

Fondamenti di Programmazione (A)

I4 - Puntatori

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

Puntate precedenti

- Array mono-dimensionali
 - Array statici
 - Array semi-dinamici
- Array bi-dimensionali
 - Matrici
- Stringhe
- Struct

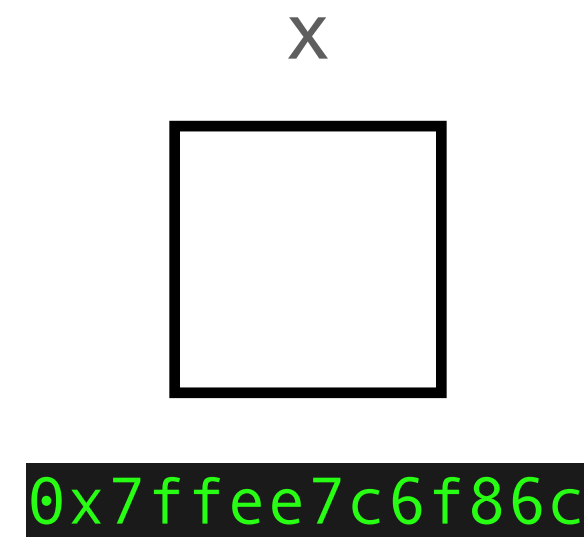
Variabili

Variabili

- *Astrazione* di una cella di memoria

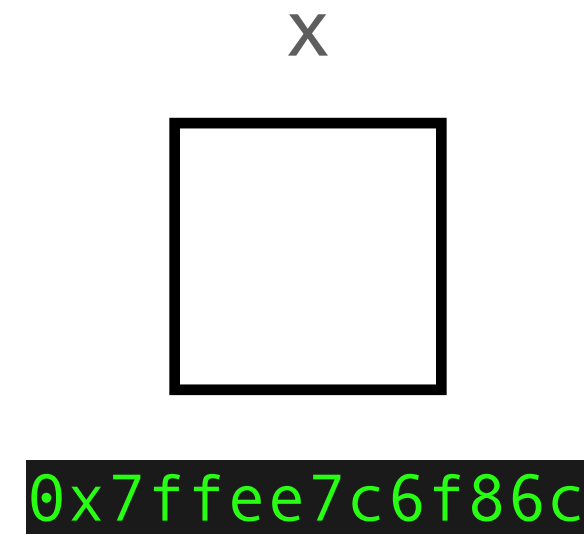
Variabili

- *Astrazione* di una cella di memoria



Variabili

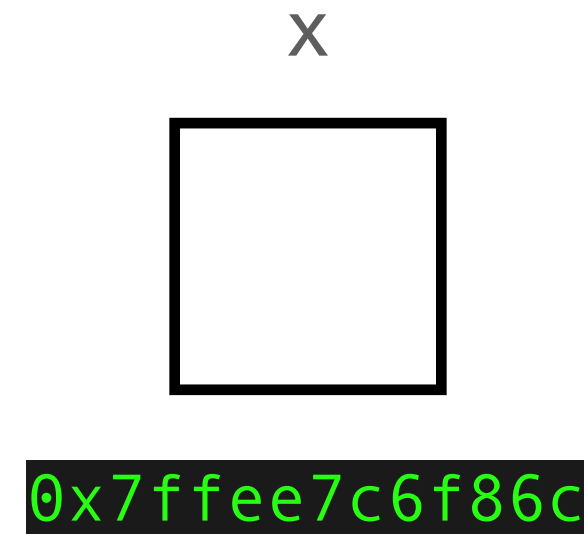
- *Astrazione* di una cella di memoria



- **Tipo**: quali dati posso essere memorizzati all'interno della variabile

Variabili

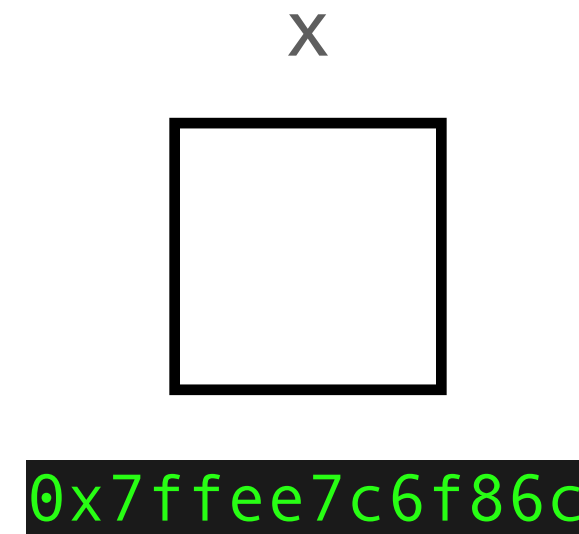
- *Astrazione* di una cella di memoria



- **Tipo**: quali dati posso essere memorizzati all'interno della variabile
- **Valore**: il valore contenuto dalla variabile in un certo momento

Variabili

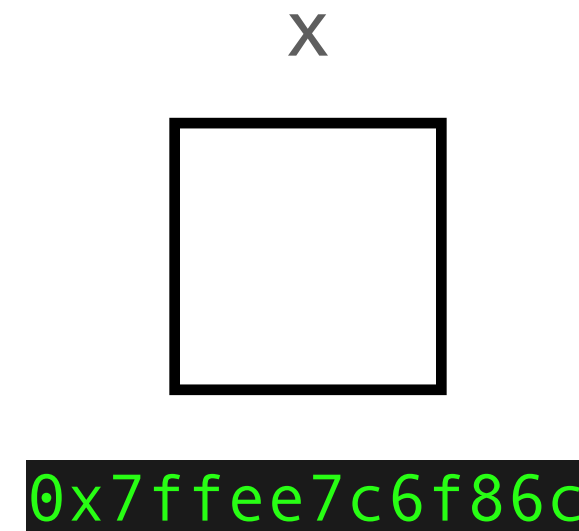
- *Astrazione* di una cella di memoria



- **Tipo**: quali dati posso essere memorizzati all'interno della variabile
- **Valore**: il valore contenuto dalla variabile in un certo momento
- **Indirizzo di memoria/della cella**: a quale indirizzo di memoria si trova la variabile

Variabili

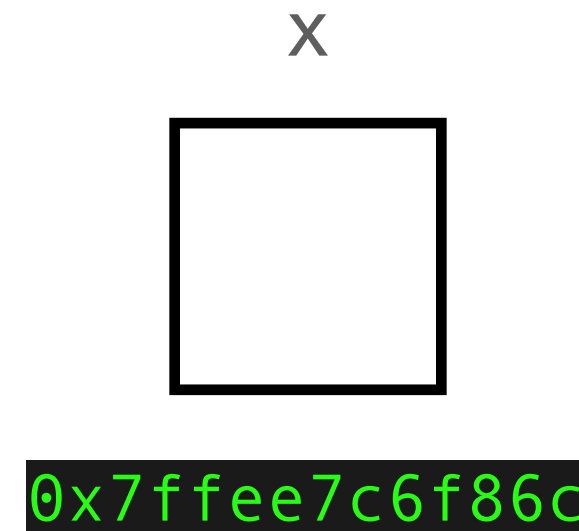
- *Astrazione* di una cella di memoria



- **Tipo**: quali dati posso essere memorizzati all'interno della variabile
- **Valore**: il valore contenuto dalla variabile in un certo momento
- **Indirizzo di memoria/della cella**: a quale indirizzo di memoria si trova la variabile
- Almeno un'operazione di lettura e una di scrittura del valore della variabile

Variabili

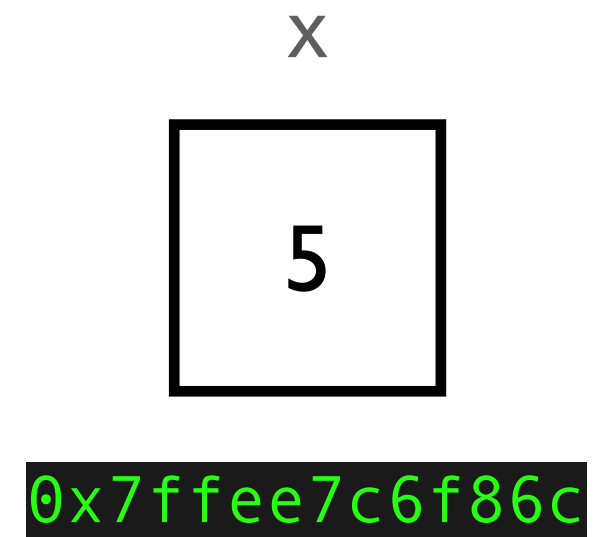
- *Astrazione* di una cella di memoria



- **Tipo**: quali dati posso essere memorizzati all'interno della variabile
- **Valore**: il valore contenuto dalla variabile in un certo momento
- **Indirizzo di memoria/della cella**: a quale indirizzo di memoria si trova la variabile
- Almeno un'operazione di lettura e una di scrittura del valore della variabile

Ad una variabile è spesso associato un **nome**, un identificatore utilizzato per riferirsi in maniera simbolica alla variabile

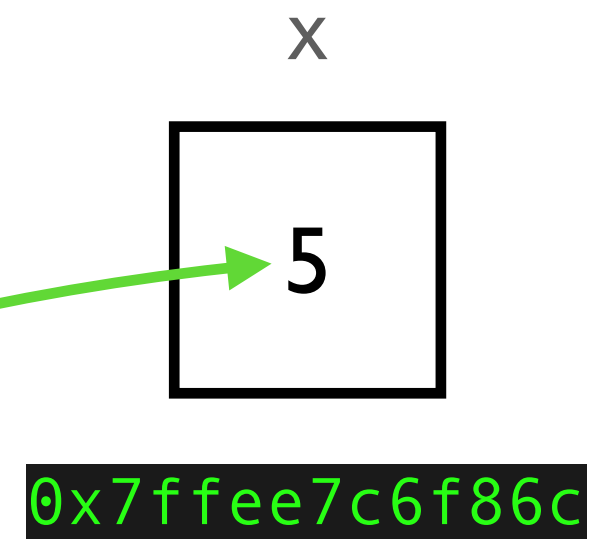
Operatore di indirizzo (*referencing*)



`X` accesso al valore della variabile tramite il suo nome

`&X` accesso all'indirizzo della variabile

Operatore di indirizzo (*referencing*)



`X` accesso al valore della variabile tramite il suo nome

`&X` accesso all'indirizzo della variabile

Operatore di indirizzo (*referencing*)



Operatore di indirizzo (*referencing*)



- Accesso al valore di una variabile: accede all'**indirizzo di memoria** tramite il suo **nome**

Operatore di indirizzo (*referencing*)



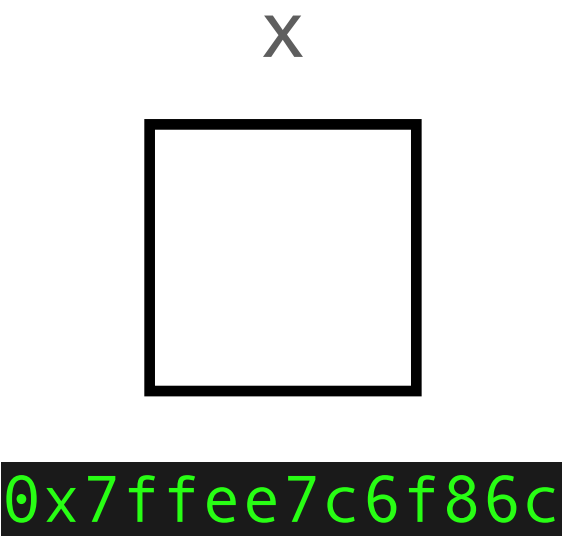
- Accesso al valore di una variabile: accede all'**indirizzo di memoria** tramite il suo **nome**
- C++ permette di manipolare e utilizzare indirizzi di memoria come qualsiasi altro tipo di dato

Esercizio

- **Problema:** ottenere il valore e l'indirizzo di una variabile

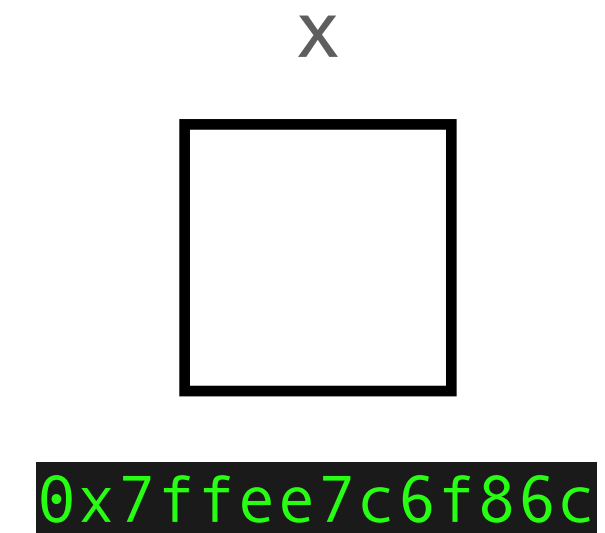
Puntatori

$\&x$



Puntatori

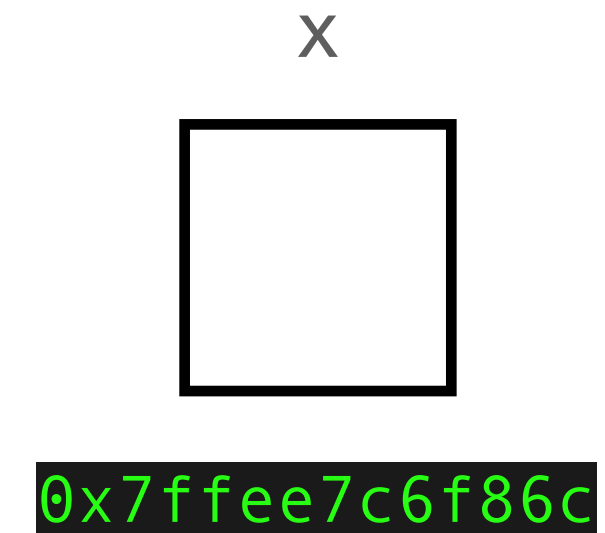
$\&x$



- Se x è di tipo `int`, $\&x$ è di tipo **puntatore a** `int`
- Puntatore a tipo: si dichiara utilizzando il carattere `*` come suffisso al tipo di dato puntato

Puntatori

`&x`



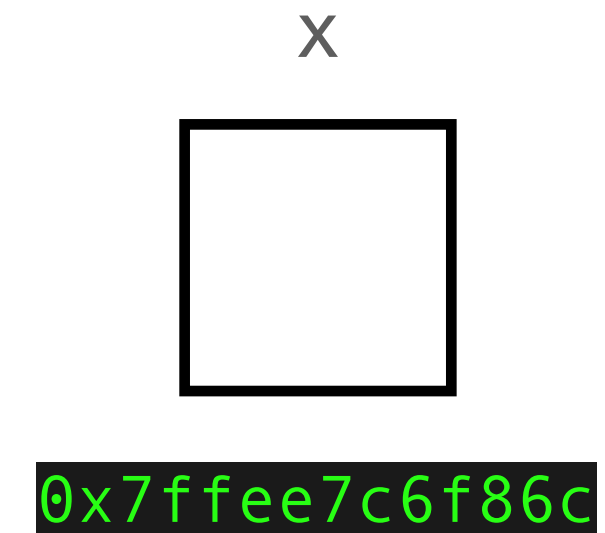
- Se `x` è di tipo `int`, `&x` è di tipo **puntatore a `int`**
- Puntatore a tipo: si dichiara utilizzando il carattere `*` come suffisso al tipo di dato puntato

`int *`

tipo puntatore a `int`

Puntatori

$\&x$



- Se x è di tipo `int`, $\&x$ è di tipo **puntatore a `int`**
- Puntatore a tipo: si dichiara utilizzando il carattere `*` come suffisso al tipo di dato puntato

`int *`

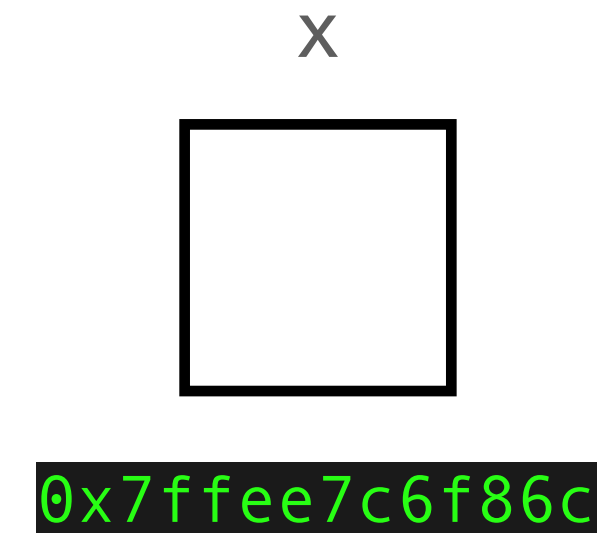
tipo puntatore a `int`

`float *`

tipo puntatore a `float`

Puntatori

$\&x$



- Se x è di tipo `int`, $\&x$ è di tipo **puntatore a** `int`
- Puntatore a tipo: si dichiara utilizzando il carattere `*` come suffisso al tipo di dato puntato

`int *`

tipo puntatore a `int`

`float *`

tipo puntatore a `float`

`char *`

tipo puntatore a `char`

Puntatori

Puntatori

`int`

contiene valori interi

occupa 4 byte

Puntatori

`int`

contiene valori interi

occupa 4 byte

`int *`

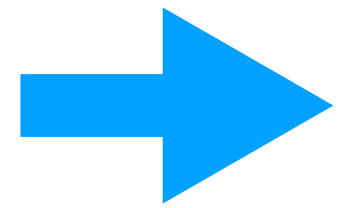
contiene **indirizzi di memoria
i quali contengono interi**

occupa 4 byte (su macchine 32-bit)

occupa 8 byte (su macchine 64-bit)

Puntatori

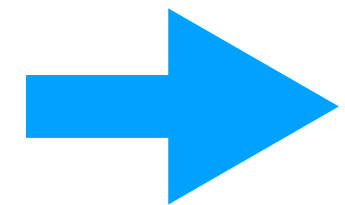
Esempio



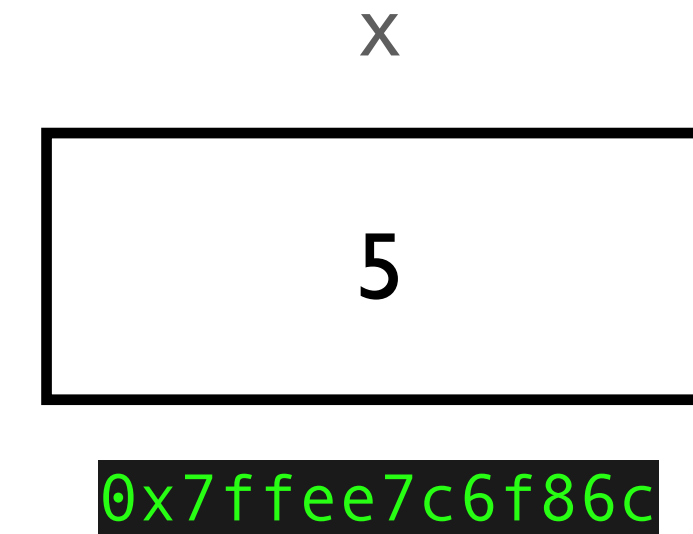
```
int x = 5;
int* p = &x;
cout << "Valore: " << x << endl;
cout << "Indirizzo: " << p << endl;
return 0;
```

Puntatori

Esempio

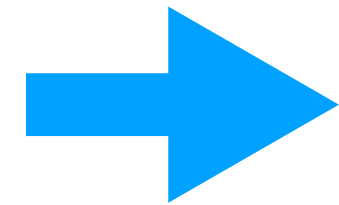


```
int x = 5;  
int* p = &x;  
cout << "Valore: " << x << endl;  
cout << "Indirizzo: " << p << endl;  
return 0;
```

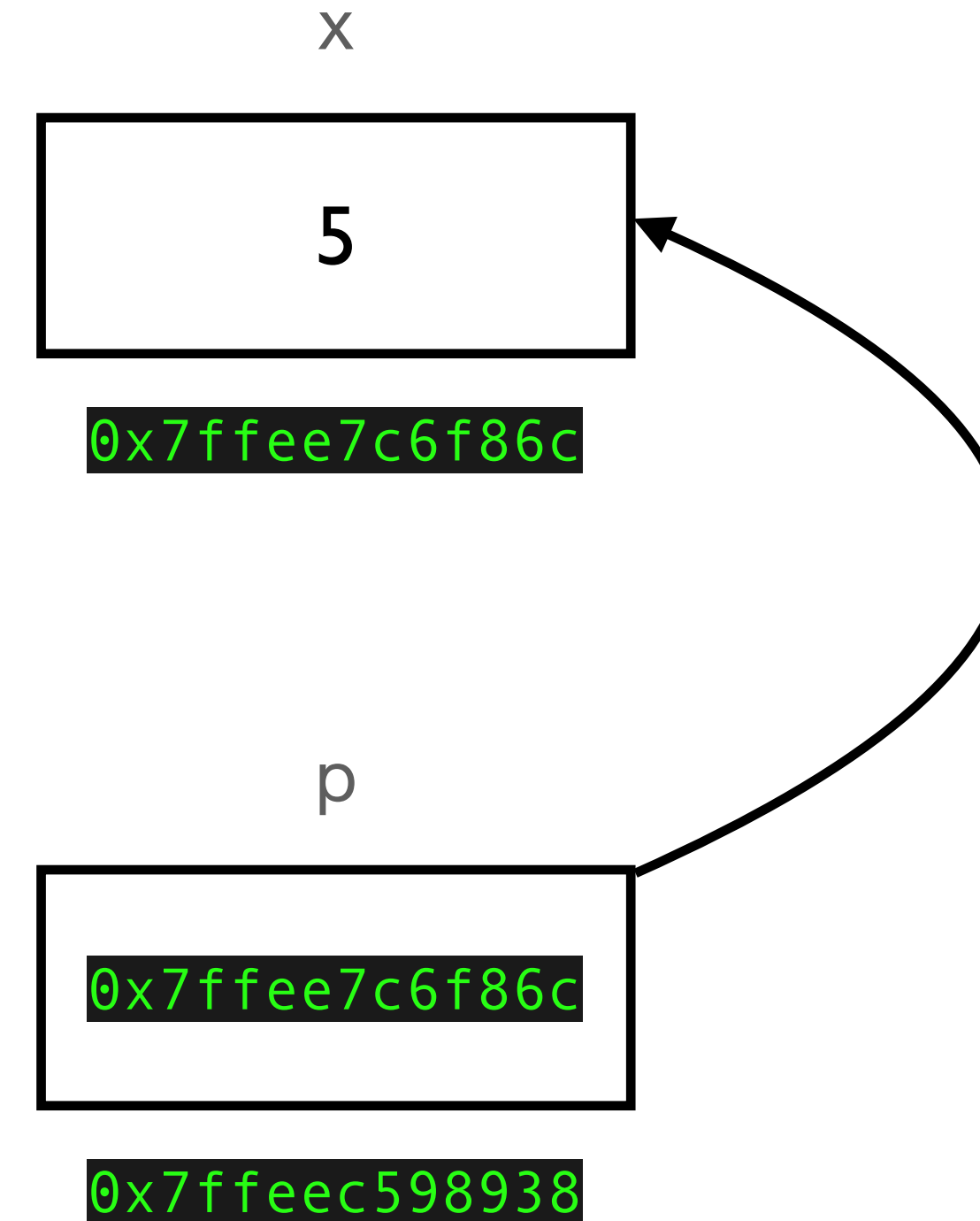


Puntatori

Esempio



```
int x = 5;  
int* p = &x;  
cout << "Valore: " << x << endl;  
cout << "Indirizzo: " << p << endl;  
return 0;
```

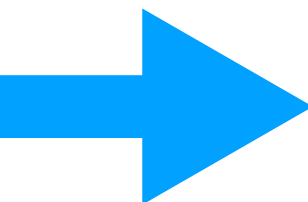


La variabile **p punta a x**: la variabile **p** contiene l'indirizzo di memoria di **x**

La variabile **p** è allocata a sua volta ad un altro indirizzo di memoria

Puntatori

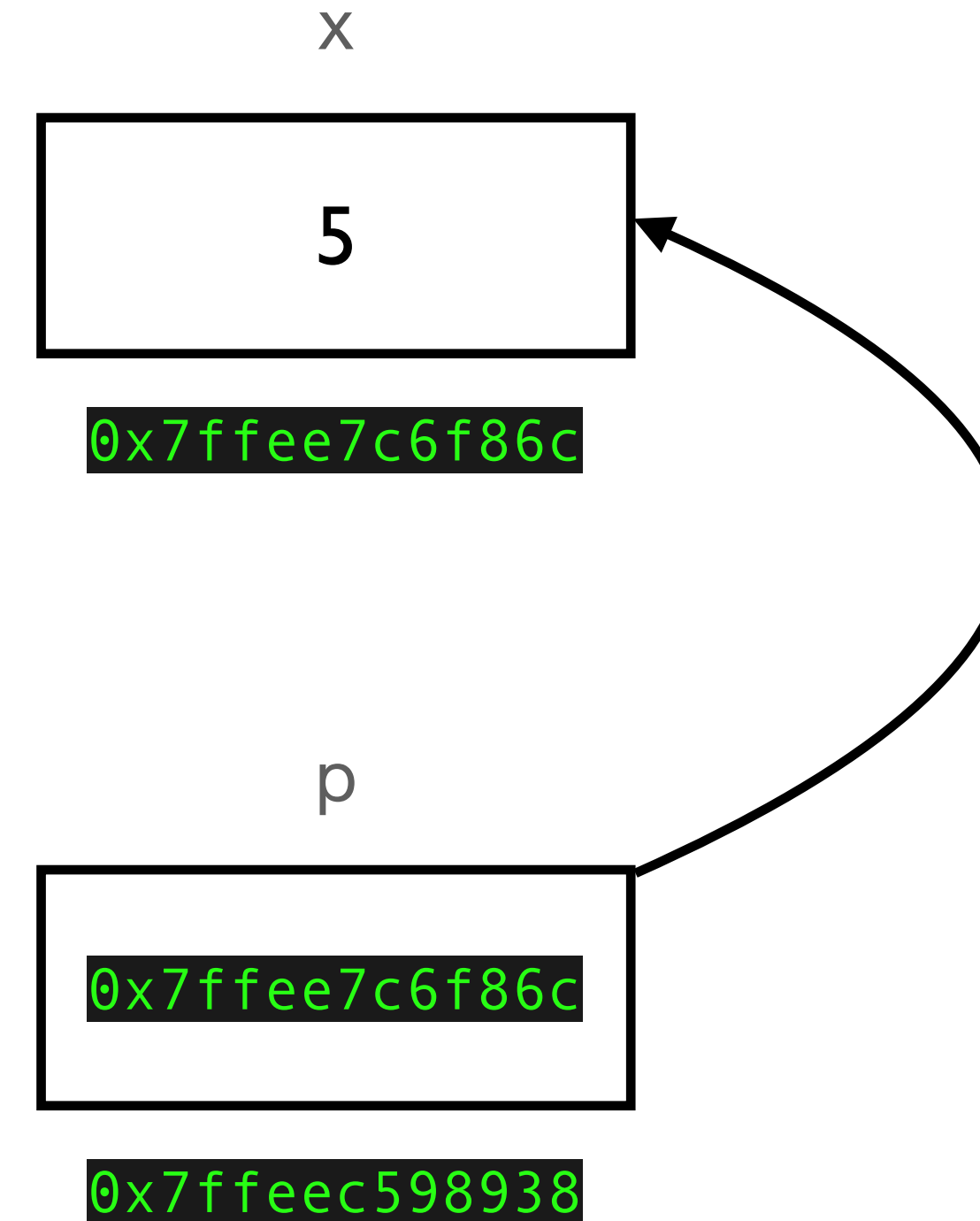
Esempio



```
int x = 5;
int* p = &x;
cout << "Valore: " << x << endl;
cout << "Indirizzo: " << p << endl;
return 0;
```

Output

5

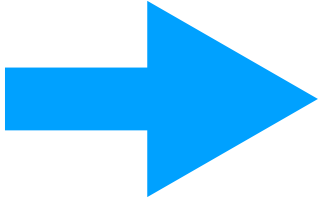


La variabile p **punta a** x: la variabile p contiene l'indirizzo di memoria di x

La variabile p è allocata a sua volta ad un altro indirizzo di memoria

Puntatori

Esempio

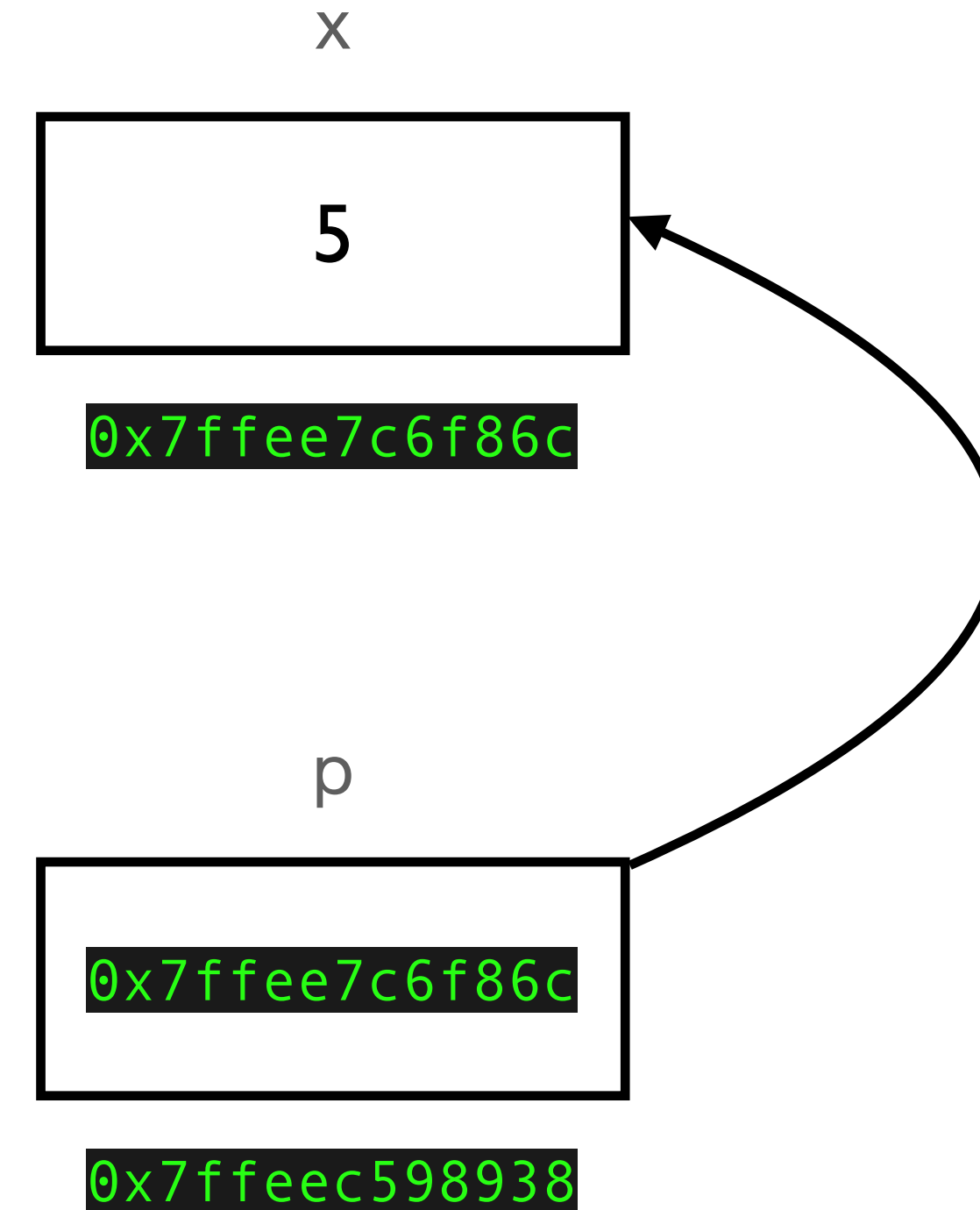


```
int x = 5;  
int* p = &x;  
cout << "Valore: " << x << endl;  
cout << "Indirizzo: " << p << endl;  
return 0;
```

Output

5

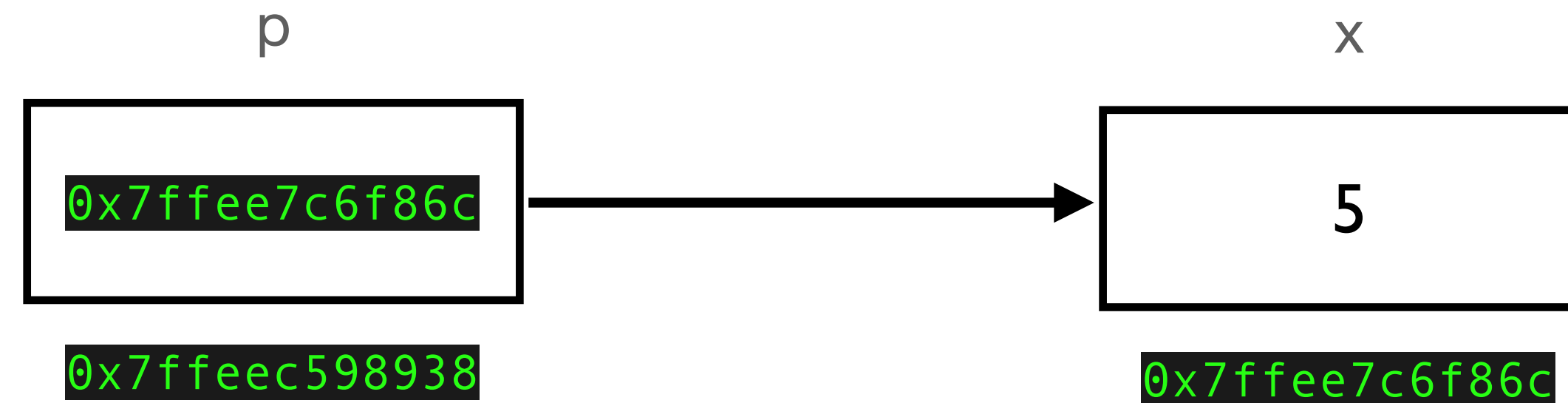
0x7ffee7c6f86c



La variabile **p punta a x**: la variabile **p** contiene l'indirizzo di memoria di **x**

La variabile **p** è allocata a sua volta ad un altro indirizzo di memoria

Puntatori



- Un puntatore contiene un indirizzo di memoria
- Un puntatore può essere contenuto in una variabile
- All'indirizzo di memoria contenuto nel puntatore c'è una variabile

Puntatori



- Un puntatore contiene un indirizzo di memoria
- Un puntatore può essere contenuto in una variabile
- All'indirizzo di memoria contenuto nel puntatore c'è una variabile

Operatore di *dereferencing*



- Come riferirsi a x tramite p?

Operatore di *dereferencing*



- Come riferirsi a x tramite p ?

$*p$

Operatore di *dereferencing*



- Come riferirsi a x tramite p ?

$*p$

- Valuta al nome della variabile puntata da p

Operatore di *dereferencing*



- Come riferirsi a x tramite p ?

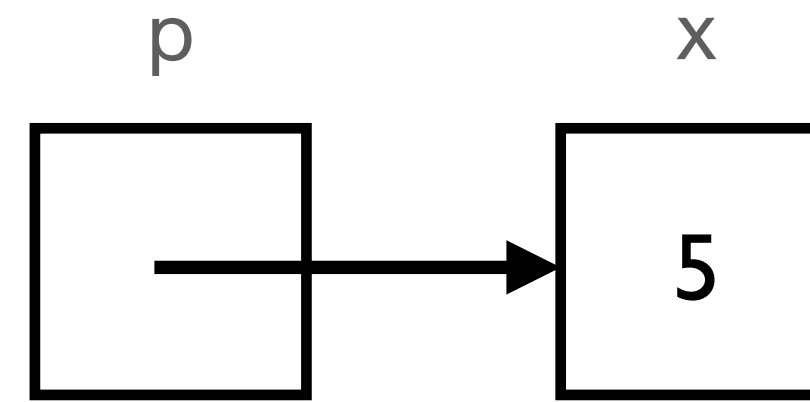
$*p$

- Valuta al nome della variabile puntata da p

“ciò a cui punta la variabile p ”

Operatore di *dereferencing*

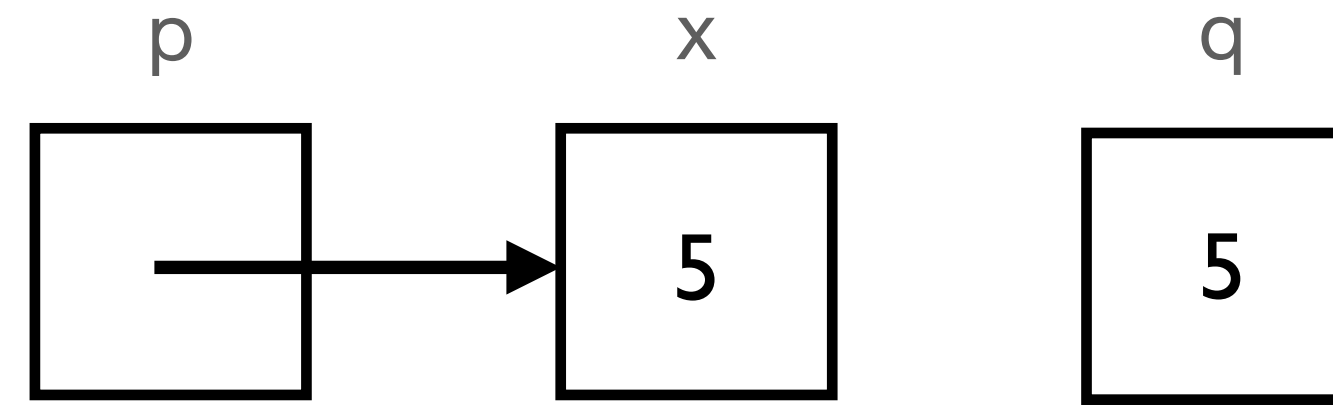
Esempio



```
int x = 5;  
int* p = &x;  
int q = *p;  
*p = 7;
```

Operatore di *dereferencing*

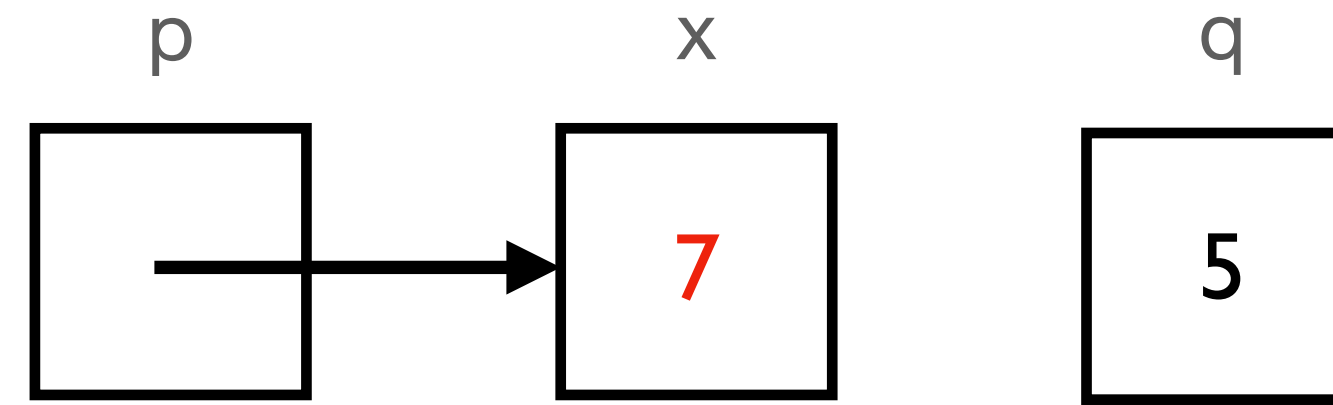
Esempio



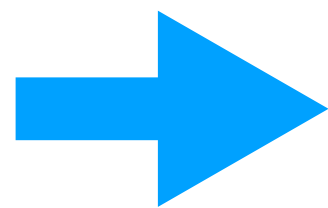
```
int x = 5;  
int* p = &x;  
int q = *p;  
*p = 7;
```

Operatore di *dereferencing*

Esempio

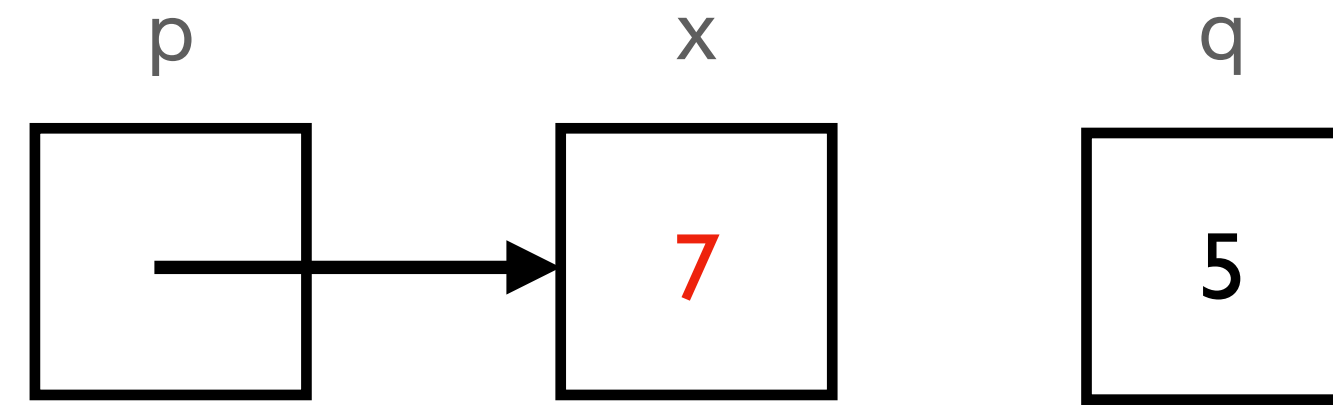


```
int x = 5;  
int* p = &x;  
int q = *p;  
*p = 7;
```



Operatore di *dereferencing*

Esempio



```
int x = 5;  
int* p = &x;  
int q = *p;  
*p = 7;
```

Il valore di q rimane 5

Operatore di *dereferencing*

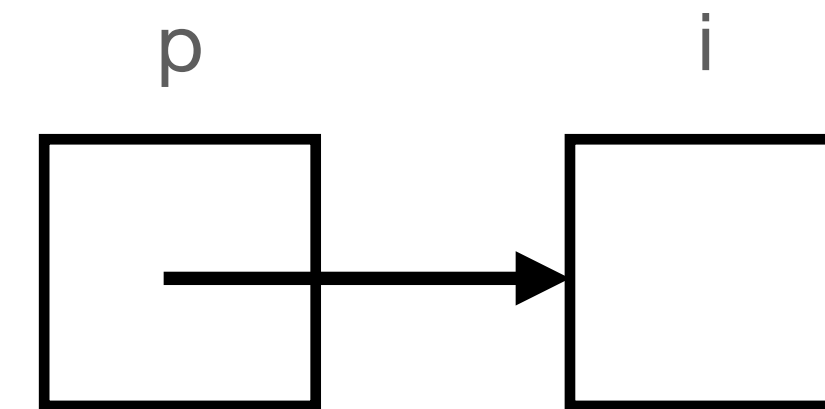
Esempio

```
int i;  
int* p = &i;  
  
for (i = 0; i < 10; i++)  
    cout << *p << endl;  
return 0;
```


Operatore di *dereferencing*

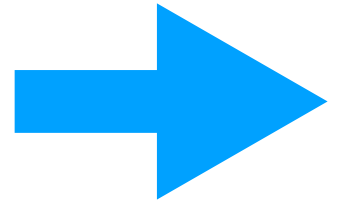
Esempio

```
int i;  
int* p = &i;  
  
for (i = 0; i < 10; i++)  
    cout << *p << endl;  
return 0;
```



Operatore di *dereferencing*

Esempio

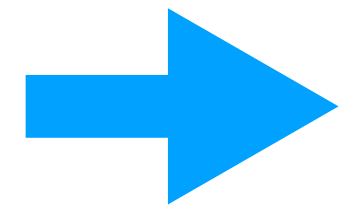


```
int i = 5;
int* p1 = &i;
int* p2 = &i;

cout << *p1 << " " << *p2 << endl;
*p1 = *p1 + 1;
cout << *p1 << " " << *p2 << endl;
return 0;
```

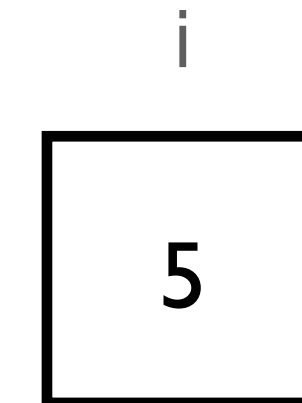
Operatore di *dereferencing*

Esempio



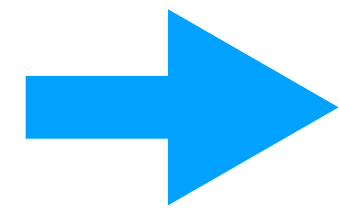
```
int i = 5;  
int* p1 = &i;  
int* p2 = &i;
```

```
cout << *p1 << " " << *p2 << endl;  
*p1 = *p1 + 1;  
cout << *p1 << " " << *p2 << endl;  
return 0;
```



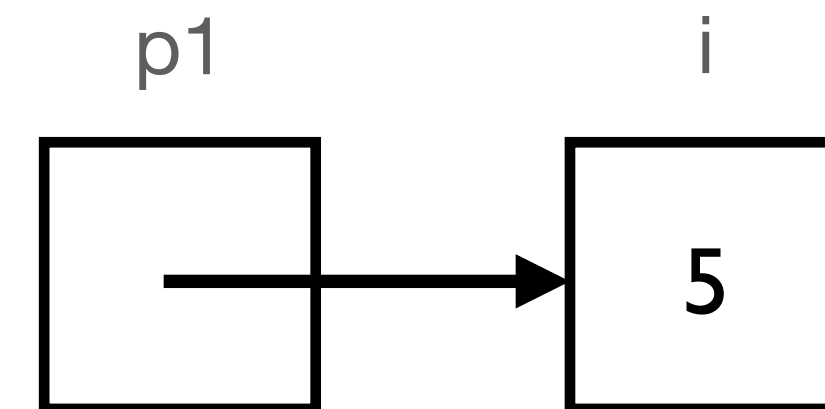
Operatore di *dereferencing*

Esempio



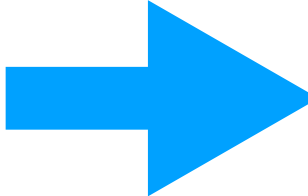
```
int i = 5;
int* p1 = &i;
int* p2 = &i;

cout << *p1 << " " << *p2 << endl;
*p1 = *p1 + 1;
cout << *p1 << " " << *p2 << endl;
return 0;
```



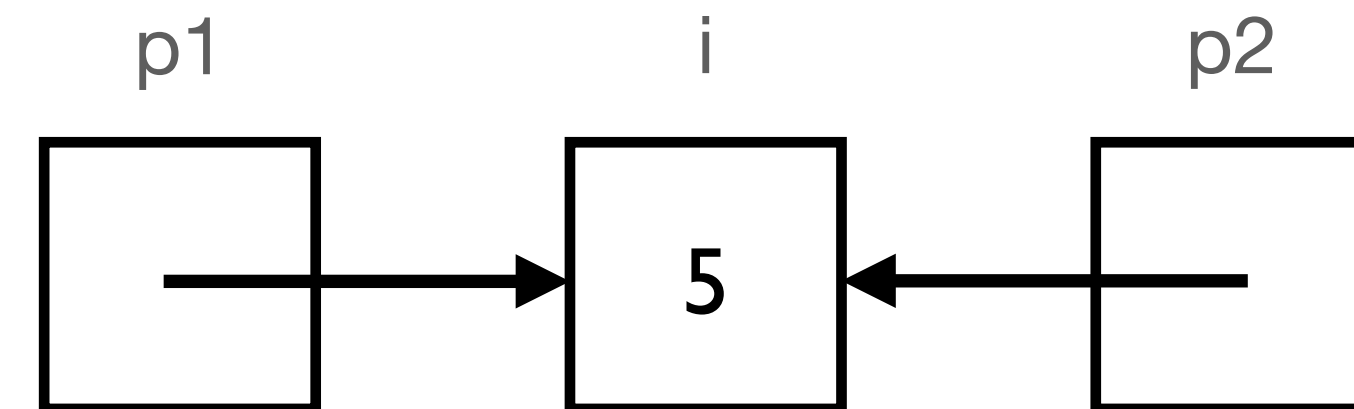
Operatore di *dereferencing*

Esempio



```
int i = 5;
int* p1 = &i;
int* p2 = &i;

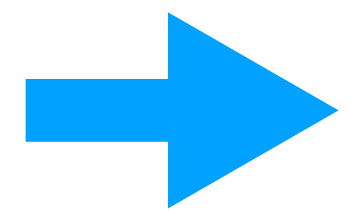
cout << *p1 << " " << *p2 << endl;
*p1 = *p1 + 1;
cout << *p1 << " " << *p2 << endl;
return 0;
```



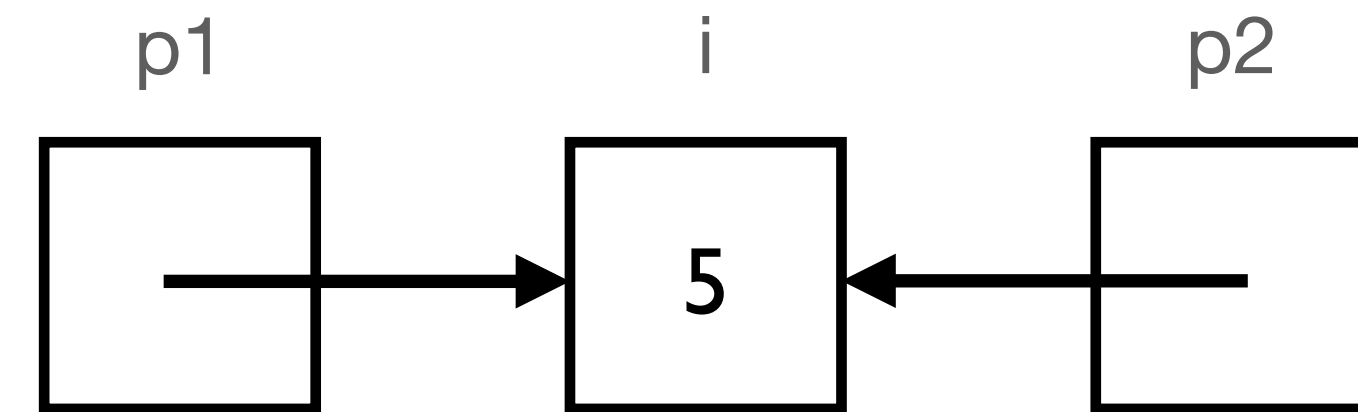
Operatore di *dereferencing*

Esempio

```
int i = 5;  
int* p1 = &i;  
int* p2 = &i;
```



```
cout << *p1 << " " << *p2 << endl;  
*p1 = *p1 + 1;  
cout << *p1 << " " << *p2 << endl;  
return 0;
```



Output

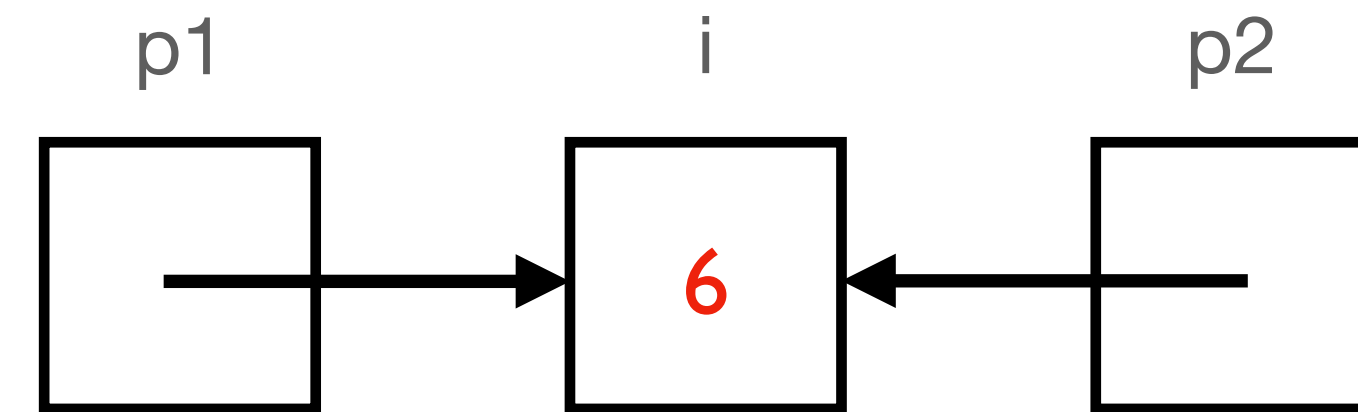
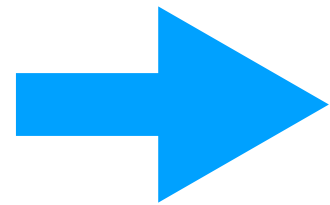
5 5

Operatore di *dereferencing*

Esempio

```
int i = 5;
int* p1 = &i;
int* p2 = &i;

cout << *p1 << " " << *p2 << endl;
*p1 = *p1 + 1;
cout << *p1 << " " << *p2 << endl;
return 0;
```



Output

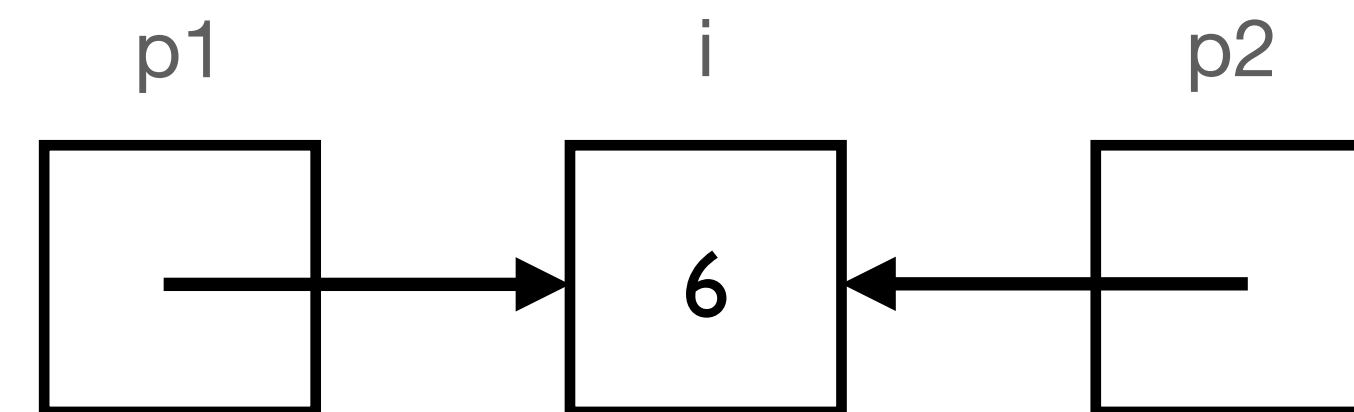
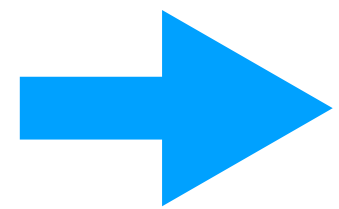
5 5

Operatore di *dereferencing*

Esempio

```
int i = 5;
int* p1 = &i;
int* p2 = &i;

cout << *p1 << " " << *p2 << endl;
*p1 = *p1 + 1;
cout << *p1 << " " << *p2 << endl;
return 0;
```

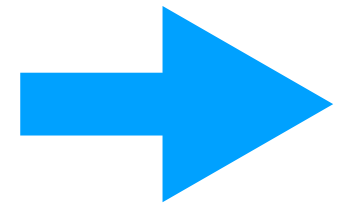


Output

5 5
6 6

Operatore di *dereferencing*

Esempio



```
int i = 5, j = 5;
int* p1 = &i;
int* p2 = &i;

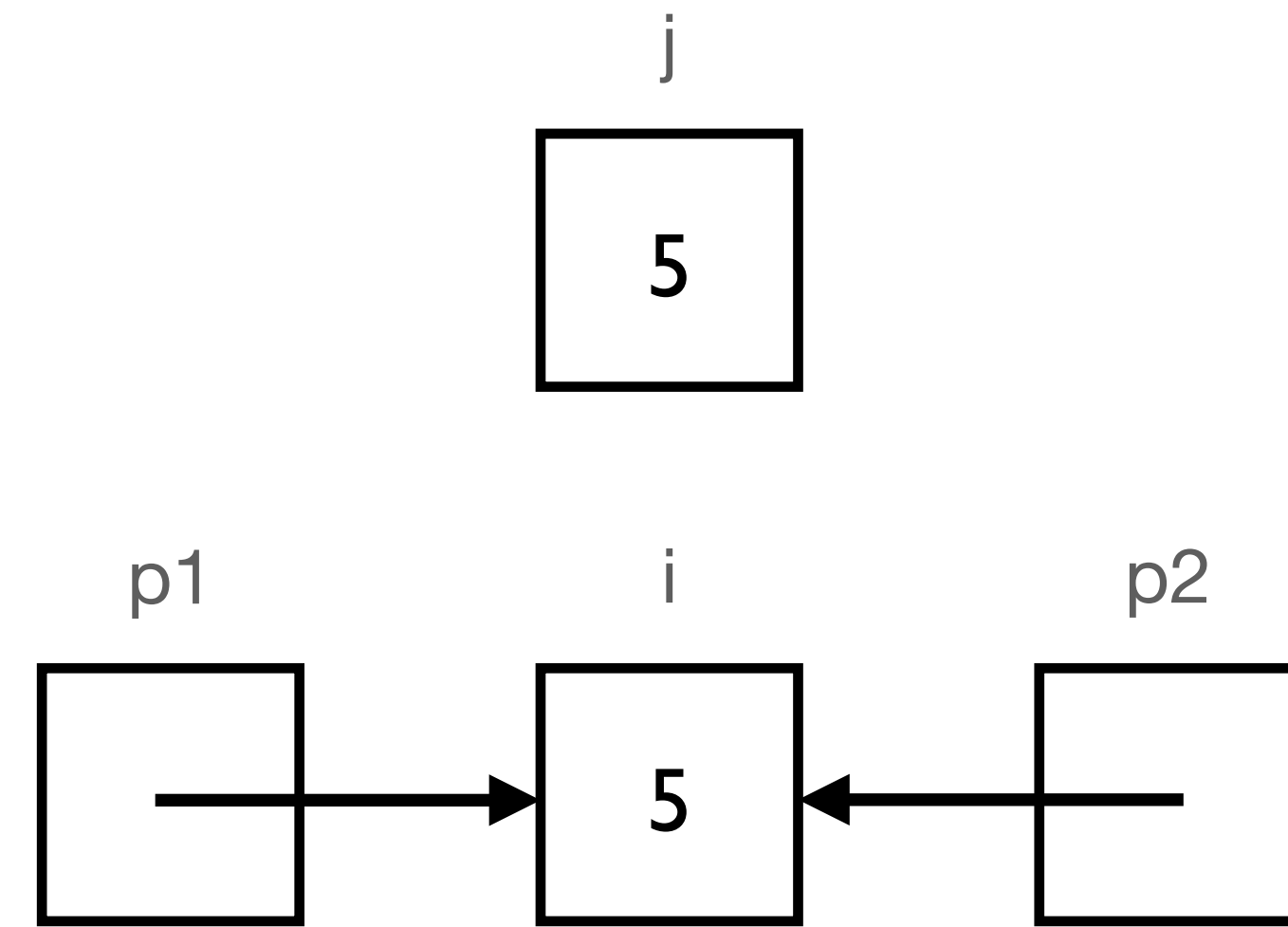
cout << *p1 << " " << *p2 << endl;
p2 = &j;
*p1 = *p1 + 1;
cout << *p1 << " " << *p2 << endl;
return 0;
```

Operatore di *dereferencing*

Esempio

```
int i = 5, j = 5;
int* p1 = &i;
int* p2 = &i;

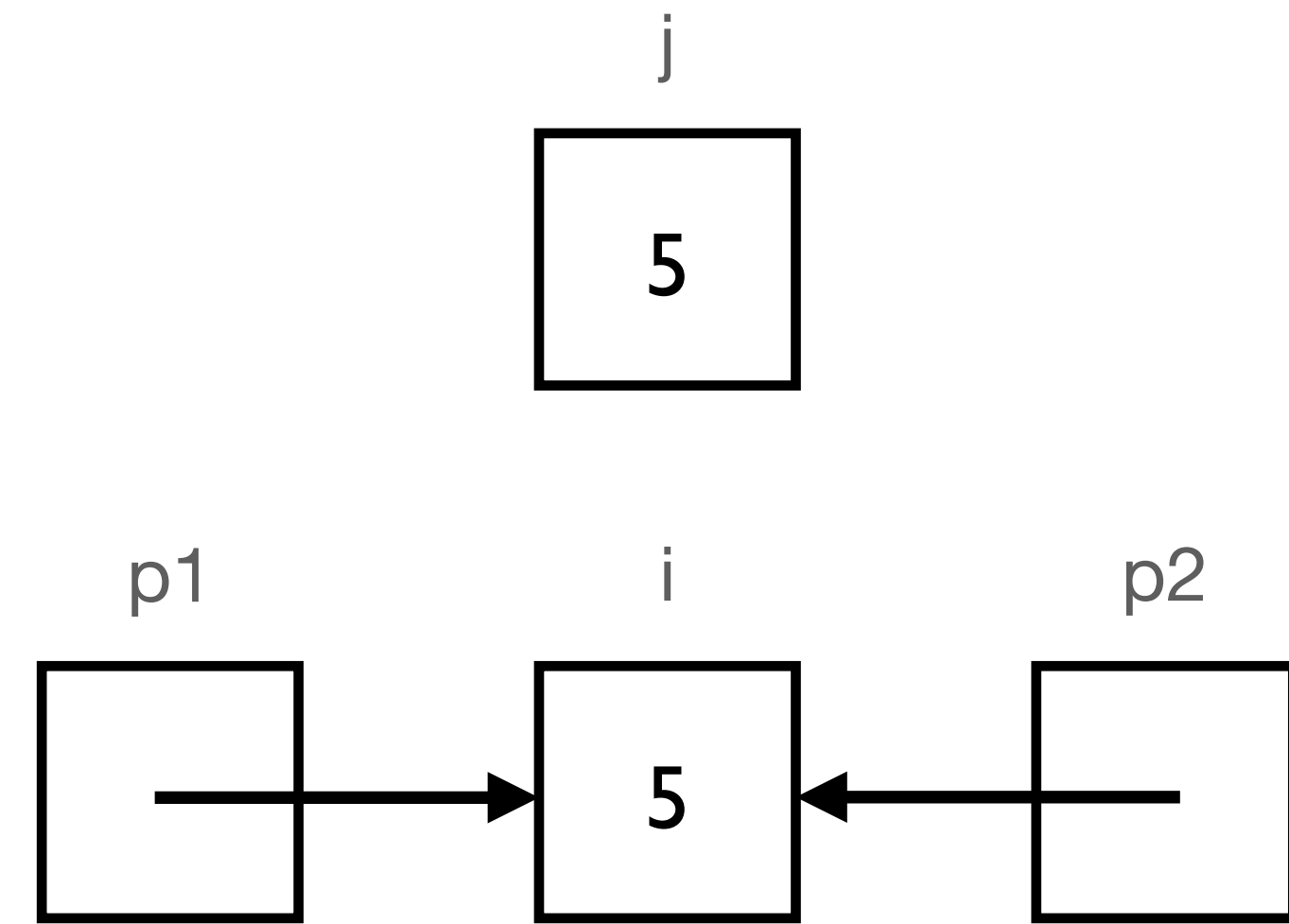
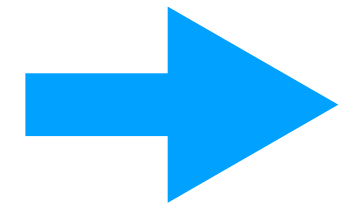
cout << *p1 << " " << *p2 << endl;
p2 = &j;
*p1 = *p1 + 1;
cout << *p1 << " " << *p2 << endl;
return 0;
```



Operatore di *dereferencing*

Esempio

```
int i = 5, j = 5;  
int* p1 = &i;  
int* p2 = &i;  
  
cout << *p1 << " " << *p2 << endl;  
p2 = &j;  
*p1 = *p1 + 1;  
cout << *p1 << " " << *p2 << endl;  
return 0;
```



Output

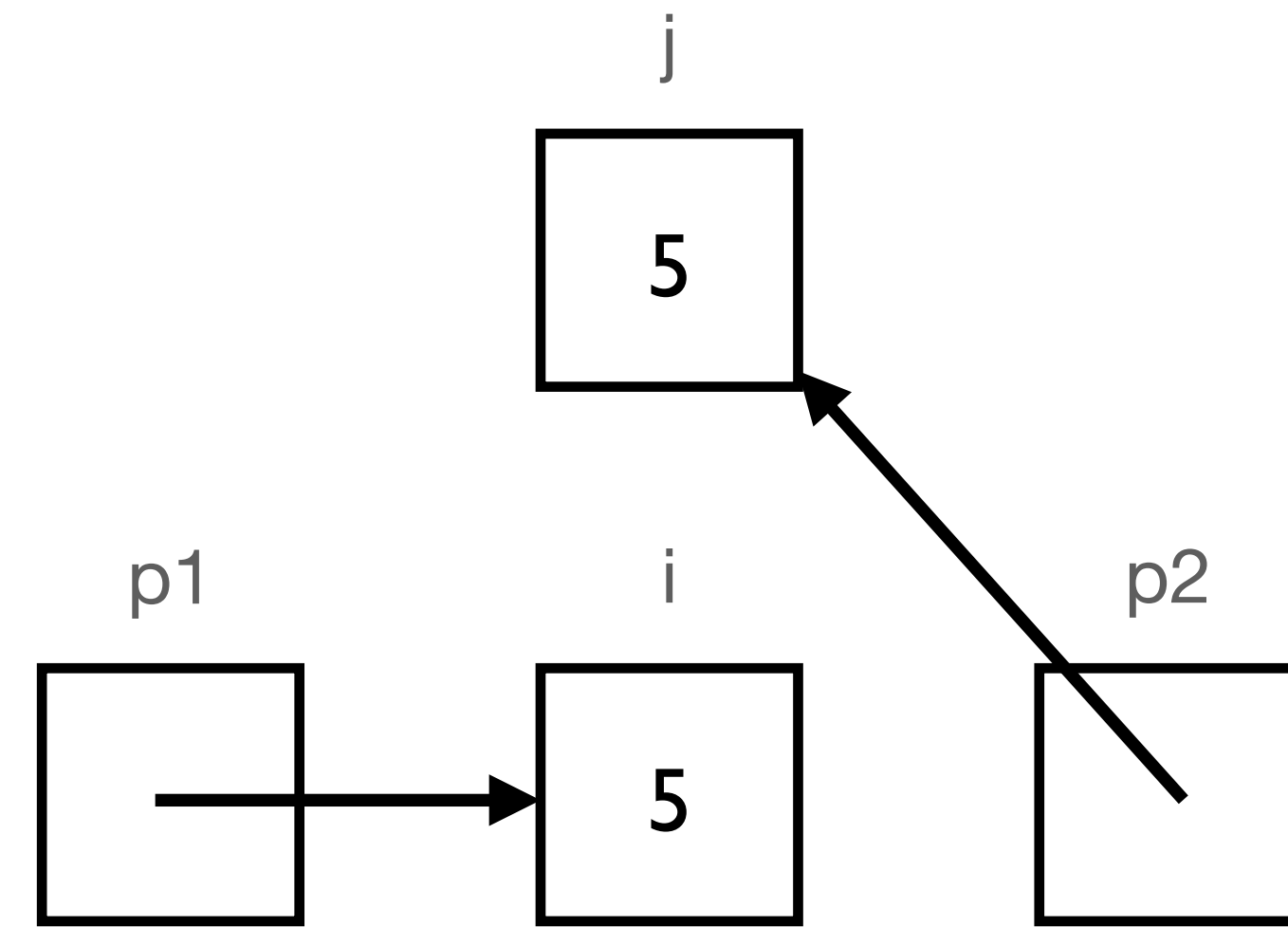
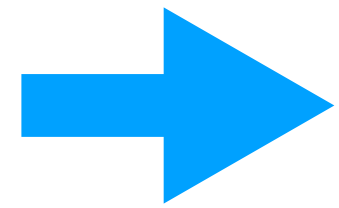
5 5

Operatore di *dereferencing*

Esempio

```
int i = 5, j = 5;
int* p1 = &i;
int* p2 = &i;

cout << *p1 << " " << *p2 << endl;
p2 = &j;
*p1 = *p1 + 1;
cout << *p1 << " " << *p2 << endl;
return 0;
```



Output

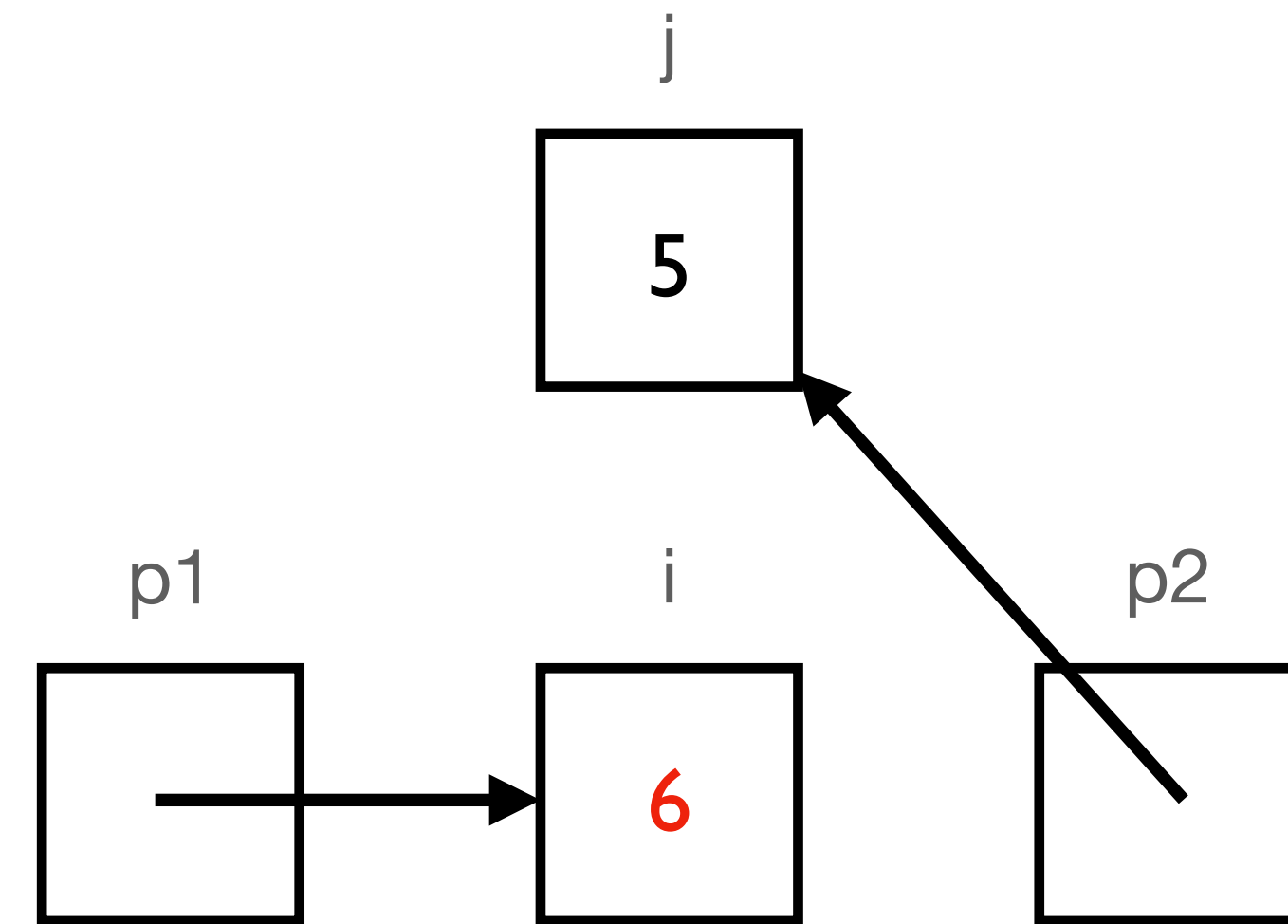
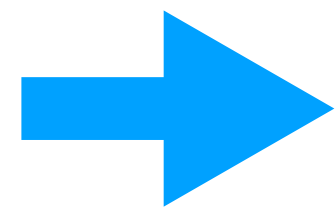
5 5

Operatore di *dereferencing*

Esempio

```
int i = 5, j = 5;
int* p1 = &i;
int* p2 = &i;

cout << *p1 << " " << *p2 << endl;
p2 = &j;
*p1 = *p1 + 1;
cout << *p1 << " " << *p2 << endl;
return 0;
```



Output

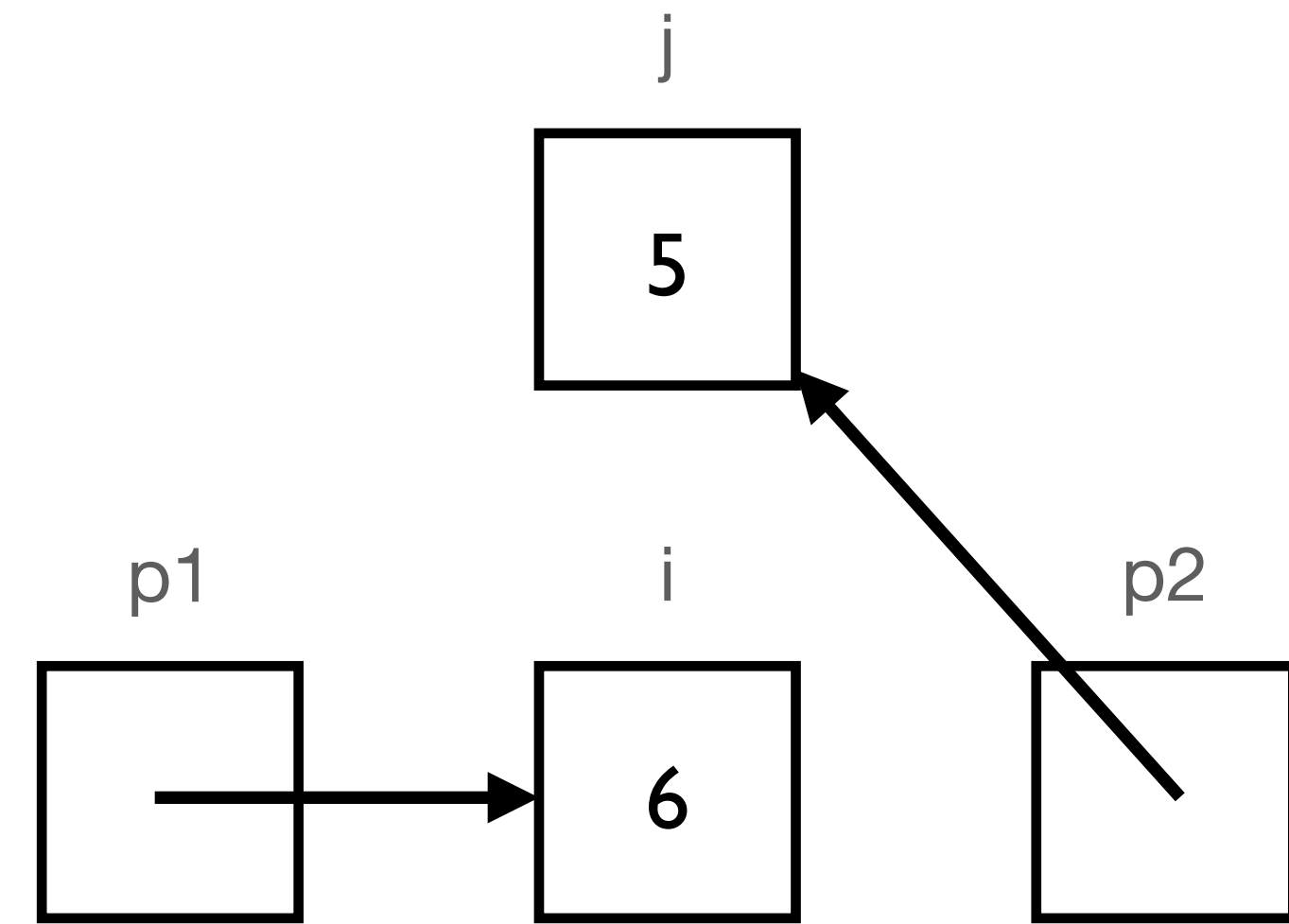
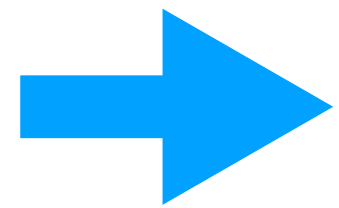
5 5

Operatore di *dereferencing*

Esempio

```
int i = 5, j = 5;
int* p1 = &i;
int* p2 = &i;

cout << *p1 << " " << *p2 << endl;
p2 = &j;
*p1 = *p1 + 1;
cout << *p1 << " " << *p2 << endl;
return 0;
```



Output

5 5
6 5

Puntatori

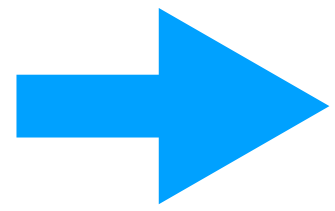
Puntatore a puntatore

- Un tipo puntatore può puntare a **qualsiasi** tipo, anche ad un altro puntatore

Puntatori

Puntatore a puntatore

- Un tipo puntatore può puntare a **qualsiasi** tipo, anche ad un altro puntatore

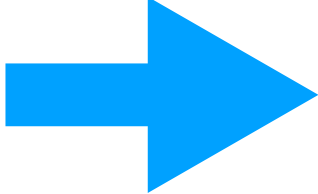


```
int i = 42;  
int* p1 = &i;  
int** p2 = &p1;  
int*** p3 = &p2;
```

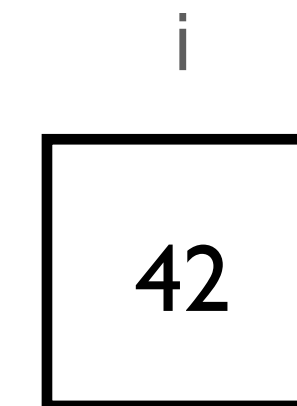

Puntatori

Puntatore a puntatore

- Un tipo puntatore può puntare a **qualsiasi** tipo, anche ad un altro puntatore



```
int i = 42;  
int* p1 = &i;  
int** p2 = &p1;  
int*** p3 = &p2;
```



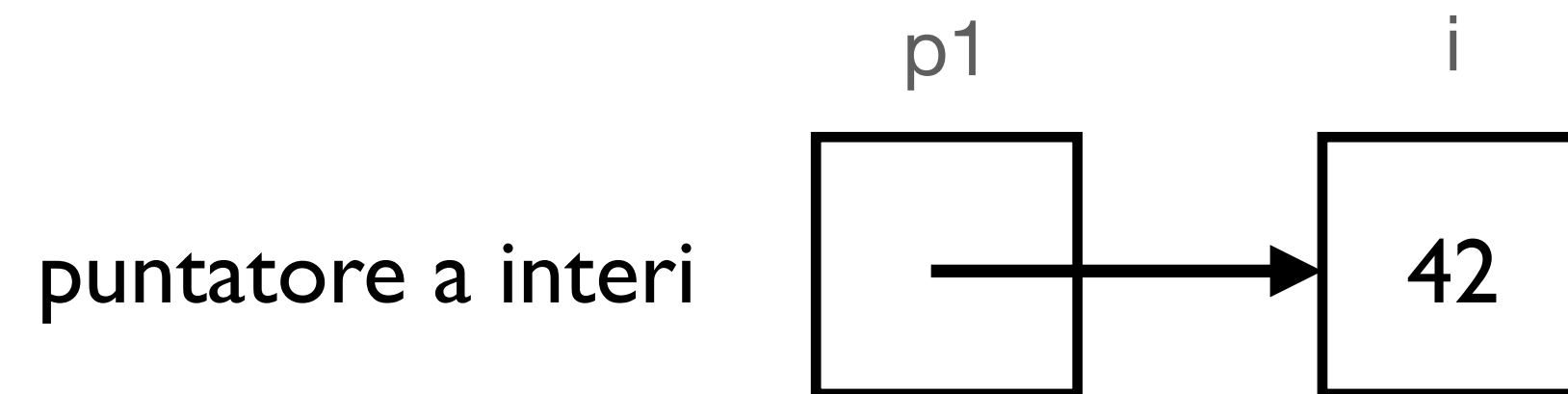
Puntatori

Puntatore a puntatore

- Un tipo puntatore può puntare a **qualsiasi** tipo, anche ad un altro puntatore

➔

```
int i = 42;  
int* p1 = &i;  
int** p2 = &p1;  
int*** p3 = &p2;
```

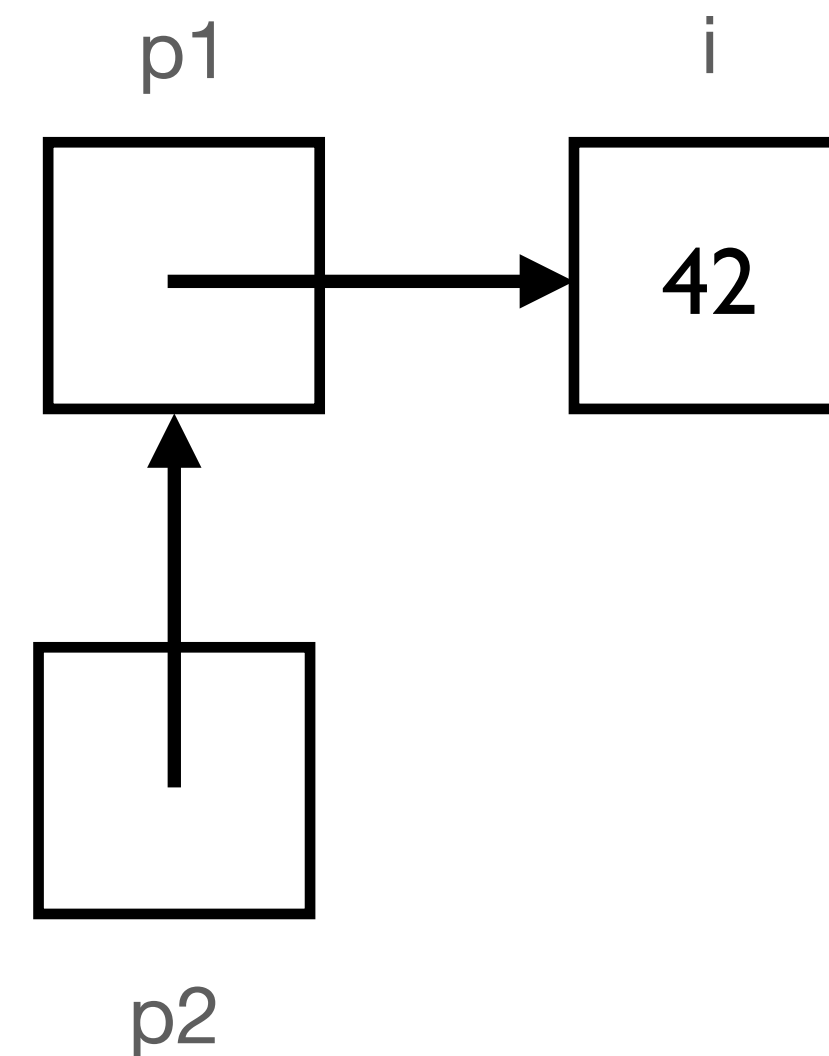
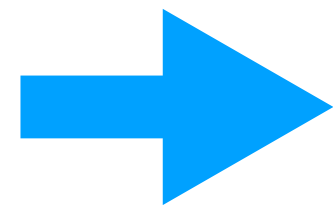


Puntatori

Puntatore a puntatore

- Un tipo puntatore può puntare a **qualsiasi** tipo, anche ad un altro puntatore

```
int i = 42;  
int* p1 = &i;  
int** p2 = &p1;  
int*** p3 = &p2;
```



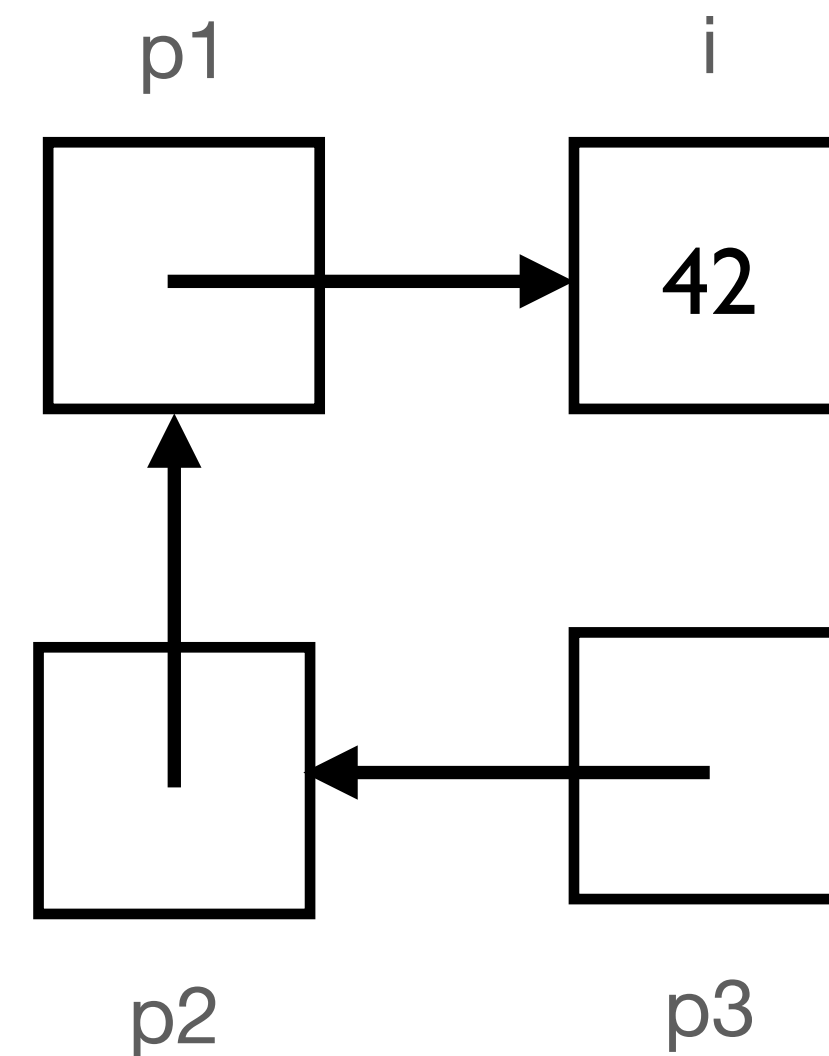
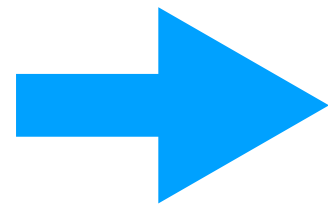
puntatore a puntatore a interi

Puntatori

Puntatore a puntatore

- Un tipo puntatore può puntare a **qualsiasi** tipo, anche ad un altro puntatore

```
int i = 42;  
int* p1 = &i;  
int** p2 = &p1;  
int*** p3 = &p2;
```



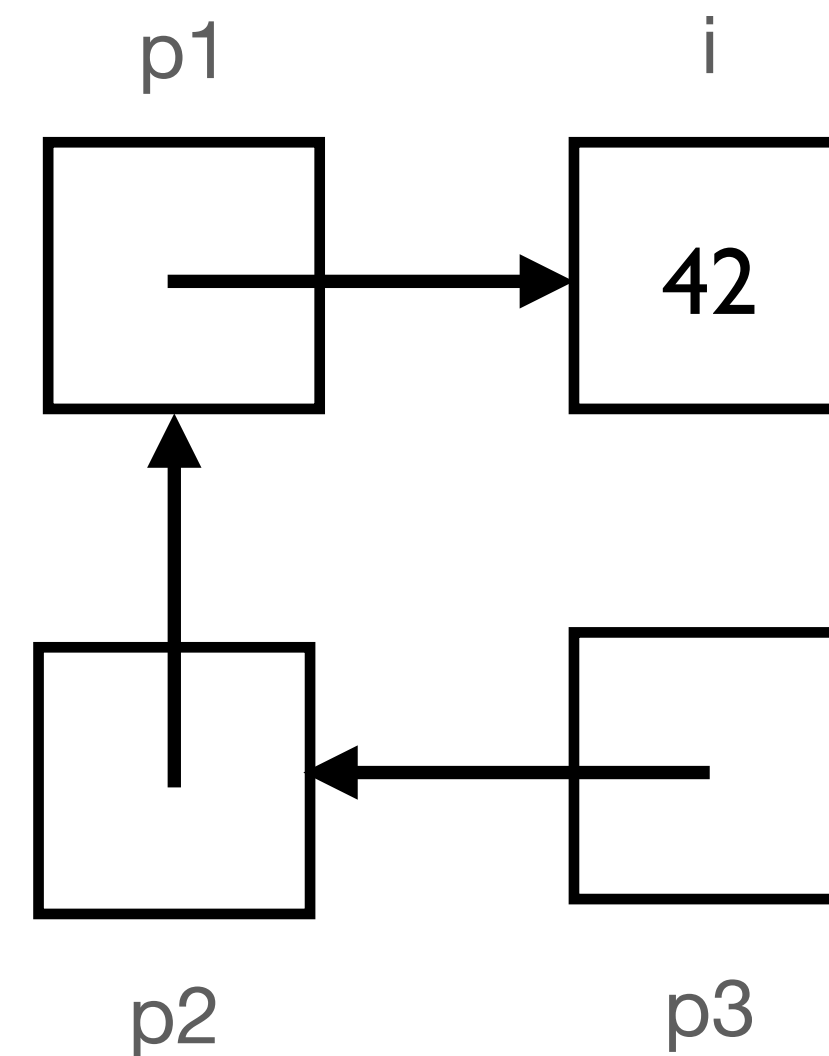
puntatore a puntatore a puntatore a interi

Puntatori

Puntatore a puntatore

- Un tipo puntatore può puntare a **qualsiasi** tipo, anche ad un altro puntatore

```
int i = 42;  
int* p1 = &i;  
int** p2 = &p1;  
int*** p3 = &p2;  
**p2 = 3;
```

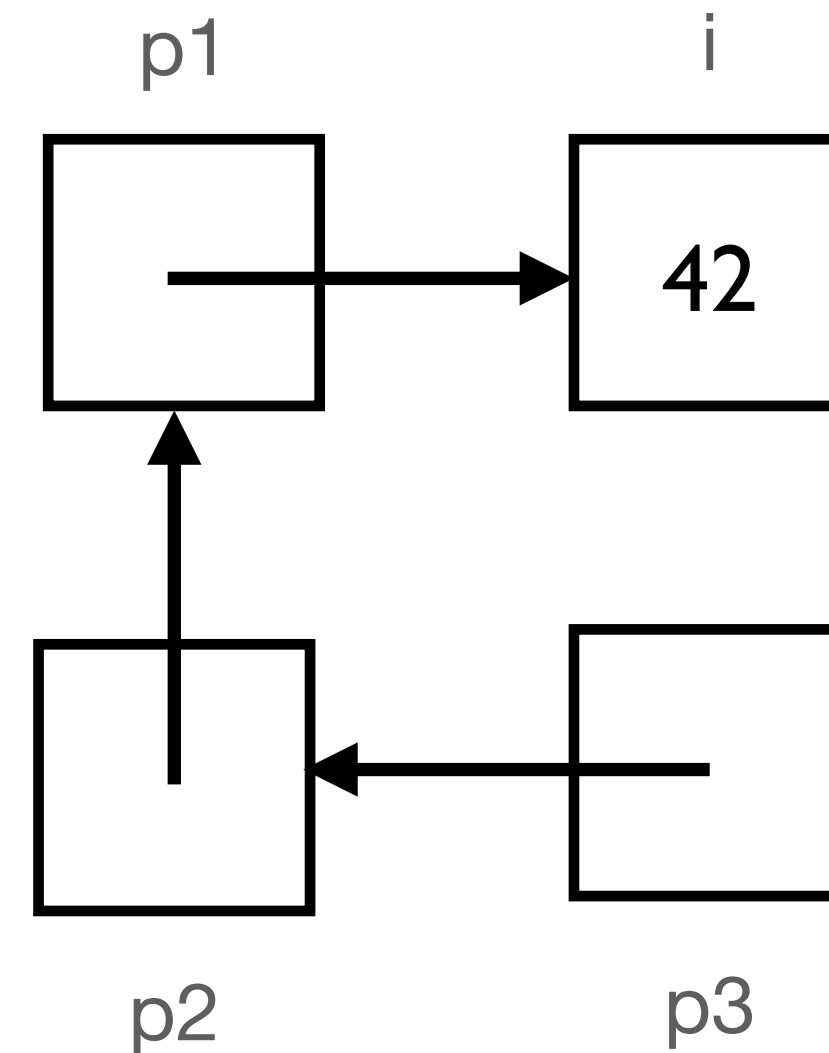
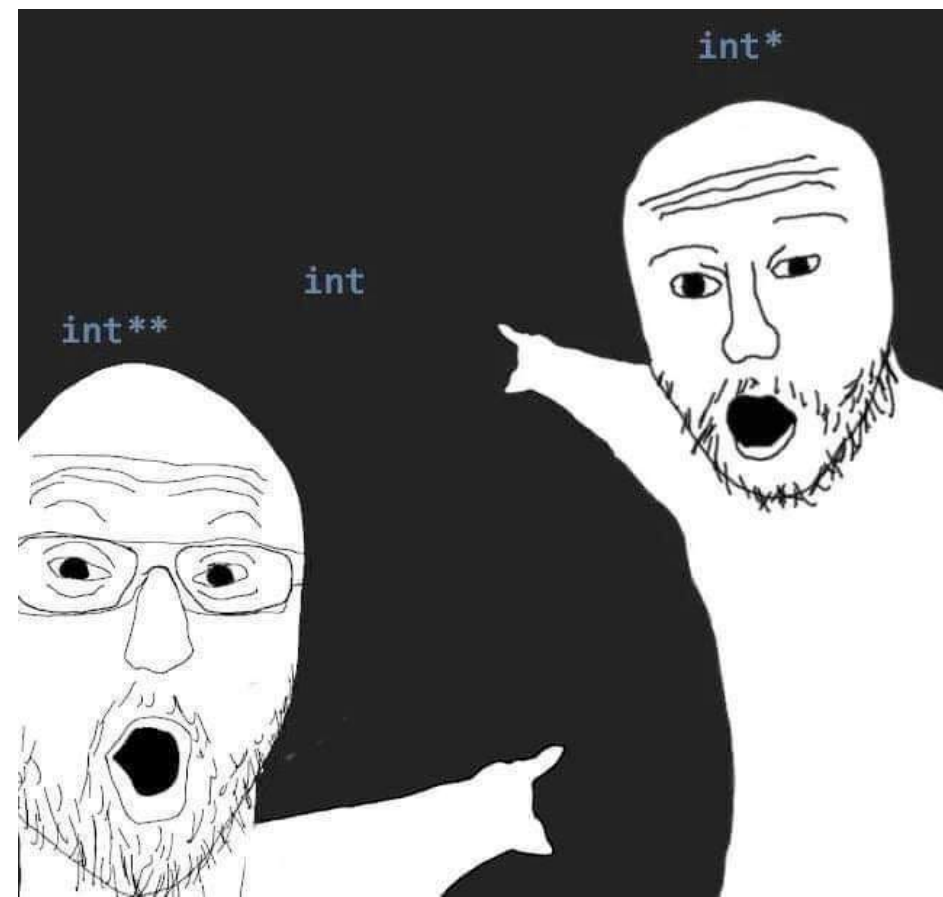


Puntatori

Puntatore a puntatore

- Un tipo puntatore può puntare a **qualsiasi** tipo, anche ad un altro puntatore

```
int i = 42;  
int* p1 = &i;  
int** p2 = &p1;  
int*** p3 = &p2;  
**p2 = 3;
```

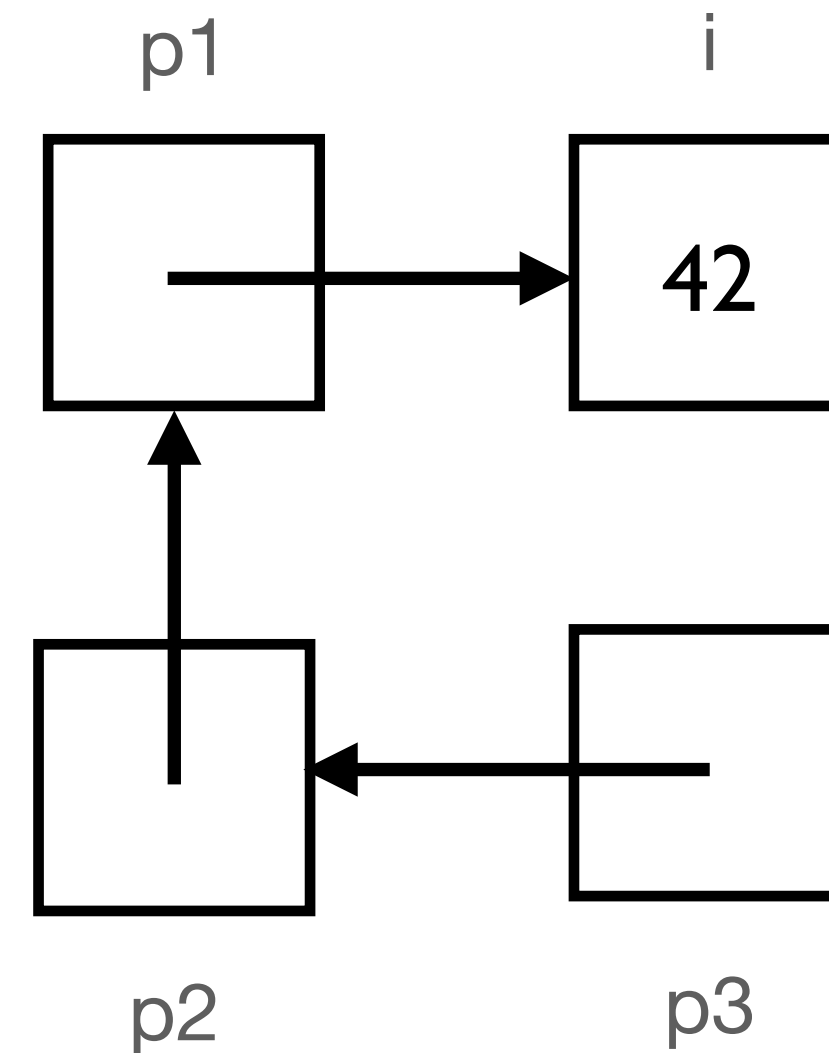
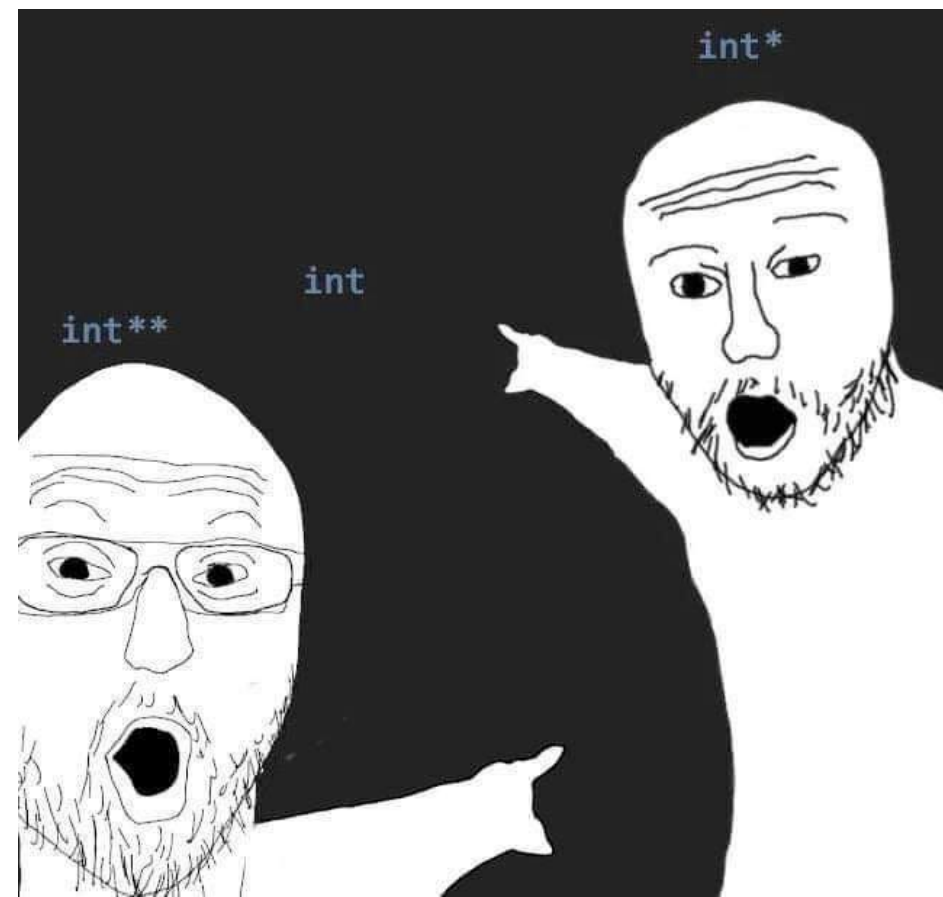


Puntatori

Puntatore a puntatore

- Un tipo puntatore può puntare a **qualsiasi** tipo, anche ad un altro puntatore

```
int i = 42;  
int* p1 = &i;  
int** p2 = &p1;  
int*** p3 = &p2;  
**p2 = 3;
```

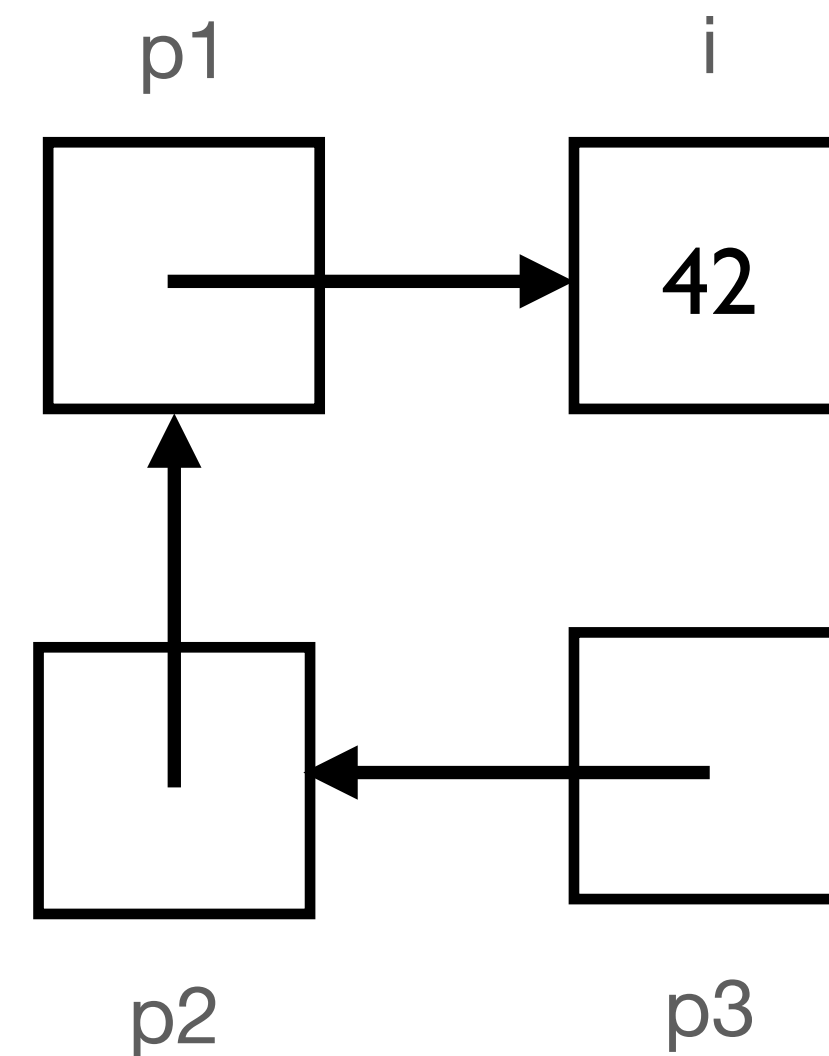
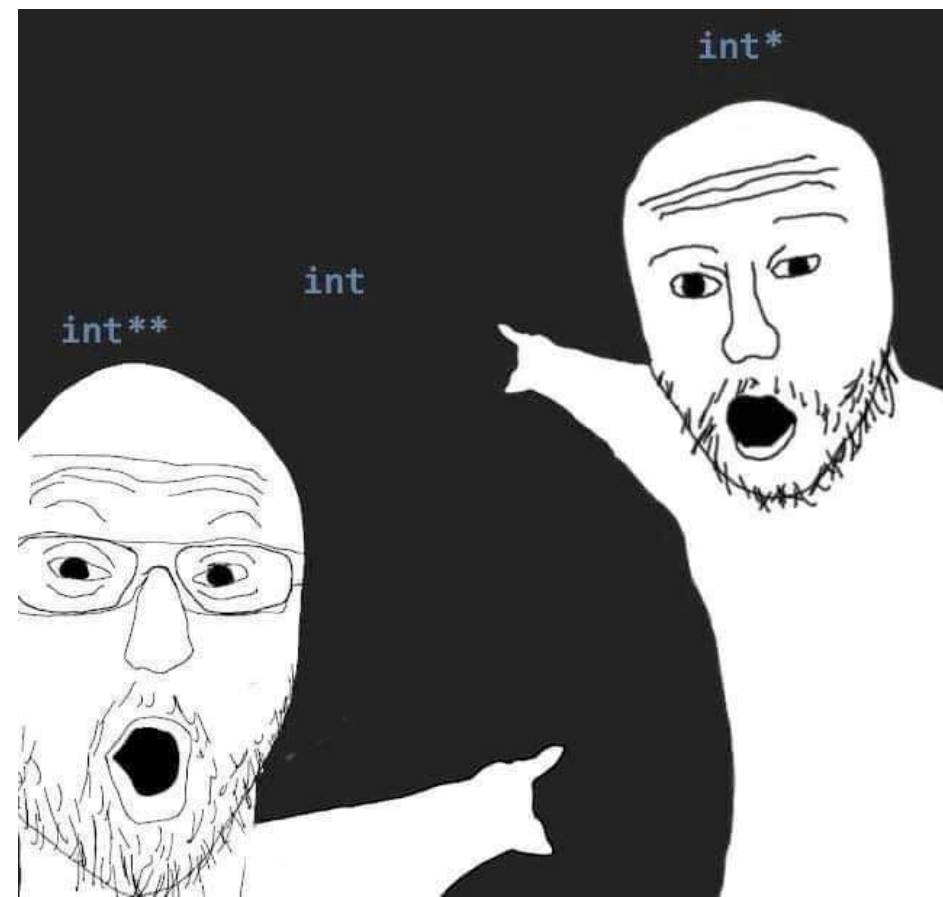


Puntatori

Puntatore a puntatore

- Un tipo puntatore può puntare a **qualsiasi** tipo, anche ad un altro puntatore

```
int i = 42;  
int* p1 = &i;  
int** p2 = &p1;  
int*** p3 = &p2;  
**p2 = 3;
```



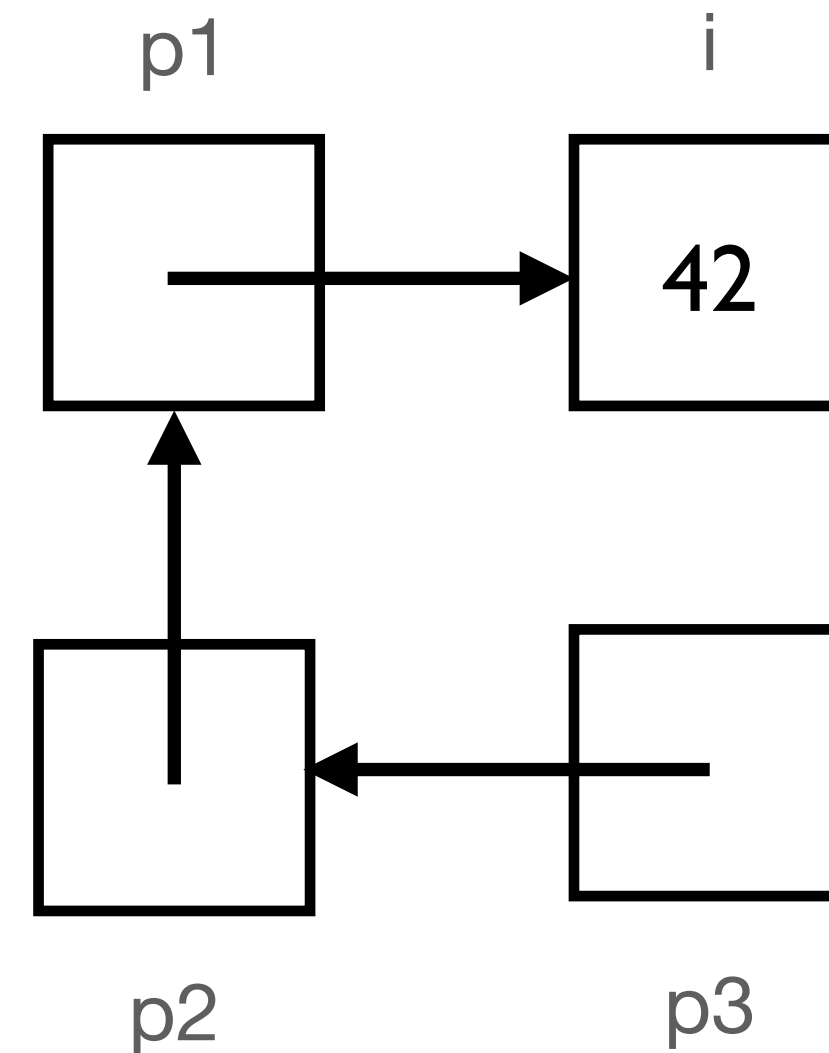
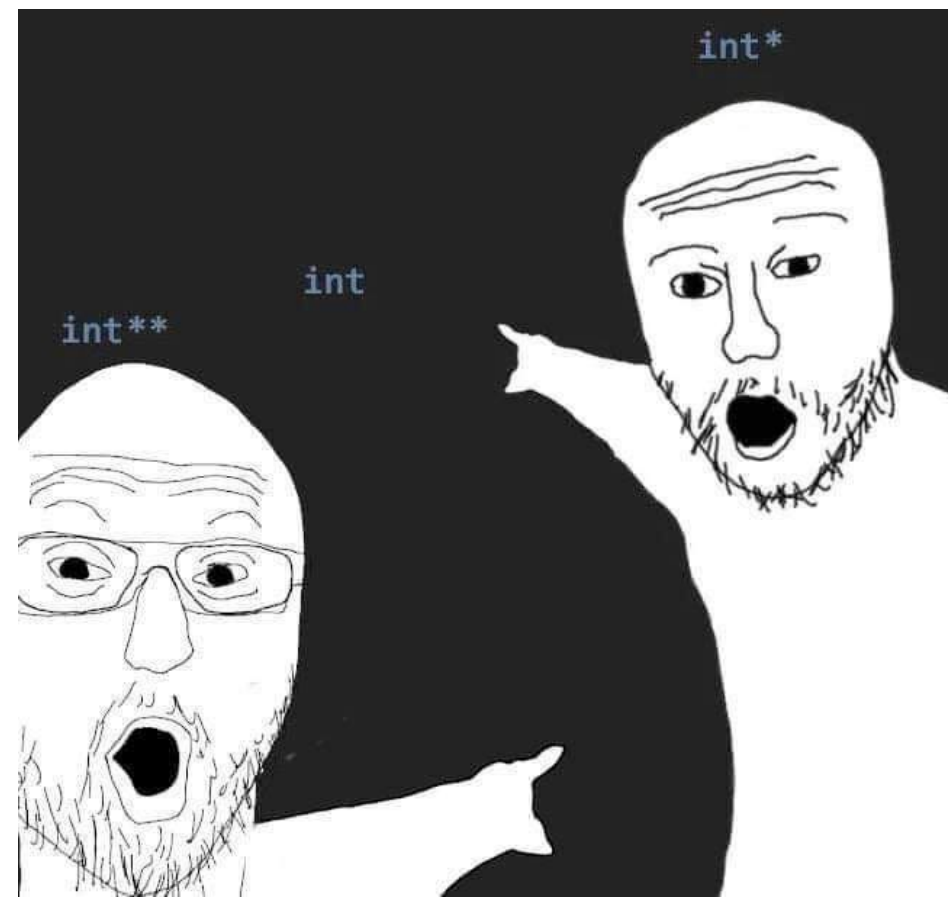
$(* (*p2))$

Puntatori

Puntatore a puntatore

- Un tipo puntatore può puntare a **qualsiasi** tipo, anche ad un altro puntatore

```
int i = 42;  
int* p1 = &i;  
int** p2 = &p1;  
int*** p3 = &p2;  
**p2 = 3;
```



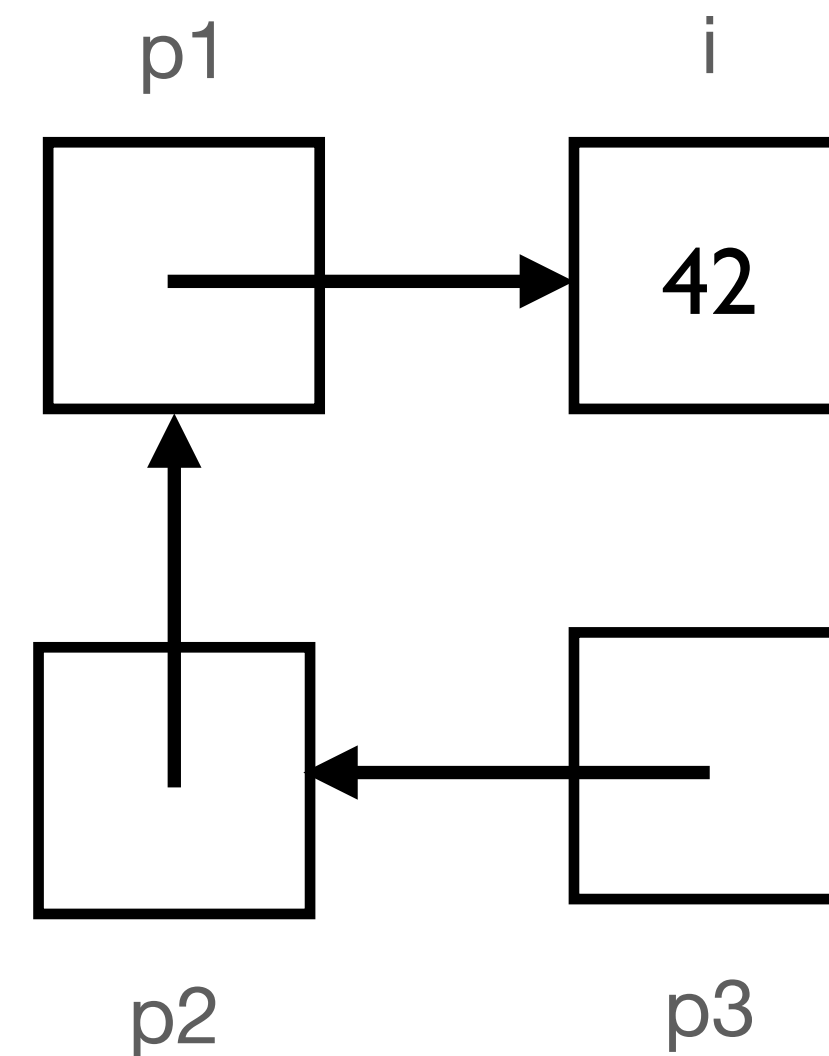
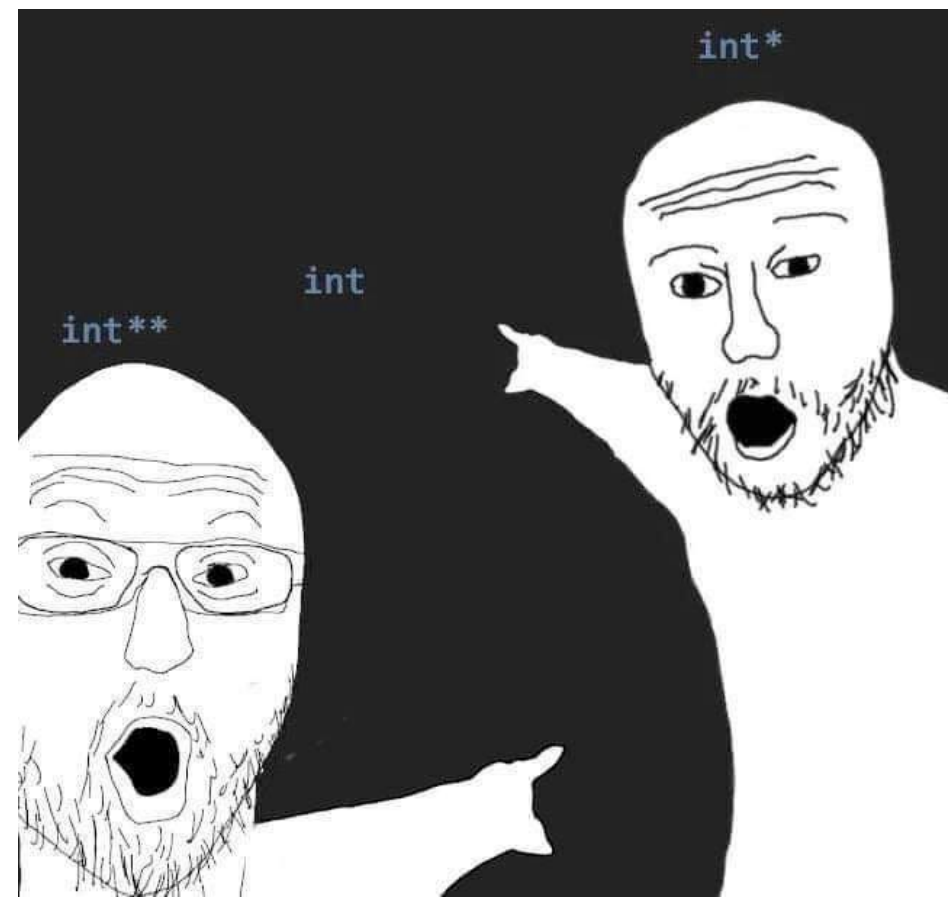
$(* (* p2)) \rightarrow (* p1)$

Puntatori

Puntatore a puntatore

- Un tipo puntatore può puntare a **qualsiasi** tipo, anche ad un altro puntatore

```
int i = 42;  
int* p1 = &i;  
int** p2 = &p1;  
int*** p3 = &p2;  
**p2 = 3;
```



$(* (*p2)) \longrightarrow (*p1) \longrightarrow i$

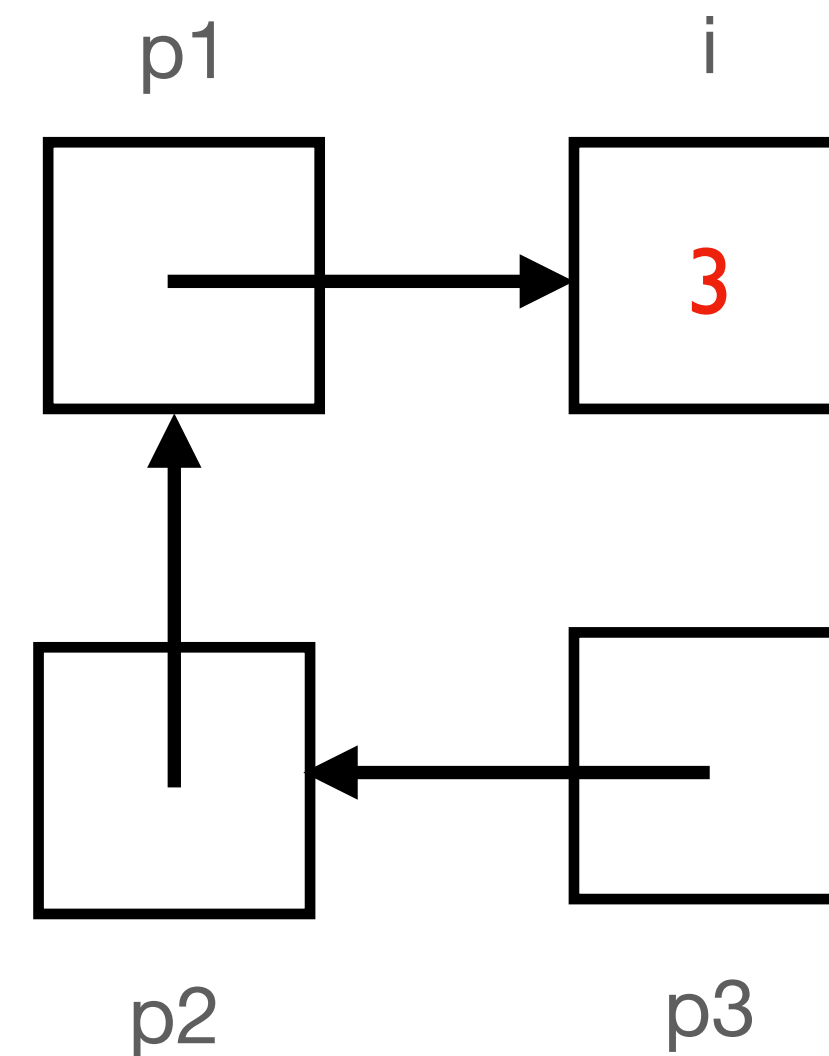
Puntatori

Puntatore a puntatore

- Un tipo puntatore può puntare a **qualsiasi** tipo, anche ad un altro puntatore

```
int i = 42;  
int* p1 = &i;  
int** p2 = &p1;  
int*** p3 = &p2;
```

→ **p2 = 3;



$(* (*p2)) \longrightarrow (*p1) \longrightarrow i$

Puntatori e array

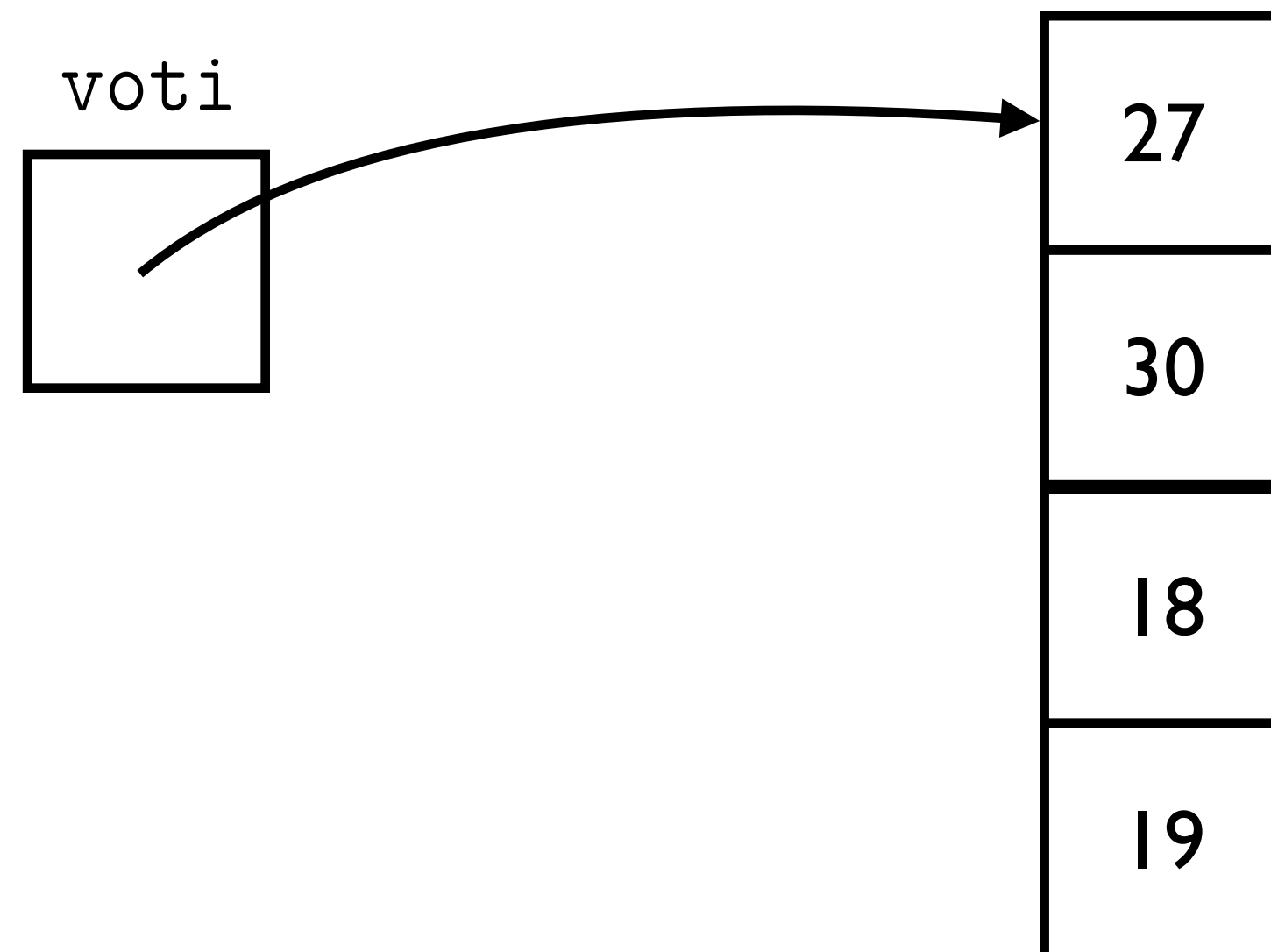
```
int voti[4] = {27, 30, 18, 19}
```

- In C++, gli array sono implementati tramite puntatori
 - `voti` è un puntatore **costante** a interi al primo elemento dell'array

Puntatori e array

```
int voti[4] = {27, 30, 18, 19}
```

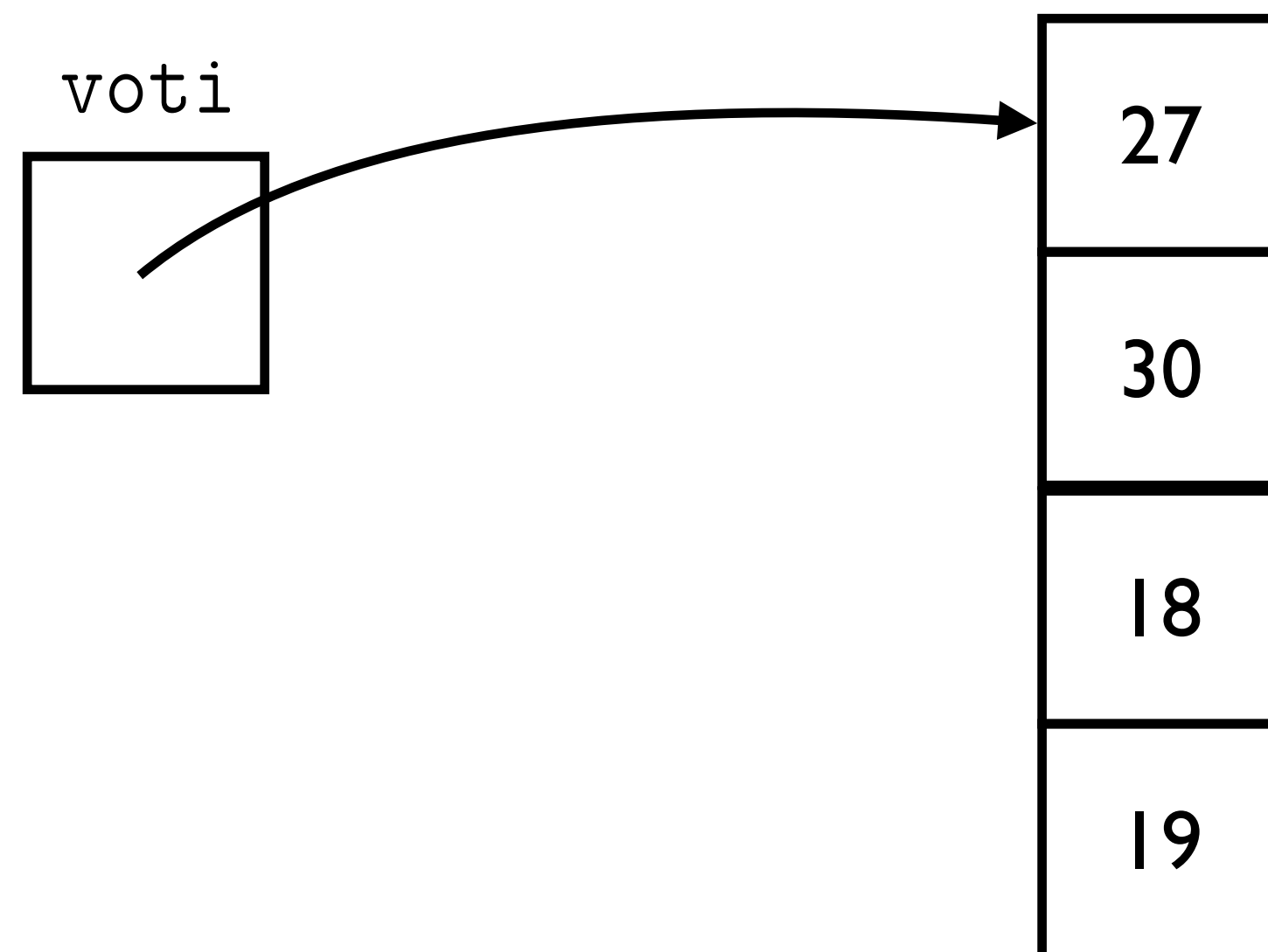
- In C++, gli array sono implementati tramite puntatori
 - `voti` è un puntatore **costante** a interi al primo elemento dell'array



Puntatori e array

```
int voti[4] = {27, 30, 18, 19}
```

- In C++, gli array sono implementati tramite puntatori
 - `voti` è un puntatore **costante** a interi al primo elemento dell'array



Lettura primo elemento di un array

Notazione vettoriale

`voti[0]`

Dereferenziazione

`*voti`

Aritmetica dei puntatori

- E' possibile effettuare (alcune) operazioni aritmetiche sui puntatori

Aritmetica dei puntatori

- E' possibile effettuare (alcune) operazioni aritmetiche sui puntatori
- Dato un puntatore p di tipo T :

Aritmetica dei puntatori

- E' possibile effettuare (alcune) operazioni aritmetiche sui puntatori
- Dato un puntatore p di tipo T :

$p + 1$ il risultato è il puntatore p incrementato di un numero pari alla dimensione del tipo T

$p - 1$ il risultato è il puntatore p decrementato di un numero pari alla dimensione del tipo T

Aritmetica dei puntatori

- E' possibile effettuare (alcune) operazioni aritmetiche sui puntatori
- Dato un puntatore p di tipo T :

$p + 1$ il risultato è il puntatore p incrementato di un numero pari alla dimensione del tipo T

$p - 1$ il risultato è il puntatore p decrementato di un numero pari alla dimensione del tipo T

- In generale, dato $i \in \mathbb{N}$

Aritmetica dei puntatori

- E' possibile effettuare (alcune) operazioni aritmetiche sui puntatori
- Dato un puntatore p di tipo T :

$p + 1$ il risultato è il puntatore p incrementato di un numero pari alla dimensione del tipo T

$p - 1$ il risultato è il puntatore p decrementato di un numero pari alla dimensione del tipo T

- In generale, dato $i \in \mathbb{N}$

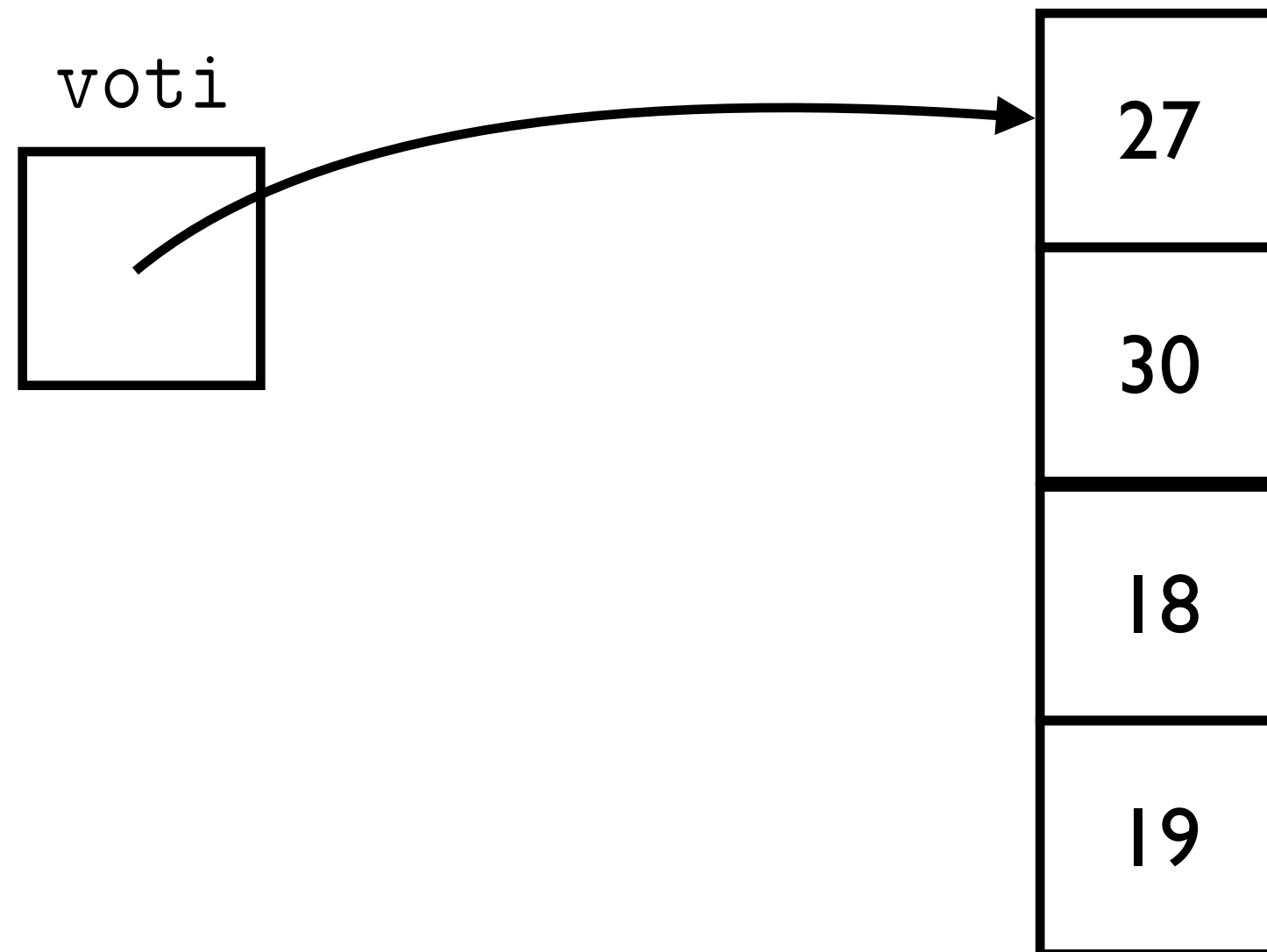
$p + i$ il risultato è il puntatore p incrementato di un numero pari alla dimensione del tipo T moltiplicato per i

$p - i$ il risultato è il puntatore p decrementato di un numero pari alla dimensione del tipo T moltiplicato per i

Aritmetica dei puntatori

Esempio

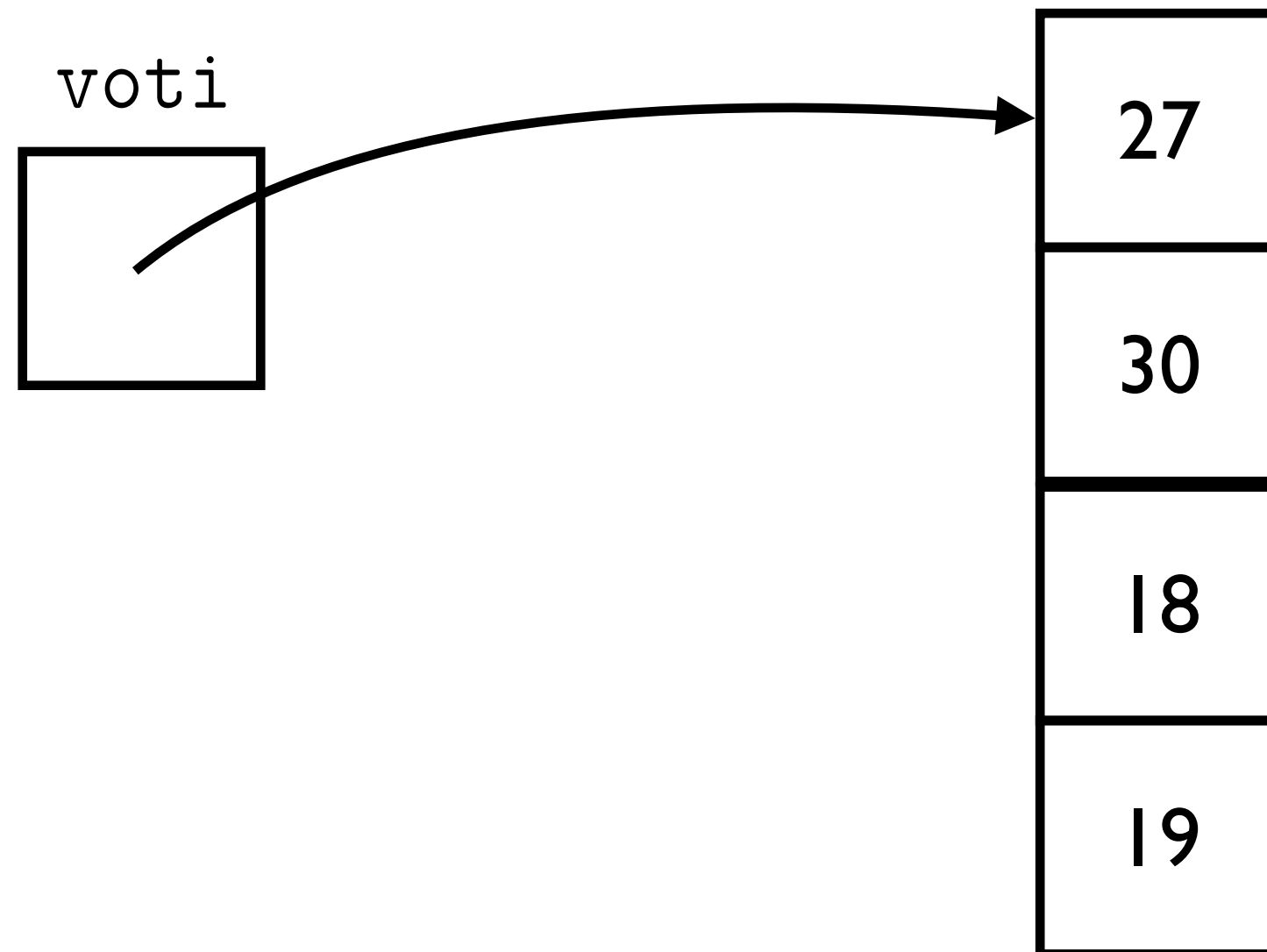
```
int voti[4] = {27, 30, 18, 19}
```



Aritmetica dei puntatori

Esempio

```
int voti[4] = {27, 30, 18, 19}
```



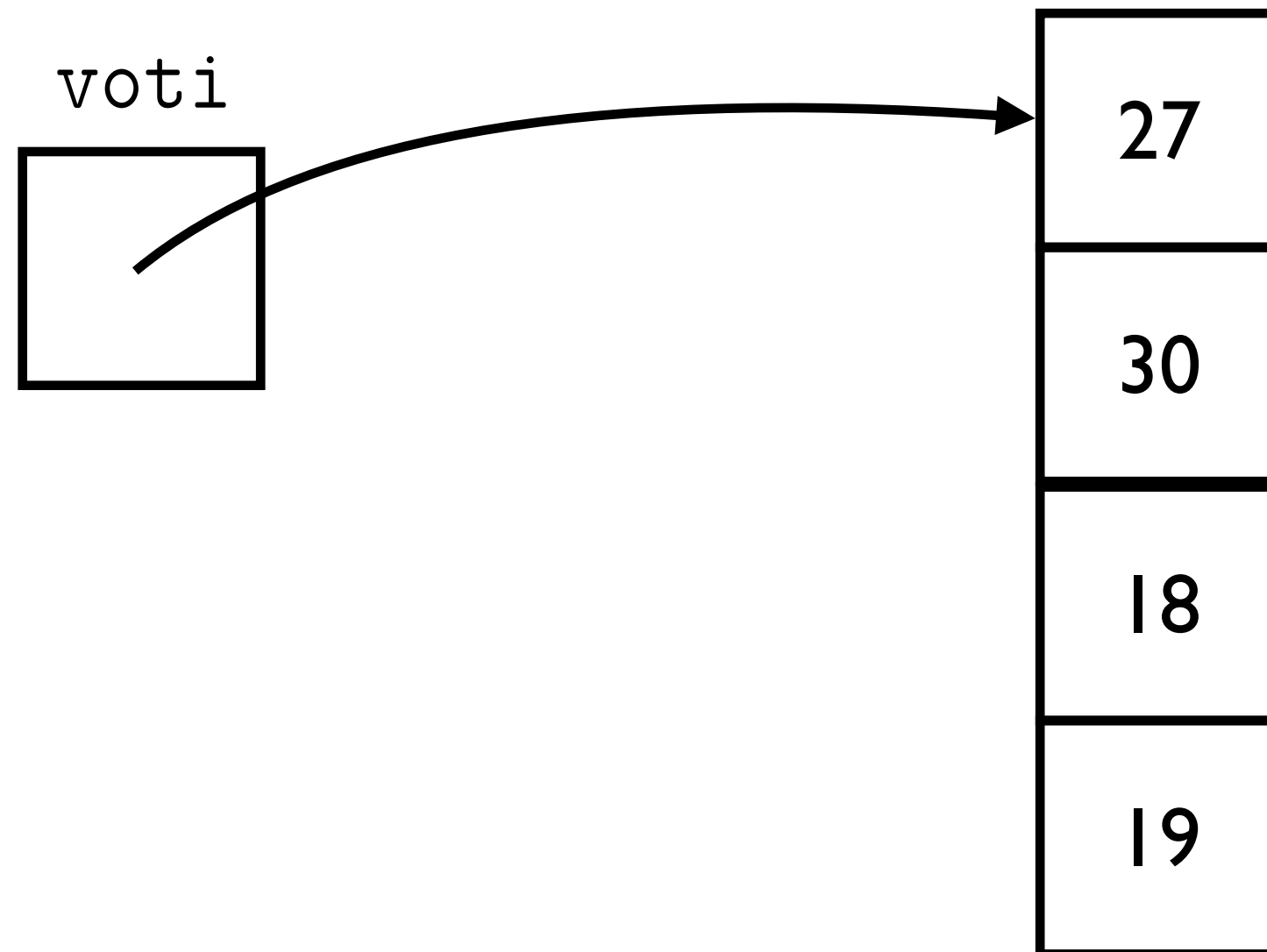
`*voti`

27

Aritmetica dei puntatori

Esempio

```
int voti[4] = {27, 30, 18, 19}
```



```
*voti
```

27

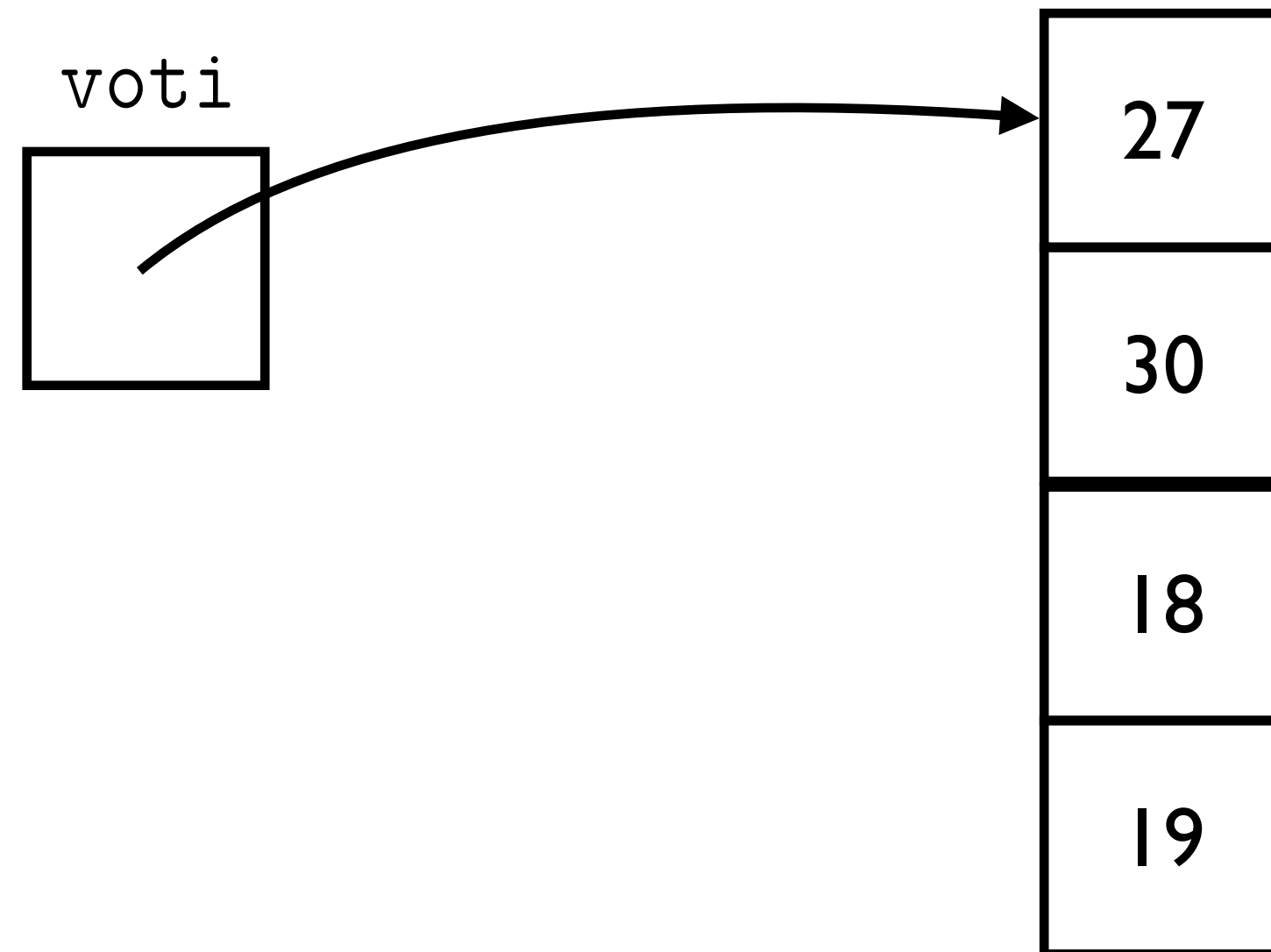
```
*(voti + 1)
```

30

Aritmetica dei puntatori

Esempio

```
int voti[4] = {27, 30, 18, 19}
```

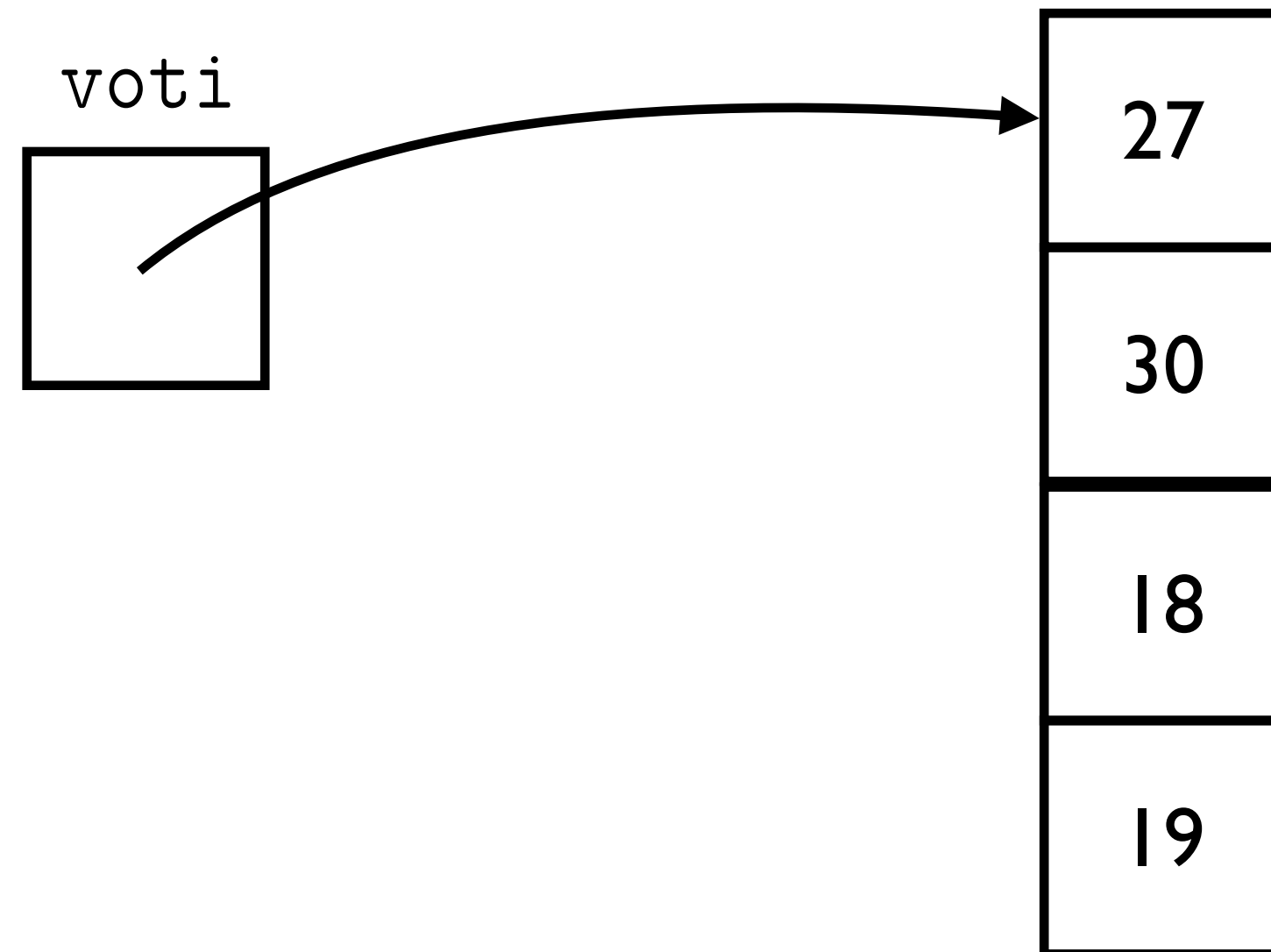


<code>*voti</code>	27
<code>*(voti + 1)</code>	30
<code>*(voti + 3)</code>	19

Aritmetica dei puntatori

Esempio

```
int voti[4] = {27, 30, 18, 19}
```



<code>*voti</code>	27
<code>*(voti + 1)</code>	30
<code>*(voti + 3)</code>	19

- **NB:** Non è consentito sommare, moltiplicare o dividere due puntatori

Puntatori e array

```
int voti[4] = {27, 30, 18, 19}
```

- In C++, gli array sono implementati tramite puntatori
 - `voti` è un **puntatore costante** a interi al primo elemento dell'array

Puntatori e array

```
int voti[4] = {27, 30, 18, 19}
```

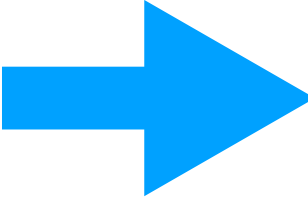
- In C++, gli array sono implementati tramite puntatori
 - `voti` è un **puntatore costante** a interi al primo elemento dell'array
- In C++, un array di tipo T è un puntatore costante a T al primo elemento dell'array: non può essere aggiornato

Puntatori e array

```
int voti[4] = {27, 30, 18, 19}
```

- In C++, gli array sono implementati tramite puntatori
 - `voti` è un **puntatore costante** a interi al primo elemento dell'array
- In C++, un array di tipo T è un puntatore costante a T al primo elemento dell'array: non può essere aggiornato
- Anche le stringhe (a la C) sono puntatori costanti a caratteri che terminano con `'\0'`

Esercizio

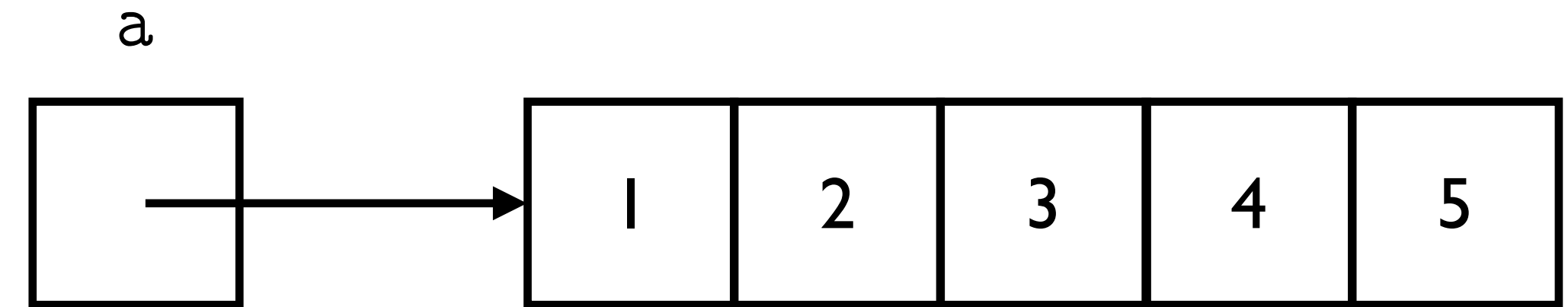


```
int a[5] = {1, 2, 3, 4, 5};  
int* p = a;  
++(*p);  
++p;  
++(*(p + 2));  
a++;
```

Esercizio

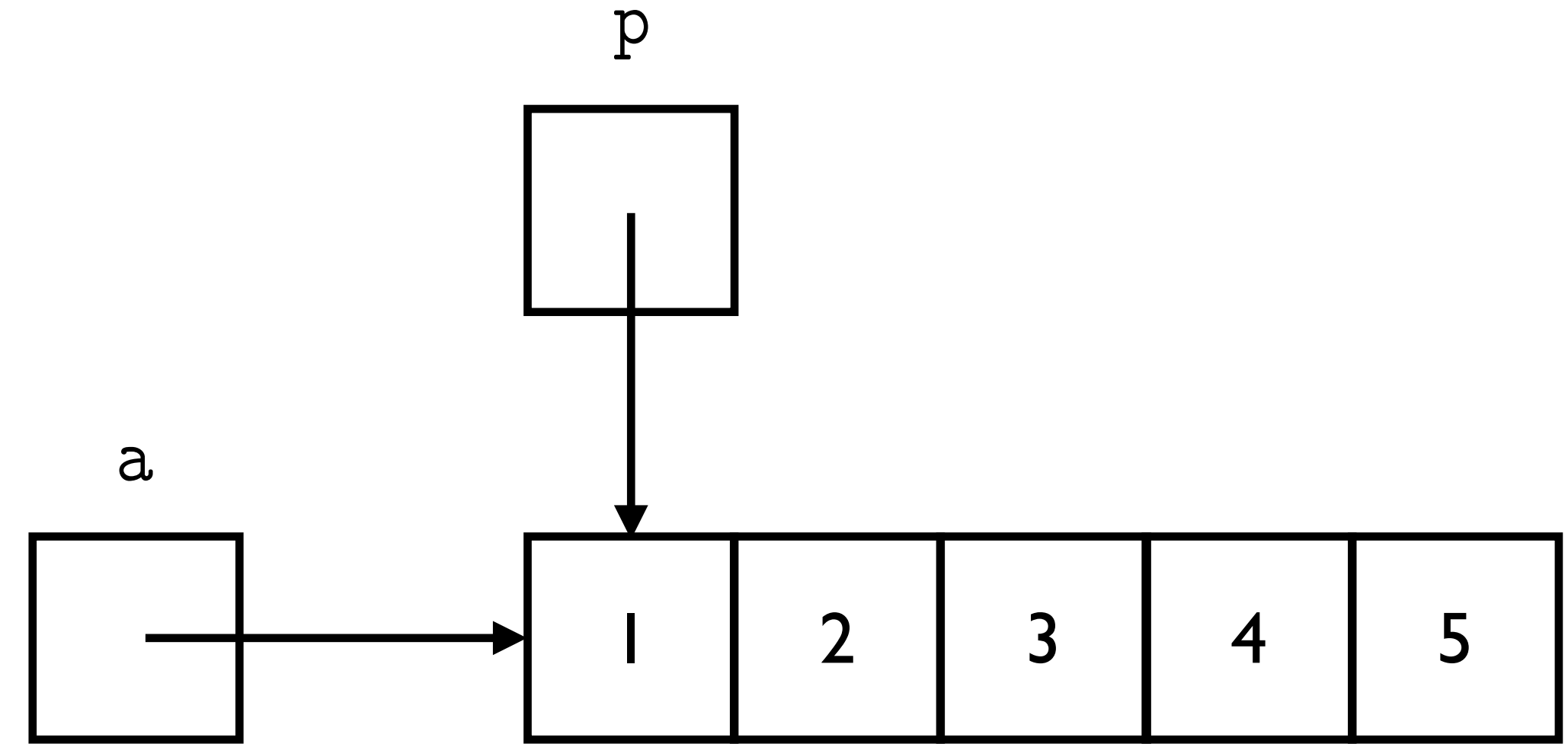
➔

```
int a[5] = {1, 2, 3, 4, 5};  
int* p = a;  
++(*p);  
++p;  
++(*(p + 2));  
a++;
```



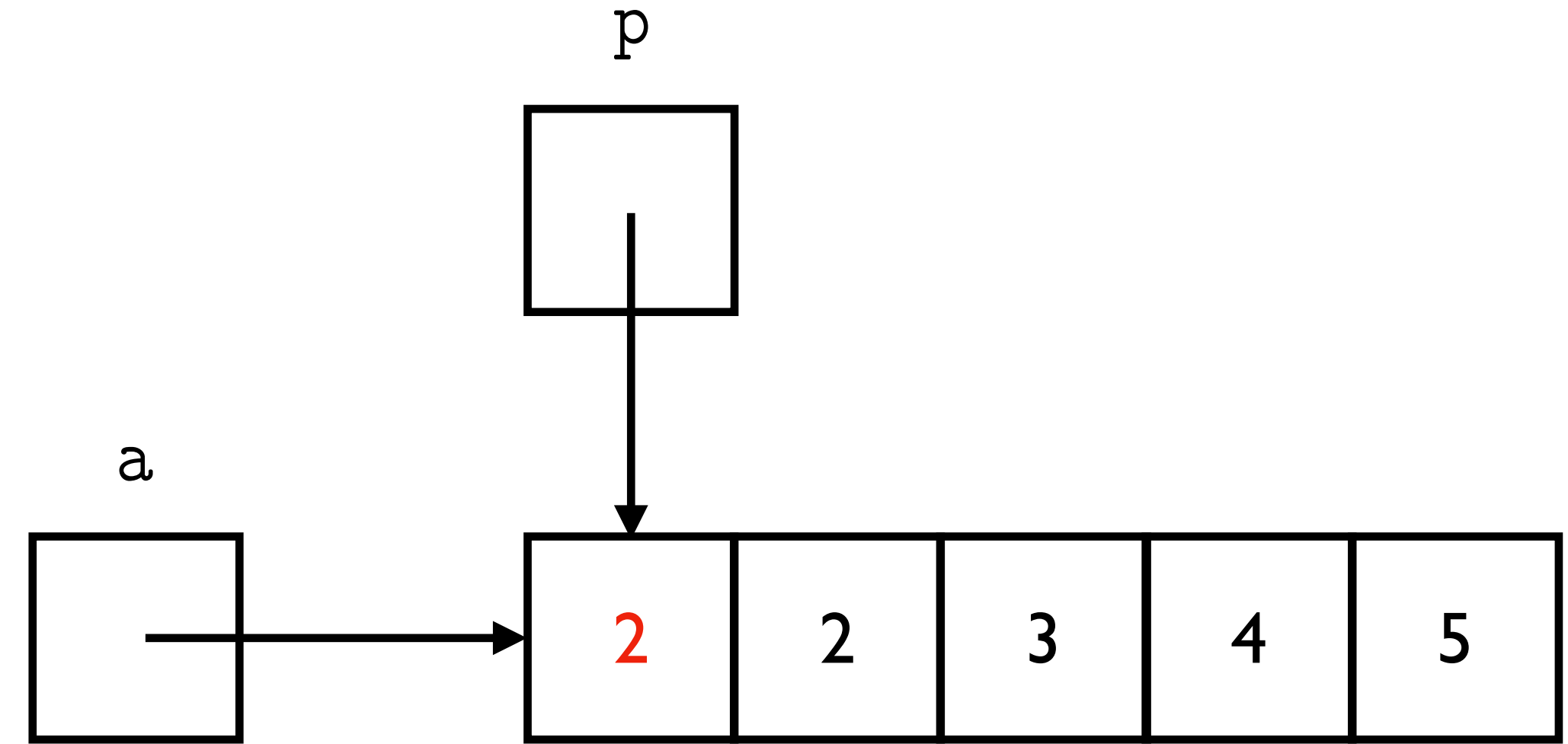
Esercizio

```
int a[5] = {1, 2, 3, 4, 5};  
int* p = a;  
→ ++(*p);  
  ++p;  
  ++(* (p + 2));  
  a++;
```



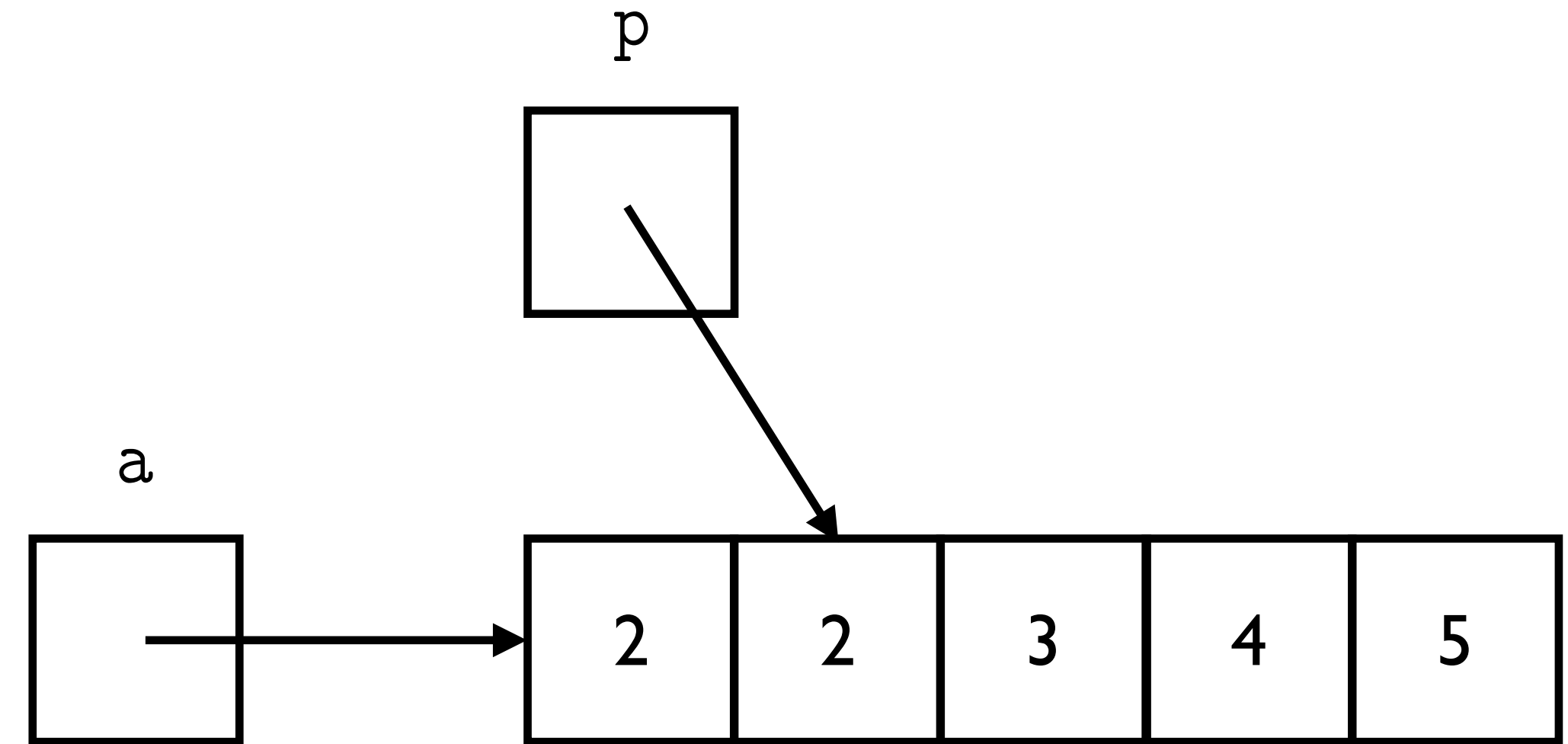
Esercizio

```
int a[5] = {1, 2, 3, 4, 5};  
int* p = a;  
→ ++(*p);  
  ++p;  
  ++(*(p + 2));  
  a++;
```



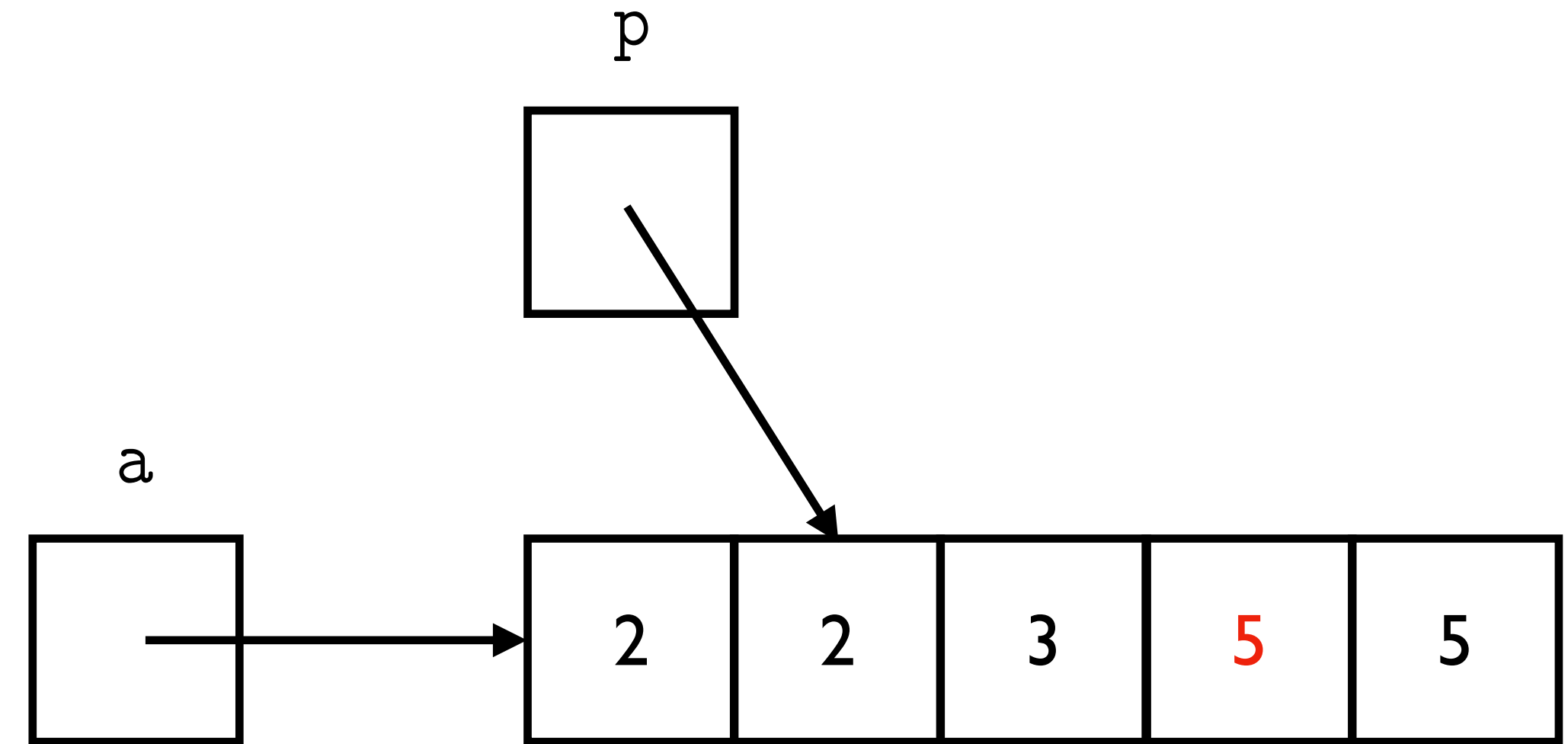
Esercizio

```
int a[5] = {1, 2, 3, 4, 5};  
int* p = a;  
++(*p);  
→ ++p;  
++(*(p + 2));  
a++;
```



Esercizio

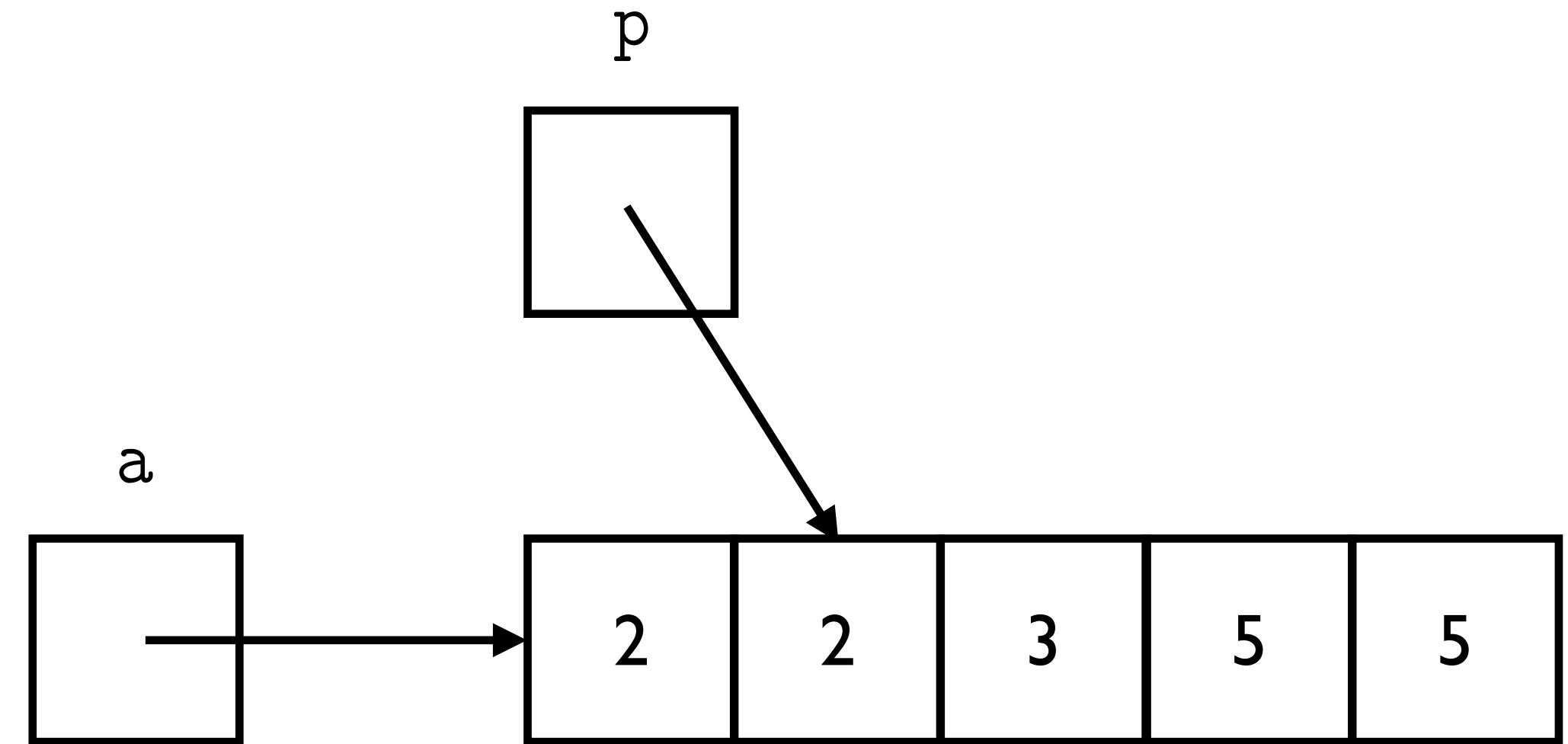
```
int a[5] = {1, 2, 3, 4, 5};  
int* p = a;  
++(*p);  
++p;  
→ ++(*(p + 2));  
a++;
```



Esercizio

```
int a[5] = {1, 2, 3, 4, 5};  
int* p = a;  
++(*p);  
++p;  
++(*(p + 2));  
a++;
```

Errore!



Puntatori nulli

Puntatori nulli

- Un puntatore inizializzato punta a qualche posizione specifica della memoria
- Un puntatore non inizializzato punta ad una posizione aleatoria della memoria

Puntatori nulli

- Un puntatore inizializzato punta a qualche posizione specifica della memoria
- Un puntatore non inizializzato punta ad una posizione aleatoria della memoria

```
int * p;
```

Puntatori nulli

- Un puntatore inizializzato punta a qualche posizione specifica della memoria
- Un puntatore non inizializzato punta ad una posizione aleatoria della memoria

```
int * p;
```

- E' importante assegnare sempre un valore ai puntatori
- Puntatore nullo: non indirizza a nessuna zona valida della memoria

Puntatori nulli

- Un puntatore inizializzato punta a qualche posizione specifica della memoria
- Un puntatore non inizializzato punta ad una posizione aleatoria della memoria

```
int * p;
```

- E' importante assegnare sempre un valore ai puntatori
- Puntatore nullo: non indirizza a nessuna zona valida della memoria

```
int * p = nullptr;
```

Puntatori nulli

- Un puntatore inizializzato punta a qualche posizione specifica della memoria
- Un puntatore non inizializzato punta ad una posizione aleatoria della memoria

```
int * p;
```

- E' importante assegnare sempre un valore ai puntatori
- Puntatore nullo: non indirizza a nessuna zona valida della memoria

```
int * p = nullptr;
```

- Se si prova a dereferenziare un puntatore nullo si ottiene un errore a run-time


```
int V[10] = {5,11,20,17,8,4,9,13,5,12};  
int i = 5;  
cout << *V + *(V + i + 1) + 1;
```

```
int x = 5, y = 5;  
int* p1 = &x;  
int* p2 = &y;  
*p1 = *p2 + 1;  
cout << x << "□" << y << endl;
```

```
int a1 [3] = {2 ,7 ,8};  
int a2 [3] = {3 ,5 ,9};  
cout << *(a1 + 1) + *a2;
```