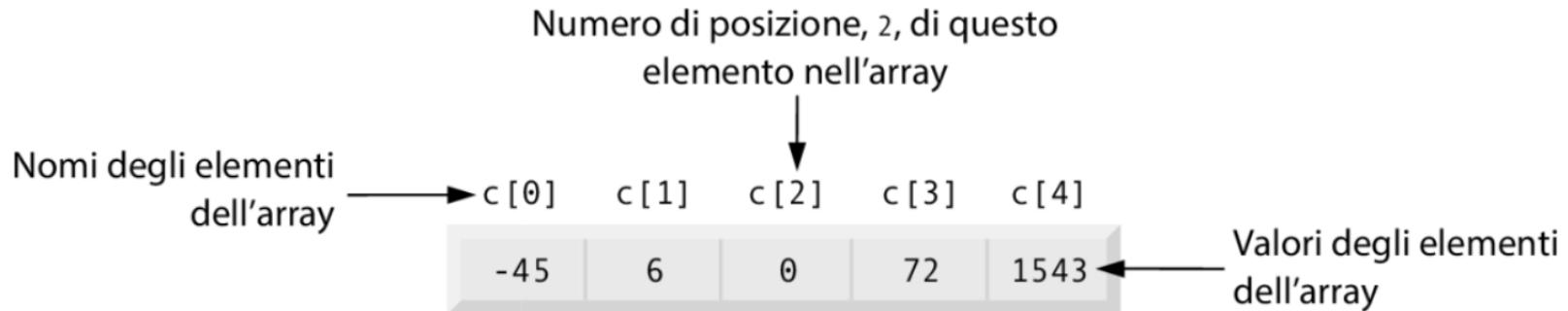


Array

Gli array (o vettori)

- Un array è un **gruppo** di elementi **dello stesso tipo** immagazzinati in modo contiguo nella memoria.
- Vedremo nelle prossime lezioni il costrutto struct del C, una struttura di dati costituita da dati correlati di tipi possibilmente differenti
- Array e struct sono entità “statiche” in quanto rimangono della stessa dimensione per tutto il loro tempo di vita

Elementi e indici



Fonte: Deitel & Deitel

- Per riferirsi ad un elemento, si utilizza il nome dell'array seguito dal numero di posizione tra parentesi quadre []
- Il numero di posizione dell'elemento nell'array è detto **indice**
 - Il primo elemento ha indice 0 (zero)
 - L'indice deve essere un intero non negativo o un'espressione intera
- Il nome dell'array è un **lvalue** (si può usare sul lato sinistro di un'assegnazione)

Definire gli array

- Specificare il tipo di elemento e il numero di elementi
 - Permette al compilatore di riservare la giusta quantità di memoria
- Esempi:
 - `int c[5];`
Array di interi c con 5 elementi (indici da 0 a 4).
 - `int b[100];`
Array di interi b con 100 elementi (indici da 0 a 99).
 - `int x[27];`
Array di interi x con 27 elementi (indici da 0 a 26).
- Un array di tipo `char` può memorizzare una stringa di caratteri

Definire un array usando un ciclo

```
1 // fig06_01.c
2 // Inizializzare gli elementi di un array a zero.
3 #include <stdio.h>
4
5 // la funzione main inizia l'esecuzione del programma
6 int main(void) {
7     int n[ 5 ]; // n e' un array di 5 interi
8
9     // imposta gli elementi dell'array n a 0
10    for ( size_t i = 0 ; i < 5 ; ++i) {
11        n[i] = 0 ; // imposta a 0 l'elemento alla locazione i
12    }
13
14    printf("%s%8s\n", "Element", "Value");
15
16    // stampa il contenuto dell'array n in formato tabellare
17    for ( size_t i = 0 ; i < 5 ; ++i) {
18        printf( "%zu%8d\n" , i, n[i]);
19    }
20 }
```

Element	Value
0	0
1	0
2	0
3	0
4	0

Definire un array usando un ciclo

- La variabile di controllo `i` è utilizzata come contatore nelle istruzioni `for`:
- Il tipo `size_t` rappresenta un tipo intero senza segno
- Raccomandato per variabili che rappresentano dimensioni o indici di array
- Definito nel file di intestazione `<stddef.h>`
- Spesso incluso anche in altri file di intestazione, come `<stdio.h>`
- Per la stampa di valori `size_t` si utilizza la specifica di conversione `%zu`

Inizializzare un array con una lista

```
1 // fig06_02.c
2 // Inizializzare gli elementi di un array con una lista di inizializzatori.
3 #include <stdio.h>
4
5 // la funzione main inizia l'esecuzione del programma
6 int main(void) {
7     int n[ 5 ] = { 32 , 27 , 64 , 18 , 95 }; // inizializza n con lista di inizializzatori
8
9     printf("%s%8s\n", "Element", "Value");
10
11    // stampa il contenuto dell'array in formato tabellare
12    for (size_t i = 0; i < 5; ++i) {
13        printf("%zu%8d\n", i, n[i]);
14    }
}
```

Element	Value
0	32
1	27
2	64
3	18
4	95

Inizializzare un array con una lista

- **Inizializzazione parziale degli array:** se ci sono meno inizializzatori degli elementi, i rimanenti sono inizializzati a 0
 - `int n[5] = {1};`
Inizializza `n[0]` a 1 ed implicitamente tutti gli altri elementi a 0
- **Errore di compilazione:** si verifica se ci sono più inizializzatori di quanti siano gli elementi nell'array
 - `int n[3] = {32, 27, 64, 18};`
Quattro inizializzatori per un array di tre elementi
- **Inizializzazione completa senza specificare dimensione:**
 - `int n[] = {1, 2, 3, 4, 5};`
Array di 5 elementi con valori da 1 a 5
 - Se la dimensione dell'array è omessa, il compilatore calcola la dimensione in base al numero di inizializzatori

Specificare la dimensione di un array con una costante simbolica

```
1 // fig06_03.c
2 // Inizializzare gli elementi dell'array s con gli interi pari da 2 a 10.
3 #include <stdio.h>
4 #define SIZE 5 // massima dimensione dell'array
5
6 // la funzione main inizia l'esecuzione del programma
7 int main(void) {
8     // la costante simbolica SIZE specifica la dimensione dell'array
9     int s[SIZE] = { 0 }; // l'array s ha un numero di elementi uguale a SIZE
10
11    for (size_t j = 0; j < SIZE; ++j) { // imposta i valori
12        s[j] = 2 + 2 * j;
13    }
14
15    printf("%s%8s\n", "Element", "Value");
16
17    // stampa il contenuto dell'array s in formato tabellare
18    for (size_t j = 0; j < SIZE; ++j) {
19        printf("%zu%8d\n", j, s[j]);
20    }
21}
```

Element	Value
0	2
1	4
2	6
3	8
4	10

Direttiva per il processore `#define`

- Utilizzata per creare costanti simboliche
 - `#define SIZE 5` Crea la costante simbolica SIZE con valore 5
- Il preprocessore sostituisce ogni occorrenza di SIZE con 5 prima della compilazione
- Migliora la leggibilità e la modificabilità del codice
- Permette di cambiare la dimensione di un array modificando un solo valore
 - Cambiare SIZE da 5 a 1000 modifica automaticamente l'array in tutte le sue occorrenze.
- Facilita la manutenzione del codice, evitando errori dovuti a valori numerici ripetuti

Direttiva per il processore `#define`

- Non usare il punto e virgola alla fine di una direttiva `#define`
- Esempio di errore:
 - `#define SIZE 5;`
sostituisce tutte le occorrenze con "5 ;", causando errori di sintassi
- Le costanti simboliche non sono variabili
 - assegnare loro un valore in un'istruzione eseguibile provoca un errore di compilazione
- Convenzione di stile: usare lettere maiuscole per le costanti simboliche (es. `SIZE`)

Sommare gli elementi di un array

```
1 // fig06_04.c
2 // Calcolare la somma degli elementi di un array.
3 #include <stdio.h>
4 #define SIZE 5
5
6 // la funzione main inizia l'esecuzione del programma
7 int main(void) {
8     // usa una lista di inizializzatori per inizializzare l'array
9     int a[SIZE] = {1, 2, 3, 4, 5};
10    int total = 0; // somma dell'array
11
12    // somma i valori contenuti nell'array a
13    for (size_t i = 0; i < SIZE; ++i) {
14        total += a[i];
15    }
16
17    printf("The total of a's values is %d\n", total);
18 }
```

The total of a's values is 15

Usare array per riassumere i risultati di un sondaggio

- A venti studenti viene chiesto di valutare la qualità del cibo nella caffetteria degli studenti con una scala da 1 a 5 (1 significa pessima e 5 eccellente)
- Mettete le 20 risposte in un array intero e riassumete i risultati del sondaggio

Usare array per riassumere i risultati di un sondaggio

```
1 // fig06_05.c
2 // Analisi di un sondaggio di studenti.
3 #include <stdio.h>
4 #define RESPONSES_SIZE 20 // definisci le dimensioni degli array
5 #define FREQUENCY_SIZE 6
6
7 // la funzione main inizia l'esecuzione del programma
8 int main(void) {
9     // inserisci le risposte del sondaggio nell'array delle risposte
10    int responses[RESPONSES_SIZE] =
11        {1, 2, 5, 4, 3, 5, 2, 1, 3, 1, 4, 3, 3, 3, 2, 3, 3, 2, 2, 5};
12
13    // inizializza i contatori per le frequenze a 0
14    int frequency[ FREQUENCY_SIZE ] = { 0 };
15
16    // per ogni risposta, seleziona il valore di un elemento dell'array
17    // responses e usa quel valore come indice nell'array frequency per
18    // determinare l'elemento da incrementare
19    for (size_t answer = 0; answer < RESPONSES_SIZE; ++answer) {
20        ++frequency[responses[answer]];
21    }
22
23    // stampa i risultati
24    printf("%s%12s\n", "Rating", "Frequency");
25
26    // stampa le frequenze in un formato tabellare
27    for ( size_t rating = 1 ; rating < FREQUENCY_SIZE ; ++rating) {
28        printf("%6zu%12d\n", rating, frequency[rating]);
29    }
30 }
```

Rating	Frequency
1	3
2	5
3	7
4	2
5	3

Fonte: Deitel & Deitel

Usare array per riassumere i risultati di un sondaggio

- Abbiamo due array principali:
 - responses: un array di 20 elementi che contiene le risposte degli studenti.
 - frequency: un array di 6 elementi usato per contare quante volte ciascuna risposta è stata data.
- frequency [0] è ignorato; le risposte incrementano da frequency [1] in poi
- Permette di usare ogni risposta direttamente come indice in frequency
- Nel progettare il programma, è fondamentale puntare alla chiarezza
 - A volte, è preferibile sacrificare un po' di efficienza nella memoria o nel tempo di esecuzione per rendere il codice più comprensibile

Usare array per riassumere i risultati di un sondaggio

- Nel ciclo `for` le risposte vengono prelevate dall'array `responses` e incrementano i contatori:
`++frequency[responses[answer]];`
- Se `answer` è 0, `responses[answer]` è 1, quindi `++frequency[1]` incrementa `frequency[1]`
- Se `answer` è 1, `responses[answer]` è 2, quindi `++frequency[2]` incrementa `frequency[2]`
- Se `answer` è 2, `responses[answer]` è 5 quindi `++frequency[5]` incrementa `frequency[5]`
- Il ciclo aggiorna il contatore corretto in `frequency` per ogni risposta

Usare array per riassumere i risultati di un sondaggio

- Voti da 1 a 5 nel sondaggio
 - In questo caso un array con 6 elementi (escludendo l'elemento zero) è sufficiente per riassumere i risultati
- Se i dati contenessero valori come 13, il programma tenterebbe di accedere a `frequency[13]`, fuori dai confini dell'array
- Il C non ha controllo automatico sui confini degli array, il che può causare accessi a posizioni non valide senza avvisi
- I programmi devono convalidare tutti i valori in ingresso per evitare che valori errati influenzino i calcoli e causino problemi di sicurezza

Usare array per riassumere i risultati di un sondaggio

- Fare riferimento a un elemento al di fuori dei limiti di un array è considerato un **errore logico**
- L'indice dell'array deve essere compreso tra 0 e dimensione dell'array meno uno (SIZE – 1)
 - Assicuratevi che l'indice non scenda sotto 0 e sia sempre minore del numero totale degli elementi dell'array
- È essenziale garantire che tutti i riferimenti all'array rimangano entro i confini definiti dell'array per evitare errori

Intervallo



Fonte: PlaygroundAI

Rappresentare con grafici a barre i valori degli elementi di un array

```
1 // fig06_06.c
2 // Stampa di una grafico a barre.
3 #include <stdio.h>
4 #define SIZE 5
5
6 // la funzione main inizia l'esecuzione del programma
7 int main(void) {
8     // usa la lista di inizializzatori per inizializzare l'array n
9     int n[SIZE] = {19, 3, 15, 7, 11};
10
11    printf("%s%13s%17s\n", "Element", "Value", "Bar Chart");
12
13    // per ogni elemento dell'array n, stampa una barra del grafico
14    for (size_t i = 0; i < SIZE; ++i) {
15        printf( "%7zu%13d%8s" , i, n[i], "" );
16
17        for ( int j = 1 ; j <= n[i]; ++j) { // stampa una barra
18            printf( "%c" , '*' );
19        }
20
21        puts(""); // termina una barra con un newline
22    }
23 }
```

Element	Value	Bar Chart
0	19	*****
1	3	***
2	15	*****
3	7	*****
4	11	*****

Lanciare un dado 60.000.000 di volte e riepilogare i risultati in un array

```
1 // fig06_07.c
2 // Lancio di un dado a sei facce 60.000.000 di volte
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <time.h>
6 #define SIZE 7
7
8 int main(void) {
9     srand(time(NULL)); // seme per il generatore di numeri casuali
10
11    int frequency[SIZE] = {0}; // inizializza tutti i conti di frequency a 0
12
13    for (int roll = 1; roll <= 60000000; ++roll) {
14        size_t face = 1 + rand() % 6;
15        ++frequency[face]; // sostituisce l'intero switch della Figura 5.5
16    }
17
18    printf("%s%17s\n", "Face", "Frequency");
19
20    // stampa gli elementi di frequency 1-6 in formato tabellare
21    for (size_t face = 1; face < SIZE; ++face) {
22        printf("%4zu%17d\n", face, frequency[face]);
23    }
24 }
```

Face	Frequency
1	9997167
2	10003506
3	10001940
4	9995833
5	10000843
6	10000711

Fonte: Deitel & Deitel

Quiz



Array, caratteri e stringhe

- Un array di caratteri può essere inizializzato usando una stringa letterale, es.:

```
char string1[] = "first";
```
- La dimensione dell'array è determinata dal compilatore in base alla lunghezza della stringa, inclusivo del carattere nullo di terminazione.
 - La stringa "first" contiene 5 caratteri più il carattere nullo '\0', quindi string1 ha 6 elementi in totale
- Il carattere nullo di terminazione segna la fine di una stringa
- È fondamentale che l'array di caratteri sia abbastanza grande da contenere tutti i caratteri della stringa e il carattere nullo di terminazione

Accesso in un array di caratteri

- E' possibile accedere ai singoli caratteri di una stringa direttamente usando la notazione con indice per gli array
- Es: `char string1[] = "first";`
 - `string1[0]` è 'f'
 - `string1[3]` è 's'
 - `string1[5]` è '\0'

Definizione di un array di caratteri

```
char string2[20];
```

- crea un array di caratteri capace di memorizzare una stringa di massimo 19 caratteri più un carattere nullo di terminazione
- L'istruzione `scanf ("%19s", string2);` legge una stringa dalla tastiera e la memorizza in `string2`
- Il nome dell'array `string2` è passato a `scanf` **senza** il carattere `&`
 - normalmente usato con `scanf` per passare l'indirizzo di memoria delle variabili non stringa

Gestione di un array di caratteri

- Assicurarsi che l'array possa contenere qualsiasi stringa immessa dall'utente
- La funzione `scanf` non verifica la dimensione dell'array e legge fino a uno spazio, tabulazione, newline o end-of-file.
- L'array `string2` deve avere una dimensione sufficiente per contenere la stringa più il carattere nullo di terminazione
- Se l'utente inserisce 20 o più caratteri, si possono verificare:
 - Arresto del programma
 - Problemi di sicurezza noti come **overflow del buffer**
- Utilizzare `%19s` con `scanf` limita la lettura a 19 caratteri e previene scrittura in memoria oltre la fine dell'array `string2`

Stampa di un array di caratteri

- Per stampare un array di caratteri che rappresenta una stringa, si usa la specifica di conversione `%s`
 - `printf ("%s\n", string2);`
stampa il contenuto dell'array `string2`.
- Come `scanf`, `printf` non verifica la dimensione dell'array di caratteri
- `printf` stampa i caratteri fino a incontrare un carattere nullo di terminazione (`'\0'`)
- Se il carattere nullo di terminazione è mancante `printf` potrebbe continuare a stampare dati non previsti, portando a risultati imprevedibili o problemi di sicurezza

Illustrazione di un array di caratteri

```
1 // fig06_08.c
2 // Trattare gli array di caratteri come stringhe.
3 #include <stdio.h>
4 #define SIZE 20
5
6 // la funzione main inizia l'esecuzione del programma
7 int main(void) {
8     char string1[SIZE] = ""; // riserva 20 caratteri
9     char string2[] = "string literal"; // riserva 15 caratteri
10
11    // memorizza la stringa inserita dall'utente nell'array string1
12    printf("%s", "Enter a string (no longer than 19 characters): ");
13    scanf("%19s", string1); // leggi non più di 19 caratteri
14
15    // stampa le stringhe
16    printf("string1 is: %s\nstring2 is: %s\n", string1, string2);
17    puts("string1 with spaces between characters is:");
18
19    // stampa i caratteri finche' non si raggiunge il carattere nullo
20    for (size_t i = 0; i < SIZE && string1[i] != '\0'; ++i) {
21        printf("%c ", string1[i]);
22    }
23
24    puts("");
25 }
```

```
Enter a string (no longer than 19 characters): Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
```

Passare gli array alle funzioni

- Per passare un array come argomento a una funzione, specificare solo il nome dell'array, senza parentesi
 - modifyArray(hourlyTemperatures, HOURS_IN_A_DAY);
 - L'array e la sua dimensione vengono solitamente passati alla funzione
 - In C, tutti gli argomenti sono passati per valore, tranne gli array che sono passati per **riferimento**
 - La funzione chiamata riceve l'indirizzo del primo elemento dell'array
 - può modificare direttamente i valori dell'array originario nella funzione chiamante
 - Il nome di un array rappresenta l'indirizzo in memoria del primo elemento dell'array
 - Poiché l'indirizzo di partenza viene passato, la funzione chiamata sa esattamente dove l'array è memorizzato e può apportare modifiche agli elementi dell'array

Passare gli array alle funzioni

```
1 // fig06_10.c
2 // Il nome di un array coincide con l'indirizzo del suo primo elemento.
3 #include <stdio.h>
4
5 // la funzione main inizia l'esecuzione del programma
6 int main(void) {
7     char array[5] = ""; // definisci un array di dimensione 5
8
9     printf( "    array = %p\n&array[0] = %p\n    &array = %p\n" ,
10            array, &array[ 0 ], &array);
11 }
```

```
array = 0031F930
&array[0] = 0031F930
&array = 0031F930
```

Fonte: Deitel & Deitel

- Stampa di vari indirizzi usando la specifica di conversione %p
- %p stampa indirizzi in formato esadecimale (base 16), che può variare in base al compilatore
- I numeri esadecimali utilizzano le cifre da 0 a 9 e le lettere da A a F (equivalenti ai numeri decimali 10-15)

Passare gli array alle funzioni

- L'output mostra che `array`, `&array` e `&array[0]` hanno lo stesso valore, confermando che rappresentano lo stesso indirizzo
- Gli indirizzi sono identici per ogni esecuzione del programma su uno specifico computer, ma l'output può variare tra diversi sistemi
- Il passaggio degli array per riferimento ha senso per ragioni di prestazioni
- Se gli array fossero passati per valore, verrebbe passata una copia di ogni elemento
- Nel caso di array grandi passati frequentemente, ciò comporterebbe un consumo significativo di tempo e memoria per copiare gli array

Passare gli array alle funzioni

- Gli array sono passati per riferimento alle funzioni, permettendo alla funzione di modificare l'intero array
- Gli elementi singoli dell'array sono passati per valore, come le variabili scalari
 - Esempi di scalari: int, float, char
- Per passare un elemento specifico dell'array a una funzione:
 - Utilizzare il nome dell'array seguito dall'indice dell'elemento come argomento nella chiamata della funzione
 - Es: `function (array[2]) ;` passa il valore dell'elemento all'indice 2 dell'array alla funzione `function`

Passare gli array alle funzioni

- La funzione modifyArray può essere dichiarata per ricevere un array di interi e il numero di elementi dell'array come segue:

```
void modifyArray(int b[], int size);
```

- int b[] indica che la funzione si aspetta un array di interi
- int size specifica il numero di elementi dell'array
- La dimensione dell'array tra le parentesi quadre ([]) non è necessaria e viene ignorata se inclusa; se specificata, deve essere positiva
 - Se la dimensione è negativa, si verifica un errore di compilazione.

Passare gli array alle funzioni

```
void modifyArray(int b[], int size);
```

- Il nome dell'array `b` nella funzione si riferisce all'array originario passato dalla funzione chiamante
- Quando la funzione `modifyArray` è chiamata, il nome dell'array passato rappresenta l'indirizzo del primo elemento dell'array nella funzione chiamante
- Es. chiamata di funzione

```
modifyArray(hourlyTemperatures, HOURS_IN_A_DAY);
```

- `hourlyTemperatures` è l'array passato alla funzione `modifyArray`
- `HOURS_IN_A_DAY` rappresenta il numero di elementi dell'array
- Il parametro `b` rappresenta l'array `hourlyTemperatures` nella funzione chiamante

Passare gli array alle funzioni

- Deitel code example 06_11.c
- Il programma evidenzia la differenza tra il passaggio di un array intero e il passaggio di un singolo elemento dell'array
- Inizialmente, il programma stampa i cinque elementi dell'array
- Successivamente, l'array e la sua dimensione vengono passati alla funzione modifyArray, dove ogni elemento di a viene moltiplicato per 2
- Il contenuto aggiornato viene quindi stampato mostrando che modifyArray ha modificato effettivamente gli elementi dell'array
- Successivamente, il valore di a [3] viene stampato e passato alla funzione modifyElement
- Questa funzione moltiplica il valore per 2 e stampa il risultato
- Tuttavia, quando il programma stampa nuovamente a [3], il valore non è stato modificato, poiché gli elementi individuali dell'array sono passati per valore e non per riferimento

Passare gli array alle funzioni

```
1 // nella funzione tryToModifyArray, l'array b e' const, per cui non lo si
2 // puo' usare per modificare il suo argomento array nella funzione chiamante
3 void tryToModifyArray(const int b[]) {
4     b[ 0 ] /= 2; // errore
5     b[ 1 ] /= 2; // errore
6     b[ 2 ] /= 2; // errore
7 }
```

Fonte: Deitel & Deitel

- In alcune situazioni, è necessario impedire che una funzione modifichi gli elementi di un array
- Il qualificatore di tipo **const** in C può impedire a una funzione di modificare un array
 - gli elementi sono trattati come costanti nella funzione
 - Ogni tentativo di modificare un elemento dell'array all'interno della funzione produce un errore di compilazione
- Questo approccio segue il principio del **privilegio minimo**, limitando le modifiche non necessarie agli array della funzione chiamante

Intervallo



Fonte: PlaygroundAI

Ordinamento di un array

- Ordinare i dati (in ordine crescente o decrescente) è una delle applicazioni più importanti in informatica
- Esempi pratici: le banche ordinano gli assegni per numero di conto per preparare estratti conto; le compagnie telefoniche ordinano gli elenchi dei clienti per nome e cognome
 - Quasi tutte le organizzazioni devono ordinare dati, spesso in grandi quantità (Big Data)
- L'ordinamento dei dati è un problema complesso che ha attirato molte ricerche in informatica
- Esistono algoritmi di ordinamento semplici, facili da scrivere, testare e correggere, ma con prestazioni generalmente scarse
- Per migliorare le prestazioni, sono spesso necessari algoritmi di ordinamento più complessi.

Bubble Sort

- La tecnica di ordinamento chiamata "bubble sort" o "sinking sort" ordina i valori come bolle che salgono verso la cima o affondano verso il fondo dell'array
 - I valori più piccoli "salgono" progressivamente verso l'inizio dell'array, mentre i valori più grandi "scendono" verso la fine
- L'algoritmo esegue diverse passate lungo l'array, confrontando coppie successive di elementi (elemento 0 con elemento 1, elemento 1 con elemento 2, e così via)
 - Se una coppia di valori è in ordine crescente (o se i valori sono uguali), i valori restano in posizione
 - Se una coppia di valori è in ordine decrescente, i valori vengono scambiati



Fonte: <https://www.programmingsimplified.com/c/source-code/c-program-bubble-sort>

Bubble Sort

```
1 // fig06_12.c
2 // Ordinare i valori di un array in ordine crescente.
3 #include <stdio.h>
4 #define SIZE 10
5
6 // la funzione main inizia l'esecuzione del programma
7 int main(void) {
8     int a[SIZE] = {2, 6, 4, 8, 10, 12, 89, 68, 45, 37};
9
10    puts("Data items in original order");
11
12    // stampa l'array originario
13    for (size_t i = 0; i < SIZE; ++i) {
14        printf("%4d", a[i]);
15    }
16
17    // bubble sort
18    // ciclo per il numero di passate
19    for (int pass = 1; pass < SIZE; ++pass) {
20        // ciclo per il numero di confronti a ogni passata
21        for (size_t i = 0; i < SIZE - 1; ++i) {
22            // confronta due elementi adiacenti e scambiali se il
23            // primo elemento e' maggiore del secondo elemento
24            if (a[i] > a[i + 1]) {
25                int hold = a[i];
26                a[i] = a[i + 1];
27                a[i + 1] = hold;
28            }
29        }
30    }
31
32    puts("\nData items in ascending order");
33
34    // stampa l'array ordinato
35    for (size_t i = 0; i < SIZE; ++i) {
36        printf("%4d", a[i]);
37    }
38
39    puts("");
40 }
```

```
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
```

Fonte: Deitel & Deitel

Bubble Sort

- Il programma confronta gli elementi successivi dell'array: prima $a[0]$ con $a[1]$, poi $a[1]$ con $a[2]$, fino a confrontare $a[8]$ con $a[9]$
- Anche se ci sono 10 elementi, vengono eseguiti solo 9 confronti per passata
- Un valore grande può scendere molte posizioni durante una singola passata, mentre un valore piccolo può salire solo di una posizione
 - Dopo la prima passata, il valore più grande è posizionato all'ultimo posto dell'array ($a[9]$)
 - Alla seconda passata, il secondo valore più grande scende fino a $a[8]$, e così via
 - Dopo la nona passata, il penultimo valore più grande arriva a $a[1]$, lasciando il valore più piccolo in $a[0]$
- Per ordinare un array di 10 elementi, sono necessarie solo nove passate

Bubble Sort

- L'ordinamento avviene tramite cicli annidati `for`
- Lo scambio tra due valori, se necessario, viene effettuato con tre assegnazioni:

```
int hold = a[i];  
a[i] = a[i + 1];  
a[i + 1] = hold;
```

- La variabile `hold` viene usata per memorizzare temporaneamente uno dei due valori da scambiare.
- Utilizzare solo due assegnazioni, come:
 - risulterebbe nella perdita di uno dei due valori.
- Pertanto, è necessaria una variabile extra come `hold` per conservare temporaneamente il valore durante lo scambio

Bubble Sort

- Complessità di tempo quadratica $O(n^2)$ dove n è il numero di elementi nell'array
 - Il tempo di esecuzione cresce quadraticamente con la dimensione dell'input
- Esegue molti scambi, il che lo rende inefficiente su array di grandi dimensioni rispetto ad altri algoritmi di ordinamento.
 - Ogni scambio richiede un'operazione di lettura e scrittura in memoria.
- È un algoritmo **stabile**, il che significa che preserva l'ordine relativo degli elementi con chiavi uguali
 - Questo può essere utile in applicazioni dove l'ordine degli elementi identici deve essere mantenuto.
- È raramente usato in applicazioni pratiche per il sorting di grandi volumi di dati.
- È semplice da implementare e capire, il che lo rende un buon punto di partenza per chi sta imparando gli algoritmi di ordinamento.

Ricerca in array

- Gli array spesso contengono grandi quantità di dati con cui lavorare
- Potrebbe essere necessario cercare un valore specifico, chiamato "valore chiave", all'interno di un array
- Questo processo di individuazione di un valore chiave è noto come "ricerca"
- Esistono diverse tecniche di ricerca, tra cui:
 - **Ricerca lineare**: una tecnica semplice che esamina ogni elemento dell'array in sequenza
 - **Ricerca binaria**: una tecnica più efficiente ma complessa, utilizzabile solo su array ordinati

Ricerca lineare in un array

```
1 // fig06_14.c
2 // Ricerca lineare in un array.
3 #include <stdio.h>
4 #define SIZE 100
5
6 // prototipo di funzione
7 int linearSearch( const int array[], int key, size_t size);
8
9 // la funzione main inizia l'esecuzione del programma
10 int main(void) {
11     int a[SIZE] = {0}; // crea l'array a
12
13     // crea alcuni dati
14     for (size_t x = 0; x < SIZE; ++x) {
15         a[x] = 2 * x;
16     }
17
18     printf("Enter integer search key: ");
19     int searchKey = 0; // valore da localizzare nell'array a
20     scanf("%d", &searchKey);
21
22     // tenta di localizzare searchKey nell'array a
23     int subscript = linearSearch(a, searchKey, SIZE );
24
25     // stampa i risultati
26     if (subscript != -1) {
27         printf("Found value at subscript %d\n", subscript);
28     } else {
29         puts("Value not found");
30     }
31
32     // confronta la chiave con ogni elemento dell'array fino a quando non viene
33     // trovata la posizione o raggiunta la fine dell'array; restituisci l'indice
34     // dell'elemento se la chiave viene trovata o -1 se la chiave non viene trovata
35     int linearSearch( const int array[], int key, size_t size) {
36         // ciclo attraverso l'array
37         for ( size_t n = 0 ; n < size; ++n) {
38             if (array[n] == key) {
39                 return n; // restituisci la posizione della chiave
40             }
41
42         return -1 ; // chiave non trovata
43     }
44 }
```

```
Enter integer search key: 36
Found value at subscript 18
```

```
Enter integer search key: 37
Value not found
```

- Una ricerca lineare confronta ogni elemento dell'array con la **chiave di ricerca**
- Dal momento che l'array non è in un ordine particolare, è altrettanto probabile che il valore venga trovato nel primo elemento quanto nell'ultimo
- In media, dunque, il programma dovrà confrontare la chiave di ricerca con *metà* degli elementi dell'array

Ricerca binaria in un array ordinato

- La **ricerca lineare** è adatta per array piccoli o non ordinati, ma diventa inefficiente per array di grandi dimensioni
- Per array ordinati, è preferibile utilizzare la **ricerca binaria**, che è molto più veloce
- Dopo ogni confronto, elimina metà degli elementi dell'array ordinato dalla ricerca
- Trova l'elemento centrale dell'array e lo confronta con la chiave di ricerca

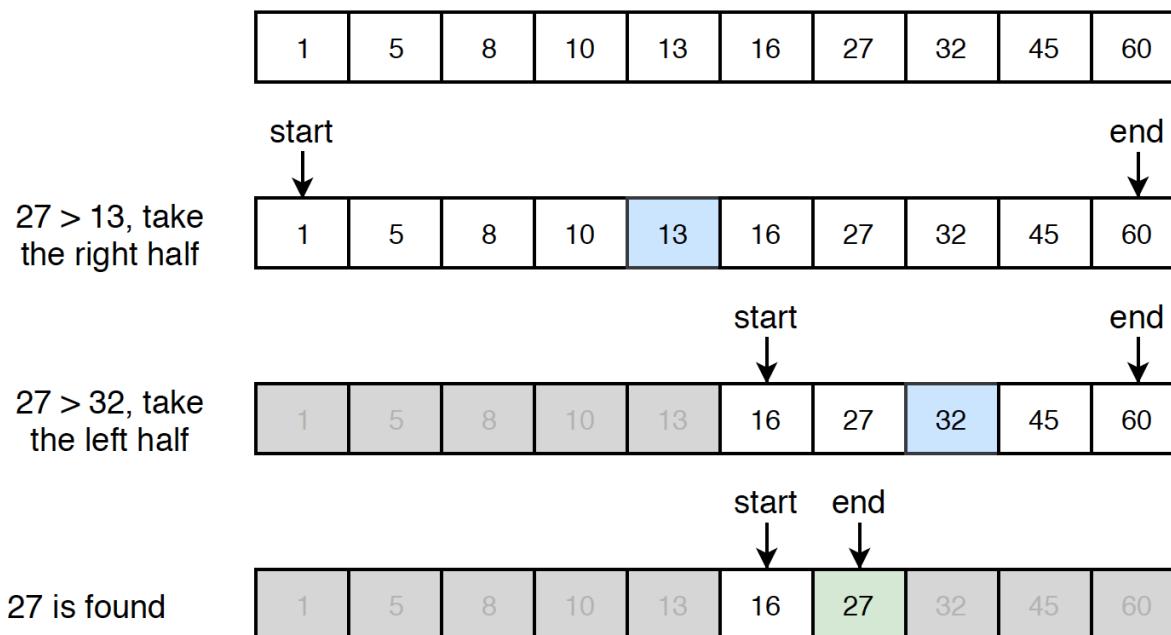
Ricerca binaria in un array ordinato

- Se la chiave di ricerca è uguale all'elemento centrale, l'algoritmo restituisce l'indice dell'elemento
- Se non sono uguali:
 - Se la chiave è minore dell'elemento centrale, ricerca nella prima metà dell'array
 - Se la chiave è maggiore, ricerca nella seconda metà dell'array
- La ricerca continua finché:
 - La chiave di ricerca è trovata nel sottoarray.
 - Oppure il sottoarray è ridotto a un solo elemento che non corrisponde alla chiave di ricerca (indicando che la chiave non è presente)

Ricerca binaria in un array ordinato

Binary Search

Search 27 in a sorted array with 10 elements



Fonte: <https://jojohuang.github.io/assets/images/algorithim/1211/binarysearch.png>

Ricerca binaria in un array ordinato

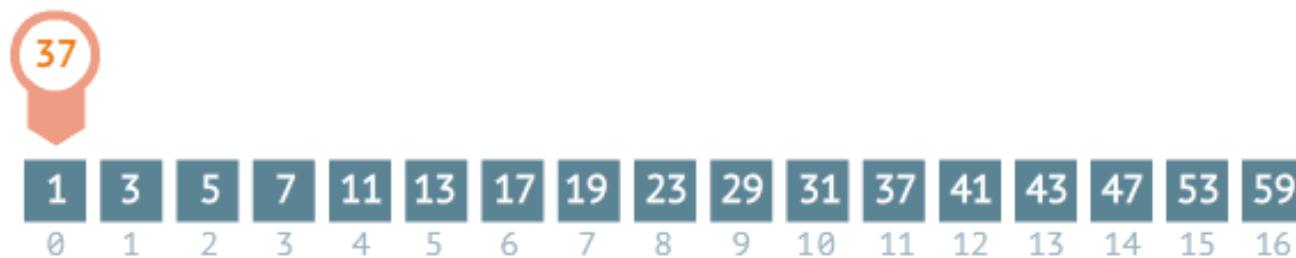
Binary search

steps: 0



Sequential search

steps: 0



Ricerca binaria in un array ordinato

- Deitel code example fig06_15.c
- Nel **peggiore** dei casi, la ricerca binaria in un array di 1023 elementi richiede solo 10 confronti
- Questo perché dividendo ripetutamente 1024 per 2 si ottengono i valori 512, 256, 128, 64, 32, 16, 8, 4, 2, 1; servono 10 divisioni (corrispondenti a 10 confronti) per ridurre 1024 a 1
 - Un array ordinato di un **miliardo** di elementi richiede un massimo di soli **30 confronti** con la ricerca binaria
 - la ricerca lineare necessiterebbe di circa 500 milioni di confronti in media
- Il numero massimo di confronti necessari per una ricerca binaria in un array può essere determinato considerando la prima potenza di 2 maggiore del numero di elementi dell'array

Quiz



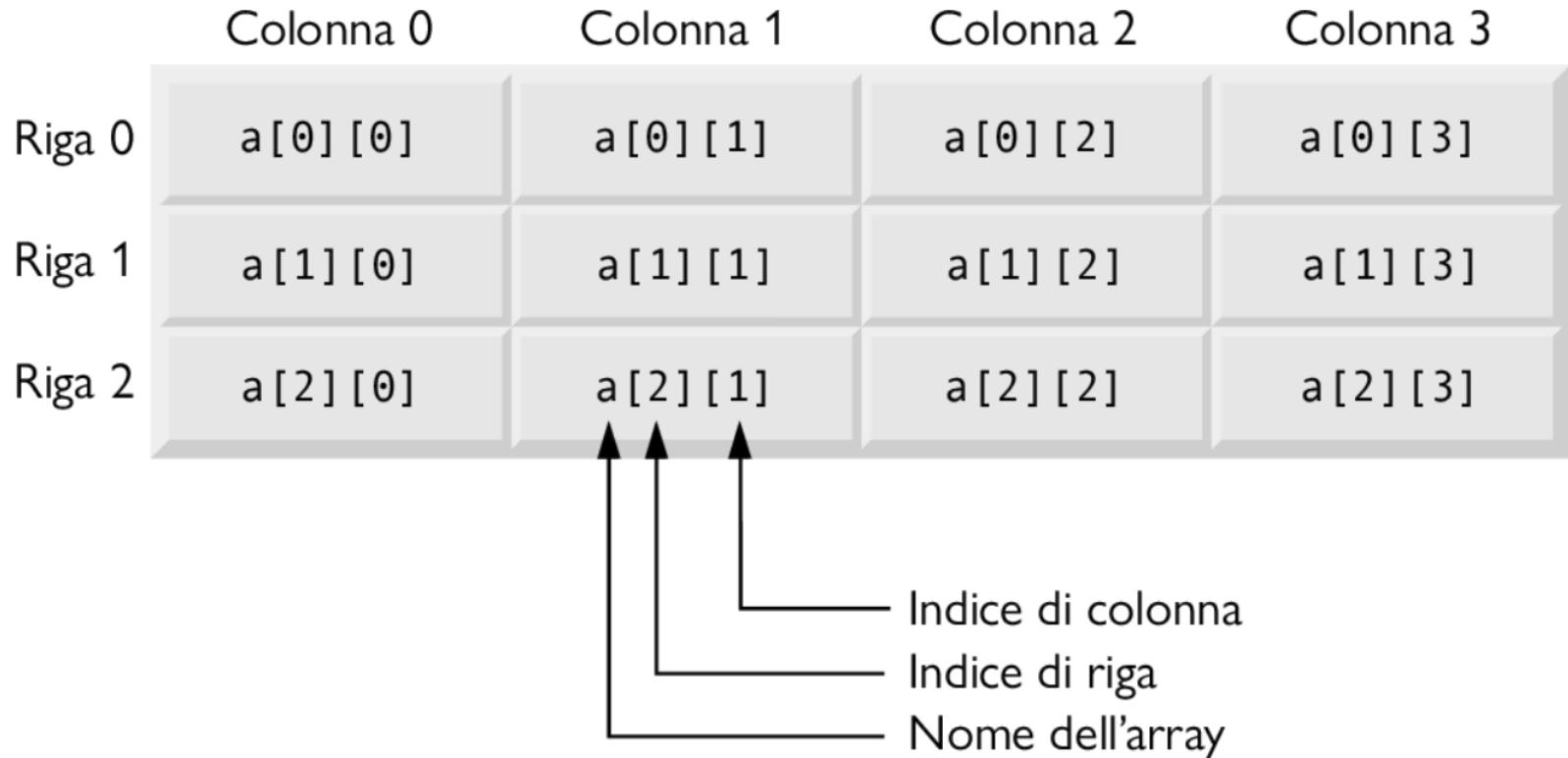
Caso pratico di introduzione alla data science: analisi dei dati di un sondaggio

- Deitel code example 06_13.c
- Il programma utilizza un array chiamato response per memorizzare 99 risposte di un sondaggio, ognuna rappresentata da un numero tra 1 e 9
- Il programma calcola tre misure statistiche: la media, la mediana e la moda dei 99 valori raccolti
- L'esempio mostra diverse manipolazioni comuni con gli array, tra cui il passaggio di array come argomenti a funzioni
- Le righe 48-52 del programma contengono stringhe letterali separate da spazi: il compilatore C le combina automaticamente in una singola stringa, migliorando la leggibilità delle stringhe lunghe

Array multidimensionali

- Possono avere diversi indici
- Un uso comune è la rappresentazione di tabelle di valori (informazioni disposte in righe e colonne)
 - Array bidimensionali
- Per identificare un elemento si usano due indici:
 - Primo indice: identifica la riga dell'elemento
 - Secondo indice: identifica la colonna dell'elemento
- Gli array multidimensionali possono avere più di due indici

Array bidimensionali



Array bidimensionali

- Ogni elemento è identificato da un nome della forma $a[i][j]$:
 - a è il nome dell'array.
 - i e j sono gli indici che identificano univocamente ogni elemento
- Esempi di indici:
 - Gli elementi nella riga 0 hanno il primo indice uguale a 0 (es. $a[0][j]$)
 - Gli elementi nella colonna 3 hanno il secondo indice uguale a 3 (es. $a[i][3]$).
- Riferirsi a un elemento come $a[x, y]$ invece di $a[x][y]$ è un **errore logico**
- In C, $a[x, y]$ viene interpretato come $a[y]$:
 - Non è un errore di sintassi, ma di interpretazione logica
- L'operatore virgola valuta le espressioni da sinistra a destra
- Il valore dell'intera espressione separata da virgole è dato dall'espressione più a destra

Array bidimensionali

```
1 // fig06_16.c
2 // Inizializzazione di array multidimensionali.
3 #include <stdio.h>
4
5 void printArray(int a[][3]); // prototipo di funzione
6
7 // la funzione main inizia l'esecuzione del programma
8 int main(void) {
9     int array1[ 2 ][ 3 ] = {{ 1 , 2 , 3 }, { 4 , 5 , 6 }};
10    puts("Values in array1 by row are:");
11    printArray(array1);
12
13    int array2[ 2 ][ 3 ] = {{ 1 , 2 , 3 }, { 4 , 5 }};
14    puts("Values in array2 by row are:");
15    printArray(array2);
16
17    int array3[ 2 ][ 3 ] = {{ 1 , 2 }, { 4 }};
18    puts("Values in array3 by row are:");
19    printArray(array3);
20 }
21
22 // funzione per stampare un array con due righe e tre colonne
23 void printArray(int a[][ 3 ]) {
24     // iterazione per righe
25     for (size_t i = 0; i <= 1; ++i) {
26         // stampa i valori delle colonne
27         for (size_t j = 0; j <= 2; ++j) {
28             printf("%d ", a[i][j]);
29         }
30
31         printf("\n"); // inizia una nuova riga di stampa
32     }
33 }
```

Values in array1 by row are:

1 2 3
4 5 6

Values in array2 by row are:

1 2 3
4 5 0

Values in array3 by row are:

1 2 0
4 0 0

Fonte: Deitel & Deitel

Stampare un array

```
void printArray(int a[][][3])
```

- Utilizzata per stampare gli elementi di ogni array
- Il parametro array è definito come `int a[][][3]`
 - In un array unidimensionale, le parentesi dell'array sono vuote (`int a[]`)
 - Per un array multidimensionale, solo il **primo indice** può essere omesso; tutti gli altri indici devono essere specificati
- Il compilatore utilizza gli indici per determinare le posizioni di memoria degli elementi degli array multidimensionali
- Gli elementi di un array sono **memorizzati consecutivamente** in memoria, indipendentemente dal numero di indici
- In un array bidimensionale la **prima riga** è memorizzata per prima, seguita dalla **seconda riga**, e così via.

Array bidimensionali e memoria

- Specificare i valori degli indici nella dichiarazione del parametro consente al compilatore di determinare la posizione di ogni elemento
- Ogni riga in un array bidimensionale è un **array unidimensionale**.
- Per localizzare un elemento in una riga specifica, il compilatore deve conoscere il numero di elementi per riga:
 - Permette di "saltare" il numero corretto di locazioni di memoria per accedere all'elemento desiderato
- **Esempio:** Per accedere a $a[1][2]$:
 - Il compilatore "salta" i tre elementi della **prima riga** per raggiungere la **seconda riga** (riga 1)
 - Poi accede all'**elemento 2** di quella riga

Impostare gli elementi in una riga

```
for (int column = 0; column <= 3; ++column) {  
    a[2][column] = 0;  
}
```

- L'istruzione seguente imposta a zero tutti gli elementi nella riga 2 di un array 3x4 di tipo int
- Il ciclo varia solo l'**indice della colonna** (column)
- Equivalente a:

```
a[2][0] = 0;  
a[2][1] = 0;  
a[2][2] = 0;  
a[2][3] = 0;
```

Calcolare il totale degli elementi in un array bidimensionale

```
int total = 0;
for (int row = 0; row <= 2; ++row) {
    for (int column = 0; column <= 3; ++column) {
        total += a[row][column];
    }
}
```

- La struttura `for` calcola il totale degli elementi una riga alla volta
- Istruzione `for` esterna:
 - Imposta l'indice `row` a 0 per iniziare a sommare gli elementi della prima riga.
 - Incrementa `row` a 1 per sommare gli elementi della seconda riga
 - Incrementa `row` a 2 per sommare gli elementi della terza riga
- Istruzione `for` interna:
 - Somma tutti gli elementi della riga corrente specificata da `row`
- Alla fine del ciclo `for` annidato, la variabile `total` contiene la somma di tutti gli elementi dell'array `a`

Manipolazioni di un array bidimensionale

- Deitel code example fig06_17.c
- Il programma usa istruzioni for per eseguire diverse manipolazioni comuni sull'array 3-per-4 studentGrades
- Ogni riga rappresenta uno studente, e ogni colonna rappresenta un voto a uno dei quattro esami che gli studenti hanno sostenuto durante il semestre
- Le manipolazioni dell'array sono eseguite da quattro funzioni:
 - La funzione minimum (righe 38-52) trova il voto più basso fra quelli di tutti gli studenti nel semestre
 - La funzione maximum (righe 55-69) trova il voto più alto fra quelli di tutti gli studenti nel semestre
 - La funzione average (righe 72-81) calcola la media nel semestre di un particolare studente
 - La funzione printArray (righe 84-98) stampa l'array bidimensionale in un formato tabellare ordinato

Array di lunghezza variabile

- Fino a ora, la dimensione degli array è stata specificata al momento della compilazione
- Se la dimensione di un array non può essere determinata fino al momento dell'esecuzione:
 - In passato, si utilizzava l'allocazione dinamica di memoria
- C supporta gli array di lunghezza variabile (Variable-Length Array, VLA):
 - La lunghezza degli array VLA è determinata da espressioni calcolate durante l'esecuzione
- Deitel code example fig06_18.c

Recap

- Introduzione agli array
- Uso di array di caratteri per le stringhe
- Array locali statici ed automatici
- Passare gli array alle funzioni
- Ordinamento di array
- Ricerca in array
- Array multidimensionali
- Array di lunghezza variabile