

# Introduzione al linguaggio C

## Puntatori

Violetta Lonati

Università degli studi di Milano  
Dipartimento di Informatica

Laboratorio di algoritmi e strutture dati  
Corso di laurea in Informatica

# Argomenti

## Puntatori

Operatori & e \*

Puntatori come argomento di funzione

Puntatori come valori restituiti

Puntatori a strutture

# Variabili di tipo puntatore

## Indirizzi di memoria

La memoria è divisa in byte, ognuno dei quali ha un indirizzo. Ogni variabile occupa uno o più byte a seconda del suo tipo.

## Variabili puntatore

- ▶ Sono variabili che hanno come **valore** un indirizzo di memoria.
- ▶ Se **p** contiene l'indirizzo di memoria in cui si trova la variabile **i**, diciamo che **p punta a i**.

## Dichiarazione di variabili puntatore

Dato che due variabili di tipo diverso occupano quantità di memoria diversa, è importante specificare che tipo di variabile può puntare un puntatore:

```
int *p;    /* p punta a variabili di tipo int */  
int a, b, *p, n[10]; /* dichiarazione composta*/
```

## Operatori di indirizzo (&) e indirezione (\*)

- ▶ L'operatore & consente di ottenere l'indirizzo di memoria di una variabile:

```
int i, *p; /* dichiaro il puntatore a intero p */  
p = &i; /* assegno a p l'indirizzo di i */
```

- ▶ E' possibile inizializzare una variabile puntatore in fase di dichiarazione:

```
int i;  
int *p = &i;
```

- ▶ L'operatore \* consente di accedere alla variabile puntata dal puntatore:

```
int i = 3, *p = &i;  
printf( "%d", *p ); /* stampa 3 */
```

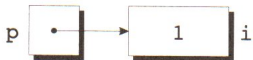
- ▶ Cambiando il valore di \*p si cambia il valore della variabile puntata!

# Esempio

```
p = &i;
```



```
i = 1;
```



```
printf(“%d\n”, i);      /* stampa 1 */  
printf(“%d\n”, *p);    /* stampa 1 */  
*p = 2;
```



```
printf(“%d\n”, i);      /* stampa 2 */  
printf(“%d\n”, *p);    /* stampa 2 */
```

# Assegnamento di puntatori

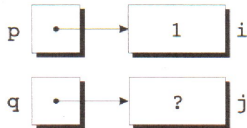
- ▶ Attenzione! Se un puntatore `p` non è inizializzato, il suo valore non è definito, quindi non è definito nemmeno `*p`; `p` potrebbe puntare ad uno spazio di memoria qualsiasi, anche riservato al sistema operativo → segmentation fault!
- ▶ La seguente porzione di programma copia il valore di `p` (cioè l'indirizzo di `i`) dentro `q`, ovvero dopo la copia anche `q` punterà alla variabile `i`.

```
int i, j, *p, *q;  
p = &i;  
q = p; /* copia di puntatori */
```

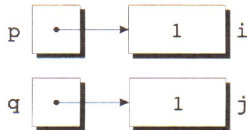
- ▶ Cambiando il valore di `*p` si cambia automaticamente anche il valore di `*q`.
- ▶ Fate attenzione a non confondere `q = p;` con `*q = *p;`: nel secondo caso, il valore della variabile puntata da `p` viene assegnato alla variabile puntata da `q`.

# Esempio

```
p = &i;  
q = &j;  
i = 1;
```



```
*q = *p;
```



## Puntatori come argomento di funzione

- ▶ Passando ad una funzione un puntatore ad una variabile, si può fare in modo che la funzione modifichi il valore della variabile stessa. Invece di passare la variabile `x` dovremo passare come argomento il suo indirizzo, ovvero `&x`.
- ▶ Al momento della chiamata il parametro `p` (di tipo puntatore) corrispondente a `&x` verrà inizializzato col valore di `&x` ovvero con l'indirizzo di `x`.

### Esempio: l'uso di `scanf`

Passo alla funzione `scanf` l'indirizzo della variabile `i` di cui voglio cambiare il valore attraverso la chiamata della funzione `scanf` stessa:

```
int i;  
scanf( "%d", &i );
```

L'operatore `&` nella `scanf` non è richiesto se come argomento uso un punt.:

```
int i, *p = &i;  
scanf( "%d", p );
```

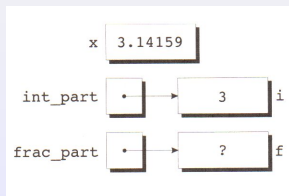
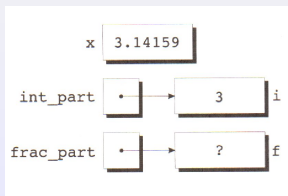
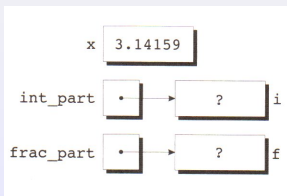


# Esempio

```
void decompose( float x, int *int_part,
               float *frac_part) {
    *int_part = (int) x;
    *frac_part = x - *int_part;
}
```

Se *i* e *f* sono rispettivamente di tipo `int` e `float`, possiamo effettuare la chiamata

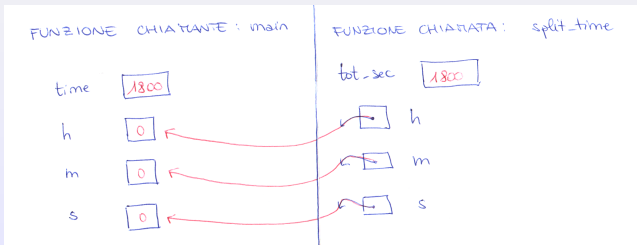
```
decompose( 3.14159, &i, &f);
```



# Esempio

```
void split_time( long int tot_sec, int *h,
                int *m, int *s ) {
    *h = tot_sec / 3600;          tot_sec %= 3600;
    *m = tot_sec / 60;           *s = tot_sec % 60;
}

int main( void ) {
    long int time = 1800;
    int h=0, m=0, s=0;
    split_time( time, &h, &m, &s );
    printf( "h=%d, m=%d, s=%d\n", h, m, s );
    return 0;
}
```



## Puntatori come valori restituiti

E' possibile scrivere funzioni che restituiscono puntatori. Ad esempio:

```
int *max( int *a, int *b) {  
    if ( *a > *b )  
        return a;  
    else  
        return b;  
}
```

Per invocare la funzione `max`, le passiamo due puntatori a variabili `int` e salviamo il risultato in una variabile puntatore:

```
int *p, i, j;  
...  
p = max( &i, &j);
```

Attenzione: non restituite mai un puntatore ad una variabile locale (a meno di averla dichiarata `static`)!

## Puntatori a strutture

Accedere ai membri di una struttura usando un puntatore è un'operazione molto frequente, tanto che il linguaggio C fornisce l'operatore `->` specifico per questo scopo:

```
typedef struct {
    float x, y;
} punto;

typedef struct {
    punto p1, p2;
} rettangolo;

/* stampa i vertici che def. il rett. puntato da r*/
void stampa( rettangolo *r ) {
    printf( "Rett. di vertici (%f, %f) e (%f, %f).\n",
           r -> p1.x, r -> p1.y,
           r -> p2.x, r -> p2.y );
}
```