

# Array e funzioni

int a[10];

a = a[0],

Marco Alberti



Dipartimento  
di Matematica  
e Informatica



Università  
degli Studi  
di Ferrara

Programmazione e Laboratorio, A.A. 2020-2021

Ultima modifica: 7 dicembre 2020

Attenzione! Questo materiale didattico è per uso personale dello studente ed è coperto da copyright.  
Ne sono vietati la riproduzione e il riutilizzo anche parziale, ai sensi e per gli effetti della legge sul diritto d'autore.

# Parametri di tipo array

E' possibile passare a una funzione un parametro di tipo array, definendolo con la sintassi consueta.

## 115\_array\_e\_funzioni/somma-elementi.c

```

1 #include <stdio.h>
2
3 int somma(int v[5]) {
4     int i, s = 0; acc reduce(+, 0, v)
5     for (i = 0; i < 5; i++)
6         s += v[i];
7     return s;
8 }
9
10 int main(void) {
11     int a[5] = {4, 1, 0, 8, 5};
12     printf("%d\n", somma(a)); // stampa 18
13 }
```



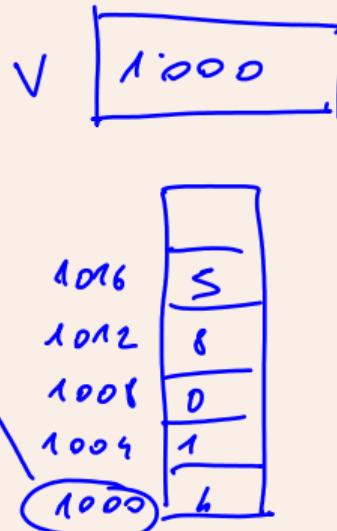
# E se cambia la dimensione?

La funzione **somma** alla diapositiva 2 funziona solo con array di 5 elementi. Usiamo una macro per facilitare eventuali modifiche della dimensione.

## 115\_array\_e\_funzioni/somma-elementi-macro.c

```

1 #include <stdio.h>
2 #define DIM 5
3
4 int somma(int v[DIM]) {
5     int i, s = 0;
6     for (i = 0; i < DIM; i++)
7         s += v[i];
8     return s;
9 }
10
11 int main(void) {
12     int a[DIM] = {4, 1, 0, 8, 5};
13     printf("%d\n", sizeof(a)); // stampa 20
14     printf("%d\n", somma(a)); // stampa 18
15 }
```



## Stessa funzione, dimensioni diverse

- E se dovessi chiamare la funzione `somma` su array di dimensioni diverse (ad esempio uno con 5 e uno con 10 elementi) nello stesso programma? La macro ha un valore unico.  
 $\text{int somma}(\text{int } v[\text{ ] }) \{ \text{dim} =$  }
- Alla funzione `somma` servirebbe un modo per determinare la dimensione dell'array passato come parametro...
- In C *questo modo non esiste!* La dimensione (`5` o `DIM`) che abbiamo messo fra quadre deve essere costante e comunque non è accessibile.
- Il parametro passato (nome dell'array) è, come sappiamo, l'indirizzo del primo elemento, non una copia dell'intero array.
- In effetti, la dimensione si può anche omettere: `int somma(int v[ ]) { ... }` è equivalente. E' compito del corpo della funzione non "sconfinare".
- Perché non calcolare la dimensione come `sizeof(v)/sizeof(int)`? Poichè il parametro `v`, cioè il nome dell'array, è un indirizzo, `sizeof(v)` restituisce la dimensione di un indirizzo (tipicamente 4, 6 o 8 byte).

# Dimensione passata come parametro

La dimensione dell'array deve essere passata dal chiamante come un ulteriore parametro intero (in questo esempio, **d1**).

## 115\_array\_e\_funzioni/somma-elementi-dimensione-variabile.c

```

1 #include <stdio.h>
2
3 int somma(int v[], int d1) {
4     int i, s = 0;
5     for (i = 0; i < d1; i++)    i=0 4
6         s += v[i];
7     return s;
8 }
9
10 int main(void) {
11     int a[5] = {4, 1, 0, 8, 5};
12     printf("%d\n", somma(a, 5)); // stampa 18
13 }
```

1010	5
1011	8
1008	0
1006	1
1000	4

# Parametri stringa

char s[] = "ciao";  
 printf("%d", strlen(s)), s [c i | a l o] \n 4

Le stringhe sono array, quindi vale quanto detto finora. Se le stringhe sono **NULL-terminated**, come quasi sempre, non occorre passare la dimensione, visto che la parte significativa termina con il carattere '\0'.

## 115\_array\_e\_funzioni/stampa-stringa.c

```

1 #include <stdio.h>
2
3 void stampa_stringa(char s[]) {
4     int i;
5     for (i = 0; s[i] != '\0'; i++)
6         printf("%c", s[i]);
7 }
8
9 int main(void) {
10     char parola[10] = "Ferrara";
11     stampa_stringa(parola);
12     printf("\n");
13     return 0;
14 }
```

1000	-
1001	\0
1002	-
1003	a
1004	r
1005	a
1006	r
1007	e
1008	r
1009	F

Ferrara

# Esempio

Che cosa stampa questo programma?

115\_array\_e\_funzioni/azzera-array.c

```

1 #include <stdio.h>
2
3 void azzera(int v[], int d1) {
4     int i;
5     for (i = 0; i < d1; i++)
6         v[i] = 0;
7 }
8
9 int main(void) {
10    int i = 0, a[3] = {1, 2, 3};
11    azzera(a, 3);
12    for (i = 0; i < 3; i++)
13        printf("%d ", a[i]);
14    printf("\n");
15    return 0;
16 }
```

void f(int n){  
 n=0,  
}  
main(){  
 int i=3;  
 f(i);  
 printf("%d", i);  
}

a[2]	3
a[1]	2
a[0]	1

1000

## Passaggio di parametri array

azzera(a);  
a[2] → void azzera(u)  
1000 v[2].



- Quando si chiama una funzione con parametro array, le si passa il nome dell'array, cioè l'indirizzo del primo elemento.
- Quindi la funzione ha accesso non a una copia dell'array nel suo record di attivazione (come avviene per scalari e strutture), ma all'array nel record di attivazione del chiamante.
- Quindi modifiche fatte all'array nella funzione chiamata si riflettono nel contenuto dell'array nel chiamante.
- In altre parole, gli elementi dell'array sono sempre passati per riferimento, anche se il nome è passato per valore come tutti i parametri.

## Azzeramento array

```
1 #include <stdio.h>
2
3 void azzera(int v[], int dl) {
4     int i;
5     for (i = 0; i < dl; i++)
6         v[i] = 0;
7 }
8
9 int main(void) {
10    int i = 0, a[3] = {1, 2, 3};
11    azzera(a, 3);
12    for (i = 0; i < 3; i++)
13        printf("%d ", a[i]);
14    printf("\n");
15    return 0;
16 }
```

i	4294951756
a[2]	0
a[1]	4294951752
a[0]	3
	4294951748
	2
	4294951744
	1

main (11)

# Azzeramento array

```
1 #include <stdio.h>
2
3 void azzera(int v[], int dl) {
4     int i;
5     for (i = 0; i < dl; i++)
6         v[i] = 0;
7 }
8
9 int main(void) {
10    int i = 0, a[3] = {1, 2, 3};
11    azzera(a, 3);
12    for (i = 0; i < 3; i++)
13        printf("%d ", a[i]);
14    printf("\n");
15    return 0;
16 }
```

i	4294951756
a[2]	0
a[1]	4294951752
a[0]	3
v	4294951748
dl	2
i	4294951744
	1
	4294951736
	4294951740
	3
	4294951724
	136254341

main (11)

azzer (5)

# Azzeramento array

```
1 #include <stdio.h>
2
3 void azzera(int v[], int dl) {
4     int i;
5     for (i = 0; i < dl; i++)
6         v[i] = 0;
7 }      0
8
9 int main(void) {
10    int i = 0, a[3] = {1, 2, 3};
11    azzera(a, 3);
12    for (i = 0; i < 3; i++)
13        printf("%d ", a[i]);
14    printf("\n");
15    return 0;
16 }
```

i	4294951756
a[2]	0
a[1]	4294951752
a[0]	3
v	4294951748
dl	2
i	4294951744
	1
	0
	4294951736
	3
	4294951740
	3
	4294951724
	0

main (11)

azzera (6)

# Azzeramento array

```
1 #include <stdio.h>
2
3 void azzera(int v[], int dl) {
4     int i;
5     for (i = 0; i < dl; i++)
6         v[i] = 0;
7 } 1
8
9 int main(void) {
10    int i = 0, a[3] = {1, 2, 3};
11    azzera(a, 3);
12    for (i = 0; i < 3; i++)
13        printf("%d ", a[i]);
14    printf("\n");
15    return 0;
16 }
```

indirizzo di  $v[i]$

$v + 1 \neq \text{sizeof}(\text{int})$

$\dots 746+4 = \dots 748$

i	4294951756
a[2]	0
a[1]	4294951752
a[0]	3
v	4294951748
dl	2
i	0

main (11)

azzera (6)

A blue bracket on the left side of the table groups the first four rows (i, a[2], a[1], a[0]), indicating they are local variables within the main function's scope. A blue circle highlights the value '2' in the 'dl' row, and a red circle highlights the value '0' in the 'a[0]' row.

# Azzeramento array

```
1 #include <stdio.h>
2
3 void azzera(int v[], int dl) {
4     int i;
5     for (i = 0; i < dl; i++)
6         v[i] = 0;
7 }
8
9 int main(void) {
10    int i = 0, a[3] = {1, 2, 3};
11    azzera(a, 3);
12    for (i = 0; i < 3; i++)
13        printf("%d ", a[i]);
14    printf("\n");
15    return 0;
16 }
```

i	4294951756
a[2]	4294951752
a[1]	4294951748
a[0]	4294951744
v	4294951736
dl	4294951740
i	4294951724

The table shows the state of variables in memory. The variable 'i' has a value of 0. The array 'a' is shown with its elements a[2], a[1], and a[0] having values 0, 0, and 0 respectively. The variable 'v' has a value of 3. The variable 'dl' has a value of 3. The variable 'i' also has a value of 2.

main (11) | azzer (6)

# Azzeramento array

```
1 #include <stdio.h>
2
3 void azzera(int v[], int dl) {
4     int i;
5     for (i = 0; i < dl; i++)
6         v[i] = 0;
7 }
8
9 int main(void) {
10    int i = 0, a[3] = {1, 2, 3};
11    azzera(a, 3);
12    for (i = 0; i < 3; i++)
13        printf("%d ", a[i]);
14    printf("\n");
15    return 0;
16 }
```

i	4294951756
a[2]	0
a[1]	4294951752
a[0]	0
v	4294951748
dl	3
i	4294951744
	3
	4294951736
	3
	4294951724
	3

main (11)

azzera (7)

# Azzeramento array

```
1 #include <stdio.h>
2
3 void azzera(int v[], int dl) {
4     int i;
5     for (i = 0; i < dl; i++)
6         v[i] = 0;
7 }
8
9 int main(void) {
10    int i = 0, a[3] = {1, 2, 3};
11    azzera(a, 3);
12    for (i = 0; i < 3; i++)
13        printf("%d ", a[i]);
14    printf("\n");
15    return 0;
16 }
```

Output

0 0 0

i	4294951756
a[2]	0
a[1]	4294951752
a[0]	0
	4294951748
	0
	4294951744
	0

main (12)

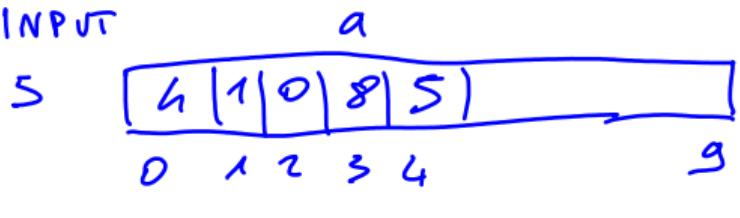
# Esercizio

## Indice massimo

Scrivere un programma che

- legga da tastiera un intero  $n$  e un array di  $n$  interi, tramite una funzione
- calcoli l'indice del massimo dell'array tramite una funzione
- visualizzi sullo schermo l'indice del massimo

INPUT



```
int indice_max(int v[], int d){
```

...  
}

$\text{indice\_max}(a, 5) \rightarrow 3$

## Notazione a puntatori per parametri di tipo array

- Poiché i parametri attuali di tipo array sono indirizzi, è possibile definire i parametri formali come puntatori anziché array.
- Ad esempio, la funzione **azzera** alla slide 7 si poteva definire come
 

```
void azzera-array(int *v, int dl) {...}
```
- Anzi, di solito le funzioni di libreria sono definite così. Ad esempio:

char \*strcpy(char \*dest, const char \*src);  
 char dest[], dest[10] = 'a';  
 void azzera (int v[], int dl)  
 int \*v  
 int a[] = {1, 2, 3};  
 azzera (a),  
 |  
 &a[0] → 1000

# Esercizio

## Procedura Insertion Sort

Scrivere una procedura che implementi l'algoritmo Insertion Sort già visto (è sufficiente adattare il codice).

```
void insertionSort (int a[], int d) {
```

```
}
```

## Esercizio

1 h 2 h X 6

## Eliminazione ripetuti

Scrivere e testare una procedura che elimini gli eventuali elementi ripetuti dall'array di interi che riceve come parametro.

```
void elrip ( int v[], int *pd){  
    }  
    dimensione logica * pd  
int main () {  
    int a[] = {1, 4, 2, 4, 1, 6};  
    int dl = 6;  
    elrip(a, &dl);  
    dl → 4  
    a → 1, 4, 2, 6, ...
```

# Esercizio

## Merge

Scrivere una procedura che riceva due array ordinati in senso crescente, e ne crei un terzo, sempre ordinato in senso crescente, contenente gli elementi dei primi due.

Ad esempio, se il primo array contiene 1, 3, 4, 4, 5 e i secondo 0, 1, 4, 6, 7, l'array risultante deve contenere 0, 1, 1, 3, 4, 4, 4, 5, 6, 7.

```
void merge (int a1[], int d1, int a2[], int d2, int a3[]){
```

```
}
```

```
    main() { int a1[5] = {1,3,4,4,5};
```

```
        int a2[5] = {0,1,4,6,7},
```

```
        int a3[20],
```

```
        merge(a1,5,a2,5,a3); ... }
```

## Array e valori di ritorno

~~char f();~~

char \* f();

Non è possibile definire una funzione con valore di ritorno di tipo array.

E' possibile invece avere un valore di ritorno di tipo puntatore (ad esempio `strcpy` ha tipo di ritorno `char *`), che è equivalente.

E' però scorretto

- ① definire una funzione con tipo di ritorno puntatore (ad esempio `int *f(void)`);
- ② creare un array come variabile locale nella funzione (ad esempio `int a[10];`);
- ③ restituire l'array (ad esempio `return a;`)

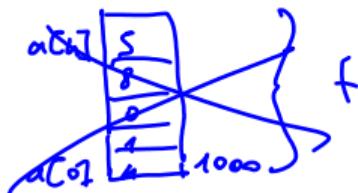
Questo perché staremmo restituendo l'indirizzo di un'area di memoria contenuta nel record di attivazione della funzione, che verrà distrutto all'uscita della funzione.

```

int main()
{
    int *v;
    v = f();
    cout << v[3];
}

int * f(void)
{
    int a[5] = {4,1,0,8,5};
    return a;
}
    
```

indefinito

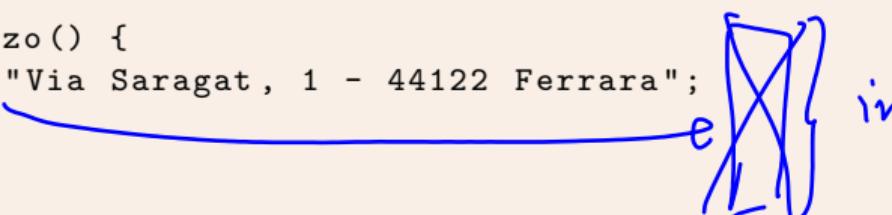


# Array come valore di ritorno

Esempio scorretto:

## 115\_array\_e\_funzioni/return-array.c

```
1 #include <stdio.h>
2
3 char* indirizzo() {
4     char s[] = "Via Saragat, 1 - 44122 Ferrara";
5     return s;
6 }
7
8 int main(void) {
9     char* ind;
10    ind = indirizzo();
11    printf("%s\n", ind);
12    return 0;
13 }
```



Il compilatore avverte della scorrettezza con un warning ("function returns address of local variable").

# Array come parametro di output

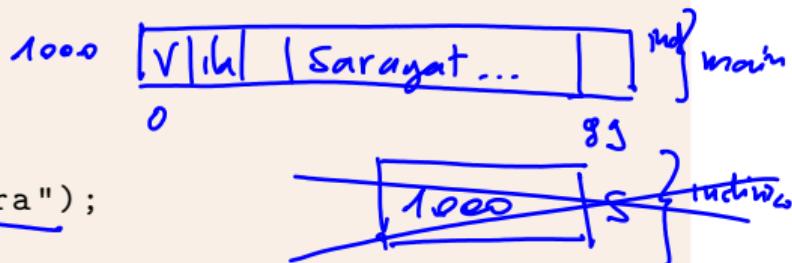
E' corretto invece allocare l'array nel chiamante e passarlo come parametro

## 115\_array\_e\_funzioni/array-parametro-output.c

```

1 #include <stdio.h>
2 #include <string.h> output
3
4 void indirizzo(char *s) {
5     strcpy(s, "Via Saragat, 1 - 44122 Ferrara");
6 }
7
8 int main(void) {
9     char ind[100];
10    indirizzo(ind);
11    printf("%s\n", ind);
12    return 0;
13 }
```

*origin*



# Array campi di strutture

115\_array\_e\_funzioni/persona.c

```

1 typedef struct {
2     char nome[30];
3     int eta;
4 } Persona;
```

```

intf (Persona p) {
    p.nome[0] = 'd';
    return p.eta;
}
```

```

main() {
    Persona q = {"Caesar",
                  44};
    f(q);
}
```

- Sappiamo che le strutture possono essere passate a funzioni come parametri (per valore, quindi creando una copia).
- Che cosa succede se la struttura contiene un array? Gli elementi dell'array sono passati per valore o per riferimento?
- Poiché l'array fa parte della struttura copiata, i suoi elementi sono una copia di quelli nella struttura passata come parametro attuale.
- Quindi in particolare eventuali modifiche non si riflettono sul parametro attuale.

# Passaggio parametro

Ad esempio, scriviamo una procedura che modifica l'array contenuto nel suo parametro struttura:

115\_array\_e\_funzioni/campo.c

```

1 void f(Persona p) {
2     p.nome[0] = 'A';
3 }
4
5 int main() {
6     Persona per = {"Mario", 25};
7     f(per); // per.nome rimane invariato
8     return 0;
9 }
```

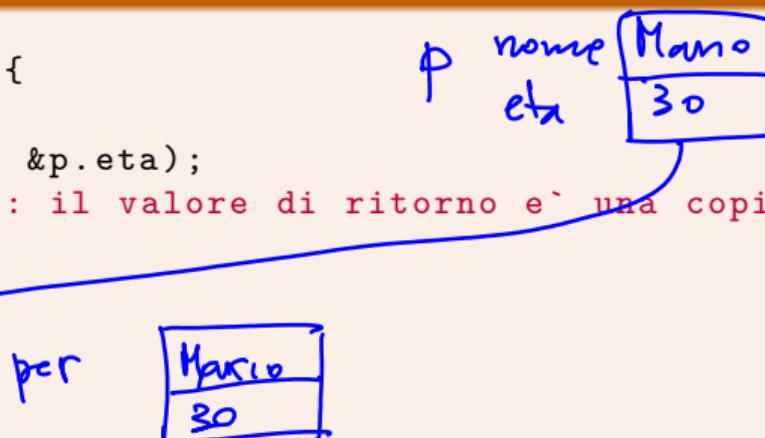
void f (Persona \* pp) {  
 (\*pp).nome[0] = 'A'  
 }  
 ↑  
 f (&per)

# Valore di ritorno

E' corretto passare una variabile locale struttura contenente un array come valore di ritorno: l'intera struttura viene copiata.

## 115\_array\_e\_funzioni/valore-ritorno.c

```
1 Persona leggiPersona() {  
2     Persona p;  
3     scanf("%s%d", p.nome, &p.eta);  
4     return p; // corretto: il valore di ritorno e' una copia di p  
5 }  
6  
7 int main() {  
8     Persona per;  
9     per = leggiPersona();  
10    return 0;  
11 }
```



## Esercizio (da prova teorica del 22/2/2019)

Detta  $m$  la rappresentazione come intero decimale del proprio numero di matricola, la funzione di prototipo `int f(int d)` restituisce

- la  $d$ -esima cifra di  $m$  a partire da sinistra se  $d$  è compreso fra 1 e il numero di cifre di  $m$ ;
- 0 altrimenti.

Per esempio, se il numero di matricola è `654321`, la chiamata `f(2)` restituisce `5`, mentre le chiamate `f(0)` e `f(7)` restituiscono `0`.

Quali caratteri stampa il programma costituito dal codice alla slide 21 e dalla definizione della funzione `f`? Motivare la risposta.

# Codice per l'esercizio alla slide 20

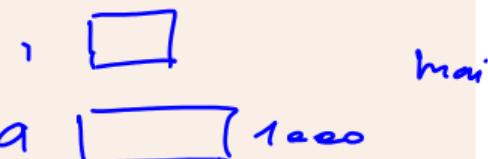
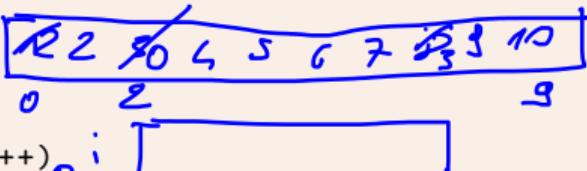
$$\begin{matrix} 2 & 3 & < & 7 & 3 \\ \star & (\alpha + z) & & & \\ \alpha & \in & \mathbb{Z} & & \end{matrix} = 0$$

## 115\_array\_e\_funzioni/20190222-t1.c

```

1 #include <stdio.h>
2
3 int f(int d);
4
5 void g(int a[]) {
6     int i, int + a1000;
7     for (i = 0; i < 3; i++) {
8         *(a + f(a[f(i)])) = f(i);
9     }
10    .
11 int main(int argc, char* argv[]) {
12     int i, a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
13     g(a);
14     for (i = 0; i < 10; i++)
15         printf("%d ", a[i]);
16     printf("\n");
17     return 0;
18 }
```

f(2) → 3



2 2 0 4 5 6 7 3 3 10