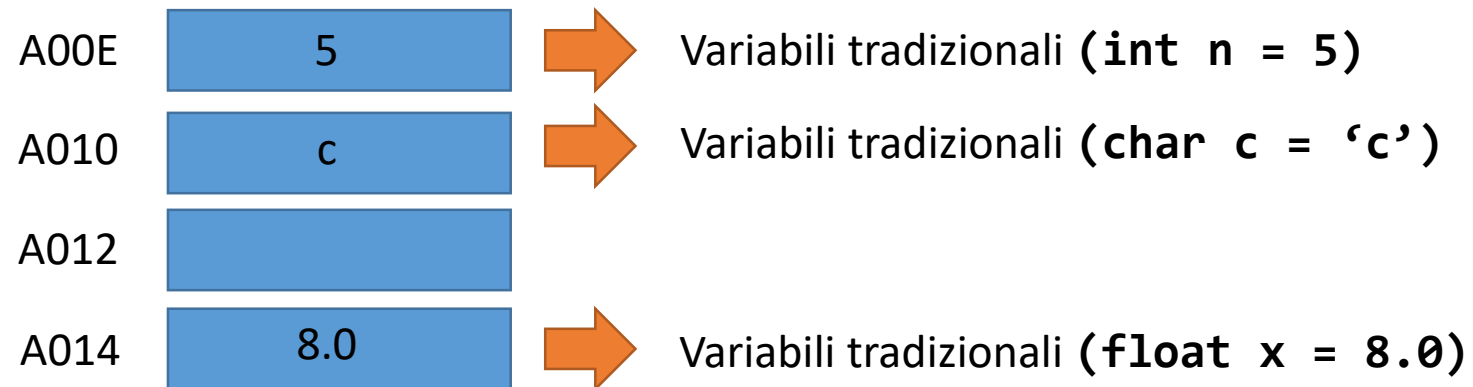


# Puntatori

- Cosa sono?
  - Una particolare tipologia di variabili
  - **Dov'è la particolarità?** Possono memorizzare solo **indirizzi di memoria**.
    - Normalmente le variabili contengono un **valore compatibile col tipo della variabile**.
    - Le variabili di tipo puntatore contengono **un indirizzo di memoria**

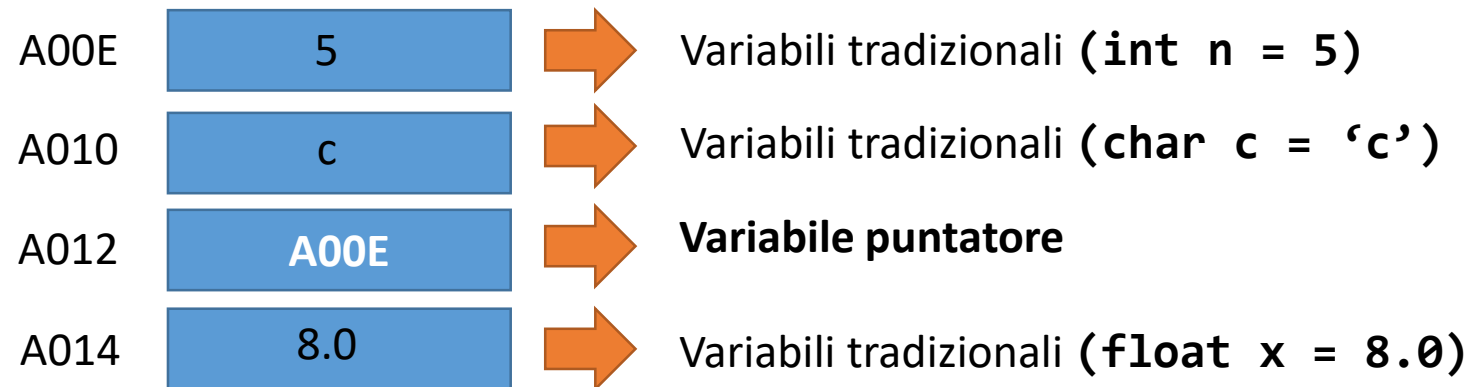
# Puntatori

- Cosa sono?
  - Una particolare tipologia di variabili
  - **Dov'è la particolarità?** Possono memorizzare solo **indirizzi di memoria**.
    - Normalmente le variabili contengono un **valore compatibile col tipo della variabile**.
    - Le variabili di tipo puntatore contengono un **indirizzo di memoria**

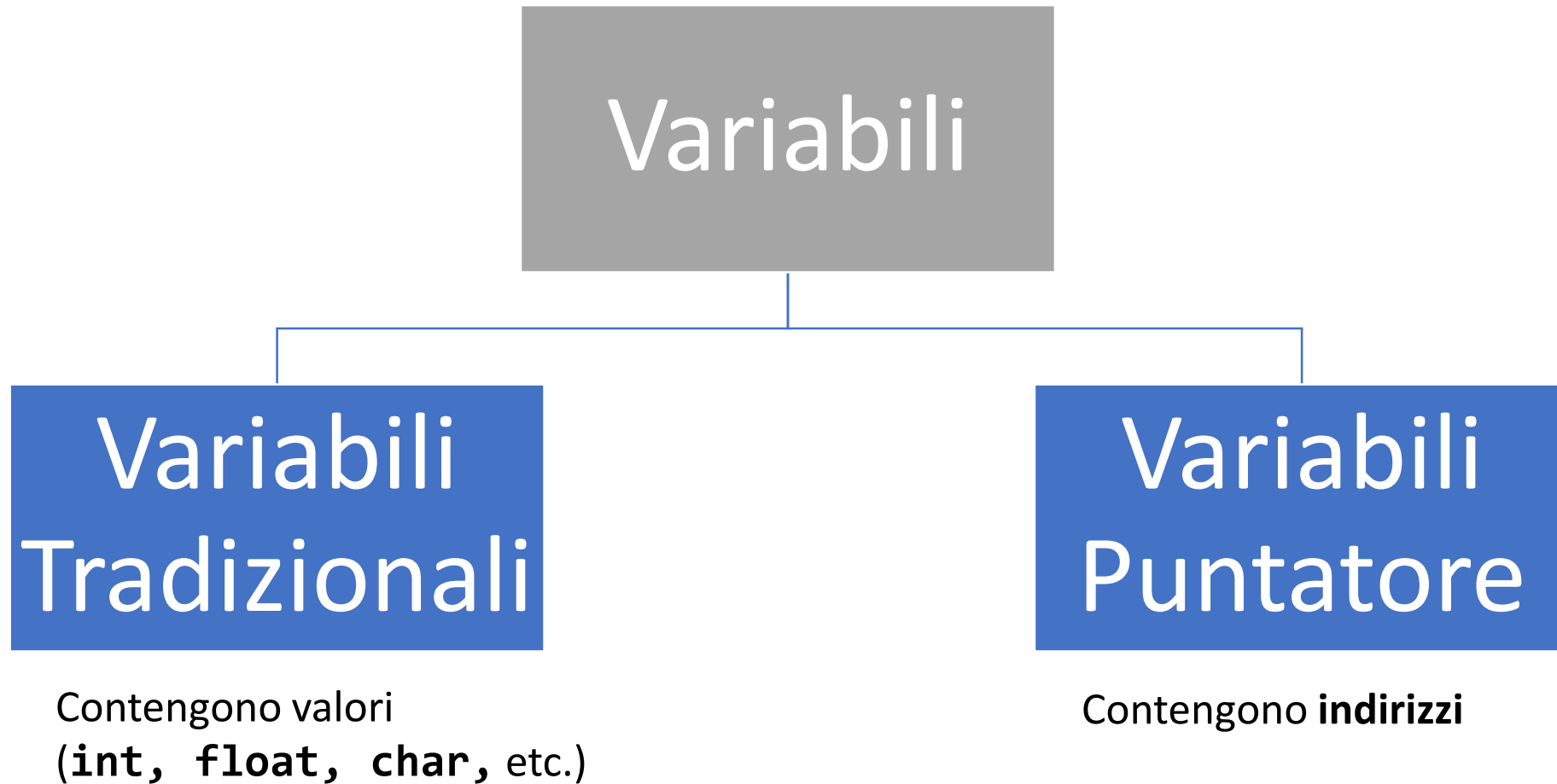


# Puntatori

- Cosa sono?
  - Una particolare tipologia di variabili
  - **Dov'è la particolarità?** Possono memorizzare solo **indirizzi di memoria**.
    - Normalmente le variabili contengono un **valore compatibile col tipo della variabile**.
    - Le variabili di tipo puntatore contengono **un indirizzo di memoria**

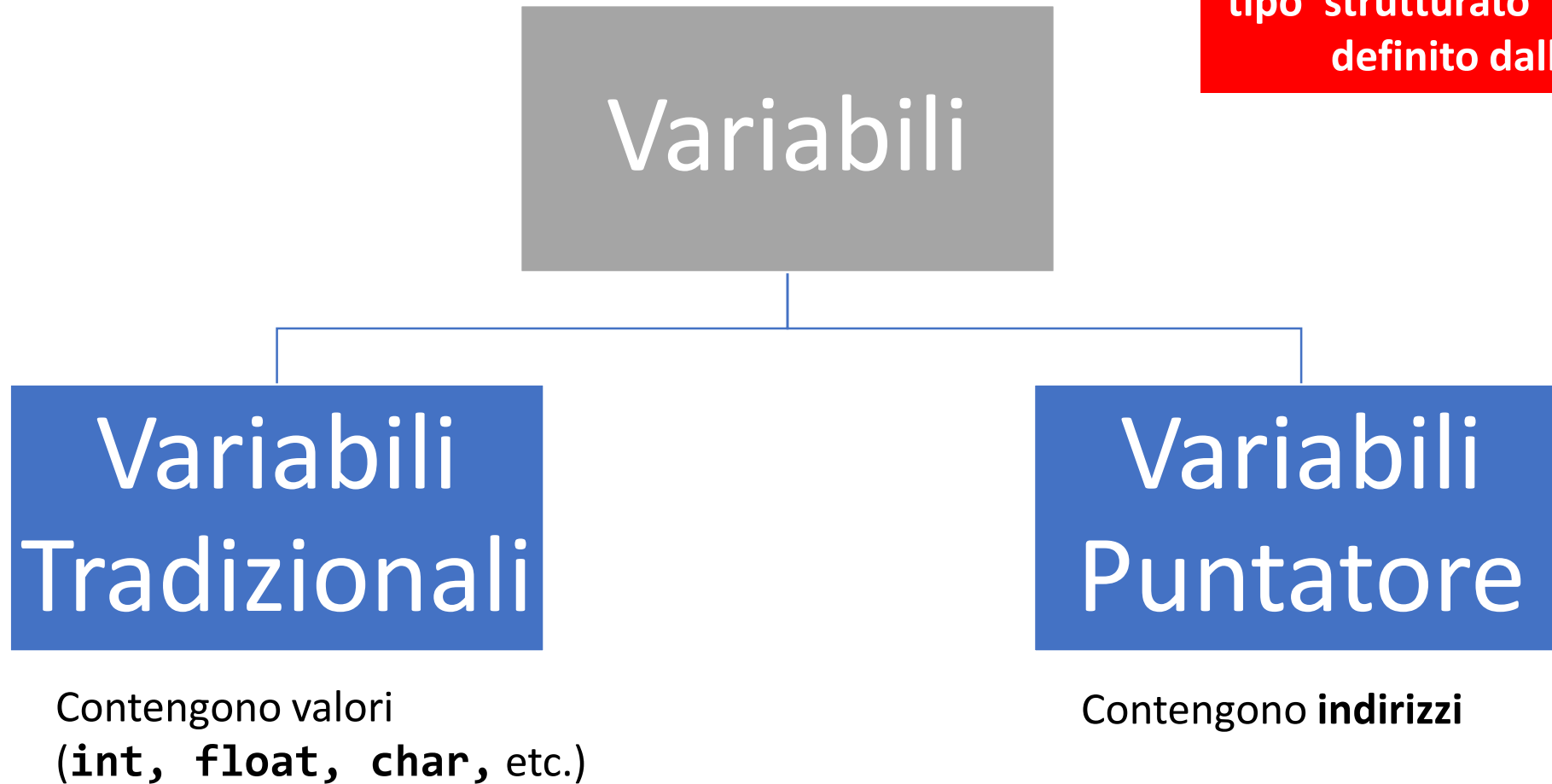


# Recap



# Recap

Questo schema non tiene in considerazione le variabili di tipo 'strutturato' o i tipi di dato definito dall'utente!



# Variabili puntatore

- Come si dichiara?

# Variabili puntatore

- Come si dichiara?
  - **<tipo\_primitivo>\* <nome\_variabile>**
    - `int* a;`
    - `float* b;`
    - `char* c;`

# Variabili puntatore

- Come si dichiara?
  - **<tipo\_primitivo>\* <nome\_variabile>**
    - `int* a;`
    - `float* b;`
    - `char* c;`
  - L'asterisco può anche essere messo vicino al nome della variabile (**`int *a`**)
- Quali sono le caratteristiche di questa variabile? (es. **`int *a`**)
  - **nome:**
  - **tipo:**
  - **valore:**
  - **indirizzo:**



# Variabili puntatore

- Come si dichiara?
  - **<tipo\_primitivo>\* <nome\_variabile>**
    - `int* a;`
    - `float* b;`
    - `char* c;`
  - L'asterisco può anche essere messo vicino al nome della variabile (**`int *a`**)
- Quali sono le caratteristiche di questa variabile? (es. **`int *a`**)
  - **nome:** a
  - **tipo:** puntatore a intero (Si legge «a è un puntatore a un intero»)
  - **valore:** inizialmente casuale (**← attenzione!**)
  - **indirizzo:** definito dal compilatore

# Variabili puntatore

- Come si dichiara?
  - **<tipo\_primitivo>\* <nome\_variabile>**
    - `int* a;`
    - `float* b;`
    - `char* c;`
- Le variabili di tipo puntatore hanno anche un tipo. **Cosa significa?**

# Variabili puntatore

- Come si dichiara?
  - **<tipo\_primitivo>\* <nome\_variabile>**
    - `int* a;`
    - `float* b;`
    - `char* c;`
- Le variabili di tipo puntatore hanno anche un tipo. **Cosa significa?**
  - **Il valore delle variabili di tipo puntatore deve essere un indirizzo**
    - **Quell'indirizzo deve contenere a sua volta un valore di quel tipo**
  - **Bisogna stare attenti: il compilatore spesso non restituisce errore. Gli errori si notano solo in fase di esecuzione**

# Variabili puntatore

- Come si dichiara?
  - **<tipo\_primitivo>\* <nome\_variabile>**
    - `int* a;`
    - `float* b;`
    - `char* c;`
- Le variabili di tipo puntatore hanno anche un tipo. **Cosa significa?**
  - **Il valore delle variabili di tipo puntatore deve essere un indirizzo**
    - **Quell'indirizzo deve contenere a sua volta un valore di quel tipo**
  - **`int* a` → ad 'a' bisogna assegnare il valore di un indirizzo al cui interno deve essere **memorizzato un valore intero!****

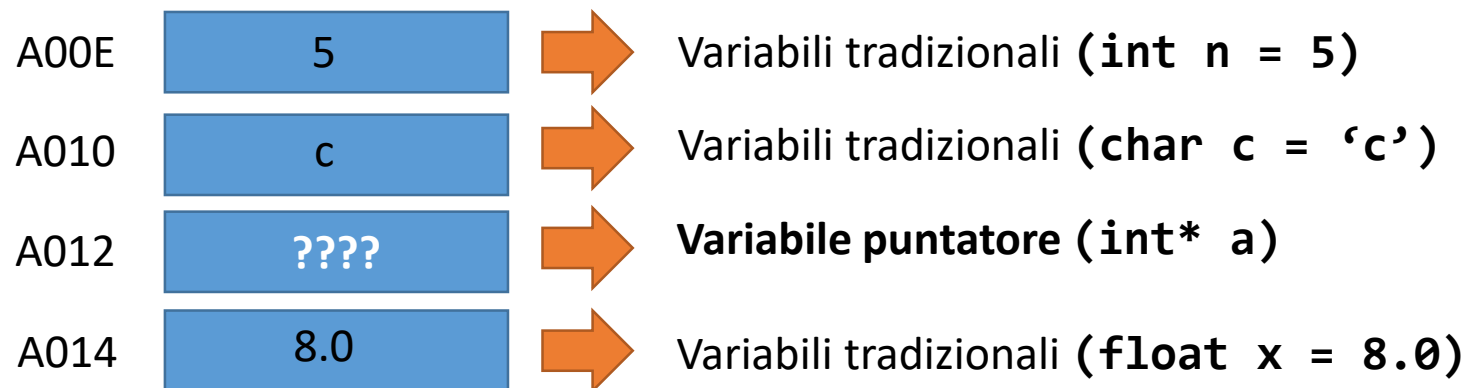
# Variabili puntatore - Esempio

- Come si dichiara?

- **<tipo\_primitivo>\* <nome\_variabile>**

- `int* a;`
- `float* b;`
- `char* c;`

- Qual è un'assegnazione valida per la variabile `int* a` ?



## Reminder

ad 'a' bisogna assegnare il valore di un indirizzo al cui interno deve essere memorizzato un valore intero!

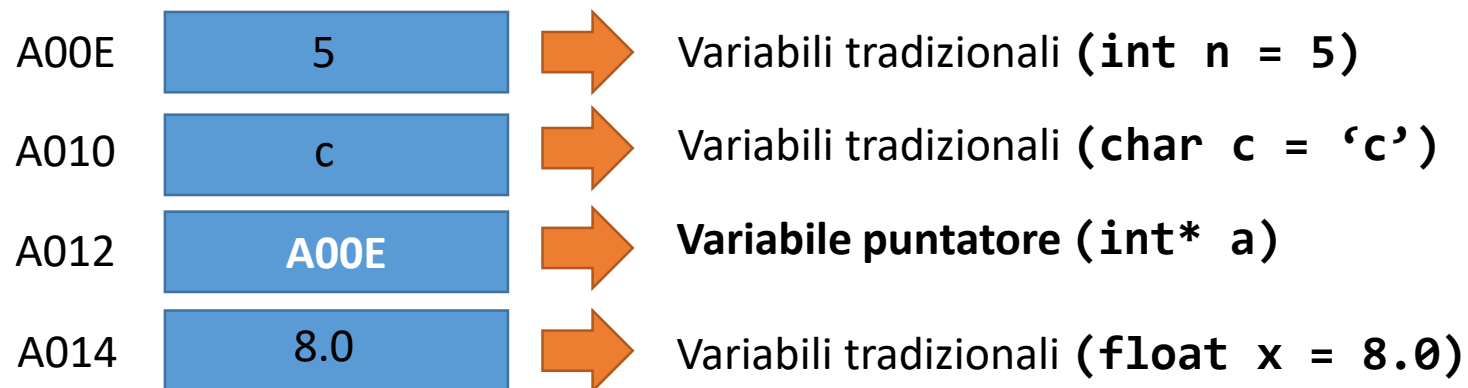
# Variabili puntatore - Esempio

- Come si dichiara?

- **<tipo\_primitivo>\* <nome\_variabile>**

- `int* a;`
- `float* b;`
- `char* c;`

- Qual è un'assegnazione valida per la variabile `int* a` ?



## Reminder

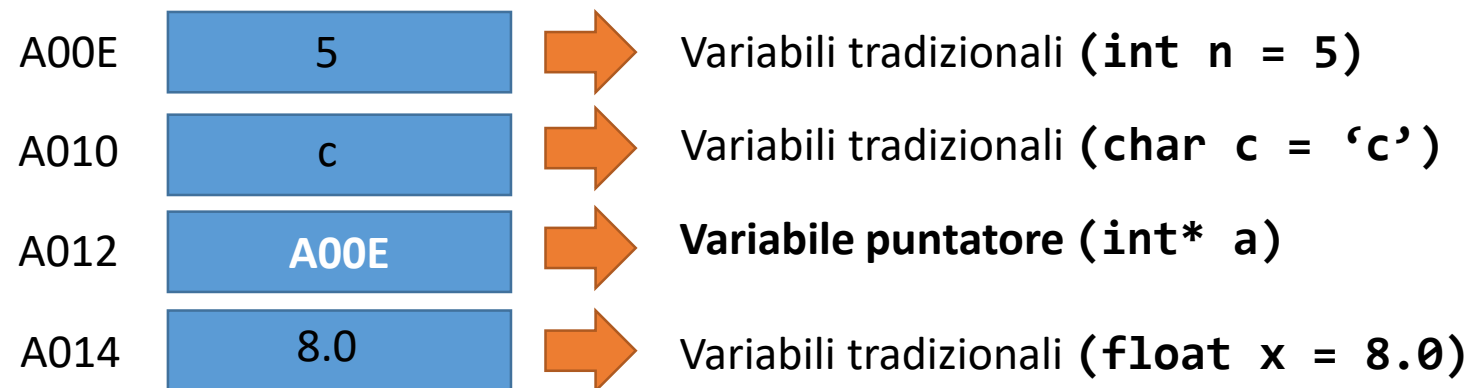
ad 'a' bisogna assegnare il valore di un indirizzo al cui interno deve essere memorizzato un valore intero!

# Variabili puntatore - Esempio

- Come si dichiara?
  - **<tipo\_primitivo>\* <nome\_variabile>**
    - `int* a;`
    - `float* b;`
    - `char* c;`
  - Qual è un'assegnazione valida per la variabile `int* a` ?

L'unica assegnazione **valida** è **A00E** perché la variabile deve 'puntare' a un indirizzo di memoria **al cui interno è stato memorizzato un intero.**

Altre assegnazioni saranno accettate ma potranno portare a problemi in fase di esecuzione



# Variabili puntatore

- Come si assegna un valore alle variabili puntatore?



# Variabili puntatore

- Come si assegna un valore alle variabili puntatore?
  - Il valore delle variabili puntatore è un indirizzo.
  - Abbiamo modo di conoscere direttamente gli indirizzi della macchina? NO
    - Si usa l'operatore indirizzo

# Variabili puntatore

- Come si assegna un valore alle variabili puntatore?
  - Il valore delle variabili puntatore è un indirizzo.
  - Abbiamo modo di conoscere direttamente gli indirizzi della macchina? NO
    - Si usa l'operatore indirizzo

```
1 #include <stdio.h>
2 int main() {
3     int a = 5;
4     float b = 3.0;
5
6     int* aPtr; // puntatore a intero
7     float* bPtr; // puntatore a float
8
9     aPtr = a; // errore
10    aPtr = &a; // ok
11    bPtr = &b; // ok
12
13    printf("\n a: %d \t\t &a: %X", a, aPtr);
14    printf("\n b: %.2f \t &b: %X", b, bPtr);
15 }
```

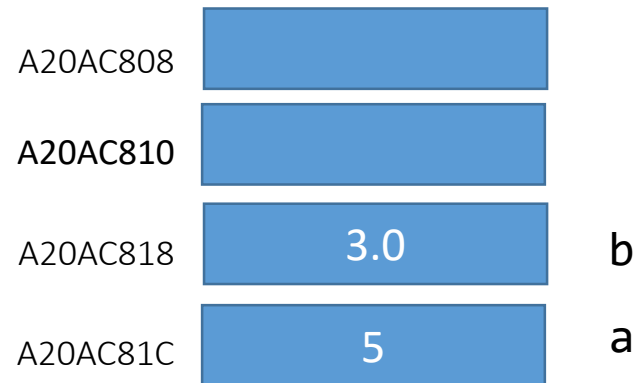
# Variabili puntatore

- Come si assegna un valore alle variabili puntatore?
  - Il valore delle variabili puntatore è un indirizzo.
  - Abbiamo modo di conoscere direttamente gli indirizzi della macchina? NO
    - Si usa l'operatore indirizzo

**Cosa succede in memoria?**

```
1 #include <stdio.h>
2 int main() {
3     int a = 5;
4     float b = 3.0;
5
6     int* aPtr; // puntatore a intero
7     float* bPtr; // puntatore a float
8
9     aPtr = a; // errore
10    aPtr = &a; // ok
11    bPtr = &b; // ok
12
13    printf("\n a: %d \t\t &a: %X", a, aPtr);
14    printf("\n b: %.2f \t &b: %X", b, bPtr);
15 }
```

# Variabili puntatore



Situazione di partenza,  
con le due variabili allocate

```
1  #include <stdio.h>
2  int main() {
3      int a = 5;
4      float b = 3.0;
5
6      int* aPtr; // puntatore a intero
7      float* bPtr; // puntatore a float
8
9      aPtr = a; // errore
10     aPtr = &a; // ok
11     bPtr = &b; // ok
12
13     printf("\n a: %d \t\t &a: %X", a, aPtr);
14     printf("\n b: %.2f \t &b: %X", b, bPtr);
15 }
```

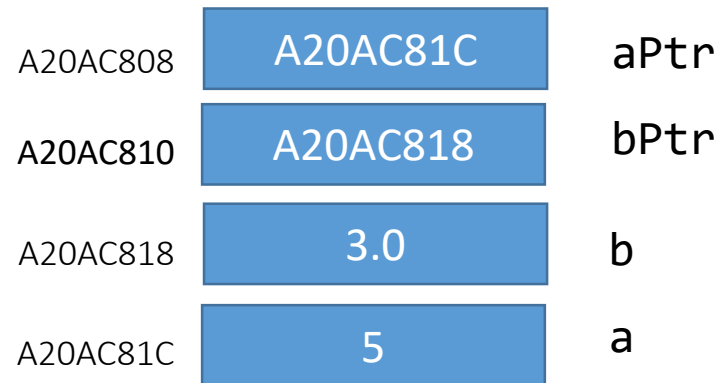
# Variabili puntatore

A20AC808	?	aPtr
A20AC810	?	bPtr
A20AC818	3.0	b
A20AC81C	5	a

Assegniamo ai puntatori i **due indirizzi delle variabili** (di tipo compatibile)

```
1 #include <stdio.h>
2 int main() {
3     int a = 5;
4     float b = 3.0;
5
6     int* aPtr; // puntatore a intero
7     float* bPtr; // puntatore a float
8
9     aPtr = a; // errore
10    aPtr = &a; // ok
11    bPtr = &b; // ok
12
13    printf("\n a: %d \t\t &a: %X", a, aPtr);
14    printf("\n b: %.2f \t &b: %X", b, bPtr);
15 }
```

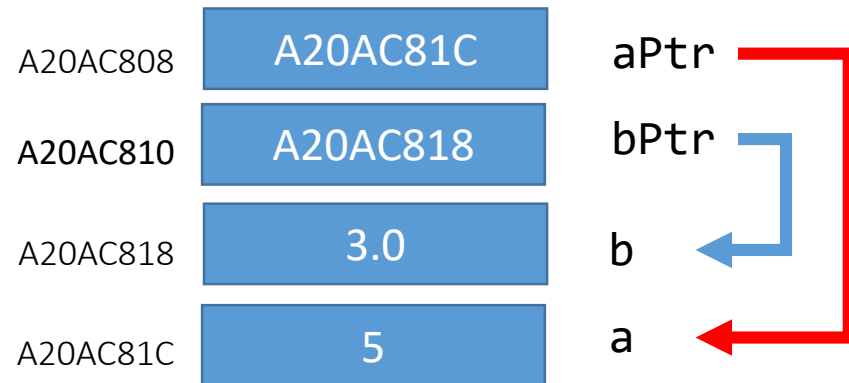
# Variabili puntatore



Assegniamo ai puntatori i **due indirizzi delle variabili** (di tipo compatibile)

```
1 #include <stdio.h>
2 int main() {
3     int a = 5;
4     float b = 3.0;
5
6     int* aPtr; // puntatore a intero
7     float* bPtr; // puntatore a float
8
9     aPtr = a; // errore
10    aPtr = &a; // ok
11    bPtr = &b; // ok
12
13    printf("\n a: %d \t\t &a: %X", a, aPtr);
14    printf("\n b: %.2f \t &b: %X", b, bPtr);
15 }
```

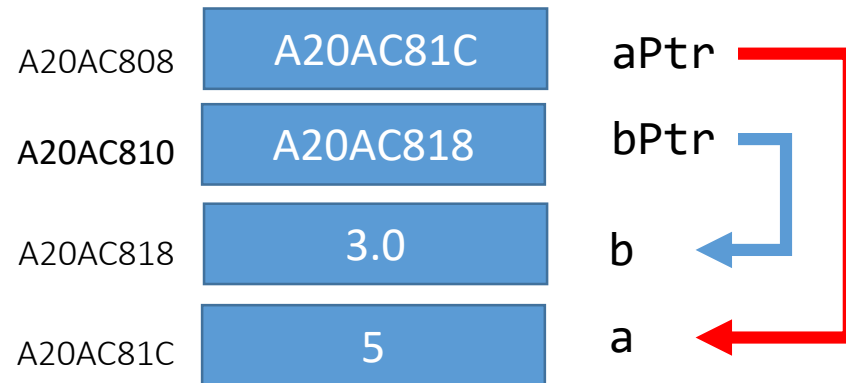
# Variabili puntatore



**Nota:** dopo l'assegnazione è possibile rappresentare graficamente i puntatori

```
1  #include <stdio.h>
2  int main() {
3      int a = 5;
4      float b = 3.0;
5
6      int* aPtr; // puntatore a intero
7      float* bPtr; // puntatore a float
8
9      aPtr = a; // errore
10     aPtr = &a; // ok
11     bPtr = &b; // ok
12
13     printf("\n a: %d \t\t &a: %X", a, aPtr);
14     printf("\n b: %.2f \t &b: %X", b, bPtr);
15 }
```

# Variabili puntatore

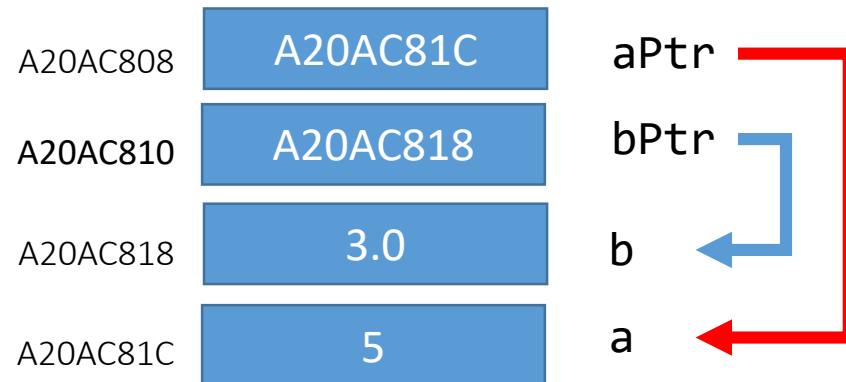


Stampiamo in output i quattro valori

```
1 #include <stdio.h>
2 int main() {
3     int a = 5;
4     float b = 3.0;
5
6     int* aPtr; // puntatore a intero
7     float* bPtr; // puntatore a float
8
9     aPtr = a; // errore
10    aPtr = &a; // ok
11    bPtr = &b; // ok
12
13    printf("\n a: %d \t\t &a: %X", a, aPtr);
14    printf("\n b: %.2f \t &b: %X", b, bPtr);
15 }
```



# Variabili puntatore



Stampiamo in output i quattro valori

```
gcc version 4.6.3
a: 5                    &a: A20AC81C
b: 3.00                &b: A20AC818
```

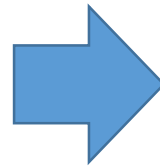
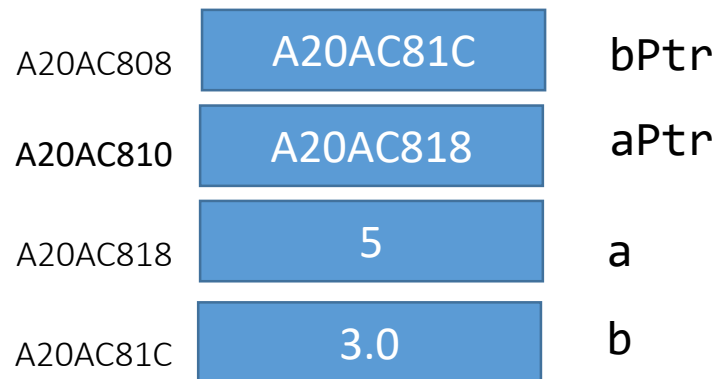
```
1  #include <stdio.h>
2  int main() {
3      int a = 5;
4      float b = 3.0;
5
6      int* aPtr; // puntatore a intero
7      float* bPtr; // puntatore a float
8
9      aPtr = a; // errore
10     aPtr = &a; // ok
11     bPtr = &b; // ok
12
13     printf("\n a: %d \t\t &a: %X", a, aPtr);
14     printf("\n b: %.2f \t &b: %X", b, bPtr);
15 }
```

# Operatore di indirizione

- Dato un puntatore, il linguaggio C mette a disposizione anche un operatore di **indirizione**
- **A che serve?**
  - **Serve a risalire al valore memorizzato** nella cella di memoria dell'indirizzo puntato dalla variabile

# Operatore di indirizzazione

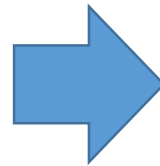
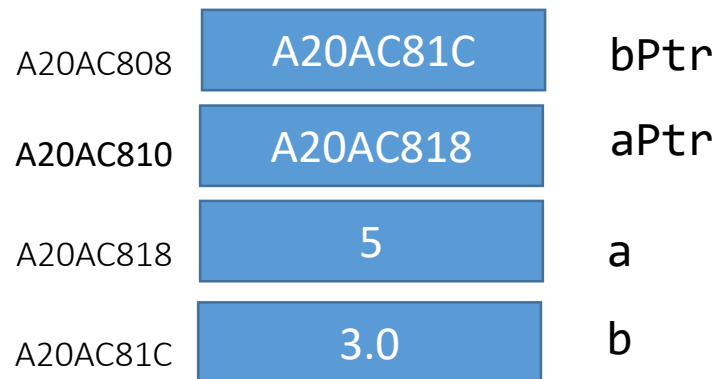
- Dato un puntatore, il linguaggio C mette a disposizione anche un operatore di **indirizzazione**
- **A che serve?**
  - **Serve a risalire al valore memorizzato** nella cella di memoria dell'indirizzo puntato dalla variabile



**Esempio: bPtr** è un variabile di tipo puntatore. La variabile punta a un indirizzo. **Attraverso l'operatore di indirizzazione posso risalire al valore memorizzato nella cella di memoria identificato dall'indirizzo.**

# Operatore di indirazione

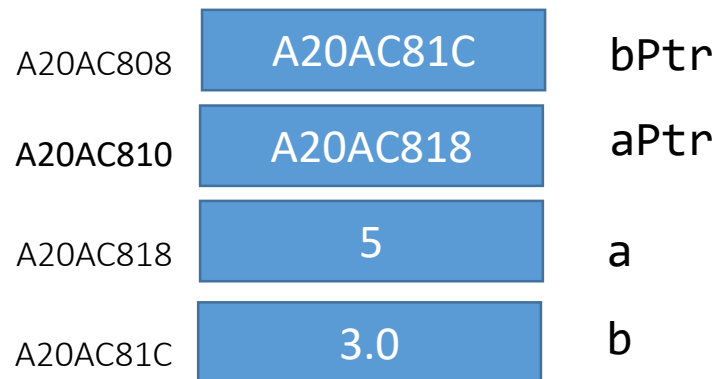
- L'operatore di indirazione si esprime con l'asterisco accanto a una variabile di tipo puntatore (es. **\*bPtr**)
- Qual è il valore di **\*bPtr**?



**Esempio:** **bPtr** è un variabile di tipo puntatore. La variabile punta a un indirizzo. **Attraverso l'operatore di indirazione posso risalire al valore memorizzato nella cella di memoria identificato dall'indirizzo.**

# Operatore di indirizzazione

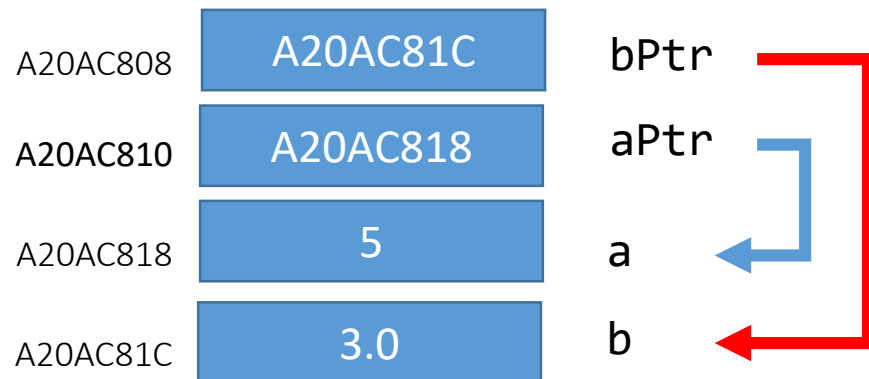
- L'operatore di indirizzazione si esprime con l'asterisco accanto a una variabile di tipo puntatore (es. **\*bPtr**)
- In questo caso il valore di **\*bPtr** è uguale a 3.0



**Esempio:** **bPtr** è un variabile di tipo puntatore. La variabile punta a un indirizzo. **Attraverso l'operatore di indirizzazione posso risalire al valore memorizzato nella cella di memoria identificato dall'indirizzo.**

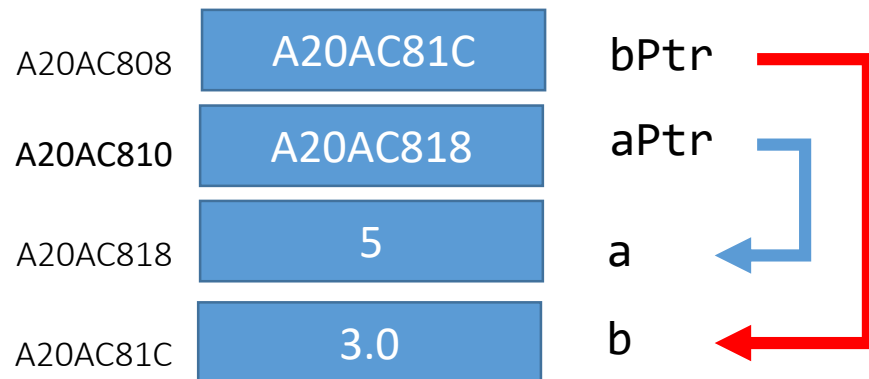
# Operatore di indirizzazione

- L'operatore di indirizzazione si esprime con l'asterisco accanto a una variabile di tipo puntatore (es. **\*bPtr**)
- In questo caso il valore di **\*bPtr** è uguale a 3.0
  - Il concetto di indirizzazione è più semplice se pensiamo alla rappresentazione grafica dei puntatori → è il valore contenuto nella locazione di memoria «puntata»



# Operatore di indirizzazione

- L'operatore di indirizzazione si esprime con l'asterisco accanto a una variabile di tipo puntatore (es. **\*bPtr**)
- **Attenzione a non fare confusione tra asterischi**
  - Asterisco usato per dichiarare una variabile (**float\* bPtr**)
  - Asterisco usato come operatore di indirizzazione sul puntatore (**\*bPtr**)



# Recap

```
1  #include <stdio.h>
2  int main() {
3      int a = 5;
4      float b = 3.0;
5
6      int* aPtr; // puntatore a intero
7      float* bPtr; // puntatore a float
8
9      aPtr = a; // errore
10     aPtr = &a; // ok
11     bPtr = &b; // ok
12
13     printf("\n a: %d \t\t &a: %X", a, aPtr);
14     printf("\n b: %.2f \t &b: %X", b, bPtr);
15 }
```

**a** è un intero → il suo valore è 5  
**b** è un float → il suo valore è 3.0



# Recap

```
1  #include <stdio.h>
2  int main() {
3      int a = 5;
4      float b = 3.0;
5
6      int* aPtr; // puntatore a intero
7      float* bPtr; // puntatore a float
8
9      aPtr = a; // errore
10     aPtr = &a; // ok
11     bPtr = &b; // ok
12
13     printf("\n a: %d \t\t &a: %X", a, aPtr);
14     printf("\n b: %.2f \t &b: %X", b, bPtr);
15 }
```

**a** è un intero → il suo valore è 5

**b** è un float → il suo valore è 3.0

**aPtr** è un puntatore a un intero →

**bPtr** è un puntatore a un float →

# Recap

```
1  #include <stdio.h>
2  int main() {
3      int a = 5;
4      float b = 3.0;
5
6      int* aPtr; // puntatore a intero
7      float* bPtr; // puntatore a float
8
9      aPtr = a; // errore
10     aPtr = &a; // ok
11     bPtr = &b; // ok
12
13     printf("\n a: %d \t\t &a: %X", a, aPtr);
14     printf("\n b: %.2f \t &b: %X", b, bPtr);
15 }
```

**a** è un intero → il suo valore è 5

**b** è un float → il suo valore è 3.0

**aPtr** è un puntatore a un intero →

il suo valore è l'indirizzo di **a**

**bPtr** è un puntatore a un float →

il suo valore è l'indirizzo di **b**

# Recap

```
1  #include <stdio.h>
2  int main() {
3      int a = 5;
4      float b = 3.0;
5
6      int* aPtr; // puntatore a intero
7      float* bPtr; // puntatore a float
8
9      aPtr = a; // errore
10     aPtr = &a; // ok
11     bPtr = &b; // ok
12
13     printf("\n a: %d \t\t &a: %X", a, aPtr);
14     printf("\n b: %.2f \t &b: %X", b, bPtr);
15 }
```

**a** è un intero → il suo valore è 5

**b** è un float → il suo valore è 3.0

**aPtr** è un puntatore a un intero →

il suo valore è l'indirizzo di **a**

**bPtr** è un puntatore a un float →

il suo valore è l'indirizzo di **b**

**\*aPtr** è l'operatore di indirezione sul  
puntatore **aPtr** →

**\*bPtr** è l'operatore di indirezione sul  
puntatore **bPtr** →

# Recap

```
1  #include <stdio.h>
2  int main() {
3      int a = 5;
4      float b = 3.0;
5
6      int* aPtr; // puntatore a intero
7      float* bPtr; // puntatore a float
8
9      aPtr = a; // errore
10     aPtr = &a; // ok
11     bPtr = &b; // ok
12
13     printf("\n a: %d \t\t &a: %X", a, aPtr);
14     printf("\n b: %.2f \t &b: %X", b, bPtr);
15 }
```

**a** è un intero → il suo valore è 5

**b** è un float → il suo valore è 3.0

**aPtr** è un puntatore a un intero →

il suo valore è l'indirizzo di **a**

**bPtr** è un puntatore a un float →

il suo valore è l'indirizzo di **b**

**\*aPtr** è l'operatore di indirezione sul

puntatore **aPtr** → il suo valore è 5

**\*bPtr** è l'operatore di indirezione sul

puntatore **bPtr** → il suo valore è 3.0

# Recap

**Quando usiamo l'operatore di indirazione parliamo di «dereferenziare una variabile».** Dereferenziando il puntatore, in questo caso, ritorniamo al valore della variabile di partenza.

```
8 #include <stdio.h>
9
10 int main() {
11     int a = 5;
12     float b = 3.0; // dichiaro le due variabili di input dell'esercizio
13
14     int* aPtr;
15     float* bPtr; // dichiaro due variabili di tipo puntatore
16
17     // aPtr = a --> ERRORE, devo assegnare un indirizzo
18     aPtr = &a;
19     bPtr = &b; // assegno ai puntatori due indirizzi di variabile *di tipo compatibile*
20
21     printf("Valore di a: %d \t \t \t Indirizzo di a: %X \n", a, &a);
22     printf("Valore di b: %.1f \t \t Indirizzo di b: %X \n", b, &b); // stampo i valori delle variabili
23
24     printf("Valore di aPtr: %X \t \t Indirazione di aPtr: %d \n", aPtr, *aPtr);
25     printf("Valore di bPtr: %X \t \t Indirazione di bPtr: %.1f \n", bPtr, *bPtr); // stampo i valori delle variabili
26
27 }
28
```

# Recap

```
Valore di a: 5           Indirizzo di a: 61FF24
Valore di b: 3.0        Indirizzo di b: 61FF20
Valore di aPtr: 61FF24  Indirazione di aPtr: 5
Valore di bPtr: 61FF20  Indirazione di bPtr: 3.0
```

```
8 #include <stdio.h>
9
10 int main() {
11     int a = 5;
12     float b = 3.0; // dichiaro le due variabili di input dell'esercizio
13
14     int* aPtr;
15     float* bPtr; // dichiaro due variabili di tipo puntatore
16
17     // aPtr = a --> ERRORE, devo assegnare un indirizzo
18     aPtr = &a;
19     bPtr = &b; // assegno ai puntatori due indirizzi di variabile *di tipo compatibile*
20
21     printf("Valore di a: %d \t \t \t Indirizzo di a: %X \n", a, &a);
22     printf("Valore di b: %.1f \t \t Indirizzo di b: %X \n", b, &b); // stampo i valori delle variabili
23
24     printf("Valore di aPtr: %X \t \t Indirazione di aPtr: %d \n", aPtr, *aPtr);
25     printf("Valore di bPtr: %X \t \t Indirazione di bPtr: %.1f \n", bPtr, *bPtr); // stampo i valori delle variabili
26
27 }
28
```

**L'indirizzo della variabile a (&a) corrisponde al puntatore aPtr**

# Recap

```
Valore di a: 5           Indirizzo di a: 61FF24
Valore di b: 3.0        Indirizzo di b: 61FF20
Valore di aPtr: 61FF24  Indirazione di aPtr: 5
Valore di bPtr: 61FF20  Indirazione di bPtr: 3.0
```

```
8 #include <stdio.h>
9
10 int main() {
11     int a = 5;
12     float b = 3.0; // dichiaro le due variabili di input dell'esercizio
13
14     int* aPtr;
15     float* bPtr; // dichiaro due variabili di tipo puntatore
16
17     // aPtr = a --> ERRORE, devo assegnare un indirizzo
18     aPtr = &a;
19     bPtr = &b; // assegno ai puntatori due indirizzi di variabile *di tipo compatibile*
20
21     printf("Valore di a: %d \t \t \t Indirizzo di a: %X \n", a, &a);
22     printf("Valore di b: %.1f \t \t Indirizzo di b: %X \n", b, &b); // stampo i valori delle variabili
23
24     printf("Valore di aPtr: %X \t \t Indirazione di aPtr: %d \n", aPtr, *aPtr);
25     printf("Valore di bPtr: %X \t \t Indirazione di bPtr: %.1f \n", bPtr, *bPtr); // stampo i valori delle variabili
26
27 }
28
```

**L'indirizzo della variabile a (&a) corrisponde al puntatore aPtr**  
**Il valore dereferenziato di aPtr corrisponde al valore della variabile a**

# Note importanti (1)

- **Attenzione a non fare confusione tra asterischi**
  - Asterisco usato per dichiarare una variabile (**float\* bPtr**)
  - Asterisco usato come operatore di indirazione sul puntatore (**\*bPtr**)
- **In una dichiarazione**, l'asterisco serve per **dichiarare una variabile** di tipo puntatore
  - Es.: **int \*pi;**
- **In una espressione**, l'asterisco funge da **operatore di dereferenziazione**
  - Es.: **int b = \*pi;**



# Note importanti (2)

- Gli operatori di dereferenziazione (\*) e di indirizzo (&) sono uno l'inverso dell'altro (e si annullano a vicenda). **Che significa?**

# Note importanti (2)

- Gli operatori di dereferenziazione (\*) e di indirizzo (&) sono uno l'inverso dell'altro (e si annullano a vicenda). **Che significa?**
- data la dichiarazione `int a`
  - `*&a` equivale a scrivere `a`
  - (Il valore memorizzato nell'indirizzo di memoria della variabile `a == a`)

# Note importanti (2)

- Gli operatori di dereferenziazione (\*) e di indirizzo (&) sono uno l'inverso dell'altro (e si annullano a vicenda). **Che significa?**
- data la dichiarazione **int a**
  - **\*&a** equivale a scrivere **a**
  - (Il valore memorizzato nell'indirizzo di memoria della variabile **a == a**)
- data la dichiarazione **int \*pi;**
  - **&\*pi** ha valore uguale a **pi**
  - (L'indirizzo di memoria del contenuto puntato dal puntatore **pi == pi**)

# Note importanti (3)

- Anche le variabili puntatore devono essere **inizializzate**
- Di base il compilatore assegna un indirizzo random alle variabili puntatore
  - **Problema:** non sappiamo cosa è memorizzato in quelle locazioni di memoria!

# Note importanti (3)

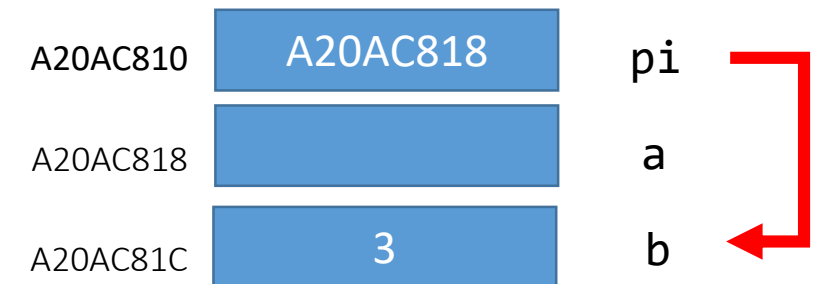
- Anche le variabili puntatore devono essere **inizializzate**
- Di base il compilatore assegna un indirizzo random alle variabili puntatore
  - **Problema:** non sappiamo cosa è memorizzato in quelle locazioni di memoria!
  - **Ulteriori operazioni fatte con i puntatori – senza inizializzare la variabile - rischiano di rendere corrotta la memoria e di creare problemi in esecuzione**

# Note importanti (3)

- Anche le variabili puntatore devono essere **inizializzate**
- Di base il compilatore assegna un indirizzo random alle variabili puntatore
  - **Problema:** non sappiamo cosa è memorizzato in quelle locazioni di memoria!
  - **Ulteriori operazioni fatte con i puntatori – senza inizializzare la variabile - rischiano di rendere corrotta la memoria e di creare problemi in esecuzione**

- **Esempio**

```
int a; int* pi; // puntatore non inizializzato
int b = 3; // variabile inizializzata
a = *pi;
*pi = 500;
```

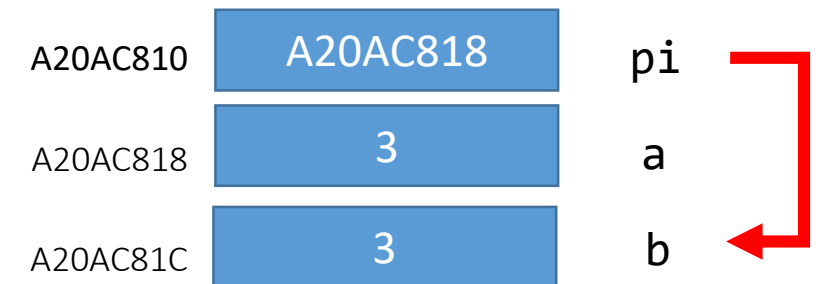


# Note importanti (3)

- Anche le variabili puntatore devono essere **inizializzate**
- Di base il compilatore assegna un indirizzo random alle variabili puntatore
  - **Problema:** non sappiamo cosa è memorizzato in quelle locazioni di memoria!
  - **Ulteriori operazioni fatte con i puntatori – senza inizializzare la variabile - rischiano di rendere corrotta la memoria e di creare problemi in esecuzione**

- **Esempio**

```
int a; int* pi; // puntatore non inizializzato
int b = 3; // variabile inizializzata
a = *pi; // assegno ad a il valore puntato
*pi = 500;
```

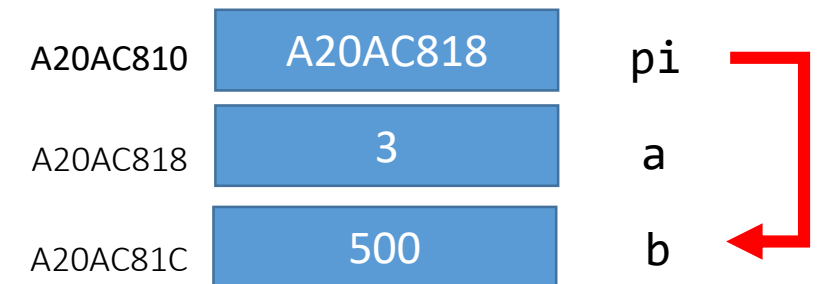


# Note importanti (3)

- Anche le variabili puntatore devono essere **inizializzate**
- Di base il compilatore assegna un indirizzo random alle variabili puntatore
  - **Problema:** non sappiamo cosa è memorizzato in quelle locazioni di memoria!
  - **Ulteriori operazioni fatte con i puntatori – senza inizializzare la variabile - rischiano di rendere corrotta la memoria e di creare problemi in esecuzione**

- **Esempio**

```
int a; int* pi; // puntatore non inizializzato
int b = 3 // variabile inizializzata
a = *pi; // assegno ad a il valore puntato
*pi = 500 // senza volerlo ho corrotto il valore
           // della variabile b!
```





# Note importanti (3)

- Anche le variabili puntatore devono essere **inizializzate**
- Di base il compilatore assegna un indirizzo random alle variabili puntatore
  - **Problema:** non sappiamo cosa è memorizzato in quelle locazioni di memoria!
  - **Ulteriori operazioni fatte con i puntatori – senza inizializzare la variabile - rischiano di rendere corrotta la memoria e di creare problemi in esecuzione**
- Per **inizializzare le variabili puntatore** si utilizza **NULL**
  - `int x;`
  - `int *pi = NULL;`
  - `float *pf = NULL`



# Passaggio dei Parametri

- La principale applicazione dei puntatori è legata al passaggio dei parametri
- In C i parametri vengono passati per **valore**
  - Il valore del parametro attuale **viene copiato** nel parametro formale (che è una variabile locale della funzione)
  - E' il metodo più sicuro per **evitare modifiche accidentali** ai parametri
  - Se all'interno di una funzione effettuiamo delle modifiche ai valori dei parametri, **queste modifiche vengono perse!**
- **A volte questo non è sufficiente!**

# Passaggio dei Parametri - Esempio

```
void scambia(int a, int b) {  
    int t; // variabile locale di appoggio  
  
    t = a; // scambio dei valori  
    a = b;  
    b = t;  
}
```

```
main() {  
    int x = 33, y = 5;  
    scambia(x, y);  
    printf("x = %d, y = %d\n", x, y);  
}
```

# Passaggio dei Parametri - Esempio

```
void scambia(int a, int b) {  
    int t; // variabile locale di appoggio  
  
    t = a; // scambio dei valori  
    a = b;  
    b = t;  
}
```

1.  
Inizializzazione

```
main() {  
    int x = 33, y = 5;  
    scambia(x, y);  
    printf("x = %d, y = %d\n", x, y);  
}
```

Area Dati scambia(a,b)

Area Dati main()

x 33

y 5

# Passaggio dei Parametri - Esempio

```
void scambia(int a, int b) {  
    int t; // variabile locale di appoggio  
  
    t = a; // scambio dei valori  
    a = b;  
    b = t;  
}
```

```
main() {  
    int x = 33, y = 5;  
    scambia(x, y);  
    printf("x = %d, y = %d\n", x, y);  
}
```

## 2. Chiamata della Funzione

Area Dati scambia(a,b)

a 33

b 5

t

Area Dati main()

x 33

y 5

# Passaggio dei Parametri - Esempio

```
void scambia(int a, int b) {  
    int t; // variabile locale di appoggio  
  
    t = a; // scambio dei valori  
    a = b;  
    b = t;  
}
```

## 3. Scambio dei Valori

```
main() {  
    int x = 33, y = 5;  
    scambia(x, y);  
    printf("x = %d, y = %d\n", x, y);  
}
```

### Area Dati scambia(a,b)

a 5

b 33

t 33

### Area Dati main()

x 33

y 5

# Passaggio dei Parametri - Esempio

```
void scambia(int a, int b) {  
    int t; // variabile locale di appoggio  
  
    t = a; // scambio dei valori  
    a = b;  
    b = t;  
}
```

```
main() {  
    int x = 33, y = 5;  
    scambia(x, y);  
    printf("x = %d, y = %d\n", x, y);  
}
```

4.  
Uscita dalla  
Funzione

Area Dati main()

x 33

y 5



# Passaggio dei Parametri - Esempio

```
void scambia(int a, int b) {  
    int t; // variabile locale di appoggio  
  
    t = a; // scambio dei valori  
    a = b;  
    b = t;  
}
```

```
main() {  
    int x = 33, y = 5;  
    scambia(x, y);  
    printf("x = %d, y = %d\n", x, y);  
}
```

4.  
Uscita dalla  
Funzione

Attraverso l'utilizzo dei **puntatori** possiamo simulare il **passaggio per riferimento**, che risolve questi problemi.

Area Dati main()

x 33

y 5

# Come si implementa il passaggio dei parametri per riferimento?

**Si passano gli indirizzi invece dei valori!**

# Regola fondamentale del passaggio dei parametri

**Il parametro deve essere modificato dalla funzione?**

# Regola fondamentale del passaggio dei parametri

**Il parametro deve essere modificato dalla funzione?**

**Se si, devo passarlo per riferimento**

**Altrimenti, è sufficiente passarlo per valore**

**`scambia(x,y)` è una funzione che «modifica» i suoi parametri, perché il valore di x passa ad y e viceversa**

# Regola fondamentale del passaggio dei parametri

**Il parametro deve essere modificato dalla funzione?**

**Se si, devo passarlo per riferimento**

**Altrimenti, è sufficiente passarlo per valore**

**Nota importante:** i vettori fanno eccezione alla regola, perché sono passati automaticamente per riferimento

# Passaggio dei Parametri - Esempio

```
void scambia(int a, int b) {  
    int t; // variabile locale di appoggio  
  
    t = a; // scambio dei valori  
    a = b;  
    b = t;  
}
```

```
main() {  
    int x = 33, y = 5;  
    scambia(x, y);  
    printf("x = %d, y = %d\n", x, y);  
}
```

4.  
Uscita dalla  
Funzione

Attraverso l'utilizzo dei **puntatori** possiamo simulare il **passaggio per riferimento**, che risolve questi problemi.

Area Dati main()

x 33

y 5

# Passaggio dei Parametri - Esempio

```
void scambia(int* a, int* b) {  
    int t; // variabile locale di appoggio  
  
    t = *a; // scambio dei valori  
    *a = *b;  
    *b = t;  
}
```

```
main() {  
    int x = 33, y = 5;  
    scambia(&x, &y);  
    printf("x = %d, y = %d\n", x, y);  
}
```

Attraverso l'utilizzo dei **puntatori** possiamo simulare il **passaggio per riferimento**, che risolve questi problemi.

Invece di passare il **valore** delle variabili, **ne passiamo l'indirizzo!**  
In questo modo, le modifiche vengono ereditate dalle variabili originali

# Passaggio dei Parametri - Esempio

```
void scambia(int* a, int* b) {  
    int t; // variabile locale di appoggio  
  
    t = *a; // scambio dei valori  
    *a = *b;  
    *b = t;  
}
```

```
main() {  
    int x = 33, y = 5,  
    scambia(&x, &y);  
    printf("x = %d, y = %d\n", x, y);  
}
```

Attraverso l'utilizzo dei **puntatori** possiamo simulare il **passaggio per riferimento**, che risolve questi problemi.

Invece di passare il **valore** delle variabili, **ne passiamo l'indirizzo!**  
In questo modo, le modifiche vengono ereditate dalle variabili originali



# Passaggio dei Parametri - Esempio

```
void scambia(int* a, int* b) {  
    int t; // variabile locale di appoggio  
  
    t = *a; // scambio dei valori  
    *a = *b;  
    *b = t;  
}
```

## 1. Inizializzazione

```
main() {  
    int x = 33, y = 5;  
    scambia(&x, &y);  
    printf("x = %d, y = %d\n", x, y);  
}
```

Area Dati scambia(a,b)

a

b

t

Area Dati main()

x

33

y

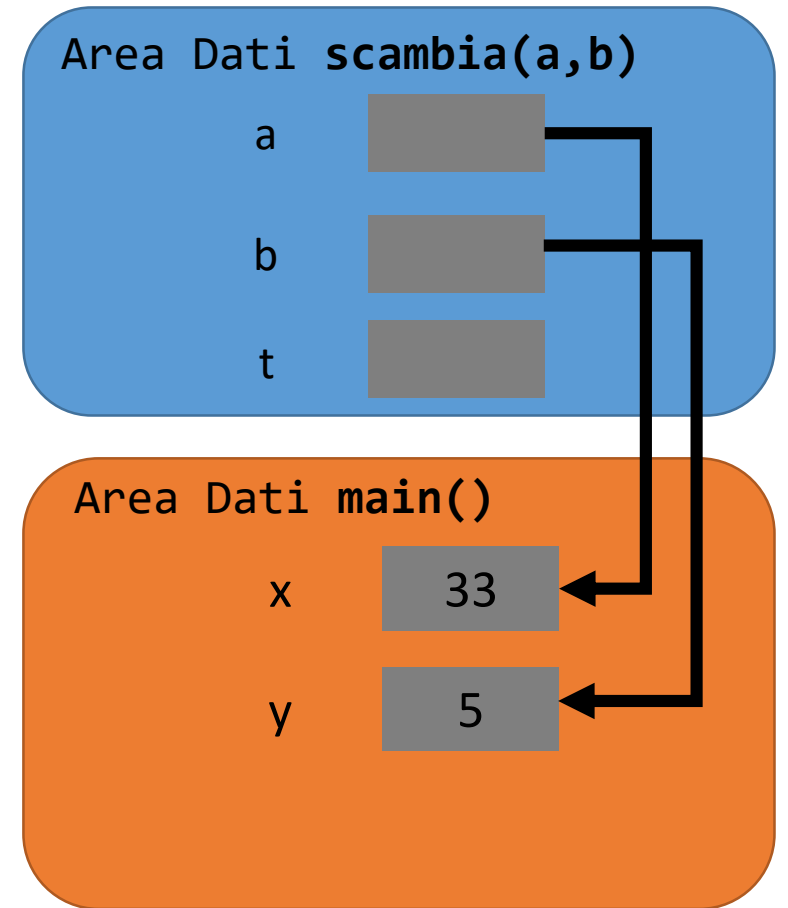
5

# Passaggio dei Parametri - Esempio

```
void scambia(int* a, int* b) {  
    int t; // variabile locale di appoggio  
  
    t = *a; // scambio dei valori  
    *a = *b;  
    *b = t;  
}
```

```
main() {  
    int x = 33, y = 5;  
    scambia(&x, &y);  
    printf("x = %d, y = %d\n", x, y);  
}
```

2.  
Chiamata della  
Funzione

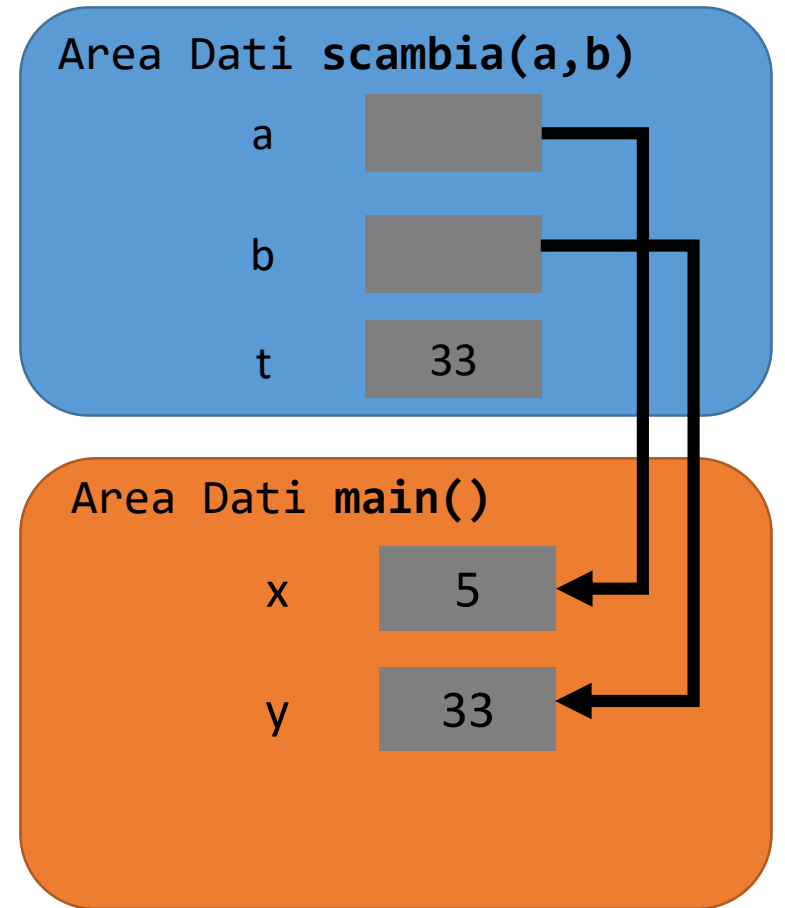


# Passaggio dei Parametri - Esempio

```
void scambia(int* a, int* b) {  
    int t; // variabile locale di appoggio  
  
    t = *a; // scambio dei valori  
    *a = *b;  
    *b = t;  
}
```

3.  
Scambio dei valori

```
main() {  
    int x = 33, y = 5;  
    scambia(&x, &y);  
    printf("x = %d, y = %d\n", x, y);  
}
```

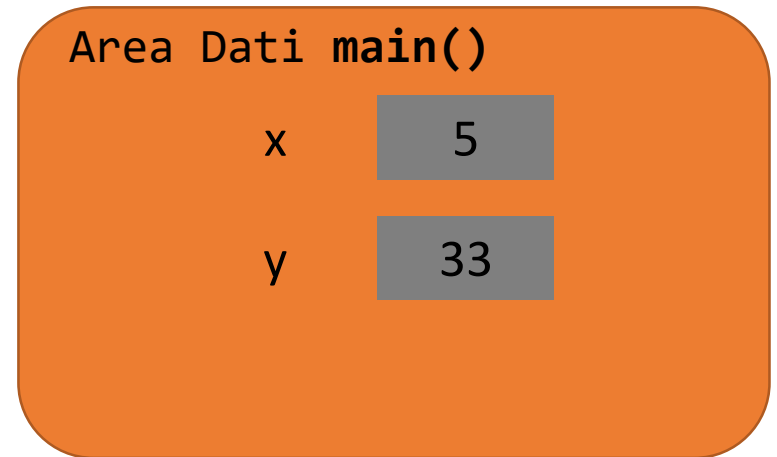


# Passaggio dei Parametri - Esempio

```
void scambia(int* a, int* b) {  
    int t; // variabile locale di appoggio  
  
    t = *a; // scambio dei valori  
    *a = *b;  
    *b = t;  
}
```

```
main() {  
    int x = 33, y = 5;  
    scambia(&x, &y);  
    printf("x = %d, y = %d\n", x, y);  
}
```

4.  
Uscita dalla  
funzione



# Passaggio dei Parametri - Recap

- La principale applicazione dei puntatori è legata al passaggio dei parametri
- Attraverso l'utilizzo dei puntatori **possiamo simulare** il passaggio per riferimento, **passando alla funzione l'indirizzo delle variabili** invece che i loro valori
- **Quando serve?**
  - **Serve NECESSARIAMENTE** quando la funzione deve modificare i valori dei **parametri** (es. scambio di valori). Esistono invece altre situazioni in cui è consigliabile, ma non obbligatorio.

# Utilizzo dei puntatori

- Una ulteriore applicazione dei puntatori è legata alla possibilità di far restituire alle funzioni più di un valore
- **Le funzioni, di base, possono restituire solo un valore!**

# Utilizzo dei puntatori

- Una ulteriore applicazione dei puntatori è legata alla possibilità di far restituire alle funzioni più di un valore
- **Le funzioni, di base, possono restituire solo un valore!**

- **Esempio**

```
int scambio-somma(int* a, int* b) {  
    int t; int somma;  
    t = *a; // scambio dei valori  
    *a = *b;  
    *b = t;  
  
    somma = *a + *b;  
    return somma;  
}
```

# Utilizzo dei puntatori

- Una ulteriore applicazione dei puntatori è legata alla possibilità di far restituire alle funzioni più di un valore
- **Le funzioni, di base, possono restituire solo un valore!**

- **Esempio**

```
int scambio-somma(int* a, int* b) {  
    int t; int somma;  
    t = *a; // scambio dei valori  
    *a = *b;  
    *b = t;  
  
    somma = *a + *b;  
    return somma;  
}
```

Formalmente, la funzione restituisce solo un valore intero, **ma in realtà restituisce tre valori** perché **scambia anche i valori di a e b.**



# Esercizio 8.1

- **Scrivere un programma che generi random il tempo sul giro, in millisecondi, di cinque diverse macchine di Formula 1. Il tempo deve essere compreso tra 1 minuto e 20 e 1 minuto e 30. I valori devono essere memorizzati in un vettore.**
- Scrivere una funzione che converta il tempo in millisecondi in minuti, secondi e decimi di secondo, e li mostri in output
  - Suggerimento: utilizzare i puntatori per restituire più di un valore
  - **Esempio: 82738 ms. = 1' 22" 738**
- **Scrivere una funzione che ordini i tempi dal più rapido al più lento (opzionale)**
  - Suggerimento: utilizzare lo scambio e il passaggio dei parametri per riferimento

# Esercizio 8.1 (Soluzione)

```
#include <stdio.h>
#include <stdlib.h>
#include "time.h"
#define NUM_AUTO 5 // numero di auto da valutare
#define TEMPO_MIN 80000 // tempo minimo sul giro
#define TEMPO_MAX 90000 // tempo massimo sul giro

// Funzione per la conversione di un tempo in millisecondi nel formato
minuti/secondi/decimi. in questo caso è accettabile usare VOID perché
restituisce più di un valore
void converti(int millisecondi, int* minuti, int* secondi, int* decimi);
```

# Esercizio 8.1 (Soluzione)

```
int main(void) {  
    int tempi[NUM_AUTO] = {0}; // vettore contenente i tempi delle auto  
    int minuti = 0; // variabili di output da mostrare  
    int secondi = 0; int decimi = 0; int seed = time(NULL);  
    srand(seed);  
  
    for(int i = 0; i < NUM_AUTO; i++) {  
        int giro = rand() % (TEMPO_MAX-TEMPO_MIN+1) + TEMPO_MIN;  
        tempi[i] = giro; // genero random un valore sul giro e lo memorizzo  
  
        // PASSAGGIO PER RIFERIMENTO: non voglio perdere il risultato della conversione  
        converti(tempi[i], &minuti, &secondi, &decimi);  
        printf("Auto n.%d:tempo %d'%d' '%d \n", i+1, minuti, secondi, decimi);  
    }  
}
```


# Esercizio 8.1 (Soluzione)

// Funzione di conversione: implementa operazioni matematiche per la conversione

```
void converti(int millisecondi, int* minuti, int* secondi, int* decimi) {  
    *secondi = millisecondi / 1000 ;  
  
    *decimi = millisecondi - (*secondi * 1000);  
    *minuti = *secondi / 60;  
    *secondi = *secondi - (*minuti * 60);  
  
}
```





# Esercizio 8.1 (Soluzione)

// Funzione di conversione: implementa operazioni matematiche per la conversione

```
void converti(int millisecondi, int* minuti, int* secondi, int* decimi) {  
  *secondi = millisecondi / 1000 ;  Es. 88532 ms. / 1000 = 88 secondi  
  
  *decimi = millisecondi - (*secondi * 1000);  
  *minuti = *secondi / 60;  
  *secondi = *secondi - (*minuti * 60);  
  
}
```

# Esercizio 8.1 (Soluzione)

// Funzione di conversione: implementa operazioni matematiche per la conversione

```
void converti(int millisecondi, int* minuti, int* secondi, int* decimi) {  
  *secondi = millisecondi / 1000 ;    
  
  *decimi = millisecondi - (*secondi * 1000);    
  *minuti = *secondi / 60;  
  *secondi = *secondi - (*minuti * 60);  
  
}
```

# Esercizio 8.1 (Soluzione)

// Funzione di conversione: implementa operazioni matematiche per la conversione

```
void converti(int millisecondi, int* minuti, int* secondi, int* decimi) {  
  *secondi = millisecondi / 1000 ; ← Es. 88532 ms. / 1000 = 88 secondi  
  
  *decimi = millisecondi - (*secondi * 1000); ← Es. 88532 - (88 * 1000) = 532 decimi  
  *minuti = *secondi / 60; ← Es. 88 / 60 = 1 minuto  
  *secondi = *secondi - (*minuti * 60);  
  
}
```

# Esercizio 8.1 (Soluzione)

// Funzione di conversione: implementa operazioni matematiche per la conversione

```
void converti(int millisecondi, int* minuti, int* secondi, int* decimi) {  
  *secondi = millisecondi / 1000 ; ← Es. 88532 ms. / 1000 = 88 secondi  
  
  *decimi = millisecondi - (*secondi * 1000); ← 88532 - (88 * 1000) = 532 decimi  
  *minuti = *secondi / 60; ← 88 / 60 = 1 minuto  
  *secondi = *secondi - (*minuti * 60); ← 88 - (1*60) = 28  
  
}
```

**Output: 1' 28" 532**



# Domande?

Cataldo Musto - Puntatori (Parte 1)

Laboratorio di Informatica (ITPS, Track B)

Università degli Studi di Bari – A.A. 2018/2019

