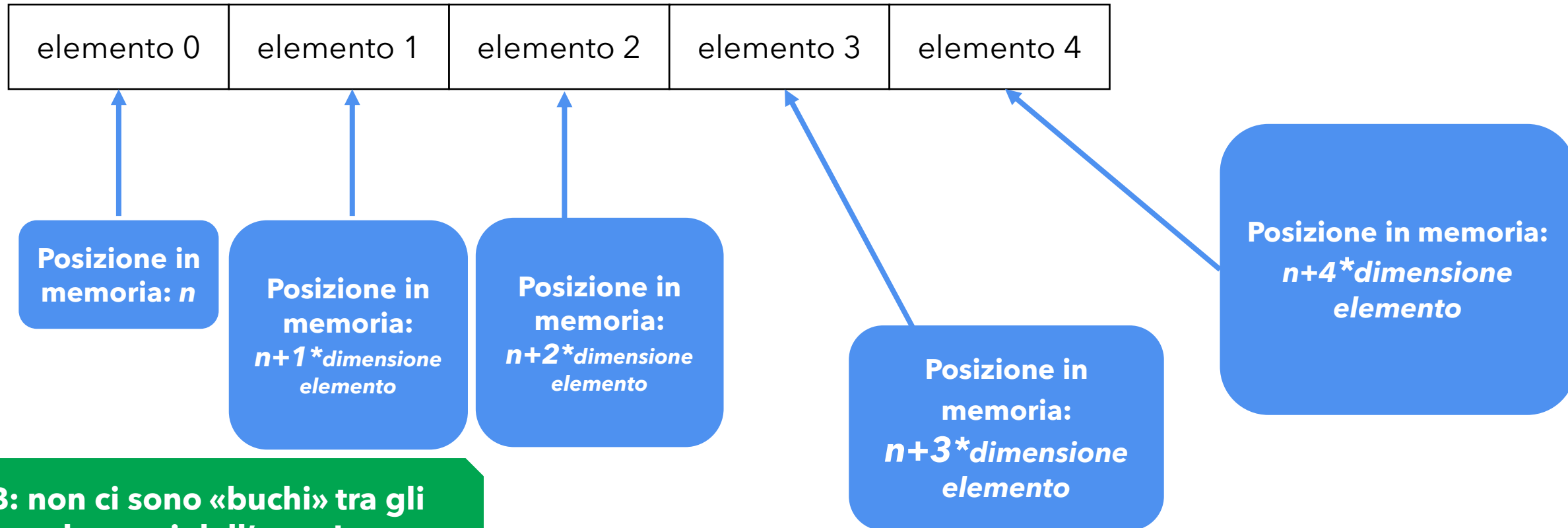


# Gli array (vettori) in C++

**Liceo G.B. Brocchi**  
**Classi seconde Scientifico - opzione scienze applicate**  
Bassano del Grappa, Settembre 2022

# Cosa sono gli array in C++ (C-style arrays)

- Un array è una sequenza di oggetti omogenei (dello **stesso tipo** e quindi della stessa dimensione in byte) allocati in posizioni di memoria contigue



# Cosa sono gli array in C++ (C-style arrays)

```
int vect_in[20];           //array of 20 integers  
vect_in[0];                //array's first element (index == 0)
```

008FFBA4: indirizzo di memoria di vect\_in[0]

008FFBA8: indirizzo di memoria di vect\_in[1]

008FFBAC: indirizzo di memoria di vect\_in[2]

008FFBB0: indirizzo di memoria di vect\_in[3]

008FFBB4: indirizzo di memoria di vect\_in[4]

008FFBB8: indirizzo di memoria di vect\_in[5]

008FFBBC: indirizzo di memoria di vect\_in[6]

. . .  
. . .

**La differenza tra l'indirizzo dell'elemento  $i$ -esimo e l'indirizzo dell'elemento  $(i-1)$ -esimo è 4 perché un int occupa 4 byte**

# Cosa sono gli array in C++ (C-style arrays)

```
char vect_ch[10];    //array of 10 characters
```

010FFE94: indirizzo di memoria di vect\_ch[0]

010FFE95: indirizzo di memoria di vect\_ch[1]

010FFE96: indirizzo di memoria di vect\_ch[2]

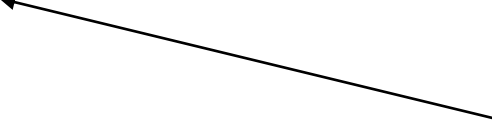
010FFE97: indirizzo di memoria di vect\_ch[3]

. . .  
. . .

**La differenza tra l'indirizzo dell'elemento  $i$ -esimo e l'indirizzo dell'elemento  $(i-1)$ -esimo è 1 perché un char occupa 1 byte**

# Cosa sono gli array in C++ (C-style arrays)

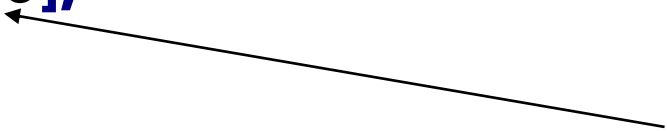
```
int size = 5;  
int v[size];
```



- Non compila! La dimensione dell'array non può essere una variabile, perché deve essere nota a *compile time* (al momento della compilazione del programma)

---

```
const int size = 5;  
int v[size];
```



- Compila! La dimensione dell'array in questo caso è una **costante** nota a compile-time

# Cosa sono gli array in C++ (C-style arrays)

```
int ai[30];
```

- Cosa conterrà la memoria allocata per l'array *ai*?



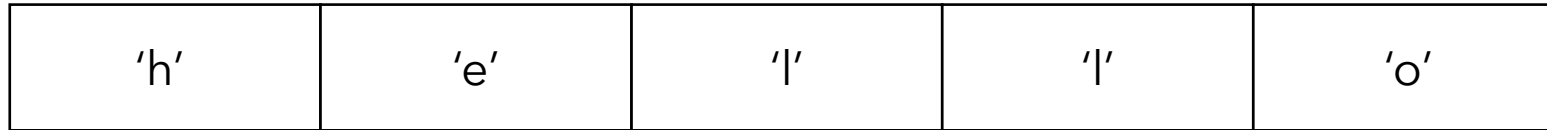
0 5242392 13 5242600 14598928 -334170950 -2 5242476 14631564 5242464 5242456 5242468  
5242484 5242488 2 2 -323339802 5242492 14631425 14768304 14700998 5242504 14587344  
14700998 5242516 14765836 5242524 14553135 0 5242548



**Memoria non inizializzata. Ci sono i valori che c'erano prima della dichiarazione nelle stesse locazioni di memoria.**

# Cosa sono gli array in C++ (C-style arrays)

```
char ac[] = {'h', 'e', 'l', 'l', 'o'};
```



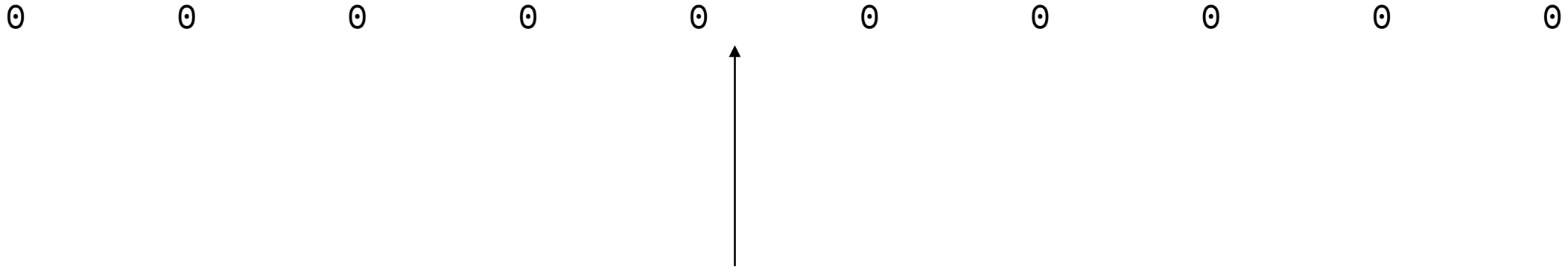
**La dimensione viene calcolata contando il numero di elementi della lista di inizializzazione**

# Cosa sono gli array in C++ (C-style arrays)

```
char ac[5] = {'h', 'e', 'l', 'l', 'o'}; //dimensione specificata ma si può omettere
```

---

```
int vi[10] = {};
```



*contenuto dell'array. I 10 elementi vengono inizializzati a 0 grazie all'inizializzatore {}*



# Cosa sono gli array in C++ (C-style arrays)

```
double vi[5] = {4.5, 6.0, 3.2, 30.2, 3.43, 3.14, 6.28};
```



**non compila!**

lecture2.cpp(134): error C2078: troppi inizializzatori

# Cosa sono gli array in C++ (C-style arrays)

```
int vi[8] = { 1, 3, 2, 4 };
```

1

3

2

4

0

0

0

0

# Cosa sono gli array in C++ (C-style arrays)

```
int vi[8] = { 1, 3, 2, 4 };
```

1

3

2

4

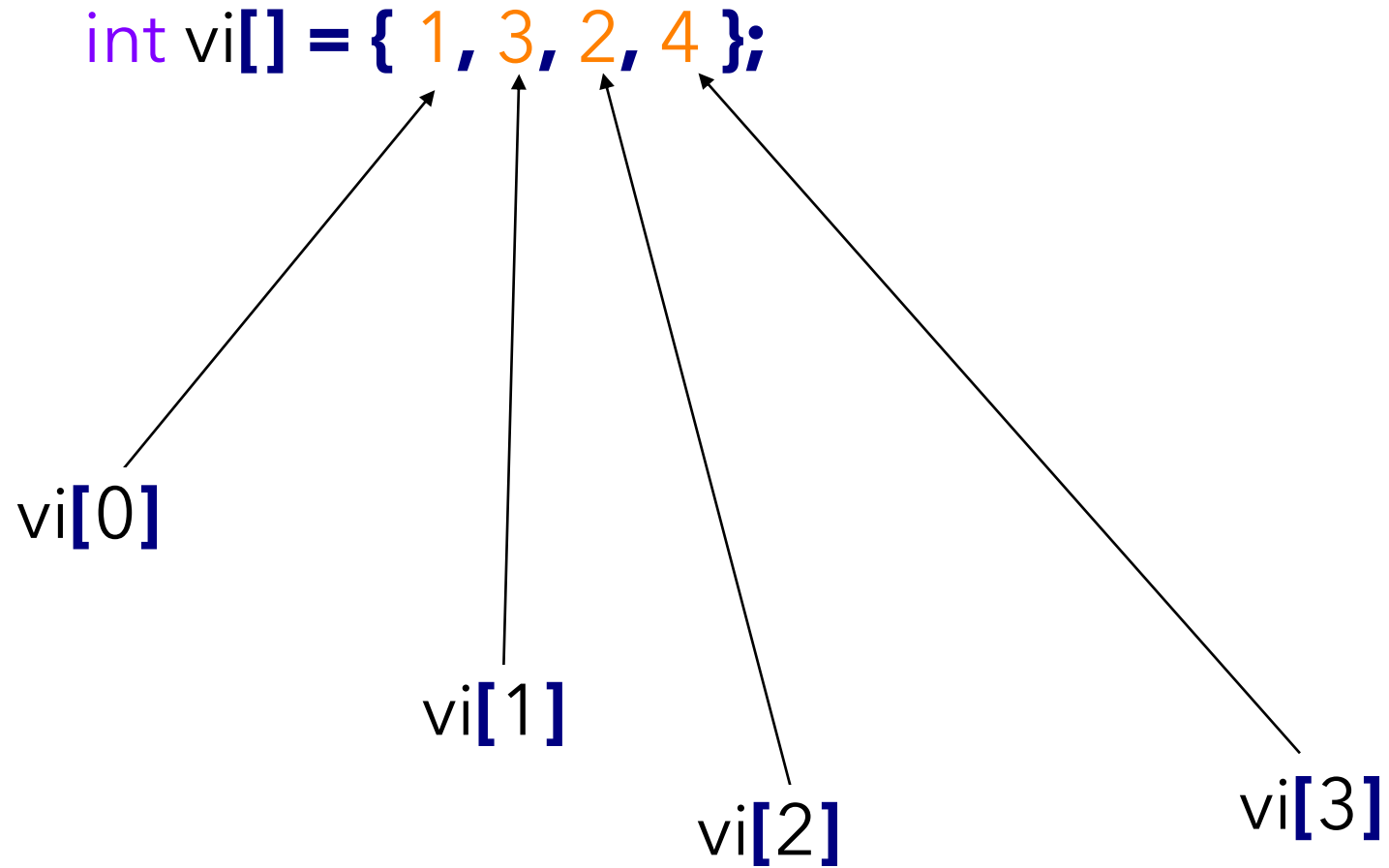
0

0

0

0

# Come accedere agli elementi di un array



# Come accedere agli elementi di un array

```
int vi[] = { 21, 12, 6, 8 };
```

21	12	6	8
vi[0]	vi[1]	vi[2]	vi[3]

```
cout << vi;
```

—————→ 0093FD04

**Vi aspettavate questo output?**

# Come accedere agli elementi di un array

```
int vi[] = { 21, 12, 6, 8 };
```

```
for (int i = 0; i < 4; i++) {  
    cout << vi[i] << '\t';  
}
```



**Ecco come si stampa il contenuto di un array**

# Come accedere agli elementi di un array

```
int vi[] = { 21, 12, 6, 8 };
```

```
for (int i = 0; i < 30; i++) {  
    cout << vi[i] << '\t';  
}
```



**Compila! In C++ non c'è alcun controllo sull'indice utilizzato per accedere ad un elemento dell'array.  
Ecco l'output:**

```
21      12      6      8      -1461440336      11532736      2921131 1  
11893664      11954752      -1461440504      2921267 2921267 9383936 0      0  
0      11532684      0      11532824      2933408 -1469777448      0  
11532752      1983932217      9383936 1983932192      11532840      2005176274  
9383936
```

# Come accedere agli elementi di un array

- Scrivere un programma che somma e memorizza in una variabile tutti gli elementi maggiori di 7 in un array di interi (utilizzare un ciclo for);
- Scrivere un programma che stampa gli elementi di un array di interi di dimensione 10 con un ciclo for. La stampa deve fermarsi prima del primo elemento dell'array con valore 42 (se c'è);



# Come accedere agli elementi di un array

```
int arr[] = { 21, 12, 6, 8 };  
int acc = 0;  
for (int i = 0; i < 4; i++) {  
    if (arr[i] > 7) {  
        acc++;  
    }  
}
```

# Come accedere agli elementi di un array

```
int arr[] = { 21, 12, 6, 8, 56, 45, 5, 8 };  
for (int i = 0; i < 8 && arr[i] != 42; i++) {  
    cout << arr[i] << '\t';  
}
```

- prima volta in cui viene valutata la condizione di permanenza:  $i == 0$ ,  $0 < 8$  e  $arr[0] == 21 \neq 42 \rightarrow \text{true}$
- seconda volta in cui viene valutata la condizione di permanenza:  $i == 1$ ,  $1 < 8$  e  $arr[1] == 12 \neq 42 \rightarrow \text{true}$
- terza volta in cui viene valutata la condizione di permanenza:  $i == 2$ ,  $2 < 8$  e  $arr[2] == 6 \neq 42 \rightarrow \text{true}$
- quarta volta in cui viene valutata la condizione di permanenza:  $i == 3$ ,  $3 < 8$  e  $arr[3] == 8 \neq 42 \rightarrow \text{true}$
- quinta volta in cui viene valutata la condizione di permanenza:  $i == 4$ ,  $4 < 8$  e  $arr[4] == 56 \neq 42 \rightarrow \text{true}$
- sesta volta in cui viene valutata la condizione di permanenza:  $i == 5$ ,  $5 < 8$  e  $arr[5] == 45 \neq 42 \rightarrow \text{true}$
- settima volta in cui viene valutata la condizione di permanenza:  $i == 6$ ,  $6 < 8$  e  $arr[6] == 5 \neq 42 \rightarrow \text{true}$
- ottava volta in cui viene valutata la condizione di permanenza:  $i == 7$ ,  $7 < 8$  e  $arr[7] == 8 \neq 42 \rightarrow \text{true}$
- nona volta in cui viene valutata la condizione di permanenza:  $i == 8$ ,  $8 < 8$  e ...  $\rightarrow \text{false}$

# Come accedere agli elementi di un array

```
int arr[] = { 21, 12, 42, 8, 56, 45, 5, 8 };  
for (int i = 0; i < 8 && arr[i] != 42; i++) {  
    cout << arr[i] << '\t';  
}
```

- prima volta in cui viene valutata la condizione di permanenza:  $i == 0$ ,  $0 < 8$  e  $arr[0] == 21 \neq 42 \rightarrow$  **true**
- seconda volta in cui viene valutata la condizione di permanenza:  $i == 1$ ,  $1 < 8$  e  $arr[1] == 12 \neq 42 \rightarrow$  **true**
- terza volta in cui viene valutata la condizione di permanenza:  $i == 2$ ,  $2 < 8$  e  $arr[2] == 42 \neq 42 \rightarrow$  **false**

# Drills

- Scrivere un programma che calcoli e memorizzi i numeri di Fibonacci fino all' n-esimo utilizzando un array costruito così:


```
const int size = ...;
```

```
int fibonacci_array[size] = {0, 1};
```

- *L'i-esimo numero di Fibonacci è la somma dei 2 precedenti. È per questo che dobbiamo partire con 0 e 1 già inseriti nell'array.*
- *L'i-esimo numero di Fibonacci deve essere memorizzato in fibonacci\_array[i]*

# Drills

Spiegare perché è comodo utilizzare la costante *size*



```
const int size = 10;
int fibonacci_array[size] = {0, 1};

for (int i = 2; i < size; i++) {
    fibonacci_array[i] = fibonacci_array[i - 1] + fibonacci_array[i - 2];
}

cout << "Fibonacci series, up to the " << size-1 << "-th element is: " << endl;
for (int i = 0; i < size; i++) {
    cout << fibonacci_array[i] << '\t';
}
cout << endl;
```

# Drills

- Scrivere un programma che stampi il rapporto tra l' $i$ -esimo e l' $(i-1)$  esimo numero della successione di Fibonacci memorizzata nell'array visto sopra. Attenzione: la divisione deve produrre un numero decimale.

Collegamento con la matematica: studiare la **Sezione Aurea**

**Facoltativo:** preparare una piccola presentazione sulla Sezione Aurea

# Drills

- Scrivere un programma che stampi il rapporto tra l'i-esimo e l'(i-1) esimo numero della successione di Fibonacci memorizzata nell'array visto sopra. Attenzione: la divisione deve produrre un numero decimale.

```
for (int i = 2; i < size; i++) {  
    cout << (double)fibonacci_array[i] / (double)fibonacci_array[i - 1] << '\t';  
}
```

# Fibonacci senza array

```
int fib_pp = 0;
int fib_p = 1;
int fib_i;

cout << fib_pp << '\t' << fib_p << '\t';

for (int i = 2; i < 10; i++){
    fib_i = fib_p + fib_pp;
    fib_pp = fib_p;
    fib_p = fib_i;

    cout << fib_i << '\t';
}

cout << endl;
```



*comodi gli array vero?*



# Fibonacci senza array

iterazione di indice 0, fittizia	fib_i	fib_pp	fib_p				
valori delle variabili	non definito	0	1				
iterazione di indice 1, fittizia	fib_i	fib_pp	fib_p				
valori delle variabili	non definito	0	1				
iterazione di indice 2	fib_i	fib_pp	fib_p				
valori delle variabili	1	1	1				
iterazione di indice 3		fib_i	fib_pp	fib_p			
valori delle variabili		2	1	2			
iterazione di indice 4			fib_i	fib_pp	fib_p		
valori delle variabili			3	2	3		
iterazione di indice 5				fib_i	fib_pp	fib_p	
valori delle variabili				5	3	5	
iterazione di indice 6					fib_i	fib_pp	fib_p
valori delle variabili					8	5	8