



UNIVERSITÀ DI PARMA

# Gli Array

*When you have more data you win*

- Array monodimensionali
  - Definizione
  - Uso
  - Inizializzazione
  - Errori
- Array multidimensionali

SUMMARY



- Fino ad ora abbiamo visto variabili “singole”
  - Gestisco uno e un solo “dato”
- Non adatte a gestire grandi moli di dati
- Eppure informatica nasce proprio per trattare dati
  - Informazione + Automatica  $\rightarrow$  Informatica
- Soluzione C  $\rightarrow$  Array

- **In C** si definisce array una struttura dati che permette di memorizzare **un numero definito** di dati tutti dello **stesso tipo**
- Caratteristiche:
  - Come per le variabili l'array ha un **nome** e uno specifico **tipo di dato**
  - Ciascun dato è detto **elemento** o componente
  - Gli elementi sono **ordinati**
    - A ogni elemento è associato un **indice numerico**

- Definizione array monodimensionale (vettore)

```
<tipo array> <nome array>[<numero elementi>];
```

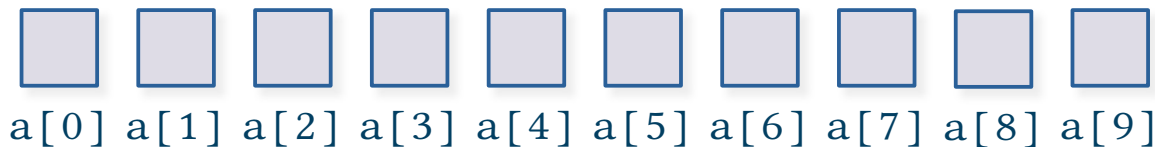
- Esempio

```
int numeri[100];
```

```
// definisco un array di nome "numeri" composto da  
// 100 elementi di tipo int
```

- Definire un array di tipo  $x$  composto da  $N$  elementi equivale a
  - Definire  $N$  variabili di tipo  $x$ 
    - Ma è più pratico
- Come accedo ai singoli elementi di un array?
  - Gli elementi sono ordinati
  - Indice da 0 a  $N-1 \rightarrow$  tra  $[ e ]$

Array  $a$  di 10  
elementi



- Salvo indice, stessa sintassi variabili “regolari” per scrittura/lettura

<code>int a [100];</code>	→ array di 100 interi
<code>...</code>	
<code>a[0] = 37;</code>	→ scrivo in primo elemento
<code>...</code>	
<code>printf(“%d”, a[8]);</code>	→ leggo nono elemento
<code>...</code>	
<code>scanf(“%d”, &amp;a[99]);</code>	→ scrivo in ultimo elemento

- Anche gli array possono essere inizializzati quando si definiscono

- Qualche differenza:

- **Inizializzazione completa:**

```
float costanti[3] = {3.14, 2.718, 1.618};
```

- **Inizializzazione completa con omissione dimensioni:**

```
short primi[] = {1,3,5,7,11,13,17,19,23,29,31,37,41,43};
```

- **Inizializzazione parziale:**

```
char lettere[50] = {'P', 'i', 'e', 'r', 'o'};
```

```
int tuttizeri[100]={0}; // C99 does not allow = {};
```

```
float sparsevector[30]={[3]=4.5, [29]=1.7}; // C99
```

In questi ultimi casi gli elementi non definiti sono comunque inizializzati a 0

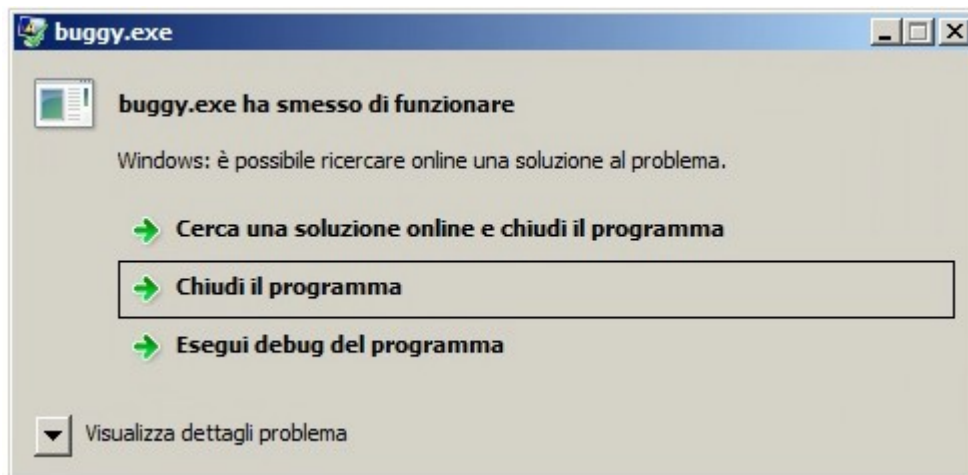


- I singoli elementi di un array sono memorizzati consecutivamente in memoria:
- L'accesso ad un elemento avviene come: indirizzo dell'array+indice
  - Ci torneremo...

```
int a[5]={3,4,8,3129,77};
```

a[0]	3	0x120
a[1]	4	0x124
a[2]	8	0x128
a[3]	3129	0x12C
a[4]	77	0x130

- Iniziando a usare gli array vedremo spesso qualche errore
- Perché?



- Il C non controlla coerenza indice  $\leftrightarrow$  dimensione array

```
int dummy[100];  
dummy[100]=7; // l'ultimo elemento ha indice 99  
dummy[-1]=0;  // -1?
```
- Le ultime due righe sono un esempio di **buffer overrun**
  - malfunzionamento del programma
  - malfunzionamento del sistema

- Gli array non sono limitati ad un solo indice
  - Ovvero a una sola dimensione
- Esempi di definizioni di array multidimensionali
  - `int matrice[10][4];` // 10 righe e 4 colonne?
  - `long b[2][3][7][12][1024];`
- Anche in questo caso possibile inizializzare
  - `int m1[2][3]={3,4,8,1,2,7};`
  - `int m2[2][3]={ {3,4,8}, {1,2,7} };`

- Sebbene intuitivamente si attribuiscono dimensioni
- Anche gli array multidimensionali sono memorizzati “vettorialmente”
- Posso vederli come
  - Array di array

```
int a[3][2]={3,4,8,3129,77,9};
```

a[0][0]	3	0x120
a[0][1]	4	0x124
a[1][0]	8	0x128
a[1][1]	3129	0x12C
a[2][0]	77	0x130
a[2][1]	9	0x134



UNIVERSITÀ DI PARMA

# Gli Array



KEEP  
CALM  
IT'S  
QUESTION  
TIME

*When you have more data you win*