

LEZIONE 6

Array Monodimensionali Statici e Algoritmi Base



Dispense di Programmazione 1 – Canale O-Z

Prof. Antonino Furnari

Corso di Laurea in Informatica

Università di Catania

Lezione 1

Array Monodimensionali Statici e Algoritmi Base

Nelle lezioni precedenti abbiamo imparato a gestire variabili scalari, cicli e strutture di controllo. Tuttavia, ogni valore elaborato veniva subito “dimenticato”: con il `while` di Lezione 4, per esempio, non era possibile tornare indietro a rileggere un dato già processato. Questa lezione introduce il costrutto che risolve il problema: l'**array monodimensionale statico**. Un array è una sequenza di variabili dello stesso tipo allocate in modo contiguo in memoria, accessibili tramite un indice numerico. Impareremo a dichiararlo, inizializzarlo e a scrivere i tre algoritmi fondamentali che si applicano a qualsiasi sequenza di dati: calcolo della media, ricerca lineare e individuazione di massimo e minimo.

1.1 Motivazione e Anatomia degli Array

1.1.1 Il Problema che Motiva l'Array

Supponiamo di voler leggere i voti di 5 esami e calcolarne la media. Con le variabili scalari che conosciamo, saremmo costretti a scrivere:

```
1 int voto1, voto2, voto3, voto4, voto5;
2 scanf("%d", &voto1);
3 scanf("%d", &voto2);
4 // ... e cos? via per ogni voto
```

Questo approccio è insostenibile per 100 esami, e soprattutto è impossibile rielaborare i dati: una volta calcolata la media, non possiamo “tornare indietro” a scoprire, ad esempio, quali voti la superano.

L'**array** risolve entrambi i problemi: raggruppa N variabili dello stesso tipo in un'unica struttura indicizzata, e le mantiene tutte in memoria per tutto il tempo necessario.

1.1.2 Dichiarazione e Layout in RAM

La sintassi di dichiarazione è:

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

Esempi pratici:

```
1 int voti[5];           // 5 interi contigui in RAM (5 x 4 byte = 20 byte)
2 double misure[10];    // 10 double contigui in RAM (10 x 8 byte = 80 byte)
3 char lettere[26];     // 26 char contigui in RAM (26 x 1 byte = 26 byte)
```

Le celle di un array sono allocate in modo **contiguo** nello Stack — la zona di memoria temporanea riservata dal sistema operativo per le variabili locali della funzione corrente. Grazie alla contiguità, la CPU può calcolare l'indirizzo di qualsiasi elemento in tempo costante, senza dover scorrere la memoria.

Ogni cella ha un proprio **indice** che parte sempre da 0 e arriva a N-1:

Tabella

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 indirizzi dell'ultima riga sono fittizi, ma mostrano come vengono salvati i dati: ogni `int` occupa 4 byte, quindi le celle si trovano una dopo l'altra, a distanza di 4 byte l'una dall'altra.

Nota

Perché gli indici partono da 0? L'indice è in realtà un **offset** (scostamento) rispetto all'indirizzo base dell'array. Il primo elemento si trova esattamente all'indirizzo base, quindi il suo scostamento è 0. Questo diventerà ancora più chiaro quando studieremo i puntatori.

Attenzione

Out-of-Bounds: L'Errore Silenzioso Accedere a `voti[5]` su un array di 5 elementi (indici validi: 0-4) è un **comportamento non definito** (*Undefined Behavior*) in C. Il compilatore **non** segnala alcun errore: il programma legge o scrive silenziosamente in una zona di memoria che appartiene ad un'altra variabile, con risultati imprevedibili. È uno degli errori più pericolosi e frequenti in C.

1.1.3 Inizializzazione e Uso di `#define N`

Un array può essere inizializzato alla dichiarazione tramite una lista di valori tra parentesi graffe:

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

Attenzione

Array non inizializzato: Garbage Value Se un array viene dichiarato senza inizializzazione, le sue celle contengono **valori spazzatura** (*garbage value*): i byte residui rimasti nello Stack dalle esecuzioni precedenti, del tutto casuali e imprevedibili.

```

1  #include <stdio.h>
2  #define N 5
3
4  int main(void) {
5      int voti[N]; // nessuna inizializzazione!
6
7      // SBAGLIATO: stampiamo i valori prima di assegnarli
8      for (int i = 0; i < N; i++) {
9          printf("voti[%d] = %d\n", i, voti[i]); // output
          ↪ imprevedibile!
10     }
11     return 0;
12 }
```

L'output di questo programma sarà completamente casuale — potrebbe essere 0 (se lo Stack è stato azzerato dal sistema operativo), oppure numeri enormi come 32767, -858993460 o qualsiasi altro residuo di memoria. Questo comportamento è **non definito** dallo standard C, il che significa che il compilatore non è obbligato a produrre un risultato coerente.

Come evitarlo: Inizializza sempre un array prima di usarlo, o con una lista esplicita, o con `\{\}\{0\}` per azzerarlo, o leggendo i valori dall'utente con `scanf` prima di accedervi.

```

1  int voti[5] = {0}; // corretto: tutti gli elementi a 0
2  int voti[5] = {28, 30, 25, 27, 18}; // corretto: valori espliciti
```

Quando rilevante, è buona pratica definire la dimensione dell'array come costante simbolica tramite `\{\}\#define`, così da poterla modificare in un solo punto:

```

1  #define N 5 // cambiando questo numero, si aggiorna tutto il codice che
    ↪ usa N
2
3  int main(void) {
4      int voti[N];
5      // ...
6  }
```

L'accesso ai singoli elementi avviene tramite l'operatore di indicizzazione `[]`:

```

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

1.1.4 La dimensione di un Array: L'operatore sizeof

L'operatore `sizeof` restituisce lo spazio occupato in memoria da un tipo di dato o da una variabile, espresso in **byte**. Quando viene applicato a un array statico, `sizeof` restituisce la **dimensione totale in byte** dell'intero array, non il numero di elementi.

Ad esempio, se abbiamo un array di 5 interi:

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

Per determinare in modo automatico il **numero di elementi** (la lunghezza dell'array) senza cablare la costante nel codice, si divide la dimensione totale dell'array per la dimensione di un singolo elemento (solitamente il primo elemento a indice 0):

```
1 int lunghezza = sizeof(voti) / sizeof(voti[0]);
2 // Equivale a: 20 byte / 4 byte = 5 elementi
```

Questa espressione viene calcolata a compile-time (durante la compilazione) ed è estremamente comoda per scrivere cicli generici in cui non vogliamo hardcodare la costante numerica:

```
1 for (int i = 0; i < (sizeof(voti) / sizeof(voti[0])); i++) {
2     // codice di scansione
3 }
```

Attenzione

Attenzione al ciclo di vita della dimensione L'operatore `sizeof` restituisce la dimensione corretta dell'array **solo** all'interno dello stesso scope (blocco di codice) in cui l'array è stato dichiarato. Se l'array viene passato come parametro a una funzione, decade a un semplice puntatore (Array Decay) e `sizeof` restituirà la dimensione del puntatore (8 byte su sistemi a 64-bit), rendendo impossibile risalire alla lunghezza originaria in questo modo.

1.2 Il Pattern di Scansione e gli Algoritmi Base

1.2.1 Il Ciclo for come Strumento Universale sugli Array

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

```
1 for (int i = 0; i < N; i++) {
2     // opera su arr[i]
3 }
```

Questo pattern è così ricorrente che vale la pena memorizzarlo come un blocco atomico di pensiero. Vediamo come usarlo per leggere e stampare un array:

```

1  #include <stdio.h>
2  #define N 5
3
4  int main(void) {
5      int voti[N];
6
7      // Lettura dell'array
8      printf("Inserisci %d voti:\n", N);
9      for (int i = 0; i < N; i++) {
10         printf("Voto %d: ", i + 1); // i+1 perche l'utente conta da 1
11         scanf("%d", &voti[i]);
12     }
13
14     // Stampa dell'array
15     printf("\nVoti inseriti: ");
16     for (int i = 0; i < N; i++) {
17         printf("%d ", voti[i]);
18     }
19     printf("\n");
20
21     return 0;
22 }

```

Si noti che usiamo **due cicli distinti**: uno per la lettura, uno per la stampa. Con un array possiamo ripercorrere i dati tutte le volte che vogliamo – cosa impossibile con variabili scalari.

1.2.2 Algoritmo: Calcolo della Media

La media aritmetica si calcola accumulando la somma in un unico ciclo e dividendo una sola volta alla fine.

```

1  #include <stdio.h>
2  #define N 5
3
4  int main(void) {
5      int voti[N] = {28, 30, 25, 27, 18};
6      int somma = 0;
7
8      for (int i = 0; i < N; i++) {
9          somma += voti[i];
10     }
11
12     double media = (double)somma / N; // cast esplicito per la divisione
    ⇨ reale!
13     printf("Media: %.2f\n", media);
14
15     return 0;
16 }

```

Attenzione

Il cast (double) è obbligatorio. Senza il cast esplicito, `somma / N` sarebbe una divisione intera che tronca il risultato decimale. Ad esempio, `128 / 5` darebbe 25 invece di 25.60. Il cast (double) `somma` converte il dividendo in virgola mobile prima della divisione.

1.2.3 La Prima Vera Dimostrazione del Valore degli Array: la Seconda Passata

Dopo aver calcolato la media, possiamo fare una **seconda passata** sull'array per scoprire quali voti la superano. Questa operazione sarebbe **impossibile** senza un array: con variabili scalari, ogni valore inserito dall'utente viene scartato dopo l'elaborazione e non può essere riletto.

```
1 // ... dopo aver calcolato 'media' con il ciclo precedente ...
2
3 printf("\nVoti superiori alla media (%.2f):\n", media);
4 for (int i = 0; i < N; i++) {
5     if (voti[i] > media) {
6         printf(" - voti[%d] = %d\n", i, voti[i]);
7     }
8 }
```

Il confronto `voti[i] > media` funziona correttamente: poiché `media` è un `double`, il compilatore promuove automaticamente `voti[i]` (intero) a `double` prima del confronto.

1.3 Algoritmi di Ricerca e di Estremi

1.3.1 Algoritmo: Ricerca Lineare (Sequential Search)

La ricerca lineare è l'algoritmo più semplice per trovare un elemento in un array non ordinato: si esaminano le celle una per una dall'inizio, finché non si trova l'elemento cercato o si esaurisce l'array.

Esistono due varianti comuni:

Pattern A: Determinare SE l'elemento esiste (flag booleano)

```
1 #define N 5
2 int voti[N] = {28, 30, 25, 27, 18};
3 int target = 30;
4 int trovato = 0; // flag: 0 = non trovato, 1 = trovato
5
6 for (int i = 0; i < N; i++) {
7     if (voti[i] == target) {
8         trovato = 1;
9         break; // inutile continuare una volta trovato
10    }
11 }
```

```

12
13 if (trovato) {
14     printf("Valore %d trovato nell'array.\n", target);
15 } else {
16     printf("Valore %d non presente.\n", target);
17 }

```

Pattern B: Determinare DOVE si trova (indice sentinella)

```

1 int indice = -1; // -1 indica "non trovato" (nessun indice valido e'
    ↪ negativo)
2
3 for (int i = 0; i < N; i++) {
4     if (voti[i] == target) {
5         indice = i;
6         break;
7     }
8 }
9
10 if (indice != -1) {
11     printf("Valore %d trovato alla posizione %d.\n", target, indice);
12 } else {
13     printf("Valore %d non presente.\n", target);
14 }

```

Nota

Perché usare -1 come sentinella? Gli indici validi di un array sono sempre ≥ 0 . Il valore -1 è quindi impossibile come indice reale e viene usato universalmente in C come segnale di "non trovato". Il break è importante per l'efficienza: una volta trovato l'elemento, non ha senso continuare la scansione.

1.3.2 Algoritmo: Massimo e Minimo su Array

L'algoritmo per trovare il massimo e il minimo di un array segue la stessa logica già vista in Lezione 4 per la sequenza fissa: si inizializzano le variabili di riferimento con il **primo elemento** dell'array (non con 0 o un valore arbitrario), poi si scorre il resto dell'array dal secondo elemento in poi.

```

1 #include <stdio.h>
2 #define N 5
3
4 int main(void) {
5     int voti[N] = {28, 30, 25, 27, 18};
6
7     int max = voti[0]; // inizializzazione corretta: primo elemento
8     int min = voti[0];
9     int pos_max = 0;   // teniamo anche l'indice della posizione
10    int pos_min = 0;
11

```

```
12  for (int i = 1; i < N; i++) { // partiamo da 1, non da 0!
13      if (voti[i] > max) {
14          max = voti[i];
15          pos_max = i;
16      }
17      if (voti[i] < min) {
18          min = voti[i];
19          pos_min = i;
20      }
21  }
22
23  printf("Voto massimo: %d (posizione %d)\n", max, pos_max);
24  printf("Voto minimo: %d (posizione %d)\n", min, pos_min);
25
26  return 0;
27 }
```

Nota

Perché il ciclo parte da $i = 1$? L'elemento `voti[0]` è già stato usato come valore di partenza per `max` e `min`. Confrontarlo con sé stesso sarebbe corretto ma ridondante. Il ciclo parte da $i = 1$ per confrontare tutti i rimanenti elementi con il riferimento iniziale.

1.3.3 Esempio Integrato: Analisi Statistica dei Voti

Quest'ultimo esempio mette insieme tutti gli algoritmi visti: leggiamo N voti validati da tastiera, calcoliamo media, massimo e minimo, e contiamo i voti superiori alla media.

```
1  #include <stdio.h>
2  #define N 5
3
4  int main(void) {
5      int voti[N];
6      int somma = 0;
7
8      // Acquisizione con validazione do-while (ripasso Lezione 5)
9      printf("Inserisci %d voti (18-30):\n", N);
10     for (int i = 0; i < N; i++) {
11         do {
12             printf("Voto %d: ", i + 1);
13             scanf("%d", &voti[i]);
14         } while (voti[i] < 18 || voti[i] > 30);
15         somma += voti[i];
16     }
17
18     // Calcolo media
19     double media = (double)somma / N;
20
21     // Calcolo max, min e conteggio in un unico ciclo
22     int max = voti[0], min = voti[0];
23     int superano_media = 0;
24     for (int i = 0; i < N; i++) {
```

```
25     if (voti[i] > max) max = voti[i];
26     if (voti[i] < min) min = voti[i];
27     if (voti[i] > media) superano_media++;
28 }
29
30 // Report finale
31 printf("\n--- Analisi Voti ---\n");
32 printf("Media:           %.2f\n", media);
33 printf("Massimo:          %d\n", max);
34 printf("Minimo:           %d\n", min);
35 printf("Sopra la media: %d/%d\n", superano_media, N);
36
37 return 0;
38 }
```

1.4 Quiz di Autovalutazione

Quiz 1

Domanda

1. Qual è il primo indice valido di un array in C?

- A) 1
- B) -1
- C) 0
- D) Dipende dal tipo dell'array

Soluzione
Risposta corretta: C
In C gli indici degli array partono sempre da 0. L'ultimo elemento valido di un array di N elementi si trova all'indice N-1.

Quiz 2

Domanda

2. Se dichiaro `int arr[8];`, quali sono gli indici validi?

- A) Da 0 a 8
- B) Da 1 a 8
- C) Da 0 a 7
- D) Da 1 a 7

Un array di dimensione N ha indici validi da 0 a $N-1$. Per `arr[8]`, gli indici validi sono da 0 a 7 . Accedere ad `arr[8]` è un accesso fuori dai limiti (out-of-bounds).

Risposta corretta: C

Soluzione

Quiz 3

Domanda

3. Cosa produce il codice `int a[3] = \{\}\{0\}\{\}\};`?

- A) Errore di compilazione: mancano tutti i valori.
- B) Un array di 3 elementi, tutti inizializzati a 0.
- C) Un array di 3 elementi, solo il primo a 0, gli altri con garbage value.
- D) Un array di 1 elemento con il valore 0.

In C, quando si fornisce un iniziatore parziale, tutti gli elementi non specificati vengono automaticamente a 0. \{\}\} è un modo comune e corretto per azzerare un intero array.

Risposta corretta: B

Soluzione

Quiz 4

Domanda

4. Perché è necessario il cast (double) somma nell'espressione (double) somma / N?

- A) Per aumentare la velocità di esecuzione.
- B) Per evitare l'overflow del tipo `int`.
- C) Per forzare una divisione in virgola mobile invece che una divisione intera che tronca il risultato.
- D) Perché `N` è sempre un `double`.

Senza il cast, somma / N sarebbe una divisione intera. Per esempio, 128 / 5 darebbe 25 invece di 25.6. Il cast (double) somma converte il dividendo in double prima della divisione, forzando un'operazione in virgola mobile.

Risposta corretta: C

Soluzione

Quiz 5

Domanda

5. Quale valore si usa convenzionalmente per inizializzare indice a "elemento non trovato" nella ricerca lineare?

- A) 0
- B) N (la dimensione dell'array)
- C) -1
- D) `INT\{}\_MAX`

Il valore -1 è la convenzione universale per indicare "non trovato" negli algoritmi di ricerca in C. Poiché gli indici validi sono sempre ≥ 0 , il valore -1 non può mai essere un indice reale e serve come sentinella inequivocabile.

Risposta corretta: C

Soluzione

Quiz 6

Domanda

6. Come si deve inizializzare la variabile `max` nell'algoritmo per trovare il massimo di un array?

- A) A 0, per confrontare valori positivi.
- B) Al valore più grande possibile (`INT\{}\_MAX`).
- C) Al valore del primo elemento dell'array (`arr[0]`).
- D) Non importa, qualsiasi valore va bene.

L'inizializzazione corretta è sempre il primo elemento dell'array. Inizializzare a 0 sarebbe sbagliato se tutti gli elementi fossero negativi. L'unica strategia sempre corretta è usare il primo elemento come valore di partenza.

Risposta corretta: C

Soluzione

Quiz 7

Domanda

7. Nel ciclo `for (int i = 1; i < N; i++)` per trovare il massimo, perché si parte da `i = 1`?

- A) Per evitare un accesso fuori dai limiti.
- B) Perché `arr[0]` è già usato come valore iniziale di `max`: confrontarlo di nuovo sarebbe ridondante.

- C) Perché il primo elemento non può mai essere il massimo.
- D) Per una convenzione del linguaggio C.

Soluzione
Risposta corretta: B
Poiché abbiamo già assegnato `max = arr[0]`, il primo elemento ha già "concorso" come candidato. Iniziare il ciclo di confronto da `i = 1` evita un confronto inutile dell'elemento con se stesso.

Quiz 8

Domanda

8. Cosa accade se si accede a un indice fuori dai limiti in C (es. `arr[N]` su un array di `N` elementi)?

- A) Il compilatore genera sempre un errore che blocca la compilazione.
- B) Il programma si arresta immediatamente con un messaggio di errore chiaro.
- C) Il comportamento è non definito (Undefined Behavior): il programma potrebbe leggere dati casuali, corrompere la memoria o crashare.
- D) Il programma ritorna automaticamente all'ultimo elemento valido.

Soluzione
Risposta corretta: C
L'accesso out-of-bounds è uno dei comportamenti non definiti più pericolosi del C. Il compilatore non garantisce alcun controllo a run-time sugli indici. Il programma potrebbe sembrare funzionare correttamente in alcuni casi e crashare in altri.

Quiz 9

Domanda

9. Qual è il vantaggio di usare `\{\}\#define N 5` rispetto a usare direttamente `5` nel codice?

- A) Riduce il tempo di compilazione.
- B) Permette di modificare la dimensione in un solo posto, aggiornando automaticamente tutto il codice che usa `N`.
- C) Rende l'array allocato nello Heap invece che nello Stack.
- D) Non ci sono vantaggi, è solo una convenzione stilistica.

Usare una costante simbolica è il principio del "numero magico unico": se la dimensione cambia, basta modificarla in un solo posto. Senza questa pratica, bisognerebbe trovare e modificare manualmente ogni occorrenza del numero 5, rischiando di dimenticarne qualcuna.

Risposta corretta: B

Soluzione

Quiz 10

Domanda

10. Quale affermazione sulla "seconda passata" su un array è corretta?

- A) È impossibile scorrere un array due volte nello stesso programma.
- B) Un array permette di elaborare gli stessi dati con più cicli sequenziali, cosa impossibile con variabili scalari.
- C) La seconda passata è sempre più veloce della prima perché i dati sono in cache.
- D) La seconda passata richiede che l'array sia ordinato.

Questo è il vantaggio fondamentale degli array rispetto alle variabili scalari: i dati rimangono in memoria per tutta la durata del programma. Un esempio pratico è la stampa dei voti superiori alla media: possiamo prima calcolare la media, poi fare una seconda passata per confrontare ogni elemento con essa — operazione impossibile se avessimo letto i voti uno alla volta senza memorizzarli.

Risposta corretta: B

Soluzione

1.5 Esercizi Pratici

Esercizio 1: Stampa dell'Array al Contrario

Domanda

Scrivete un programma che legga N interi da tastiera (con `\n` e `\t` e li stampi nell'ordine inverso rispetto a quello di inserimento.

Per stampare l'array al contrario, il secondo ciclo for parte dall'ultimo indice valido (N - 1) e decrementa i fino a 0 incluso (i >= 0). Non è necessario creare un secondo array: basta percorrere il primo in senso inverso.

```

1  #include <stdio.h>
2  #define N 6
3
4  int main(void) {
5      int arr[N];
6
7      printf("Inserisci %d numeri:\n", N);
8      for (int i = 0; i < N; i++) {
9          printf("Numero %d: ", i + 1);
10         scanf("%d", &arr[i]);
11     }
12     printf("Array al contrario: ");
13     for (int i = N - 1; i >= 0; i--) {
14         printf("%d ", arr[i]);
15     }
16     printf("\n");
17     return 0;
18 }
19
20

```

Soluzione

Esercizio 2: Somma e Media

Domanda

Scrivete un programma che legga N interi (con `\{\}\#define N 8`) e stampi la loro somma e la loro media con 2 decimali.

Accumulo la somma durante il ciclo di lettura per evitare un ciclo extra. La media richiede il cast (double) su somma per ottenere la divisione in virgola mobile.

```

1  #include <stdio.h>
2  #define N 8
3
4  int main(void) {
5      int arr[N];
6      int somma = 0;
7
8      printf("Inserisci %d numeri interi:\n", N);
9      for (int i = 0; i < N; i++) {
10         printf("Numero %d: ", i + 1);
11         scanf("%d", &arr[i]);
12         somma += arr[i];
13     }
14     double media = (double)somma / N;
15     printf("\nSomma: %d\n", somma);
16     printf("Media: %.2f\n", media);
17
18     return 0;
19 }
20

```

Soluzione

Esercizio 3: Contare gli Elementi Maggiori di una Soglia

Domanda

Scrivete un programma che legga N temperature in gradi Celsius (con `\{\}\#define N 7`) e conti quante superano 36.5. Usate un array di `double`.

Si noti l'uso di `\{\}\%lf` in `scanf` per leggere i valori `double`. La seconda passata con il ciclo di conteggio dimostra il valore dell'array: elaboriamo i dati separatamente rispetto alla lettura.

```

1  #include <stdio.h>
2  #define N 7
3
4  int main(void) {
5      double temp[N];
6      int febre = 0;
7
8      printf("Inserisci %d temperature:\n", N);
9      for (int i = 0; i < N; i++) {
10         printf("Temperatura %d: ", i + 1);
11         scanf("%lf", &temp[i]);
12     }
13
14     for (int i = 0; i < N; i++) {
15         if (temp[i] > 36.5) {
16             febre++;
17         }
18     }
19
20     printf("\nTemperature sopra 36.5: %d su %d\n", febre, N);
21
22     return 0;
23 }
```

Soluzione

Esercizio 4: Massimo e sua Posizione

Domanda

Scrivete un programma che legga `N` interi (con `\{\}\#define N 6`) e stampi il valore massimo e l'indice in cui si trova.

max viene inizializzato al primo elemento (arr[0]) e pos\{}_max a 0. Il ciclo parte da i = 1. Ogni volta che troviamo un elemento maggiore dell'attuale massimo, aggiorniamo sia il valore che la posizione.

```

1  #include <stdio.h>
2  #define N 6
3
4  int main(void) {
5      int arr[N];
6
7      printf("Inserti %d numeri:\n", N);
8      for (int i = 0; i < N; i++) {
9          printf("Numero %d: ", i + 1);
10         scanf("%d", &arr[i]);
11     }
12     int max = arr[0];
13     int pos_max = 0;
14
15     for (int i = 1; i < N; i++) {
16         if (arr[i] > max) {
17             max = arr[i];
18             pos_max = i;
19         }
20     }
21
22     printf("Massimo: %d (indice %d)\n", max, pos_max);
23
24     return 0;
25 }

```

Soluzione

Esercizio 5: Ricerca Lineare con Indice

Domanda

Scrivete un programma che legga N interi (con \{}\#define N 5), chieda un valore da cercare e stampi l'indice della prima occorrenza, oppure un messaggio se non è presente.

La variabile `indice` viene inizializzata a `-1` come sentinella. Il `break` interrompe la ricerca appena l'elemento viene trovato, evitando scansioni inutili.

```

1  #include <stdio.h>
2  #define N 5
3
4  int main(void) {
5      int arr[N];
6
7      printf("Inserisci %d numeri:\n", N);
8      for (int i = 0; i < N; i++) {
9          printf("Numero %d: ", i + 1);
10         scanf("%d", &arr[i]);
11     }
12     int target;
13     printf("Numero da cercare: ");
14     scanf("%d", &target);
15
16     int indice = -1;
17     for (int i = 0; i < N; i++) {
18         if (arr[i] == target) {
19             indice = i;
20             break;
21         }
22     }
23     if (indice != -1) {
24         printf("trovato all'indice %d.\n", indice);
25     } else {
26         printf("non presente nell'array.\n", target);
27     }
28     return 0;
29 }

```

Soluzione

Esercizio 6: Minimo e Massimo Simultanei

Domanda

Scrivete un programma che legga `N` interi (con `\{\}\#define N 7`) e stampi sia il valore minimo che il massimo in un unico ciclo di scansione.

separate.

Entrambe le variabili vengono inizializzate ad `arr[0]`. Nello stesso ciclo `for`, aggiorniamo massimo e minimo con due `if` indipendenti, evitando due passate

```

1  #include <stdio.h>
2  #define N 7
3
4  int main(void) {
5      int arr[N];
6
7      printf("Inserisci %d numeri:\n", N);
8      for (int i = 0; i < N; i++) {
9          printf("Numero %d: ", i + 1);
10         scanf("%d", &arr[i]);
11     }
12     int max = arr[0];
13     int min = arr[0];
14
15     for (int i = 1; i < N; i++) {
16         if (arr[i] > max) max = arr[i];
17         if (arr[i] < min) min = arr[i];
18     }
19
20     printf("Massimo: %d\n", max);
21     printf("Minimo: %d\n", min);
22
23     return 0;
24 }
25

```

Soluzione

Esercizio 7: Voti Superiori alla Media

Domanda

Scrivete un programma che legga N voti interi (con `\#\define N 6`), calcoli la media e stampi gli indici e i valori dei voti che la superano.

Questo esercizio esemplifica la "seconda passata": prima calcoliamo la media, poi scorriamo nuovamente l'array per trovare gli elementi che la superano.

```

1  #include <stdio.h>
2  #define N 6
3
4  int main(void) {
5      int voti[N];
6      int somma = 0;
7
8      printf("Inserisci %d voti:\n", N);
9      for (int i = 0; i < N; i++) {
10         printf("Voto %d: ", i + 1);
11         scanf("%d", &voti[i]);
12         somma += voti[i];
13     }
14     double media = (double)somma / N;
15     printf("Media: %.2f\n", media);
16     printf("\nVoti superiori alla media:\n");
17     int trovati = 0;
18     for (int i = 0; i < N; i++) {
19         if (voti[i] > media) {
20             printf(" - voti[%d] = %d\n", i, voti[i]);
21             trovati++;
22         }
23     }
24     if (trovati == 0) {
25         printf("Nessun voto superiore alla media.\n");
26     }
27     return 0;
28 }

```

Soluzione

Esercizio 8: Array con Inizializzazione e Stampa

Domanda

Formattata

Dichiarate e inizializzate un array di 5 interi con i valori `{12, 7, 25, 3, 18}`. Stampate ogni elemento con il formato `arr[i] = valore`, poi stampate la somma totale.

L'array viene inizializzato alla dichiarazione. Un unico ciclo gestisce sia la stampa formattata che l'accumulo della somma.

```

1  #include <stdio.h>
2  #define N 5
3
4  int main(void) {
5      int arr[N] = {12, 7, 25, 3, 18};
6      int somma = 0;
7
8      printf("Contenuto dell'array:\n");
9      for (int i = 0; i < N; i++) {
10         printf("arr[%d] = %d\n", i, arr[i]);
11         somma += arr[i];
12     }
13
14     printf("\nSomma totale: %d\n", somma);
15
16     return 0;
17 }
```

Soluzione

Esercizio 9: Contare le Occorrenze di un Valore

Domanda

Scrivete un programma che legga N interi (con `\{\}\#define N 8`), chieda un valore target e stampi quante volte compare nell'array.

A differenza della ricerca con **break**, qui **non** usciamo dal ciclo appena troviamo il valore: dobbiamo contare **tutte** le occorrenze. Il ciclo scorre l'intero array senza interruzioni!

```

1  #include <stdio.h>
2  #define N 8
3
4  int main(void) {
5      int arr[N];
6
7      printf("Inserisci %d numeri:\n", N);
8      for (int i = 0; i < N; i++) {
9          printf("Numero %d: ", i + 1);
10         scanf("%d", &arr[i]);
11     }
12     int target;
13     printf("Valore da contare: ");
14     scanf("%d", &target);
15     int contatore = 0;
16     for (int i = 0; i < N; i++) {
17         if (arr[i] == target) {
18             contatore++;
19         }
20     }
21     printf("%d compare %d volta/e nell'array.\n", target, contatore);
22     return 0;
23 }

```

Soluzione

Esercizio 10: Analisi Completa di Temperature Giornaliere

Domanda

Scrivete un programma che simuli l'analisi di N temperature giornaliere (con `\{\}\#define N 7`). Il programma deve:

1. Leggere le 7 temperature (double tra -50.0 e 60.0 , validate con `do-while`).
2. Calcolare e stampare la temperatura media settimanale.
3. Stampare il giorno con la temperatura massima e quello con la minima.
4. Contare e stampare quanti giorni hanno avuto temperatura superiore alla media.

Soluzione

Questo esercizio integra tutti i pattern della lezione: acquisizione validata con `while`, media come divisione `double`, `max/min` con tracciamento della posizione, e conteggio degli elementi superiori alla media in un'unica passata. Si noti che `pos\{}_max + 1` viene usato nella stampa per convertire dall'indice base-0 al numero del giorno da 1 a 7, più naturale per l'utente.

```

1  #include <stdio.h>
2  #define N 7
3
4  int main(void) {
5      double temp[N];
6      double somma = 0.0;
7
8      printf("Inserisci le temperature dei %d giorni:\n", N);
9      for (int i = 0; i < N; i++) {
10         do {
11             printf("Giorno %d (gradi C): ", i + 1);
12             scanf("%lf", &temp[i]);
13             while (temp[i] > 60.0 || temp[i] < -50.0) {
14                 somma += temp[i];
15             }
16         } while (temp[i] > 60.0 || temp[i] < -50.0);
17         double media = somma / N;
18
19         double max = temp[0], min = temp[0];
20         int pos_max = 0, pos_min = 0;
21         int sopra_media = 0;
22
23         for (int i = 0; i < N; i++) {
24             if (temp[i] > max) { max = temp[i]; pos_max = i; }
25             if (temp[i] < min) { min = temp[i]; pos_min = i; }
26             if (temp[i] > media) sopra_media++;
27         }
28
29         printf("\n--- Analisi Settimanale ---\n");
30         printf("Media: %.2f gradi C\n", media);
31         printf("Massima: %.2f gradi C (Giorno %d)\n", max, pos_max + 1);
32         printf("Minima: %.2f gradi C (Giorno %d)\n", min, pos_min + 1);
33         printf("Giorni sopra media: %d/%d\n", sopra_media, N);
34
35         return 0;
36     }

```