



Programmazione 1 (Canale O-Z)

Lezione 6: Array Monodimensionali Statici e Algoritmi Base

Docente: Prof. Antonino Furnari

Email: antonino.furnari@unict.it

Sito Web: antoninofurnari.github.io/prog1-2627

🤔 Il Problema: Troppi Esami

Vogliamo leggere i voti di 5 esami e calcolarne la media. Con variabili scalari:

```
int voto1, voto2, voto3, voto4, voto5;  
scanf("%d", &voto1);  
scanf("%d", &voto2);  
// ... e così via per ogni voto
```

- Con 100 esami? Con 10.000 misure?
- **Rileggere** i voti dopo la media? **Impossibile:** ogni valore viene scartato.



? Come raggruppare N variabili dello stesso tipo in un'unica struttura indicizzata?

L'Array: Definizione e Sintassi

Un array è una sequenza di celle di memoria **contigue**, ciascuna dello stesso tipo, accessibili tramite un **indice numerico**.

```
tipo nome_array[dimensione];
```

Esempi:

```
int    voti[5];           // 5 x 4 byte = 20 byte contigui
double misure[10];       // 10 x 8 byte = 80 byte contigui
char   lettere[26];       // 26 x 1 byte = 26 byte contigui
```

- Le celle sono allocate nello **Stack**, la zona di RAM temporanea della funzione corrente.
- Grazie alla contiguità, la CPU calcola l'indirizzo di qualsiasi elemento **in tempo costante**.

1234 Indicizzazione Base-0 e Layout in RAM

Indice	0	1	2	3	4
Elemento	voti[0]	voti[1]	voti[2]	voti[3]	voti[4]
Indirizzo (fittizio)	1000	1004	1008	1012	1016

- Gli indici partono sempre da `0` e arrivano a `N-1` (mai a `N`!).
- Ogni `int` occupa 4 byte: le celle sono **una dopo l'altra**, a distanza di 4 byte.
- L'indice è un **offset** rispetto all'indirizzo base: il primo elemento è esattamente all'indirizzo base (scostamento `0`). Lo capiremo ancora meglio con i puntatori.

? Cosa succede se accediamo a `voti[5]` ?

Out-of-Bounds: L'Errore Silenzioso

Accedere a `voti[5]` su un array di 5 elementi (indici validi: `0 – 4`) è **Undefined Behavior**.

- Il compilatore **non** segnala alcun errore.
 - Il programma legge o scrive silenziosamente in una zona di memoria che appartiene a **un'altra variabile**.
 - Risultati imprevedibili: dati casuali, memoria corrotta, crash.
- È uno degli errori **più pericolosi e frequenti** in C: controllate sempre i limiti dei vostri cicli!



Inizializzazione alla Dichiarazione

```
int voti[5] = {28, 30, 25, 27, 18}; // inizializzazione completa
int zeri[5] = {0};                  // tutti gli elementi inizializzati a 0
int parziale[5] = {10, 20};        // [2], [3], [4] posti automaticamente a 0
```

- Con un inizializzatore **parziale**, gli elementi non specificati valgono `0`.
- `{0}` è il modo comune per **azzerare** un intero array.

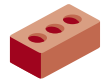
? E se non inizializziamo affatto l'array?

Array Non Inizializzato: Garbage Value

```
#include <stdio.h>
#define N 5
int main(void) {
    int voti[N]; // nessuna inizializzazione!
    // SBAGLIATO: stampiamo i valori prima di assegnarli
    for (int i = 0; i < N; i++) {
        printf("voti[%d] = %d\n", i, voti[i]); // output imprevedibile!
    }
    return 0;
}
```

- Le celle contengono i **byte residui** dello Stack: 0, 32767, -858993460, ... del tutto casuali.
-  Inizializzare sempre: con una lista, con {0} o leggendo i valori con scanf prima di usarli.

 **Live:** gcc -Wall garbage.c -o garbage && ./garbage, eseguire più volte e osservare i valori.



#define N e Accesso agli Elementi

Definire la dimensione come **costante simbolica**: si modifica in un solo punto.

```
#define N 5 // cambiando questo numero, si aggiorna tutto il codice che usa N

int main(void) {
    int voti[N];
    // ...
}
```

L'accesso ai singoli elementi avviene con l'operatore `[]`:

```
voti[0] = 28; // scrittura: assegna 28 al primo elemento
printf("%d\n", voti[2]); // lettura: stampa il terzo elemento (indice 2)
```

La Dimensione di un Array: `sizeof`

`sizeof` applicato a un array statico restituisce la **dimensione totale in byte**, non il numero di elementi.

```
int voti[5];  
printf("Spazio occupato: %zu byte\n", sizeof(voti)); // Stampa 20 (5 elementi * 4 byte)
```

Numero di elementi = spazio totale / spazio di una cella (calcolato a compile-time):

```
int lunghezza = sizeof(voti) / sizeof(voti[0]); // 20 byte / 4 byte = 5 elementi  
  
for (int i = 0; i < (sizeof(voti) / sizeof(voti[0])); i++) {  
    // codice di scansione  
}
```

⚠ Funziona **solo nello scope** in cui l'array è dichiarato: passato a una funzione, l'array decade a puntatore (*Array Decay*) e `sizeof` restituisce la dimensione del puntatore (8 byte a 64 bit).

Il Pattern Universale: `for` sull'Array

Quasi tutti gli algoritmi su array seguono lo stesso schema: un `for` che scorre gli indici da `0` a `N-1`.

```
for (int i = 0; i < N; i++) {  
    // opera su arr[i]  
}
```

- Memorizzatelo come un **blocco atomico di pensiero**.
- Condizione `i < N` (e non `i ≤ N`): l'ultimo indice valido è `N-1`.

? Come usiamo questo pattern per leggere 5 voti da tastiera e poi ristamparli?



Lettura e Stampa di un Array

```
#include <stdio.h>
#define N 5
int main(void) {
    int voti[N];
    // Lettura dell'array
    printf("Inserisci %d voti:\n", N);
    for (int i = 0; i < N; i++) {
        printf("Voto %d: ", i + 1); // i+1 perche l'utente conta da 1
        scanf("%d", &voti[i]);
    }
    // Stampa dell'array
    printf("\nVoti inseriti: ");
    for (int i = 0; i < N; i++) {
        printf("%d ", voti[i]);
    }
    printf("\n");
    return 0;
}
```

- **Due cicli distinti:** con un array possiamo ripercorrere i dati **quante volte vogliamo**.

 **Live:** gcc -Wall voti.c -o voti && ./voti, inserire 28 30 25 27 18.



Algoritmo: Calcolo della Media

```
#include <stdio.h>
#define N 5

int main(void) {
    int voti[N] = {28, 30, 25, 27, 18};
    int somma = 0;

    for (int i = 0; i < N; i++) {
        somma += voti[i];
    }

    double media = (double)somma / N; // cast esplicito per la divisione reale!
    printf("Media: %.2f\n", media);
    return 0;
}
```

Media: 25.60

- Schema: **accumulatore** → somma nel ciclo, divisione **una sola volta** alla fine.

! Il Cast (double) è Obbligatorio

- `somma` vale `28 + 30 + 25 + 27 + 18 = 128`.
- Senza cast, `somma / N` è una **divisione intera**: `128 / 5` ➡ `25`.
- Con `(double)somma / N` il dividendo diventa `double` ➡ `25.60`.

🎬 **Live:** togliere il cast, ricompilare con `gcc -Wall media.c -o media` ed eseguire ➡ `Media: 25.00`.

? Ora vogliamo sapere **quali** voti superano la media. Con le variabili scalari della Lezione 4 avremmo potuto farlo?



La Seconda Passata: il Vero Valore degli Array

Con variabili scalari sarebbe **impossibile**: ogni valore letto veniva scartato. Con l'array, facciamo una **seconda passata**:

```
// ... dopo aver calcolato 'media' ...  
  
printf("\nVoti superiori alla media (%.2f):\n",  
      media);  
for (int i = 0; i < N; i++) {  
    if (voti[i] > media) {  
        printf("    - voti[%d] = %d\n", i, voti[i]);  
    }  
}
```

```
Voti superiori alla media (25.60):  
- voti[0] = 28  
- voti[1] = 30  
- voti[3] = 27
```

- `voti[i] > media`: `media` è `double`, quindi `voti[i]` viene **promosso** automaticamente a `double`.

Ricerca Lineare — Pattern A: SE Esiste (Flag)

```
#define N 5
int voti[N] = {28, 30, 25, 27, 18};
int target = 30;
int trovato = 0; // 0 = non trovato, 1 = trovato

for (int i = 0; i < N; i++) {
    if (voti[i] == target) {
        trovato = 1;
        break; // inutile continuare
    }
}

if (trovato) {
    printf("Valore %d trovato nell'array.\n", target);
} else {
    printf("Valore %d non presente.\n", target);
}
```

- Si esaminano le celle **una per una** dall'inizio, finché si trova l'elemento o si esaurisce l'array.
- Come cercare un calzino rosso in un cassetto disordinato!

Pattern A: Traccia Passo per Passo

`voti = {28, 30, 25, 27, 18}`, `target = 30`

i	voti[i]	voti[i] = 30 ?	trovato	Azione
—	—	—	0	inizializzazione
0	28	✗	0	prosegue
1	30	✓	1	<code>break</code> : esce dal ciclo

Valore 30 trovato nell'array.

- Gli indici 2, 3, 4 **non vengono mai esaminati**: il `break` evita scansioni inutili.

? E se volessimo sapere anche **dove** si trova il valore?

Ricerca Lineare — Pattern B: DOVE si Trova (Indice)

```
int indice = -1; // -1 indica "non trovato" (nessun indice valido e' negativo)

for (int i = 0; i < N; i++) {
    if (voti[i] == target) {
        indice = i;
        break;
    }
}

if (indice != -1) {
    printf("Valore %d trovato alla posizione %d.\n", target, indice);
} else {
    printf("Valore %d non presente.\n", target);
}
```

Valore 30 trovato alla posizione 1.

- -1 è la sentinella universale per "non trovato": gli indici validi sono sempre ≥ 0 .



Algoritmo: Massimo e Minimo

```
#include <stdio.h>
#define N 5

int main(void) {
    int voti[N] = {28, 30, 25, 27, 18};

    int max = voti[0]; // inizializzazione corretta: primo elemento
    int min = voti[0];
    int pos_max = 0;    // teniamo anche l'indice della posizione
    int pos_min = 0;

    for (int i = 1; i < N; i++) { // partiamo da 1, non da 0!
        if (voti[i] > max) {
            max = voti[i];
            pos_max = i;
        }
        if (voti[i] < min) {
            min = voti[i];
            pos_min = i;
        }
    }

    printf("Voto massimo: %d (posizione %d)\n", max, pos_max);
    printf("Voto minimo:  %d (posizione %d)\n", min, pos_min);
    return 0;
}
```



Massimo e Minimo: Traccia Passo per Passo

i	voti[i]	max	pos_max	min	pos_min
init	28	28	0	28	0
1	30	30	1	28	0
2	25	30	1	25	2
3	27	30	1	25	2
4	18	30	1	18	4

Voto massimo: 30 (posizione 1)
Voto minimo: 18 (posizione 4)

- Si inizializza con `voti[0]` (non con `0`: e se fossero tutti negativi?) e il ciclo parte da `i = 1`.



Esempio Integrato: Analisi Statistica dei Voti

```
#include <stdio.h>
#define N 5

int main(void) {
    int voti[N];
    int somma = 0;

    // Acquisizione con validazione do-while (ripasso L5)
    printf("Inserisci %d voti (18-30):\n", N);
    for (int i = 0; i < N; i++) {
        do {
            printf("Voto %d: ", i + 1);
            scanf("%d", &voti[i]);
        } while (voti[i] < 18 || voti[i] > 30);
        somma += voti[i];
    }

    // Calcolo media
    double media = (double)somma / N;
```

```
    // Calcolo max, min e conteggio in un unico ciclo
    int max = voti[0], min = voti[0];
    int superano_media = 0;
    for (int i = 0; i < N; i++) {
        if (voti[i] > max) max = voti[i];
        if (voti[i] < min) min = voti[i];
        if (voti[i] > media) superano_media++;
    }

    // Report finale
    printf("\n--- Analisi Voti ---\n");
    printf("Media:                %.2f\n", media);
    printf("Massimo:                %d\n", max);
    printf("Minimo:                 %d\n", min);
    printf("Sopra la media:         %d/%d\n", superano_media, N);

    return 0;
}
```

 **Live:** gcc -Wall analisi.c -o analisi && ./analisi, provare un voto non valido (35), poi 28 30 25 27 18
➡ media 25.60, max 30, min 18, sopra la media 3/5.

Riepilogo degli Algoritmi

Algoritmo	Inizializzazione chiave	Ciclo parte da
Media	<code>somma = 0</code>	<code>i = 0</code>
Massimo / Minimo	<code>max = arr[0]</code>	<code>i = 1</code>
Ricerca (flag)	<code>trovato = 0</code>	<code>i = 0</code>
Ricerca (indice)	<code>indice = -1</code>	<code>i = 0</code>

? Media e massimo si possono calcolare già durante l'acquisizione? E la stampa dei voti sopra la media?

✓ Cosa Abbiamo Imparato Oggi

- Un array è una sequenza di celle **contigue** dello stesso tipo, nello **Stack**.
- Indici da `0` a `N-1`; uscire dai limiti è **Undefined Behavior** (nessun errore dal compilatore!).
- Inizializzare sempre (`{...}`, `{0}`, `scanf`): mai leggere garbage value.
- `#define N` per la dimensione; `sizeof(arr) / sizeof(arr[0])` per contare gli elementi.
- `for (int i = 0; i < N; i++)` è il mattone di media, ricerca lineare, massimo e minimo.
- La **seconda passata** è ciò che le variabili scalari non possono fare.

 **Per casa:** esercizi della Lezione 6 sul sito del corso.

Grazie per l'attenzione!

Sito Web del Corso: antoninofurnari.github.io/prog1-2627