



**PEGASO**  
Università Telematica





# Indice

1. ESEMPIO: ORDINAMENTO CRESCENTE O DECRESCENTE .....	3
2. ESEMPIO: USO DEI PUNTATORI A FUNZIONI PER REALIZZARE UN SISTEMA GUIDATO DA MENU .....	9
RIFERIMENTI BIBLIOGRAFICI .....	12

## 1. Esempio: ordinamento crescente o decrescente

Un puntatore a una funzione contiene l'indirizzo della funzione nella memoria.

Abbiamo visto che il nome di un array è in realtà l'indirizzo in memoria del primo elemento dell'array.

In modo simile, il nome di una funzione è in realtà l'indirizzo in memoria di partenza del codice che esegue il compito della funzione.

I puntatori a funzioni possono essere passati alle funzioni, restituiti dalle funzioni, memorizzati negli array e assegnati ad altri puntatori a funzioni.

Per illustrare l'uso dei puntatori a funzioni, consideriamo una versione modificata del programma per il bubble sort.

La nuova versione consiste nella funzione `main` e nelle funzioni `bubble`, `swap`, `ascending` e `descending`.

La funzione `bubble` riceve come argomento un puntatore a una funzione – o alla funzione `ascending` o alla funzione `descending` – in aggiunta a un array di interi e alla dimensione dell'array.

Il programma richiede all'utente di scegliere se l'array deve essere ordinato in ordine crescente o decrescente.

Se l'utente inserisce 1, viene passato alla funzione `bubble` un puntatore alla funzione `ascending`, facendo sì che l'array venga ordinato in ordine crescente.

Se l'utente inserisce 2, viene passato alla funzione `bubble` un puntatore alla funzione `descending`, facendo sì che l'array venga ordinato in ordine decrescente.

```
1 // Programma multifunzione
2 // di ordinamento che usa puntatori a funzioni.
3 #include <stdio.h>
4 #define SIZE 10
```

```
5
6 // prototipi
7 void bubble(int work[], size_t size, int (*compare)(int a,int b) );
8 int ascending(int a, int b);
9 int descending(int a, int b);
10
11 int main(void)
12 {
13     // inizializza l'array a non ordinato
14     int a[SIZE] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
15
16     printf("%s", "Enter 1 to sort in ascending order,\n"
17             "Enter 2 to sort in descending order: ");
18     int order; // 1 per l'ordine crescente o 2 per l'ordine decrescente
19     scanf("%d", &order);
20
21     puts("\nData items in original order");
22
23     // invia in uscita l'array originario
24     for (size_t counter = 0; counter < SIZE; ++counter) {
25         printf("%5d", a[counter]);
26     }
27
28     // ordina l'array in ordine crescente; passa la funzione ascending
29     // come argomento per l'ordine crescente dell'ordinamento
30     if (order == 1) {
31         bubble(a, SIZE, ascending);
32         puts("\nData items in ascending order");
33     }
34     else { // passa la funzione descending
35         bubble(a, SIZE, descending);
36         puts("\nData items in descending order");
37     }
38
39     // invia in uscita l'array ordinato
40     for (size_t counter = 0; counter < SIZE; ++counter) {
41         printf("%5d", a[counter]);
42     }
43
```

```
44     puts("\n");
45 }
46
47 // bubble sort multifunzione; il parametro compare e' un puntatore
48 // alla funzione di confronto che determina il tipo di ordinamento
49 void bubble(int work[], size_t size, int (*compare)(int a, int b))
50 {
51     void swap(int *element1Ptr, int *element2Ptr); // prototipo
52
53     // ciclo di controllo per le iterazioni
54     for (unsigned int pass = 1; pass < size; ++pass) {
55
56         // ciclo di controllo per il numero di confronti per iterazione
57         for (size_t count = 0; count < size - 1; ++count) {
58
59             // se elementi adiacenti non sono in ordine, scambiali
60             if ((*compare)(work[count], work[count + 1])) {
61                 swap(&work[count], &work[count + 1]);
62             }
63         }
64     }
65 }
66
67 // scambia i valori alle locazioni di memoria a cui puntano
68 // element1Ptr ed element2Ptr
69 void swap(int *element1Ptr, int *element2Ptr)
70 {
71     int hold = *element1Ptr;
72     *element1Ptr = *element2Ptr;
73     *element2Ptr = hold;
74 }
75
76 // determina se gli elementi non sono in ordine per un ordinamento
77 // di tipo crescente
78 int ascending(int a, int b)
79 {
80     return b < a; // effettua lo scambio se b e' minore di a
81 }
82
```

```
83 // determina se gli elementi non sono in ordine per un ordinamento
84 // di tipo decrescente
85 int descending(int a, int b)
86 {
87     return b > a; // effettuare lo scambio se b e' maggiore di a
88 }
```

Output del programma:

```
Enter 1 to sort in ascending order,
Enter 2 to sort in descending order: 1
Data items in original order
2 6 4 8 10 12 89 68 45 37
Data items in ascending order
2 4 6 8 10 12 37 45 68 89
```

```
Enter 1 to sort in ascending order,
Enter 2 to sort in descending order: 2
Data items in original order
2 6 4 8 10 12 89 68 45 37
Data items in descending order
89 68 45 37 12 10 8 6 4 2
```

Il seguente parametro è dichiarato nell'intestazione della funzione bubble (riga 49)

```
int (*compare)(int a, int b)
```

Questa dichiarazione dice a `bubble` che deve aspettarsi un parametro (`compare`) che è un **puntatore a una funzione** che riceve due parametri interi e restituisce un risultato intero.

Sono necessarie le parentesi attorno a `*compare` per associare `*` a `compare` e indicare che `compare` è un puntatore.

Se non avessimo messo le parentesi, la dichiarazione sarebbe stata

```
int *compare(int a, int b)
```

che dichiara una funzione che riceve due interi come parametri e restituisce un puntatore a un intero.

La riga 7 contiene il prototipo di funzione per `bubble`.

Il terzo parametro nel prototipo avrebbe potuto essere scritto come

```
int (*)(int, int)
```

senza il nome del puntatore a funzione e senza i nomi dei parametri.

La funzione passata a `bubble` viene chiamata in un'istruzione `if` (riga 60) come segue:

```
if ((*compare)(work[count], work[count + 1]))
```

Proprio come un puntatore a una variabile è dereferenziato per accedere al valore della variabile, un puntatore a una funzione è dereferenziato per usare la funzione.

La chiamata alla funzione avrebbe potuto essere fatta senza dereferenziare il puntatore come in

```
if (compare(work[count], work[count + 1]))
```

che usa il puntatore direttamente come nome della funzione.

Preferiamo il primo metodo per chiamare una funzione per mezzo di un puntatore, perché esso fa vedere esplicitamente che `compare` è un puntatore a una funzione che è dereferenziato per chiamare la funzione.



Il secondo metodo per chiamare una funzione per mezzo di un puntatore fa sì che compare sembri una funzione vera.

Ciò può essere fonte di confusione per un programmatore che legge il codice, il quale si aspetterebbe una definizione della funzione compare, mentre essa non è *mai definita* nel file.

## 2. Esempio: Uso dei puntatori a funzioni per realizzare un sistema guidato da menu

Un uso comune dei puntatori a funzioni si ha nei sistemi guidati da menu di tipo testuale. A un utente è richiesto di selezionare un'opzione da un menu (per esempio, da 1 a 5) scrivendo il numero dell'elemento del menu.

Ogni opzione è realizzata da una funzione differente.

I puntatori alle funzioni sono memorizzati in un array di puntatori a funzioni.

La scelta dell'utente è usata come indice dell'array e i puntatori nell'array sono usati per chiamare le funzioni.

Il seguente programma fornisce un esempio generico dei meccanismi utilizzati per definire e usare un array di puntatori a funzioni.

Definiamo tre funzioni - `function1`, `function2` e `function3` - che ricevono ciascuna un argomento intero e non restituiscono niente.

Memorizziamo i puntatori a queste tre funzioni nell'array `f`, definito nella riga 14.

La definizione si legge iniziando dall'insieme di parentesi più a sinistra, "`f` è un array di 3 puntatori a funzioni che ricevono ognuna un `int` come argomento e restituiscono `void`".

L'array è inizializzato con i nomi delle tre funzioni.

Quando l'utente inserisce un valore tra 0 e 2, il valore viene usato come indice nell'array dei puntatori alle funzioni.

Nella chiamata di funzione (riga 25), `f[choice]` seleziona il puntatore nella locazione `choice` nell'array.

Il puntatore è dereferenziato per chiamare la funzione e `choice` viene passato come argomento alla funzione.

Ogni funzione stampa il valore del suo argomento e il suo nome per confermare che la funzione è stata chiamata correttamente.

Negli esercizi di questo capitolo svilupperete diversi sistemi guidati da menu di tipo testuale.

```
1 // Programma con
2 // esempio di un array di puntatori a funzioni.
3 #include <stdio.h>
4
5 // prototipi
6 void function1(int a);
7 void function2(int b);
8 void function3(int c);
9
10 int main(void)
11 {
12     // inizializza un array di 3 puntatori a funzioni che ricevono
13     // ognuna un argomento int e restituiscono void
14     void (*f[3])(int) = { function1, function2, function3 };
15
16     printf("%s", "Enter a number between 0 and 2, 3 to end: ");
17     size_t choice; // variabile che contiene la scelta dell'utente
18     scanf("%u", &choice);
19
20     // elabora la scelta dell'utente
21     while (choice >= 0 && choice < 3) {
22
23         // invoca la funzione alla locazione choice nell'array f e
24         // passa choice come argomento
25         (*f[choice])(choice);
26
27         printf("%s", "Enter a number between 0 and 2, 3 to end: ");
28         scanf("%u", &choice);
29     }
30
31     puts("Program execution completed.");
32 }
33
34 void function1(int a)
35 {
36     printf("You entered %d so function1 was called\n\n", a);
37 }
```

```
38
39 void function2(int b)
40 {
41     printf("You entered %d so function2 was called\n\n", b);
42 }
43
44 void function3(int c)
45 {
46     printf("You entered %d so function3 was called\n\n", c);
47 }
```

Enter a number between 0 and 2, 3 to end: 0

You entered 0 so function1 was called

Enter a number between 0 and 2, 3 to end: 1

You entered 1 so function2 was called

Enter a number between 0 and 2, 3 to end: 2

You entered 2 so function3 was called

Enter a number between 0 and 2, 3 to end: 3

Program execution completed.

## Riferimenti bibliografici

- Paul Deitel, Harvey Deitel, "Il linguaggio C – Fondamenti e tecniche di programmazione",  
Libro edito da Pearson Italia. Include anche utili esercizi di autovalutazione.