



UNIVERSITÀ  
degli STUDI  
di CATANIA

# **Puntatori, array vs puntatori, aritmetica dei puntatori, const vs puntatori**

Corso di programmazione I (A-E / O-Z) AA 2025/26

Corso di Laurea Triennale in Informatica

---

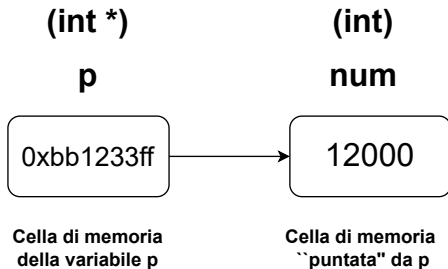
Fabrizio Messina

[fabrizio.messina@unict.it](mailto:fabrizio.messina@unict.it)

Dipartimento di Matematica e Informatica

# Cosa è una variabile puntatore

**Variabile Puntatore:** variabile che contiene un indirizzo di memoria.



Si dice che una variabile puntatore “punta” ad un certo dato, in quanto contiene il suo **indirizzo in memoria**.

# Cosa è una variabile puntatore

Un puntatore rappresenta una differente modalità di accesso alle locazioni di memoria del calcolatore.

```
1  int num = 12000;  
2  int *p = &num; // p “punta” a num
```

# Cosa è una variabile puntatore

## Dichiarazione di una variabile puntatore

```
1  int num;  
2  int *p; //dichiara p come puntatore a int  
3  p = &num; //assegna a p indirizzo di num  
4  *p = 10; //modifica il dato puntato da p
```

Il carattere **\*** anteposto al nome della variabile è denominato operatore di **dereferenziazione** o **indirizione**. Viene usato per

- definire variabili puntatore (linea 2);
- modificare il dato puntato dal puntatore stesso (linea 4)

# Cosa è una variabile puntatore

```
1  int num;  
2  int *p; //dichiara p come puntatore a int  
3  p = &num; //assegna a p indirizzo di num  
4  *p = 10; //modifica il dato puntato da p
```

Il carattere **&** è denominato operatore di **referenziazione** o “address of”.

Esso viene anteposto al nome di una variabile per “estrarre” l’indirizzo di memoria di una certa variabile.

# Cosa è una variabile puntatore

## Dichiarazione puntatore e contestuale inizializzazione

Si dichiara la variabile `p` come puntatore ad un certo tipo (ES: `int`) e assegna ad esso l'indirizzo di una certa variabile (ES: `num`).

```
1  int num = 12000;
2  int *p = &num;
3  printf("%p", p);
4  printf("%d", *p);
```

La linea 3 stampa a video un numero in **formato esadecimale** (INDIRIZZO della cella di memoria, ES: `0x112233aa`).

La linea 4 stampa a video il **valore contenuto nella variabile num** (il DATO), ovvero `12000`.

## Cosa è una variabile puntatore

```
1  int num = 12000;  
2  int k = 20;  
3  int *p = &num;  
4  *p = 34;  
5  p = &k;
```

Il puntatore `p`, in quanto variabile non costante, può essere modificato mediante riassegnamento di altri indirizzi di memoria (linea 5).

# Cosa è una variabile puntatore

Esempi svolti

14\_01.c



# Array vs puntatori

```
1  double v[] = {1.2, 10.7, 9.8};  
2  printf("%p", v) ; //stampa un indirizzo (0x..)  
3  printf("%f", *v); //stampa 1.2  
4  printf("%f", v[0]); //stampa 1.2  
5  
6  double w[] = {3.4, 6.7, 9.8};  
7  v = w; //Errore di compilazione!
```

Il **nome** di una variabile **array** è un **puntatore costante** al primo elemento dello array.

v può essere usata in **espressioni che fanno uso di aritmetica dei puntatori**, ma indirizzo contenuto in v non è modificabile.

## Array vs puntatori

```
1  double v[] = {1.2, 10.7, 9.8};  
2  double *ptr = v;  
3  printf("%f", ptr[1]); //stampa 10.7  
4  printf("%f", ptr[2]); //stampa 9.8
```

Viceversa, con le variabili che sono **puntatori**, se il puntatore punta al primo elemento di un array, si può adottare la ben nota sintassi per l'indicizzazione degli elementi dell'array.

```
1  double v[] = {1.2, 10.7, 9.8};  
2  double *ptr = v;  
3  printf("%f", *(ptr + 1)); //stampa 10.7  
4  printf("%f", *(ptr + 2)); //stampa 9.8  
5  printf("%f", *(v + 2)); //stampa 9.8
```

Se `ptr` è un puntatore, mediante la espressione  $(ptr + x)$  si ottiene l'indirizzo della locazione di memoria **distante  $x$  posizioni rispetto alla locazione puntata da `ptr`.**

**L'incremento è operato dal compilatore, e dipende dalla dimensione in byte del tipo di `ptr`.**

a[0]	0x23aaff40
a[1]	0x23aaff44
a[2]	...
a[3]	
a[4]	
a[5]	
a[6]	
a[7]	
a[8]	
a[9]	

Supponendo `sizeof(int)=4..`

```
1  int a[10];
2  printf("%p", a); // 0x..
3  printf("%p", (a+1)); //0x..
4  printf("%p", &a[1]); //0x..
```

$$0x23aaff44 = 0x23aaff40 + 4$$

Si provi

```
1  double v[] = {1.2, 10.7, 9.8};  
2  double *ptr = v;  
3  printf("%p", ptr);  
4  printf("%p", (ptr+1));  
5  printf("%p", (ptr+2));
```

Alla linea 3 sarà stampato l'indirizzo contenuto in ptr in formato esadecimale.

Alla linea 4 sarà stampato l'indirizzo contenuto in ptr + il valore restituito dall'espressione sizeof(double) in formato esadecimale.

Alla linea 5 ...

Esempi svolti

14\_02.c

# Aritmetica dei puntatori

Accesso ai valori di un array.

Notazione mediante indici vs aritmetica puntatori.

```
int v[] = {1,2,3};  
int *ptr = v;
```

Metodo di accesso	Esempio
Nome array e [ ]	v[2]
Puntatore e [ ]	ptr[2]
Nome array e aritmetica dei puntatori	*(v+2)
Puntatore e aritmetica dei puntatori	*(ptr+2)

```
1  int v[] = {1,2,3};  
2  int *ptr = v;  
3  
4  *(ptr+7) = 90; // errore a run-time  
5  v[4] = 100;   // errore a run-time
```

Le linee di codice 4 e 5 saranno **compile**, senza alcun warning.

Tuttavia quel codice rappresenta **tentativi di accesso (e di modifica) a zone di memoria non allocate per l'applicazione.**



```
1  int v[] = {1,2,3};  
2  int *ptr = v;  
3  
4  *(ptr+7) = 90; // errore a run-time  
5  v[4] = 100;   // errore a run-time
```

Nella maggior parte dei casi il **Sistema Operativo** invierà un segnale di `kill` all'applicazione a causa del **tentativo di accesso a locazioni di memoria non assegnate al processo**.

Operatori consentiti per aritmetica dei puntatori.

- **Operatori unari di incremento** ++/-- applicati ad una variabile puntatore.
- **Operatori binari di addizione e sottrazione** +/– e di **assegnamento** +=, -=, +, –, in cui un membro è un intero e l'altro membro è un puntatore.
- **Operatore di sottrazione – applicato a due puntatori.**  
Ovvero il valore di un puntatore può essere sottratto al valore di un altro puntatore.

## Incremento e decremento unario

```
1  int v[] = {1,2,3,4,5};
2  int *ptr = v;
3  printf("%d", *(++ptr)); //stampa 2
4  printf("%d", *(--ptr)); //stampa 1
5  printf("%d", *(ptr++)); //stampa 1
6  printf("%d", *(ptr)); //stampa 2
```

# Aritmetica dei puntatori

Addizione/sottrazione e assegnamento.

```
1  int v[] = {1,2,3,4,5};
2  int *ptr1 = v; //punta al dato "1"
3  int *ptr2 = &v[4]; //punta al dato "5"
4  printf("%d", *(ptr1+1)); //stampa il dato "2"
5  printf("%d", *(ptr2-1)); //stampa il dato "4"
6  ptr2 -=2;
7  printf("%d", *ptr2); //stampa il dato "3"
8  ptr1 +=1;
9  printf("%d", *ptr1); //stampa il dato "2"
10 printf("%ld", ptr2-ptr1); // stampa 1
```

# Inizializzazione di puntatori

```
1  int *ptr = NULL; //macro C
2
3  float f;
4  int *ptr3 = &f; // Warning del compilatore!
5
6  if(ptr){ // test if ptr is valid
7      //..do something
8  }
```

Linea 7, ptr all'interno di un if per verificare che il puntatore **non** sia nullo.

# Operatore di confronto

Valore vs indirizzo!

```
1  int num;  
2  int *ptr1 = &num;  
3  int *ptr2 = &num;  
4  //confronta indirizzi  
5  if(ptr1==ptr2){  
6      //...  
7  }  
8  //confronta valori  
9  if(*ptr1==*ptr2){  
10     //...  
11 }
```

## Puntatori costanti e puntatori a costanti

```
1  double d1 = 10.9;
2  double d2 = 4.5;
3
4  const double *ptr1 = &d1;
5  *ptr1 = 56.9; //errore!
6  ptr1 = &d2; //OK
```

**ptr1 è un puntatore a costante di tipo double.**

Ciò significa che il **valore alla locazione di memoria puntata da ptr1 non è modificabile** mediante ptr1. La variabile d1 può essere const o non const.

## Puntatori costanti e puntatori a costanti

```
1  double d1 = 10.9;
2  double d2 = 4.5;
3
4  double * const ptr2 = &d2;
5  ptr2 = &d1; //errore!
6  *ptr2 = 10.5; //OK
```

**ptr2 è un puntatore costante ad un tipo double.**

Ciò significa che il **puntatore è una variabile costante**: va inizializzato contestualmente alla sua dichiarazione e non potrà subire riassegnamenti, come una qualunque variabile costante.



## Puntatori costanti e puntatori a costanti

```
1  double d1 = 10.9;
2  double d2 = 4.5;
3
4  const double *const ptr3 = &d2;
5  ptr3 = &d1; //errore!
6  *ptr3 = 10.5; //errore!
```

**ptr3 è un puntatore costante ad una costante di tipo double.**

Ciò significa che il **puntatore è una variabile costante** e che tramite il puntatore stesso non è possibile modificare il valore alla locazione di memoria alla quale esso punta.

# Puntatori costanti e puntatori a costanti

Riassumendo: Const vs puntatori.

Dichiarazione	Istruzione	Ammissibile?
const double *ptr	*ptr = 45.9	NO
	ptr = &x	SI
double * const ptr	*ptr = 45.9	SI
	ptr = &x	NO
const double * const ptr	*ptr = 45.9	NO
	ptr = &x	NO

## Homework H14.1.

Dichiarare ed inizializzare tre variabili, un double, uno short unsigned, ed un char, assegnando valori a piacere.

Dichiarare altrettante variabili puntatore dello stesso tipo ed assegnare ad essi gli indirizzi delle variabili dichiarate in precedenza.

Stampare, mediante la funzione di libreria **printf()** i valori contenuti all'interno di tali variabili in due modi differenti: i) mediante le variabili stesse e ii) mediante le variabili puntatore.

Stampare, mediante printf(), i valori contenuti all'interno dei puntatori (gli indirizzi di memoria).

## Homework H14.2.

1. Dichiarare un array  $D$  di 10 elementi double ed un array  $A$  di 10 elementi int.
2. Inizializzare gli elementi di  $D$  ed  $A$  con numeri pseudo-casuali negli intervalli  $[1.25, 90]$   $[10, 50]$  rispettivamente. NB: Usare sia aritmetica dei puntatori che notazione con parentesi quadre.
3. Dichiarare due variabili puntatore  $ptr\_D$  e  $ptr\_A$  di tipo double e int rispettivamente, ed inizializzare tali puntatori in modo che puntino ai primi elementi di  $D$  ed  $A$  rispettivamente.
4. Codificare opportunamente un ciclo che stampi, su ogni riga di output, i) sia il valore, che ii) l'indirizzo in memoria, di ogni elemento di indice dispari dell'array  $D$ , e di ogni elemento di indice pari dell'array  $A$ . In particolare (segue prox. slide):

- Per stampare i dati, impiegare sia le variabili puntatore che i nomi degli array (ovvero le variabili che rappresentano gli array); usare sia aritmetica dei puntatori (e operatore '\*') che notazione con parentesi quadre in entrambi i casi;
- Per stampare gli indirizzi, impiegare sia le variabili puntatore che i nomi degli array (ovvero le variabili che rappresentano gli array); usare sia aritmetica dei puntatori che notazione con parentesi quadre (e operatore '&');

## Homework H14.3.

1. Dichiarare un array  $V$  di  $N$  double, con  $N=200$ ;
2. Inizializzare gli elementi di  $V$  con numeri pseudo-casuali nell'intervallo  $[10, 50]$ . NB: Usare sia aritmetica dei puntatori che notazione con parentesi quadre.
3. Dichiarare un puntatore al tipo **const double**. Mediante tale puntatore, stampare gli elementi con indici che siano non divisibili per due e non divisibili per tre.
4. Dichiarare un puntatore **const** al tipo **double**. Mediante tale puntatore, sostituire tutti gli elementi dell'array che abbiano gli indici specificati al punto precedente, con un valore double pseudo-casuale in  $[100, 200]$ ;

5. Per i precedenti due punti, per l'accesso agli elementi dell'array, usare i) sia la notazione con parentesi quadre che ii) la notazione che fa uso di aritmetica dei puntatori;

FINE