

Fondamenti di Programmazione (A)

II - Strutture dati (Array)

Vincenzo Arceri - Università degli Studi di Parma - vincenzo.arceri@unipr.it

Puntate precedenti

- Assegnamenti e dichiarazioni
- Statement di selezione (`if`, `switch`)
- Statement iterativi (`while`, `do – while`, `for`)
- Tipi primitivi (`int`, `float`, `bool`, `char`)

Tipi strutturati

- Collezione di dati organizzati “in qualche modo” (e.g., in sequenza)

Tipi strutturati

- Collezione di dati organizzati “in qualche modo” (e.g., in sequenza)
- In C++ (ma quasi in qualsiasi altro linguaggio)
 - array
 - struct
 - union
 - classes

Tipi strutturati

- Collezione di dati organizzati “in qualche modo” (e.g., in sequenza)
- In C++ (ma quasi in qualsiasi altro linguaggio)
 - array (oggi e nelle prossime lezioni)
 - struct (più avanti nel corso)
 - union (non trattato)
 - classes (Fondamenti di programmazione B - prossimo semestre)

Array

Definizione

Array

Definizione

- Sequenza di elementi **dello stesso tipo** che possono essere individuati da un **indice/posizione**

Array

Definizione

- Sequenza di elementi **dello stesso tipo** che possono essere individuati da un **indice/posizione**

0	1	2	3	4
5	-1	3	7	9

Array

Definizione

- Sequenza di elementi **dello stesso tipo** che possono essere individuati da un **indice/posizione**

0	1	2	3	4
5	-1	3	7	9

- Ogni elemento della sequenza è detto **cella** dell'array

Array

Definizione

- Sequenza di elementi **dello stesso tipo** che possono essere individuati da un **indice/posizione**

0	1	2	3	4
5	-1	3	7	9

- Ogni elemento della sequenza è detto **cella** dell'array
- Il numero degli elementi dell'array è la sua **lunghezza** (nell'esempio 5)

Array

Definizione

- Sequenza di elementi **dello stesso tipo** che possono essere individuati da un **indice/posizione**

0	1	2	3	4
5	-1	3	7	9

- Ogni elemento della sequenza è detto **cella** dell'array
- Il numero degli elementi dell'array è la sua **lunghezza** (nell'esempio 5)
- Un array può anche avere lunghezza = 0 (array vuoto)

Dichiarazione di un array

In C++

t A[k];

Dichiarazione di un array

In C++

$$t \ A[k];$$

- t è un tipo (a sua volta può essere strutturato) e indica il tipo che ogni cella dell'array può contenere
- k è un'espressione e indica la lunghezza dell'array
- A è un identificatore

Dichiarazione di un array

In C++

$$t \ A[k];$$

- t è un tipo (a sua volta può essere strutturato) e indica il tipo che ogni cella dell'array può contenere
- k è un'espressione e indica la lunghezza dell'array
- A è un identificatore

“Dichiarazione di una variabile A di tipo array, con lunghezza k e che può contenere valori di tipo t ”

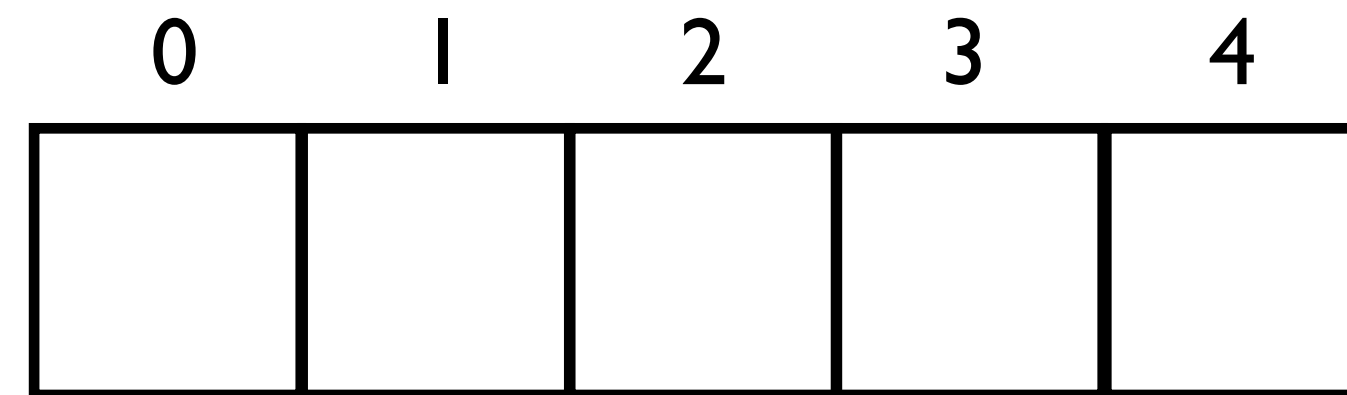
Dichiarazione di un array

Esempi

Dichiarazione di un array

Esempi

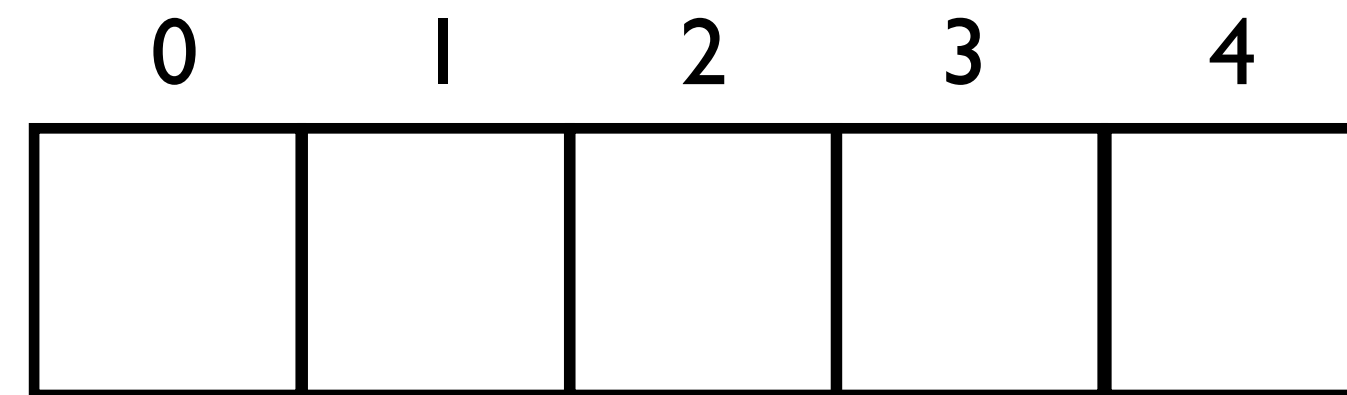
```
int a[5];
```



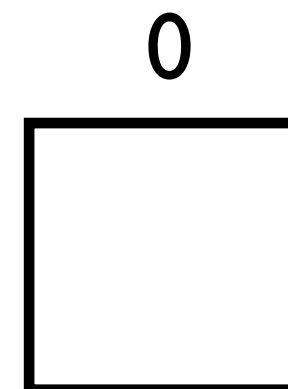
Dichiarazione di un array

Esempi

```
int a[5];
```



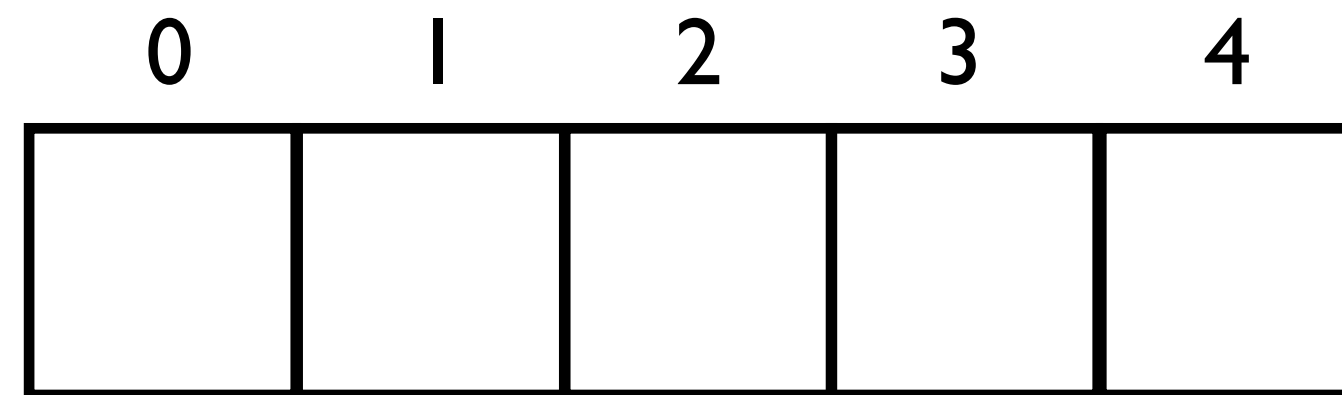
```
float x[1];
```



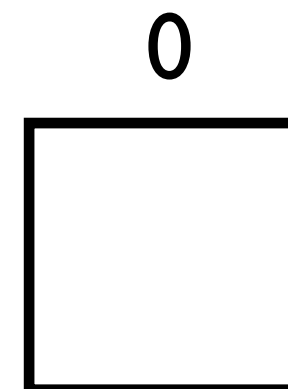
Dichiarazione di un array

Esempi

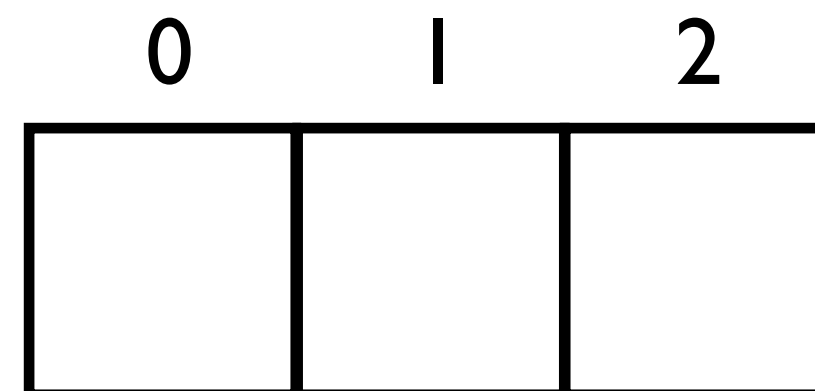
```
int a[5];
```



```
float x[1];
```



```
char w[3];
```



Dichiarazione di un array

Con inizializzazione

$$t \ A[k] = \{c_0, c_1, \dots, c_m\};$$

Dichiarazione di un array

Con inizializzazione

$$t\ A[k] = \{c_0, c_1, \dots, c_m\};$$

- Inizializzazione dei primi m elementi dell'array dichiarato
 - $m = k - 1$: vengono inizializzati tutti gli elementi dell'array
 - $m \geq k$: errore a *compile-time*
 - $m < k - 1$: vengono inizializzati i primi m elementi coi valori forniti, mentre gli altri vengono inizializzati a 0

Dichiarazione di un array

Esempi

Dichiarazione di un array

Esempi

```
int x[3] = {5, 7, -2};
```

0	1	2
5	7	-2

Dichiarazione di un array

Esempi

```
int x[3] = {5, 7, -2};
```

0	1	2
5	7	-2

```
int y[5] = {3, 8, -1};
```

0	1	2	3	4
3	8	-1	0	0

Dichiarazione di un array

Esempi

```
int x[3] = {5, 7, -2};
```

0	1	2
5	7	-2

```
int y[5] = {3, 8, -1};
```

0	1	2	3	4
3	8	-1	0	0

```
int z[2] = {3, 8, -1};
```

Errore a compile-time

Dichiarazione di un array

Con inizializzazione

Dichiarazione di un array

Con inizializzazione

- E' possibile omettere la dimensione dell'array se inizializzato
- La dimensione viene *inferita* automaticamente dal numero degli inizializzatori

Dichiarazione di un array

Con inizializzazione

- E' possibile omettere la dimensione dell'array se inizializzato
- La dimensione viene *inferita* automaticamente dal numero degli inizializzatori

```
int x[] = {1, 7, -2};
```

array di int di lunghezza 3

0	1	2
1	7	-2

Dichiarazione di un array

Con inizializzazione

- E' possibile omettere la dimensione dell'array se inizializzato
- La dimensione viene *inferita* automaticamente dal numero degli inizializzatori

```
int x[] = {1, 7, -2};
```

array di int di lunghezza 3

0	1	2
1	7	-2

- **NB:** è possibile assegnare gli elementi degli array *"in blocco"* (i.e., tramite lista di inizializzatori) **solo** in fase di dichiarazione dell'array

Dichiarazione di un array

Con inizializzazione

- E' possibile omettere la dimensione dell'array se inizializzato
- La dimensione viene *inferita* automaticamente dal numero degli inizializzatori

```
int x[] = {1, 7, -2};
```

array di int di lunghezza 3

0	1	2
1	7	-2

- **NB:** è possibile assegnare gli elementi degli array "in blocco" (i.e., tramite lista di inizializzatori) **solo** in fase di dichiarazione dell'array

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

Errore a *compile-time*

Selezione di un elemento

Selezione di un elemento

0	1	2	3	4
5	-1	3	7	9

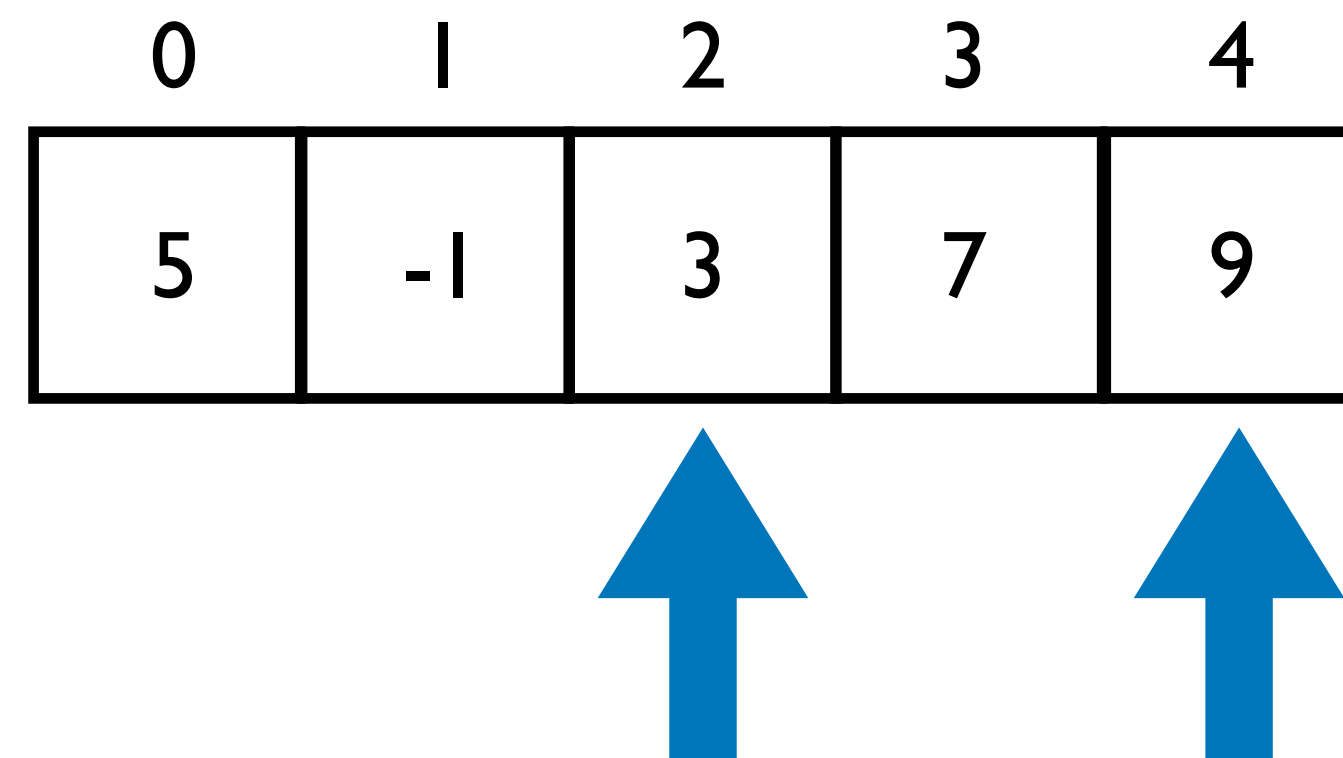
Selezione di un elemento

0	1	2	3	4
5	-1	3	7	9

- Dato un array, è possibile *accedere* (in lettura e scrittura) ai suoi elementi tramite **accesso diretto**

Selezione di un elemento

0	1	2	3	4
5	-1	3	7	9

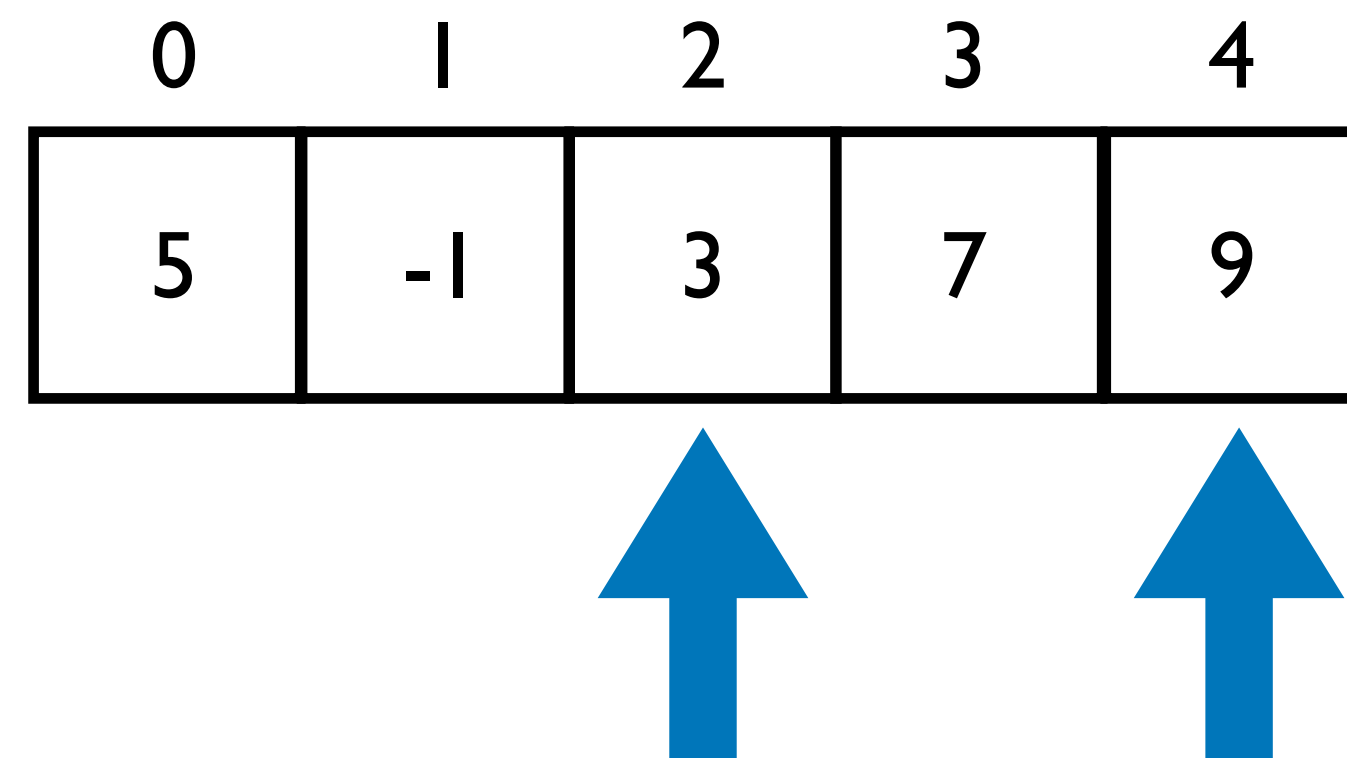


The diagram shows a horizontal array of five cells. Above each cell is its index: 0, 1, 2, 3, and 4. The cells contain the values 5, -1, 3, 7, and 9 respectively. Below the array, there are two blue arrows pointing upwards. The first arrow points to the cell at index 2 (containing the value 3). The second arrow points to the cell at index 4 (containing the value 9). This illustrates direct access to elements in an array.

- Dato un array, è possibile *accedere* (in lettura e scrittura) ai suoi elementi tramite **accesso diretto**

Selezione di un elemento

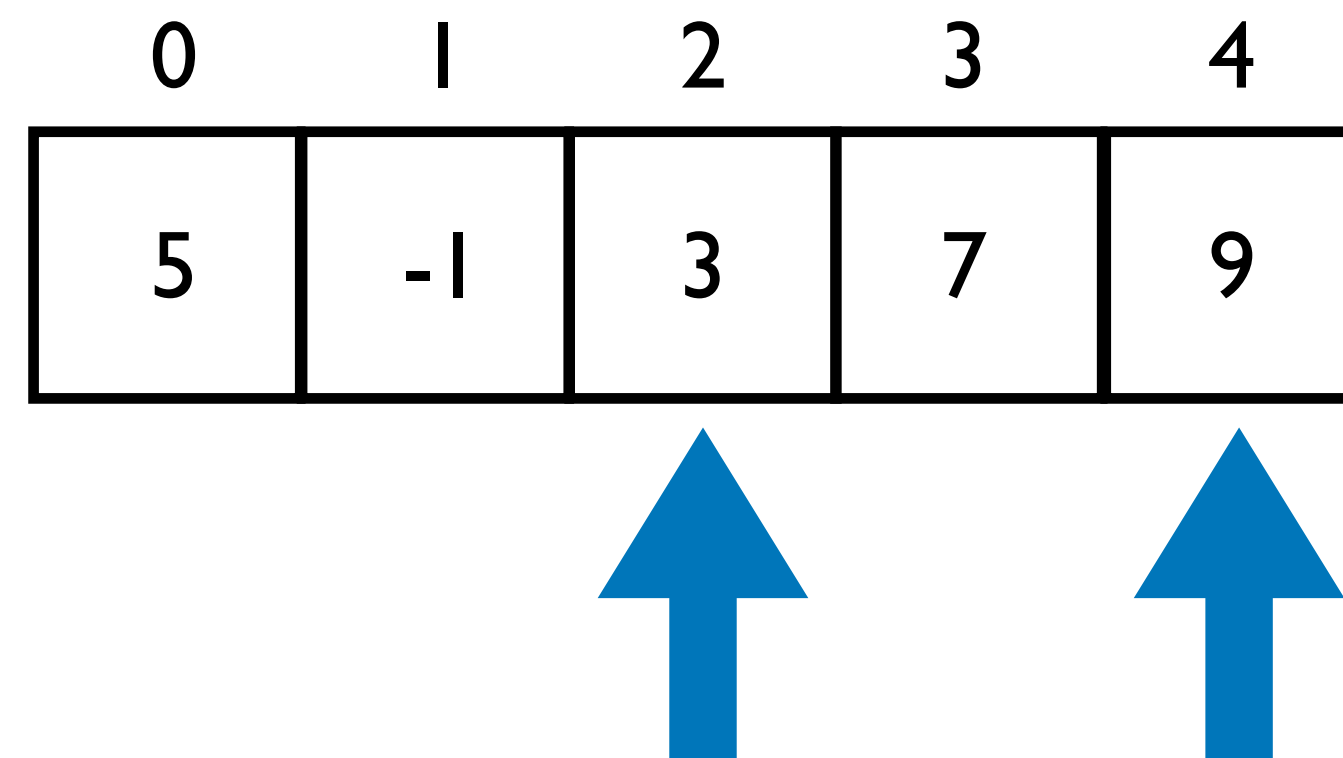
0	1	2	3	4
5	-1	3	7	9



- Dato un array, è possibile *accedere* (in lettura e scrittura) ai suoi elementi tramite **accesso diretto**

$A[exp]$

Selezione di un elemento

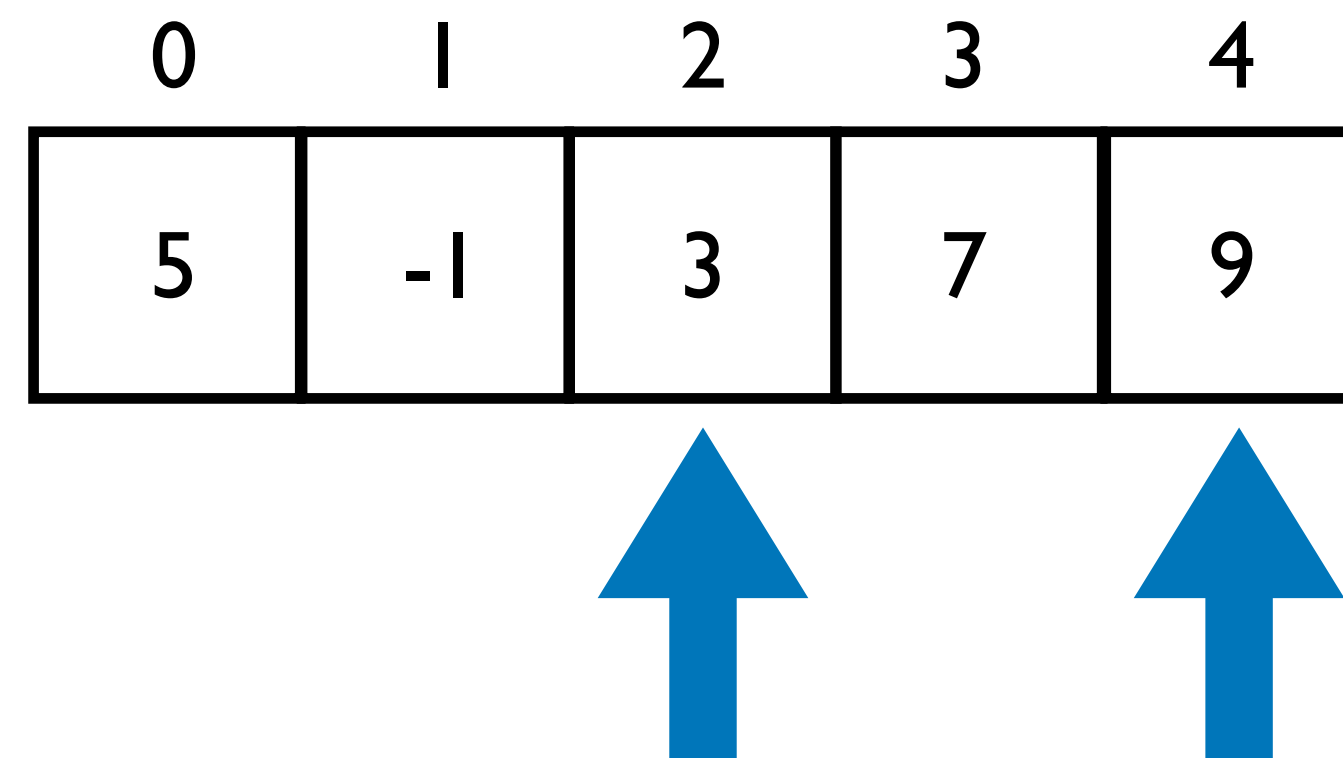


- Dato un array, è possibile *accedere* (in lettura e scrittura) ai suoi elementi tramite **accesso diretto**

$A[exp]$

A è una variabile di tipo array
 exp è un'espressione compatibile col tipo `int`

Selezione di un elemento



- Dato un array, è possibile *accedere* (in lettura e scrittura) ai suoi elementi tramite **accesso diretto**

$A[exp]$

A è una variabile di tipo array
 exp è un'espressione compatibile col tipo `int`

“Il risultato della valutazione di exp viene usato per selezionare un elemento di A ”

Selezione di un elemento

Esempio (lettura)

```
int x[4] = {1, 2, 3, -2}
```

```
cout << x[0];
```

```
cout << x[2];
```

```
int i = 1;
```

```
cout << x[i];
```

```
cout << x[i+1];
```

```
cout << x[x[i]];
```

0	1	2	3
1	2	3	-2

Selezione di un elemento

Esempio (lettura)

```
int x[4] = {1, 2, 3, -2};  
cout << x[0];  
cout << x[2];
```

output: 1

```
int i = 1;  
cout << x[i];  
cout << x[i+1];  
cout << x[x[i]];
```

0	1	2	3
1	2	3	-2

Selezione di un elemento

Esempio (lettura)

```
int x[4] = {1, 2, 3, -2};  
cout << x[0];  
cout << x[2];
```

output: 1
output: 3

```
int i = 1;  
cout << x[i];  
cout << x[i+1];  
cout << x[x[i]];
```

0	1	2	3
1	2	3	-2

Selezione di un elemento

Esempio (lettura)

```
int x[4] = {1, 2, 3, -2};  
cout << x[0];  
cout << x[2];
```

output: 1
output: 3

```
int i = 1;  
cout << x[i];  
cout << x[i+1];  
cout << x[x[i]];
```

output: 2

0	1	2	3
1	2	3	-2

Selezione di un elemento

Esempio (lettura)

```
int x[4] = {1, 2, 3, -2};  
cout << x[0];  
cout << x[2];
```

output: 1
output: 3

```
int i = 1;  
cout << x[i];  
cout << x[i+1];  
cout << x[x[i]];
```

output: 2
output: 3

0	1	2	3
1	2	3	-2

Selezione di un elemento

Esempio (lettura)

```
int x[4] = {1, 2, 3, -2}
```

```
cout << x[0];
```

```
cout << x[2];
```

output: 1

output: 3

```
int i = 1;
```

```
cout << x[i];
```

```
cout << x[i+1];
```

```
cout << x[x[i]];
```

output: 2

output: 3

output: 3

0	1	2	3
1	2	3	-2

Selezione di un elemento

Esempio (scrittura)

```
int x[4] = {1, 2, 3, -2}
```

```
x[0] = x[0] + 1;
```

```
x[1] = x[2] + x[0];
```

```
int i = 1;
```

```
x[i+1] = 1;
```

```
x[x[i+1]] = 7;
```

0	1	2	3
1	2	3	-2

Selezione di un elemento

Esempio (scrittura)

0	1	2	3
1	2	3	-2

```
int x[4] = {1, 2, 3, -2}
```

```
x[0] = x[0] + 1;
```

```
x[1] = x[2] + x[0];
```

```
int i = 1;
```

```
x[i+1] = 1;
```

```
x[x[i+1]] = 7;
```

Selezione di un elemento

Esempio (scrittura)

```
int x[4] = {1, 2, 3, -2}
```

```
x[0] = x[0] + 1;
```

```
x[1] = x[2] + x[0];
```

```
int i = 1;
```

```
x[i+1] = 1;
```

```
x[x[i+1]] = 7;
```

0	1	2	3
1	2	3	-2

Dopo l'assegnamento

0	1	2	3
2	2	3	-2

Selezione di un elemento

Esempio (scrittura)

0	1	2	3
2	2	3	-2

```
int x[4] = {1, 2, 3, -2}  
x[0] = x[0] + 1;  
x[1] = x[2] + x[0];
```

```
int i = 1;  
x[i+1] = 1;  
x[x[i+1]] = 7;
```

Selezione di un elemento

Esempio (scrittura)

```
int x[4] = {1, 2, 3, -2}  
x[0] = x[0] + 1;  
x[1] = x[2] + x[0];
```

```
int i = 1;  
x[i+1] = 1;  
x[x[i+1]] = 7;
```

0	1	2	3
2	2	3	-2

Dopo l'assegnamento

0	1	2	3
2	5	3	-2

Selezione di un elemento

Esempio (scrittura)

0	1	2	3
2	5	3	-2

```
int x[4] = {1, 2, 3, -2}
```

```
x[0] = x[0] + 1;
```

```
x[1] = x[2] + x[0];
```

```
int i = 1;
```

```
x[i+1] = 1;
```

```
x[x[i+1]] = 7;
```


Selezione di un elemento

Esempio (scrittura)

```
int x[4] = {1, 2, 3, -2}
```

```
x[0] = x[0] + 1;
```

```
x[1] = x[2] + x[0];
```

```
int i = 1;
```

```
x[i+1] = 1;
```

```
x[x[i+1]] = 7;
```

0	1	2	3
2	5	3	-2

Dopo l'assegnamento

0	1	2	3
2	5	1	-2

Selezione di un elemento

Esempio (scrittura)

0	1	2	3
2	5	1	-2

```
int x[4] = {1, 2, 3, -2}  
x[0] = x[0] + 1;  
x[1] = x[2] + x[0];
```

```
int i = 1;  
x[i+1] = 1;  
x[x[i+1]] = 7;
```

Selezione di un elemento

Esempio (scrittura)

```
int x[4] = {1, 2, 3, -2}
```

```
x[0] = x[0] + 1;
```

```
x[1] = x[2] + x[0];
```

```
int i = 1;
```

```
x[i+1] = 1;
```

```
x[x[i+1]] = 7;
```

0	1	2	3
2	5	1	-2

Dopo l'assegnamento

0	1	2	3
2	7	1	-2

Array

Dimensione degli array

- **Problema:** leggere da tastiera i voti di uno studente e calcolare la base del voto di laurea (supponendo che tutti i voti abbiano lo stesso peso)

$$\frac{media_voti \times 110}{30}$$

Array

Dimensione degli array

- **Problema:** leggere da tastiera i voti di uno studente e calcolare la base del voto di laurea (supponendo che tutti i voti abbiano lo stesso peso)

$$\frac{media_voti \times 110}{30}$$

- **NB:** non conosco a priori quanti voti devono essere inseriti da tastiera!

Array

Dimensione degli array

```
int a[5];
```

- La dimensione di un array deve essere specificata al momento della sua dichiarazione

Problema: la dimensione dell'array potrebbe non essere nota a priori!

Array

Dimensione degli array

```
int a[5];
```

- La dimensione di un array deve essere specificata al momento della sua dichiarazione

Problema: la dimensione dell'array potrebbe non essere nota a priori!

Soluzione: stabilisco una dimensione **massima** dell'array sufficiente a contenere la sequenza di elementi attesa dal programma

Array

Dimensione degli array

- **Problema:** leggere da tastiera i voti di uno studente e calcolare la base del voto di laurea (supponendo che tutti i voti abbiano lo stesso peso)

$$\frac{media_voti \times 110}{30}$$

- **NB:** non conosco a priori quanti voti devono essere inseriti da tastiera!
- Dichiaro un array di dimensione MAX_VOTI = 50 (stima in eccesso dei voti che il programma si aspetta)

Allocazione di un array

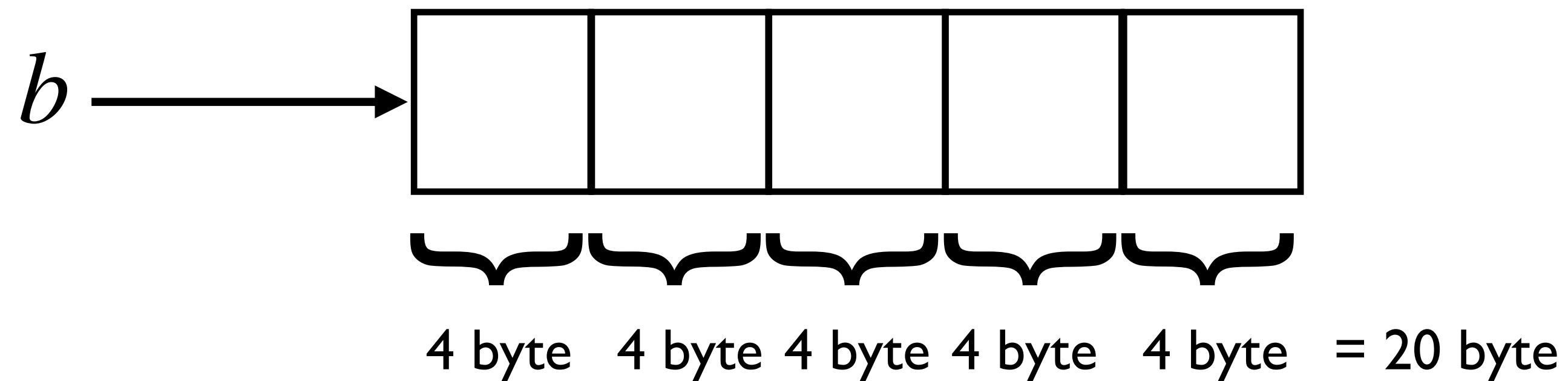
```
int x[5];
```

- Creazione di un array di 5 elementi di tipo `int`
 - Allocazione di 5 celle di memoria contigue di tipo `int` (`int` occupa 4 byte!)
 - Indirizzo di base b dell'area di memoria dell'array

Allocazione di un array

```
int x[5];
```

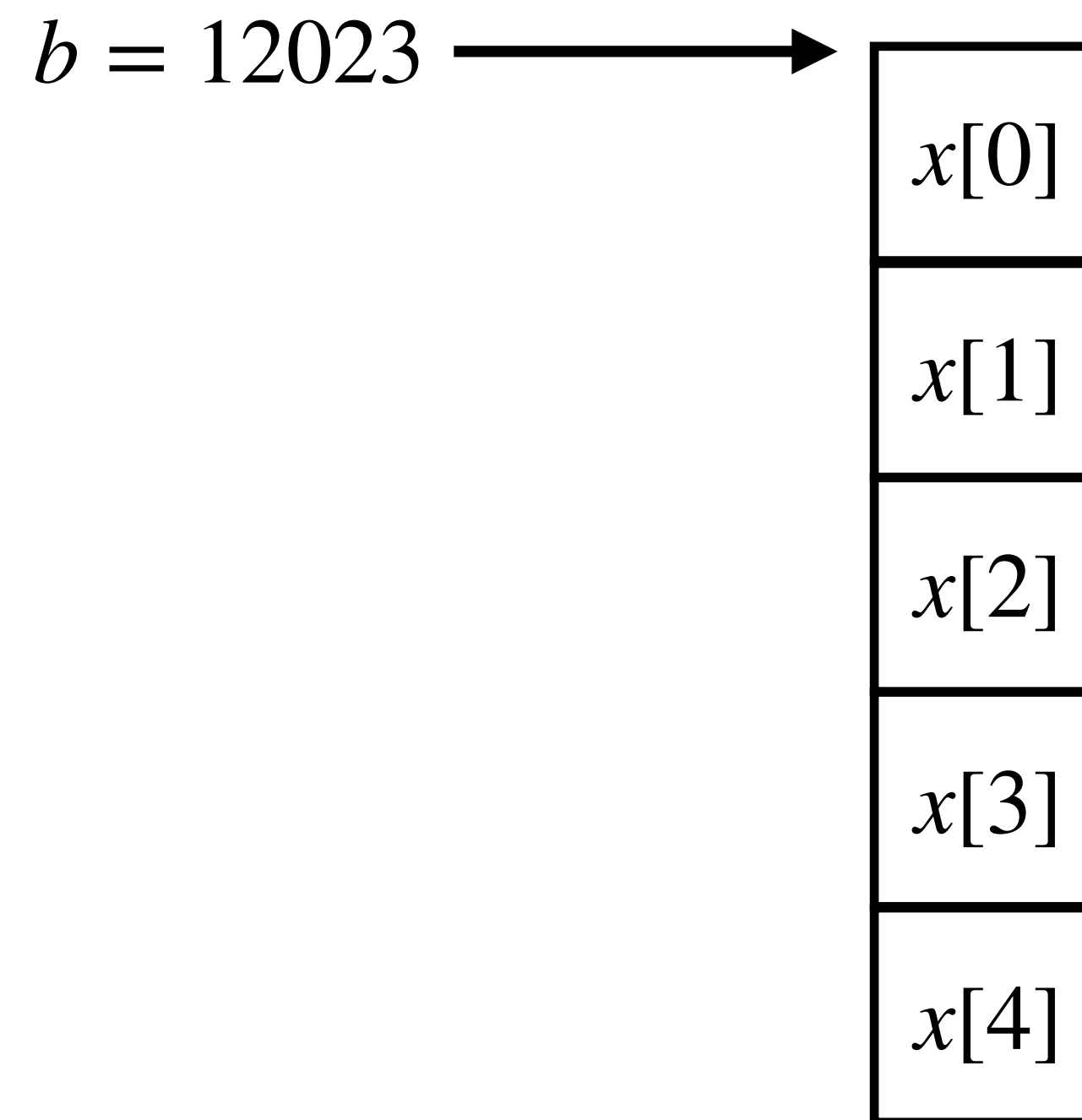
- Creazione di un array di 5 elementi di tipo `int`
 - Allocazione di 5 celle di memoria contigue di tipo `int` (`int` occupa 4 byte!)
 - Indirizzo di base b dell'area di memoria dell'array



Allocazione di un array

Esempio

```
int x[5];
```

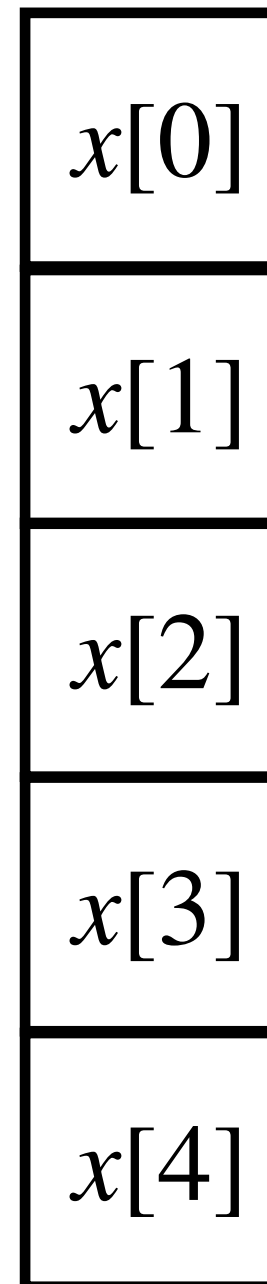


Allocazione di un array

Esempio

```
int x[5];
```

$b = 12023$ →



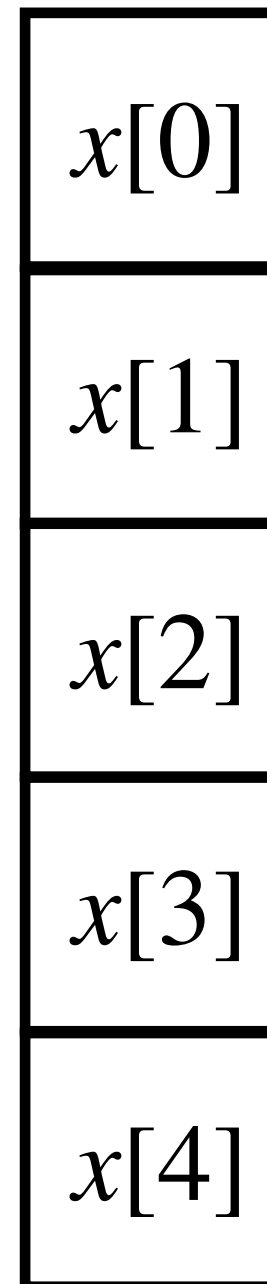
$12023 + 4 \text{ byte} = 12027$

Allocazione di un array

Esempio

```
int x[5];
```

$b = 12023$ →



$12023 + 4 \text{ byte} = 12027$

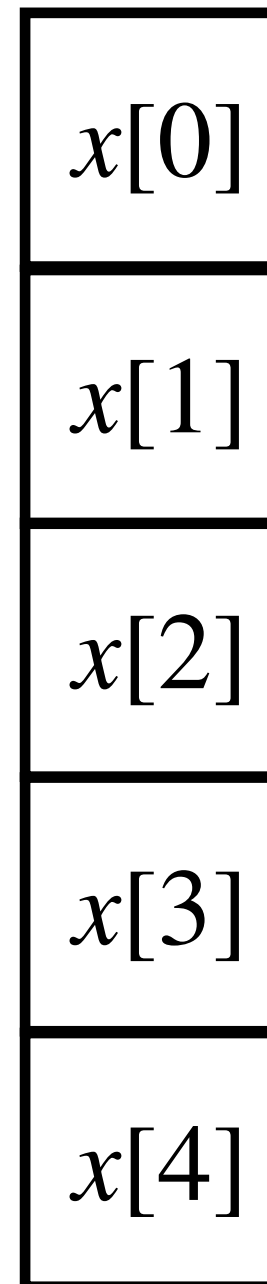
$12023 + 8 \text{ byte} = 12031$

Allocazione di un array

Esempio

```
int x[5];
```

$b = 12023$ →



$$12023 + 4 \text{ byte} = 12027$$

$$12023 + 8 \text{ byte} = 12031$$

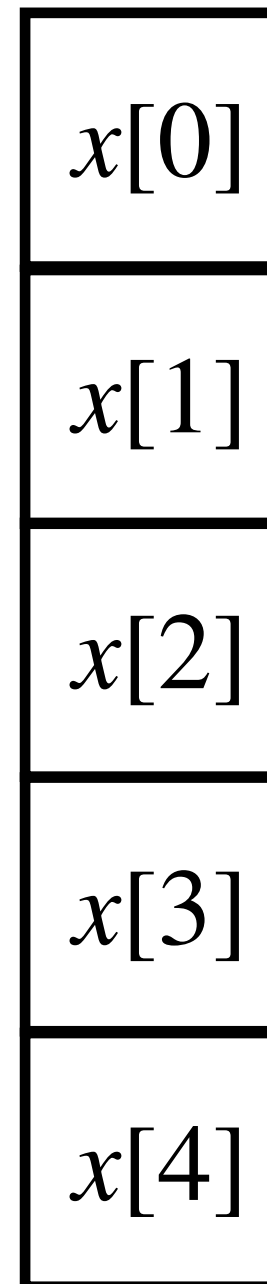
$$12023 + 12 \text{ byte} = 12035$$

Allocazione di un array

Esempio

```
int x[5];
```

$b = 12023$ →



$$12023 + 4 \text{ byte} = 12027$$

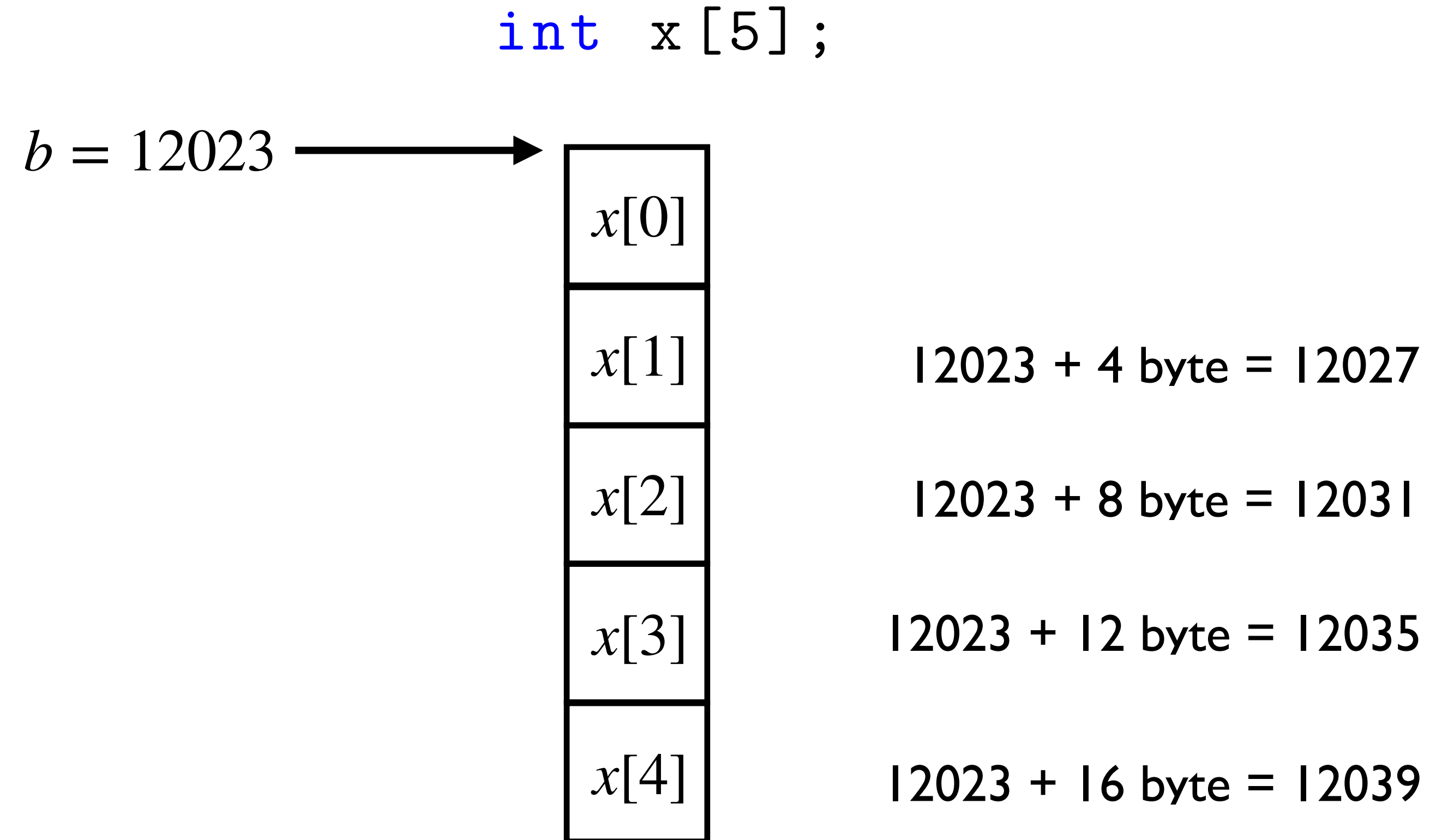
$$12023 + 8 \text{ byte} = 12031$$

$$12023 + 12 \text{ byte} = 12035$$

$$12023 + 16 \text{ byte} = 12039$$

Allocazione di un array

Esempio



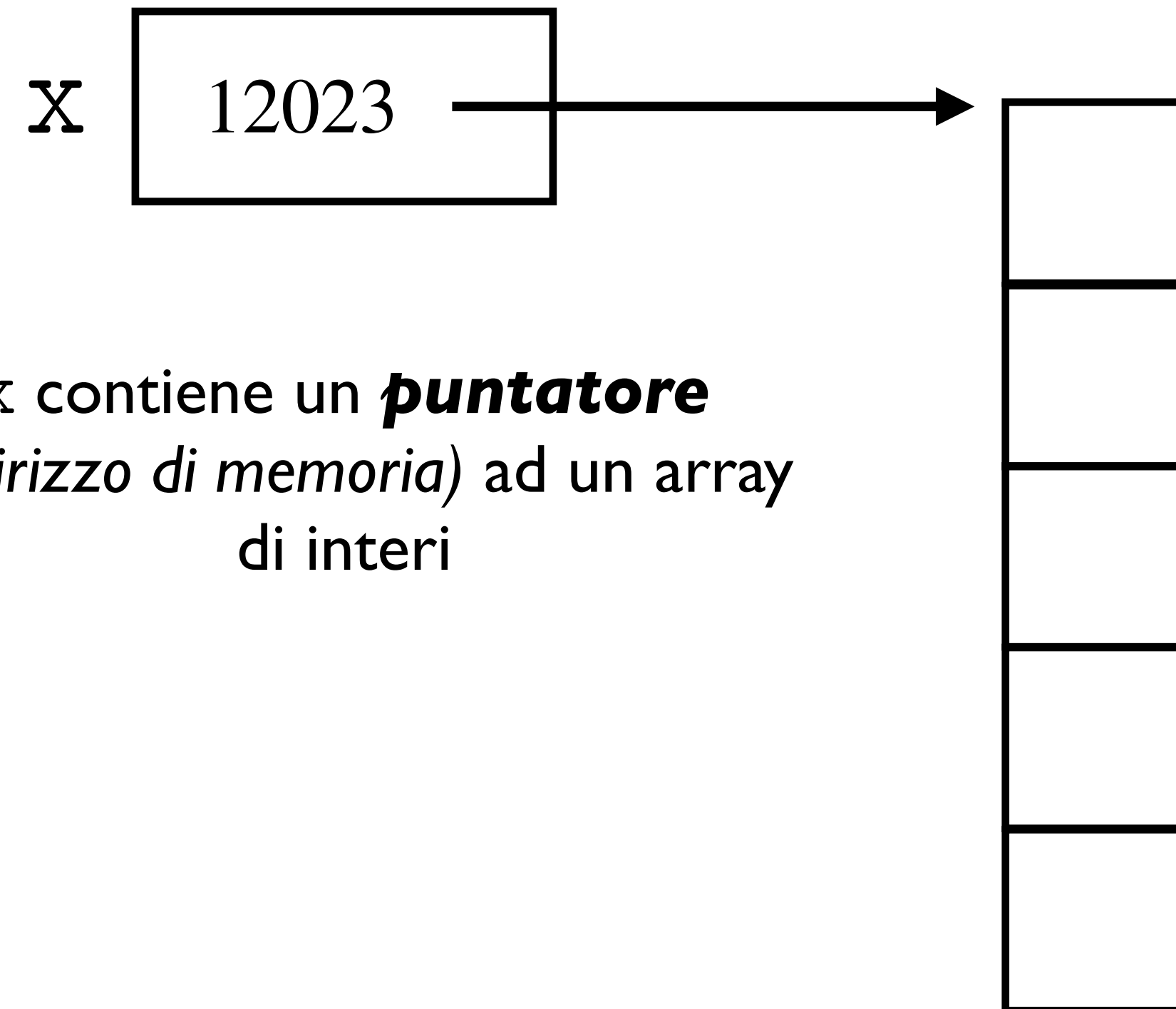
- Accesso diretto all'elemento i di un array: $b + i * d$

d = dimensione di una cella dell'array (e.g., per `int` sono 4 byte)

Allocazione di un array

Rappresentazione grafica in memoria

```
int x[5];
```



`x` contiene un **puntatore**
(indirizzo di memoria) ad un array
di interi

Array “semi-dinamici”

- Finora abbiamo lavorato con array *statici*
 - ◆ La dimensione dell’array è costante e nota a *compile-time*

```
int a[5];
```

```
int voti[MAX_VOTI];
```

Array “semi-dinamici”

- Finora abbiamo lavorato con array *statici*
 - ◆ La dimensione dell’array è costante e nota a *compile-time*

```
int a[5];
```

```
int voti[MAX_VOTI];
```

- Array semi-dinamici
 - ◆ La dimensione dell’array è un’espressione intera e decisa a *run-time*

Array “semi-dinamici”

- Finora abbiamo lavorato con array *statici*
 - ◆ La dimensione dell’array è costante e nota a *compile-time*

```
int a[5];           int voti[MAX_VOTI];
```

- Array semi-dinamici
 - ◆ La dimensione dell’array è un’espressione intera e decisa a *run-time*

```
int length = 0;  
cin >> length;  
int A[length];
```

Array

Statici vs semi-dinamici

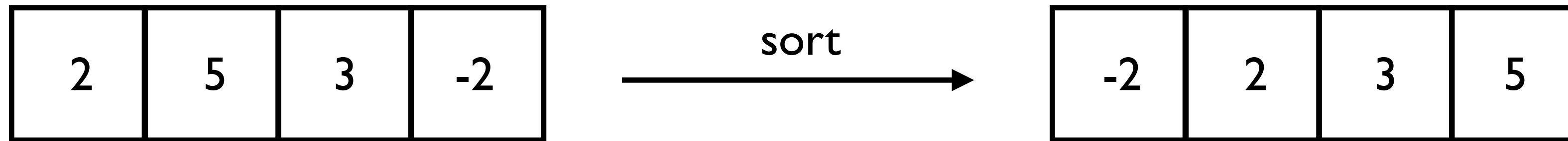
- L'uso degli array semi-dinamici è del tutto identico a quello degli array statici (lettura, scrittura,...)
- **Uniche differenze:**
 - Gli array semi-dinamici non possono essere dichiarati tramite dichiarazione globale, quelli statici sì
 - Gli array semi-dinamici non possono essere utilizzati come tipo di un campo di una struct

Esercizi

- Massimo e minimo di un array

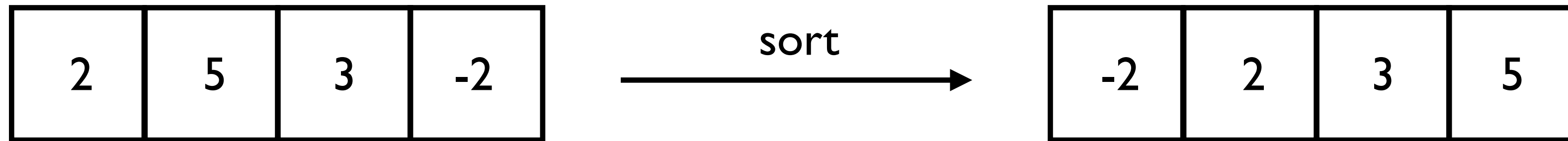
Ordinare un array

Bubble sort



Ordinare un array

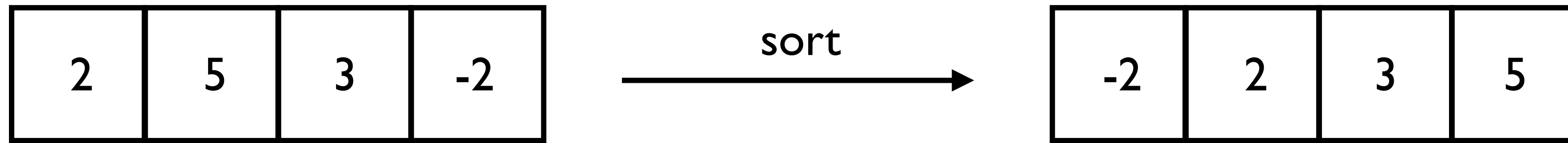
Bubble sort



- Algoritmo di ordinamento di una lista di dati

Ordinare un array

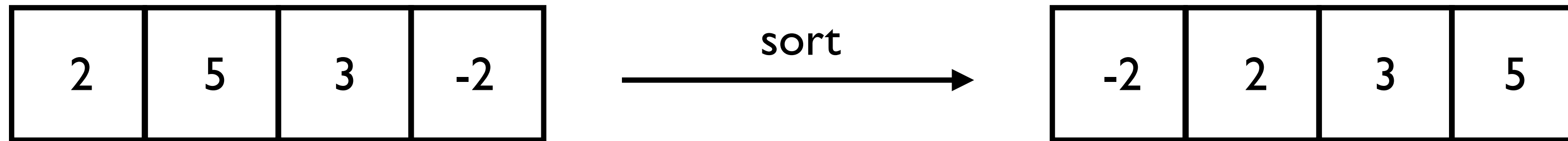
Bubble sort



- Algoritmo di ordinamento di una lista di dati
- Confronto due elementi alla volta, procedendo in un senso stabilito (supponiamo da sinistra verso destra)

Ordinare un array

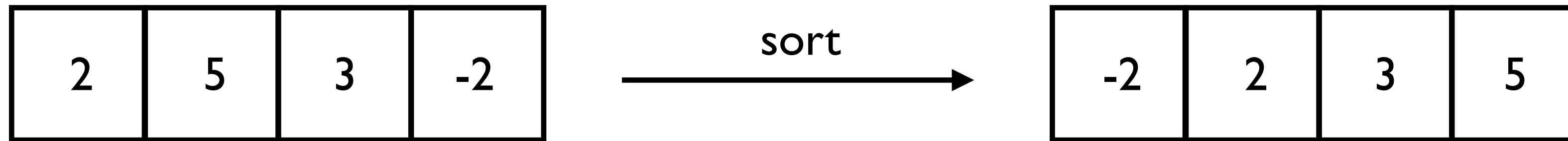
Bubble sort



- Algoritmo di ordinamento di una lista di dati
- Confronto due elementi alla volta, procedendo in un senso stabilito (supponiamo da sinistra verso destra)
- Il primo col secondo, il secondo col terzo, il terzo col quarto... i con $i + 1$

Ordinare un array

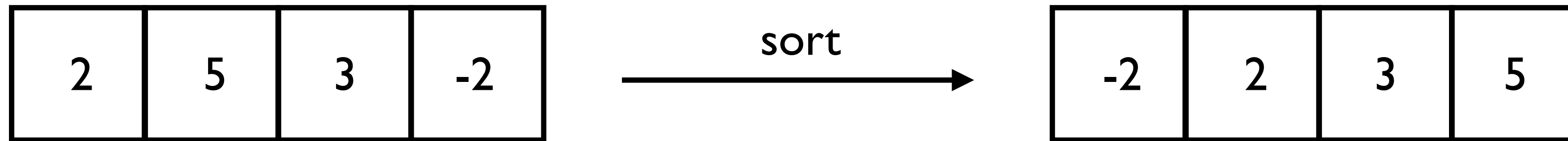
Bubble sort



- Algoritmo di ordinamento di una lista di dati
- Confronto due elementi alla volta, procedendo in un senso stabilito (supponiamo da sinistra verso destra)
- Il primo col secondo, il secondo col terzo, il terzo col quarto... i con $i + 1$
 - se $a[i] > a[i + 1]$ li scambio

Ordinare un array

Bubble sort



- Algoritmo di ordinamento di una lista di dati
- Confronto due elementi alla volta, procedendo in un senso stabilito (supponiamo da sinistra verso destra)
- Il primo col secondo, il secondo col terzo, il terzo col quarto... i con $i + 1$
 - se $a[i] > a[i + 1]$ li scambio
 - altrimenti passo a confrontare $a[i + 1]$ con $a[i + 2]$ e ripeto

Ordinare un array

Bubble sort

6 5 3 1 8 7 2 4

Ordinare un array

Bubble sort

6 5 3 1 8 7 2 4