

# Fondamenti di Programmazione (A)

9 - Costrutti di controllo del flusso - Statement `do` – `while` e `for`

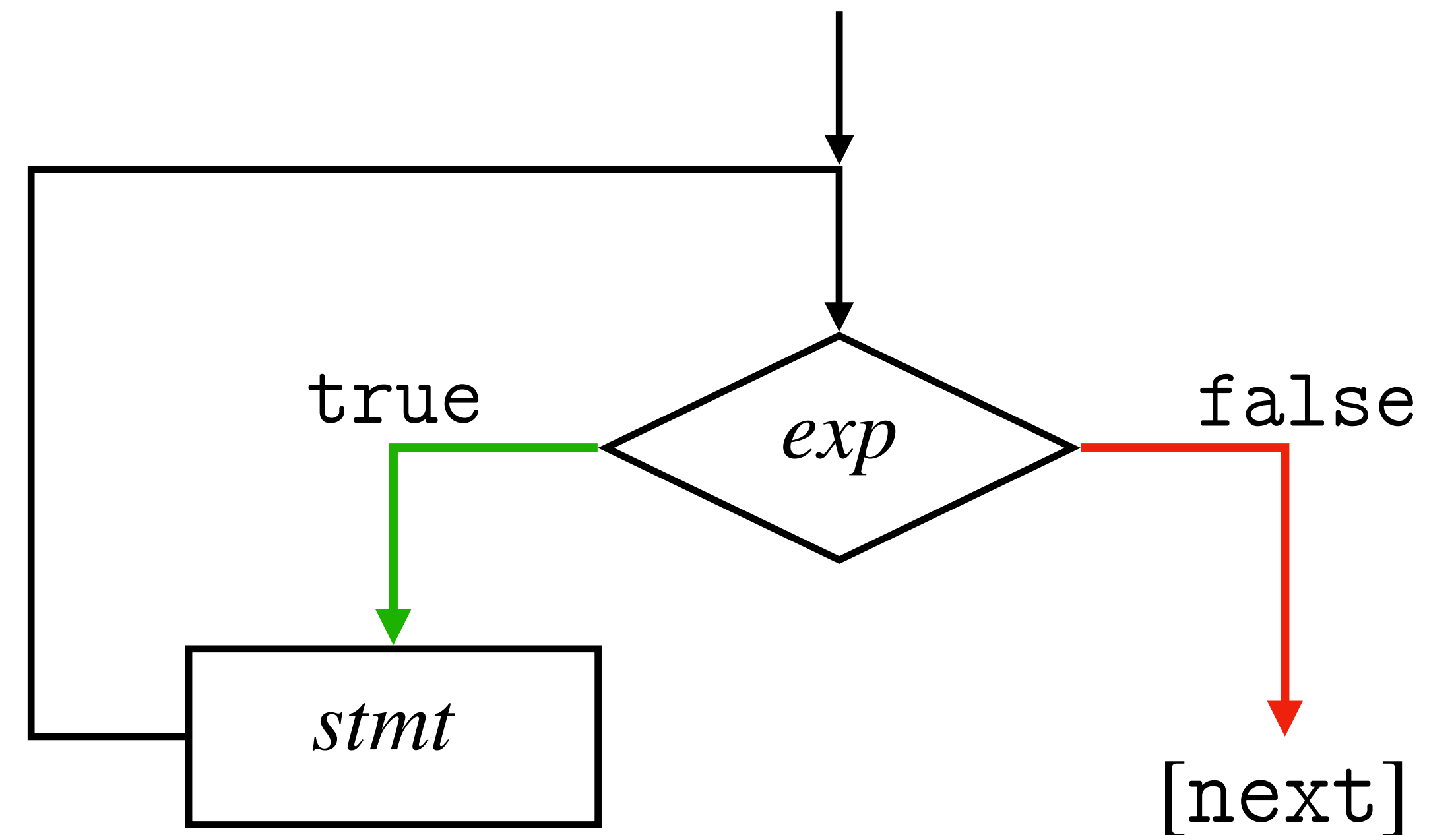
Vincenzo Arceri - Università degli Studi di Parma - [vincenzo.arceri@unipr.it](mailto:vincenzo.arceri@unipr.it)

# Puntate precedenti

- Assegnamenti e espressioni
- Statement `if` (comando di selezione)
- Statement `while` (comando iterativo)

# Ciclo `while`

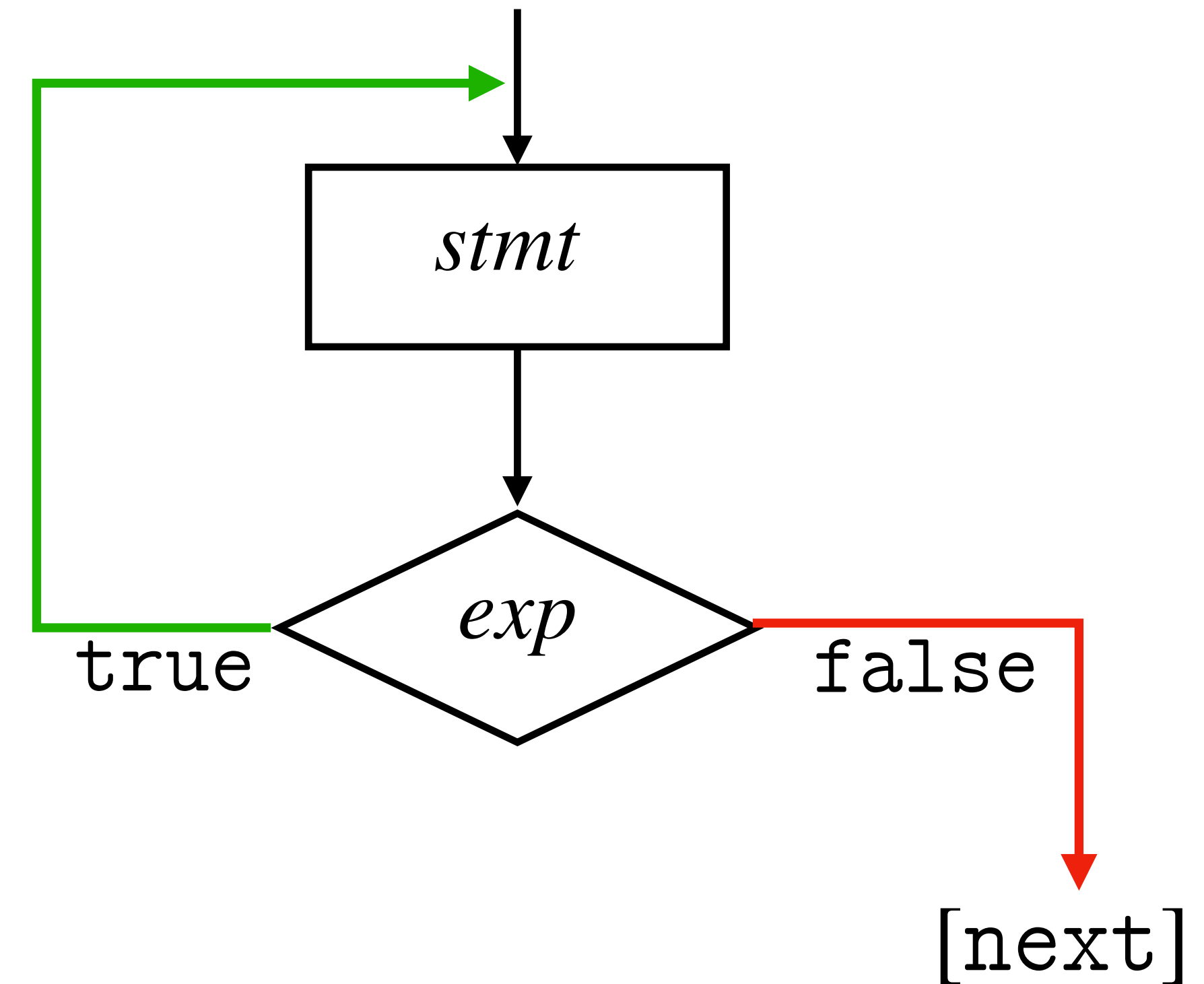
```
while (exp)  
    stmt  
[next]
```



**Prima** controllo la condizione dei ciclo, **poi** eseguo il corpo

# Ciclo do – while

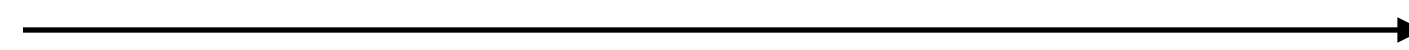
```
do  
    stmt  
while (exp) ;  
[next]
```



**Prima** eseguo il corpo del ciclo, **poi** controllo la condizione

# Ciclo do – while

*do*  
*stmt*  
*while* (*exp*) ;  
[next]



*stmt*  
*while* (*exp*)  
*stmt*  
[next]

**Prima** eseguo il corpo del ciclo, **poi** controllo la condizione

# Ciclo do — while

while vs do — while

# Ciclo do — while

while vs do — while

do

*stmt*

while (*exp*) ;

[next]

# Ciclo do – while

while vs do – while

```
do  
    stmt  
while (exp) ;  
[next]
```

*stmt* viene  
**sicuramente**  
eseguito almeno  
una volta



# Ciclo do – while

while vs do – while

```
do  
    stmt  
while (exp) ;  
[next]
```

*stmt* viene  
**sicuramente**  
eseguito almeno  
una volta

```
while (exp)  
    stmt  
[next]
```

# Ciclo do – while

while vs do – while

```
do  
    stmt  
while (exp) ;  
[next]
```

*stmt* viene  
**sicuramente**  
eseguito almeno  
una volta

```
while (exp)  
    stmt  
[next]
```

*stmt* potrebbe  
non essere mai  
eseguito

# Ciclo do – while

Esercizio (da lecture 8)

- **Problema:** preso in input un intero positivo  $n$ , stampare un triangolo rettangolo di asterischi di lato  $n$
- Il programma deve controllare che l'input inserito dall'utente sia positivo
- Il programma continua a chiedere l'input finché non inserisce un intero positivo

# Ciclo do – while

## Esercizio

- **Problema:** Scrivere un programma che legga da tastiera un numero intero fra 0 e 3, dove:
  - ♦ 0: addizione
  - ♦ 1: sottrazione
  - ♦ 2: moltiplicazione
  - ♦ 3: divisione

Una volta letta l'operazione, il programma deve leggere due numeri interi e stampare a video il risultato dell'operazione desiderata

Dopodiché, il programma deve di nuovo chiedere un intero fra 0 e 3 per effettuare nuove operazioni: per uscire dal programma, inserire -1

# Statement break

- Lo statement `break` è utilizzato per terminare un ciclo
- Quando viene raggiunto un comando `break`, l'esecuzione del ciclo viene interrotta e il controllo passa all'istruzione che segue dopo il ciclo che lo contiene

```
int i = 0;
while (i < 10) {
    if (i == 4)
        break;
    i++;
}

cout << i << endl;
```

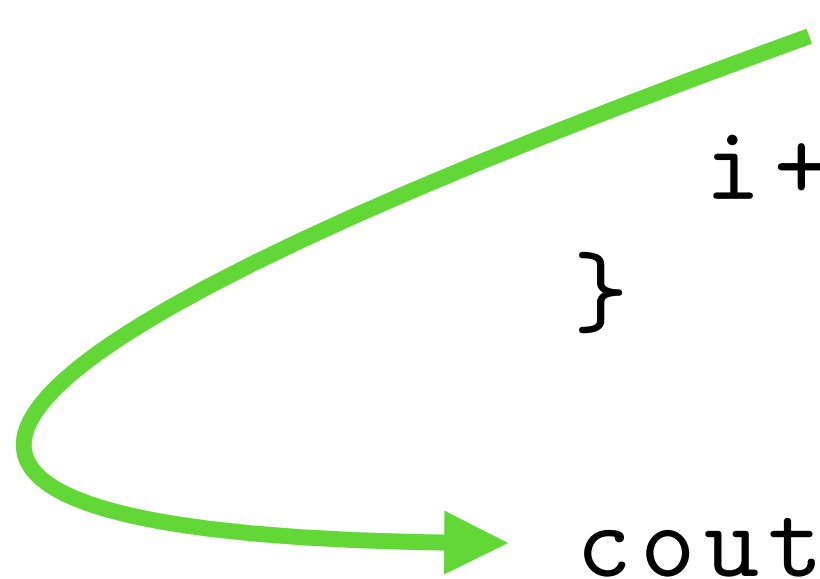
# Statement break

- Lo statement `break` è utilizzato per terminare un ciclo
- Quando viene raggiunto un comando `break`, l'esecuzione del ciclo viene interrotta e il controllo passa all'istruzione che segue dopo il ciclo che lo contiene

```
int i = 0;
while (i < 10) {
    if (i == 4)
        break;
    i++;
}
```

salto incondizionato

```
cout << i << endl;    stampa 4
```

A green curved arrow originates from the `break;` statement inside the `while` loop and points to the `cout << i << endl;` statement, illustrating an unconditional jump out of the loop.

# Statement `continue`

- Lo statement `continue` è utilizzato per interrompere **un'iterazione** di un ciclo
- Quando viene raggiunto un comando `continue`, l'esecuzione dell'iterazione viene interrotta e il controllo passa alla condizione del ciclo

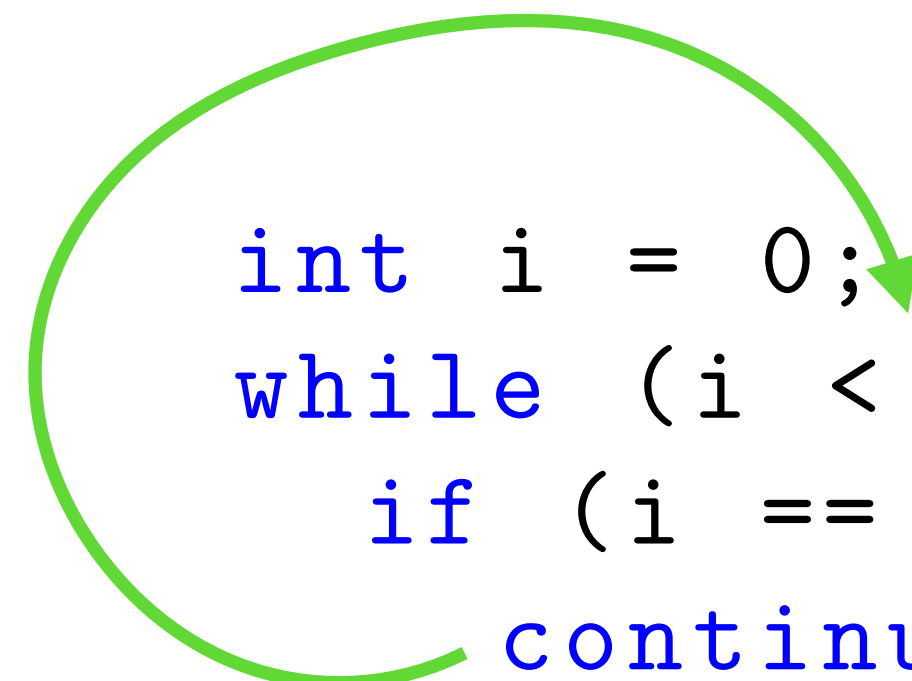
```
int i = 0;
while (i < 10) {
    if (i == 4)
        continue;
    i++;
}

cout << i << endl;
```

# Statement `continue`

- Lo statement `continue` è utilizzato per interrompere **un'iterazione** di un ciclo
- Quando viene raggiunto un comando `continue`, l'esecuzione dell'iterazione viene interrotta e il controllo passa alla condizione del ciclo

salto incondizionato



```
int i = 0;
while (i < 10) {
    if (i == 4)
        continue;
    i++;
}

cout << i << endl;
```

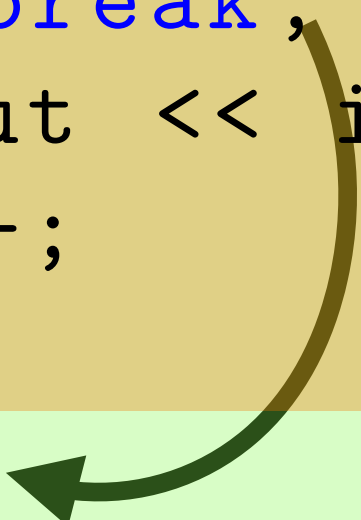


# Statement break e continue

- break e continue agiscono sul blocco che li contiene

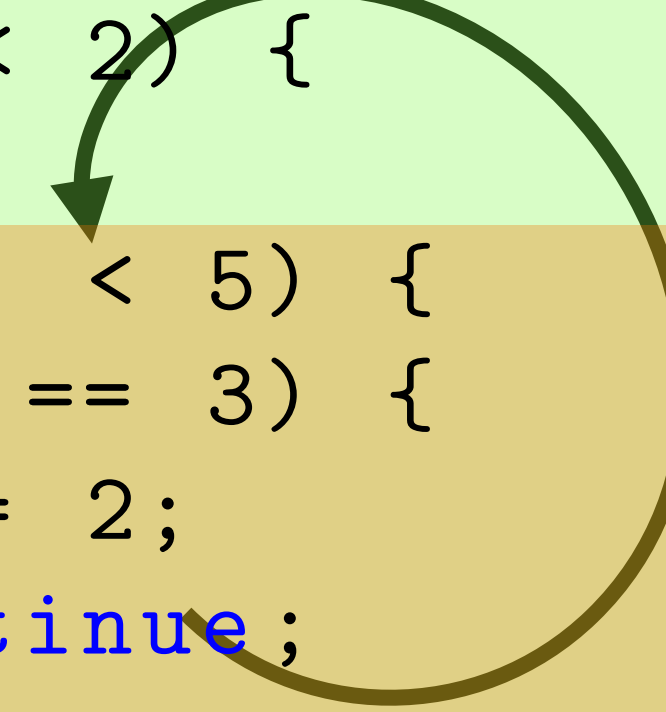
```
int i = 0;  
int j = 0;
```

```
while (i < 2) {  
    j = 0;  
    while (j < 5) {  
        if (j == 2)  
            break;  
        cout << i << j << endl;  
        j++;  
    }  
    i++;  
}
```



```
int i = 0;  
int j = 0;
```

```
while (i < 2) {  
    j = 0;  
    while (j < 5) {  
        if (j == 3) {  
            j += 2;  
            continue;  
        }  
        cout << i << j << endl;  
        j++;  
    }  
    i++;  
}
```



# Statement break e continue

Esercizio: determinare l'output dei seguenti programmi

```
int i = 0;
int j = 0;

while (i < 2) {
    j = 0;
    while (j < 5) {
        if (j == 2)
            break;
        cout << i << j << endl;
        j++;
    }
    i++;
}
```

```
int i = 0;
int j = 0;

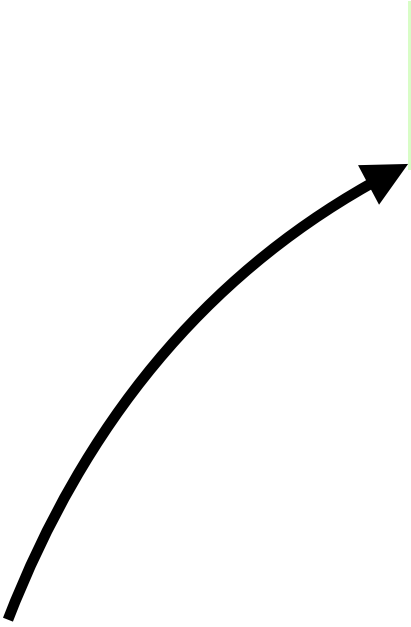
while (i < 2) {
    j = 0;
    while (j < 5) {
        if (j == 3) {
            j += 2;
            continue;
        }
        cout << i << j << endl;
        j++;
    }
    i++;
}
```

# Statement `for`

## Ciclo *limitato*

- Ciclo utilizzato per ripetere una certa operazione un numero specificato di volte

`for` ( $exp_1$ ;  $exp_2$ ;  $exp_3$ )  
*stmt*



corpo del ciclo

# Statement `for`

Ciclo *limitato*

- Ciclo utilizzato per ripetere una certa operazione un numero specificato di volte

```
for (exp1 ; exp2 ; exp3)  
    stmt
```

espressione booleana



# Statement `for`

Ciclo *limitato*

- Ciclo utilizzato per ripetere una certa operazione un numero specificato di volte

```
for (exp1 ; exp2 ; exp3)  
    stmt
```

espressione che viene eseguita **al termine di ogni iterazione**



# Statement `for`

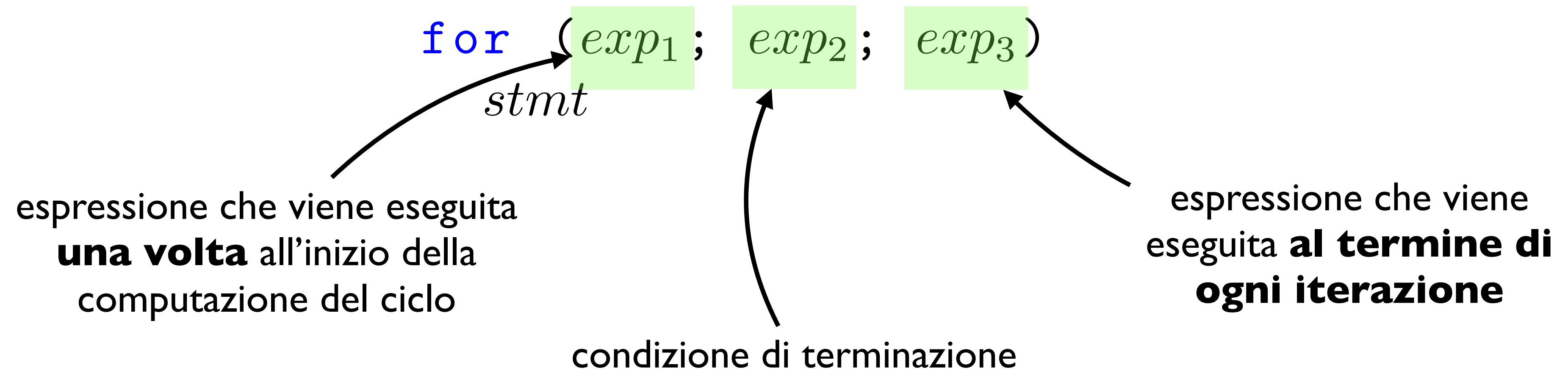
## Ciclo *limitato*

- Ciclo utilizzato per ripetere una certa operazione un numero specificato di volte

```
for (exp1; exp2; exp3)  
    stmt
```

espressione che viene eseguita **una volta** all'inizio della computazione del ciclo

# Statement for



```
{  
    exp1 ;  
    while (exp2) {  
        stmt  
        exp3 ;  
    }  
}
```

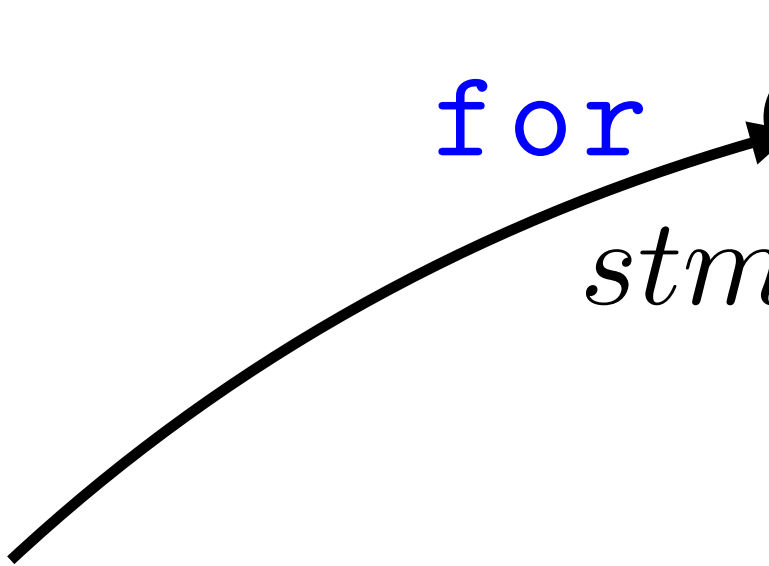
# Statement `for`

## Classico utilizzo

- Eseguire *stmt* per un intervallo di valori che va da  $c_0$  a  $c_1$

`for` *(i* =  $c_0$  ;  $i \leq c_1$  ;  $i++$ )

*stmt*



La variabile *i* è detta

- variabile di controllo del ciclo
- indice
- contatore



# Statement `for`

## Classico utilizzo

- Eseguire *stmt* per un intervallo di valori che vanno da  $c_0$  a  $c_1$

```
for (i = c0; i <= c1; i++)  
    stmt
```

Valore d'inizializzazione dell'indice



# Statement `for`

## Classico utilizzo

- Eseguire *stmt* per un intervallo di valori che vanno da  $c_0$  a  $c_1$

```
for (i = c0; i <= c1; i++)  
    stmt
```

Condizione d'entrata nel corpo del  
ciclo



# Statement `for`

## Classico utilizzo

- Eseguire *stmt* per un intervallo di valori che vanno da  $c_0$  a  $c_1$

```
for (i = c0; i <= c1; i++)  
stmt
```

Valore finale di *i*




# Statement `for`

## Classico utilizzo

- Eseguire *stmt* per un intervallo di valori che vanno da  $c_0$  a  $c_1$

```
for (i = c0 ; i <= c1 ; i++)  
    stmt
```

Passo del ciclo  
(in questo caso, 1)



# Statement `for`

## Classico utilizzo

- Eseguire *stmt* per un intervallo di valori che vanno da  $c_0$  a  $c_1$

```
for (i = c0; i <= c1; i++)  
    stmt
```

"Per  $i$  che va da  $c_0$  a  $c_1$ , con passo  $l$ , esegui *stmt*"

# Statement for

## Classico utilizzo - Esercizio

- **Problema:** stampare i primi 20 numeri positivi

.

# Statement for

## Classico utilizzo - Esercizio

- **Problema:** stampare la tabellina del 5 fino a 100

# Statement `for`

## Note sul classico utilizzo

- La variabile di controllo può essere di un qualsiasi tipo compatibile con `int`

`char bool unsigned int ...`

- **Problema:** stampare l'alfabeto (minuscolo)

.



# Statement for

## Classico utilizzo - Esercizio

- **Problema:** stampare i primi 20 numeri pari in ordine decrescente

# Statement `for`

## Esercizio

- *stmt* è uno statement qualsiasi, quindi anche **blocchi** e **altri for statement**
- **Problema:** dato in input un intero positivo  $n$ , stampare a video una matrice triangolare inferiore fatta di uni

$n = 1$

```
|
```

$n = 4$

```
| 0 0 0
| | 0 0
| | | 0
| | | |
```

$n = 5$

```
| 0 0 0 0
| | 0 0 0
| | | 0 0
| | | | 0
| | | | |
```

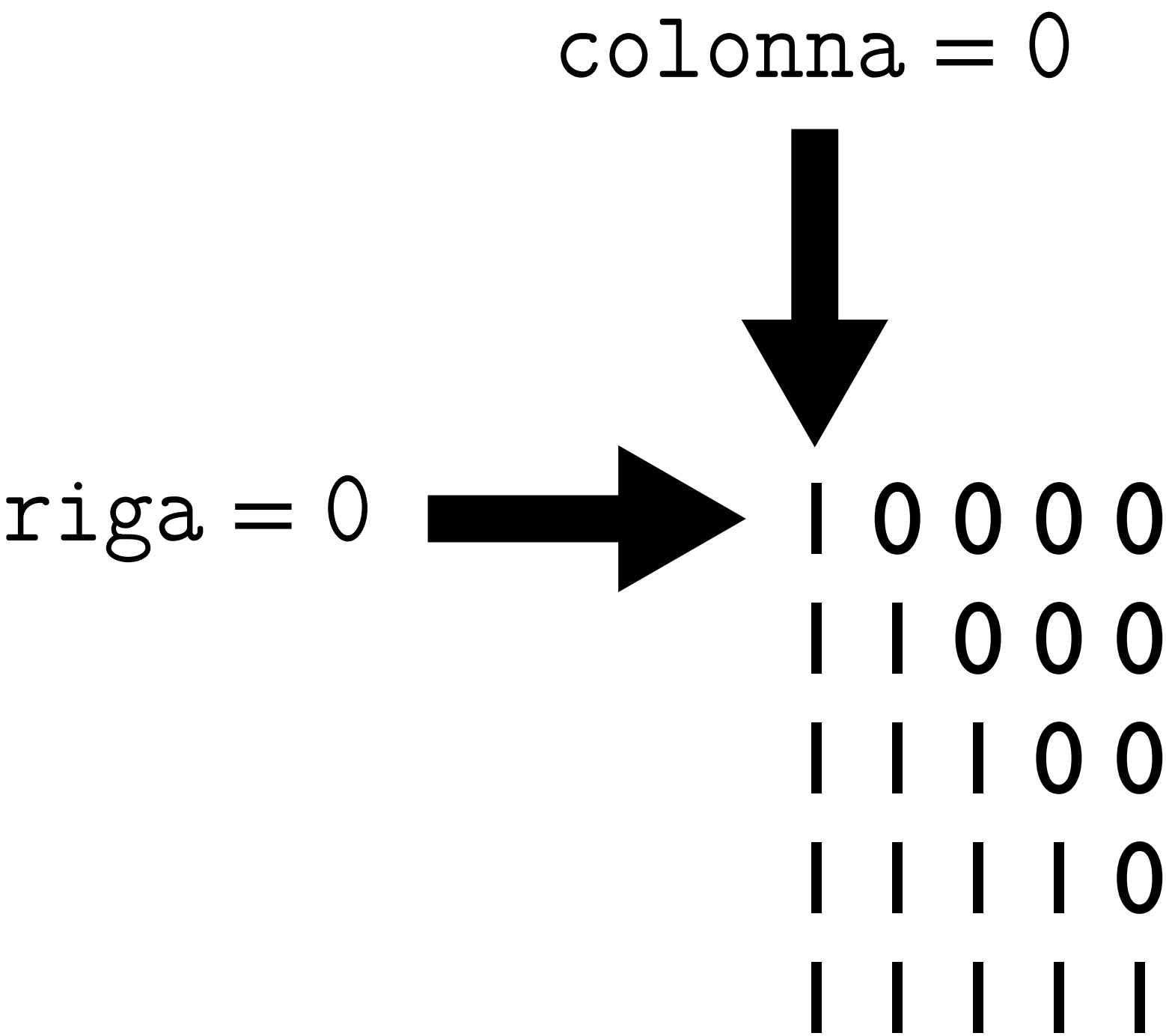
# Statement for

Esercizio (per  $n = 5$ )

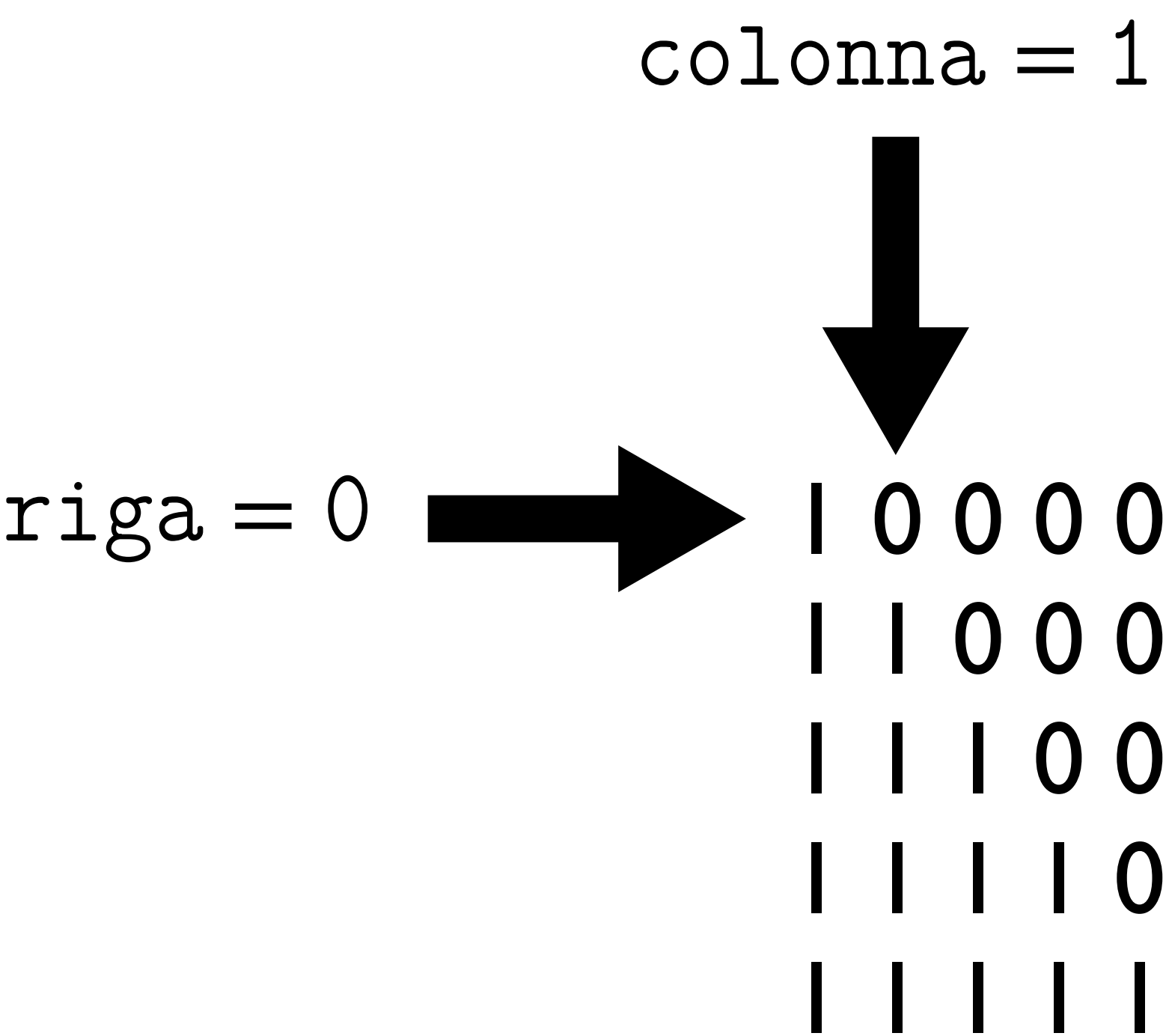
```
| 0 0 0 0
| | 0 0 0
| | | 0 0
| | | | 0
| | | | |
```

- **Idea**
  - stampo una **riga** alla volta
  - uso due indici:
    - `riga` tiene traccia di quale riga sto stampando
    - `colonna` tiene traccia di quale posizione sulla riga devo stampare

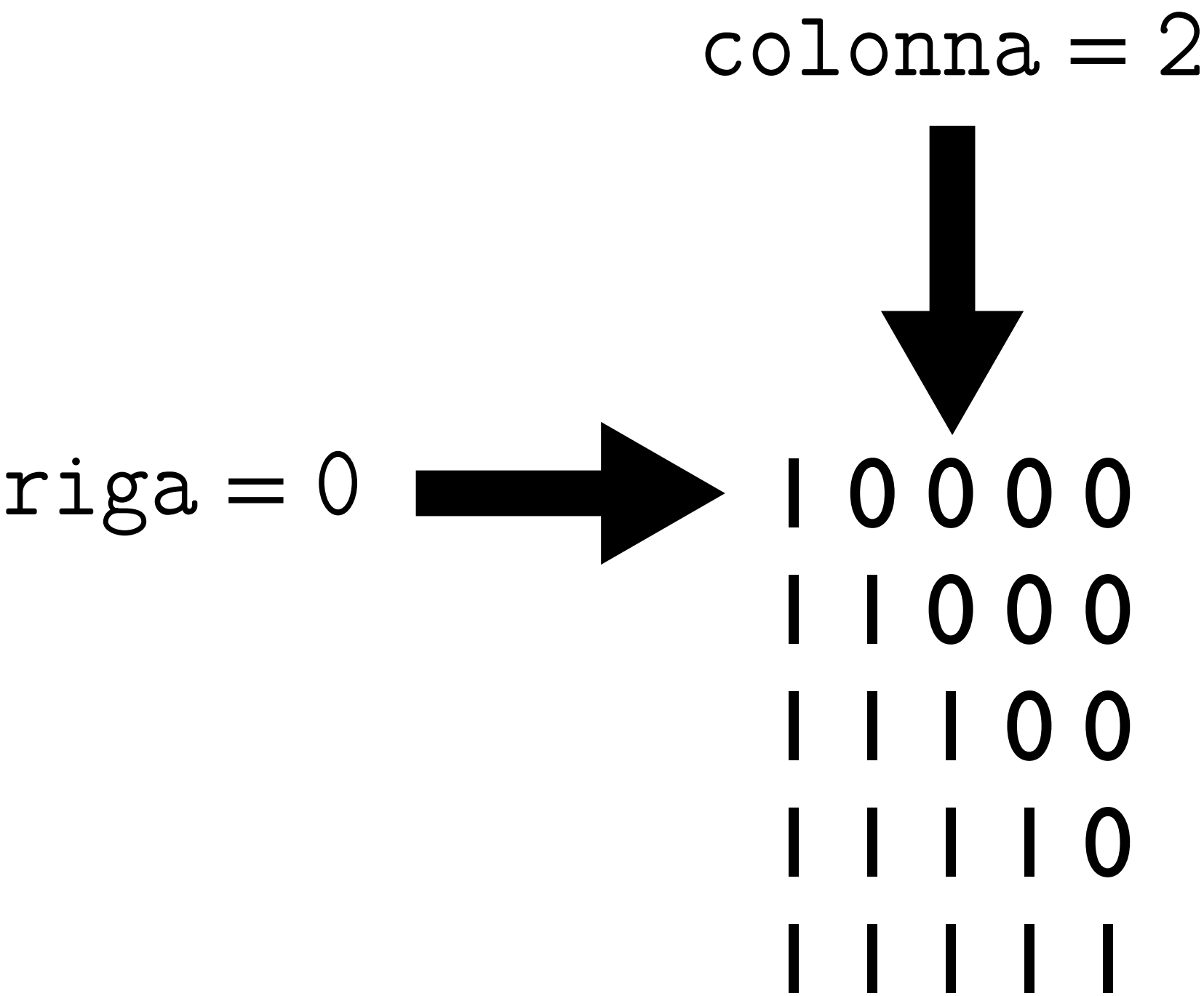
# Statement for Esercizio (per $n = 5$ )



# Statement for Esercizio (per $n = 5$ )

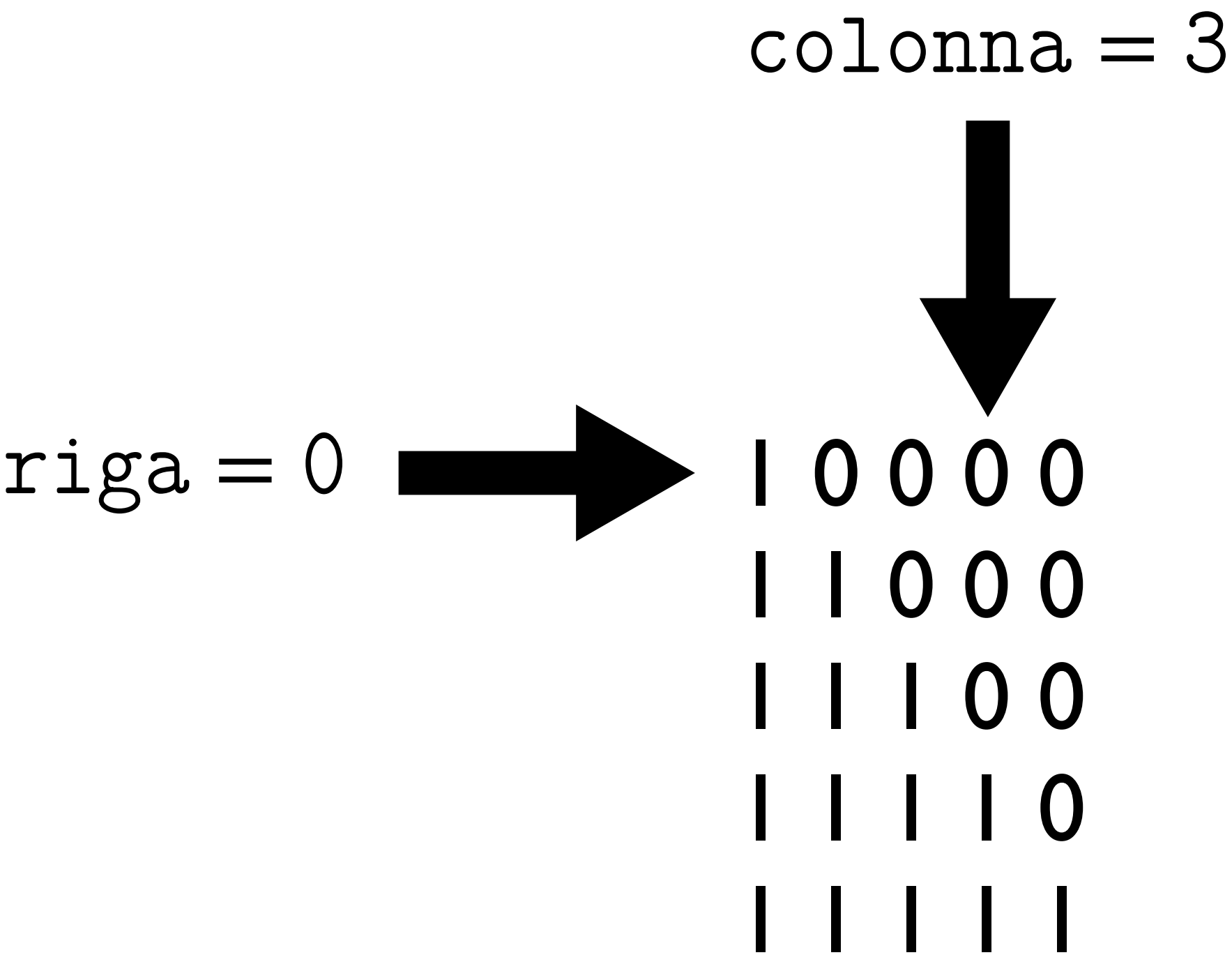


# Statement for Esercizio (per $n = 5$ )

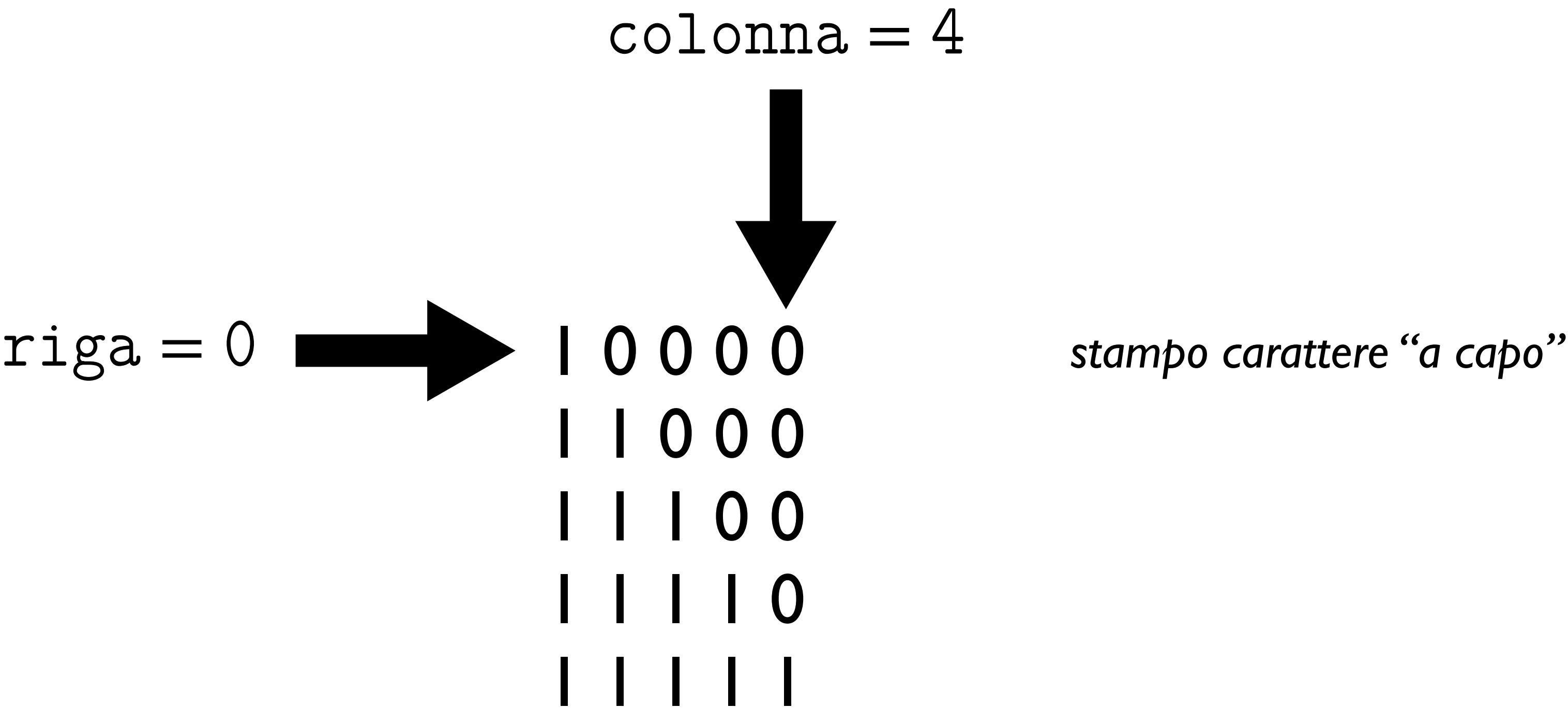


# Statement for

Esercizio (per  $n = 5$ )

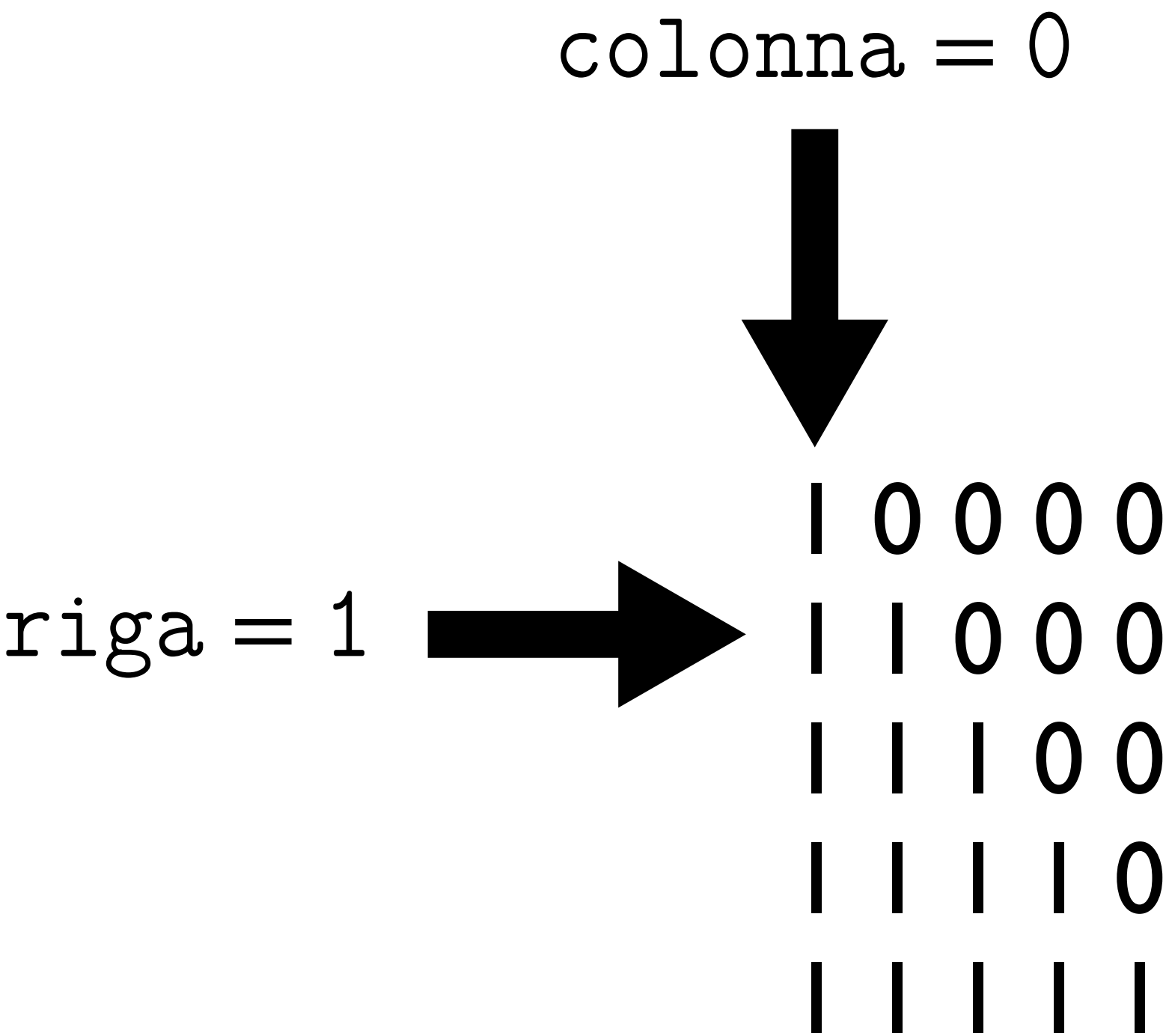


# Statement for Esercizio (per $n = 5$ )



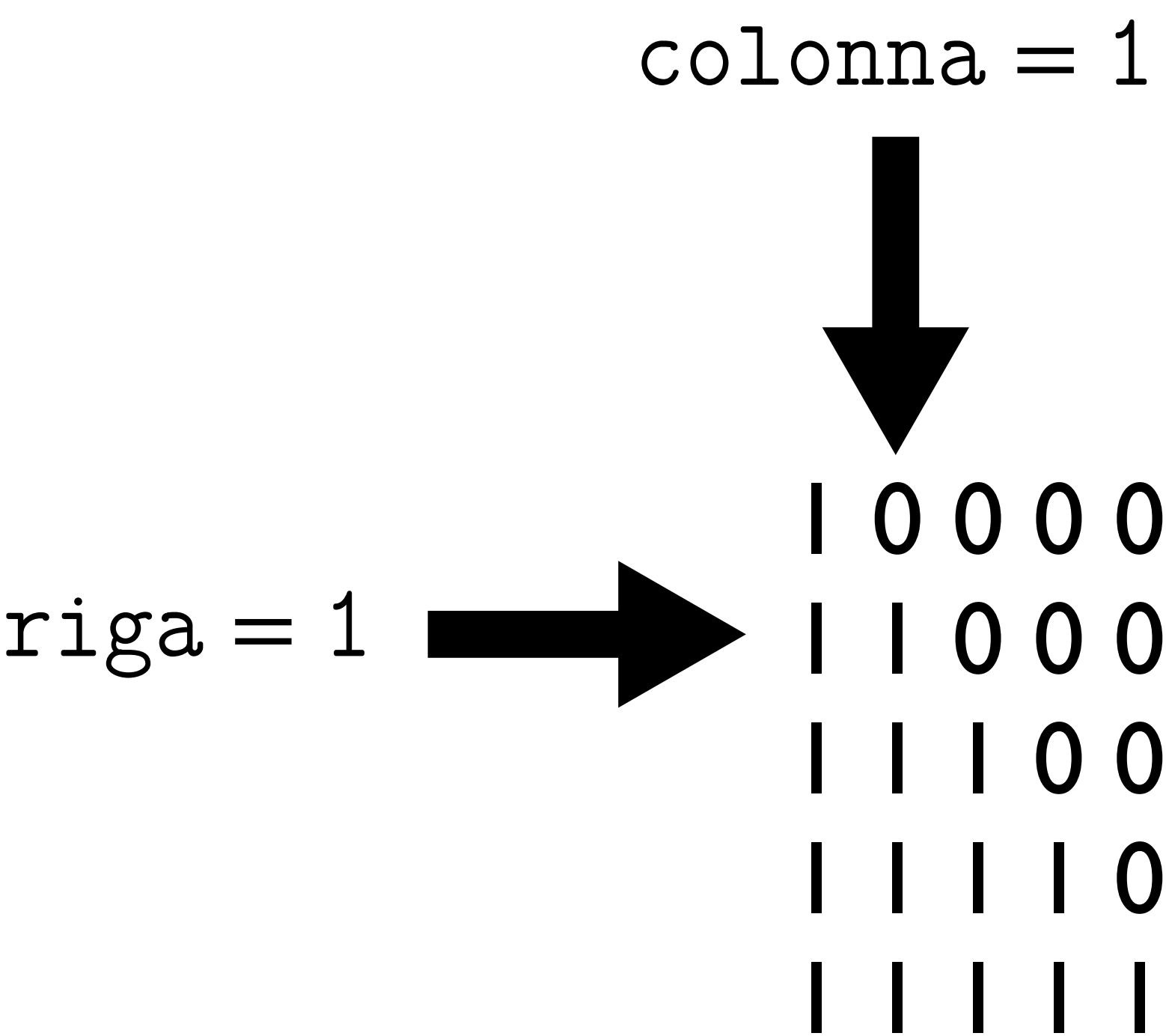


# Statement for Esercizio (per $n = 5$ )

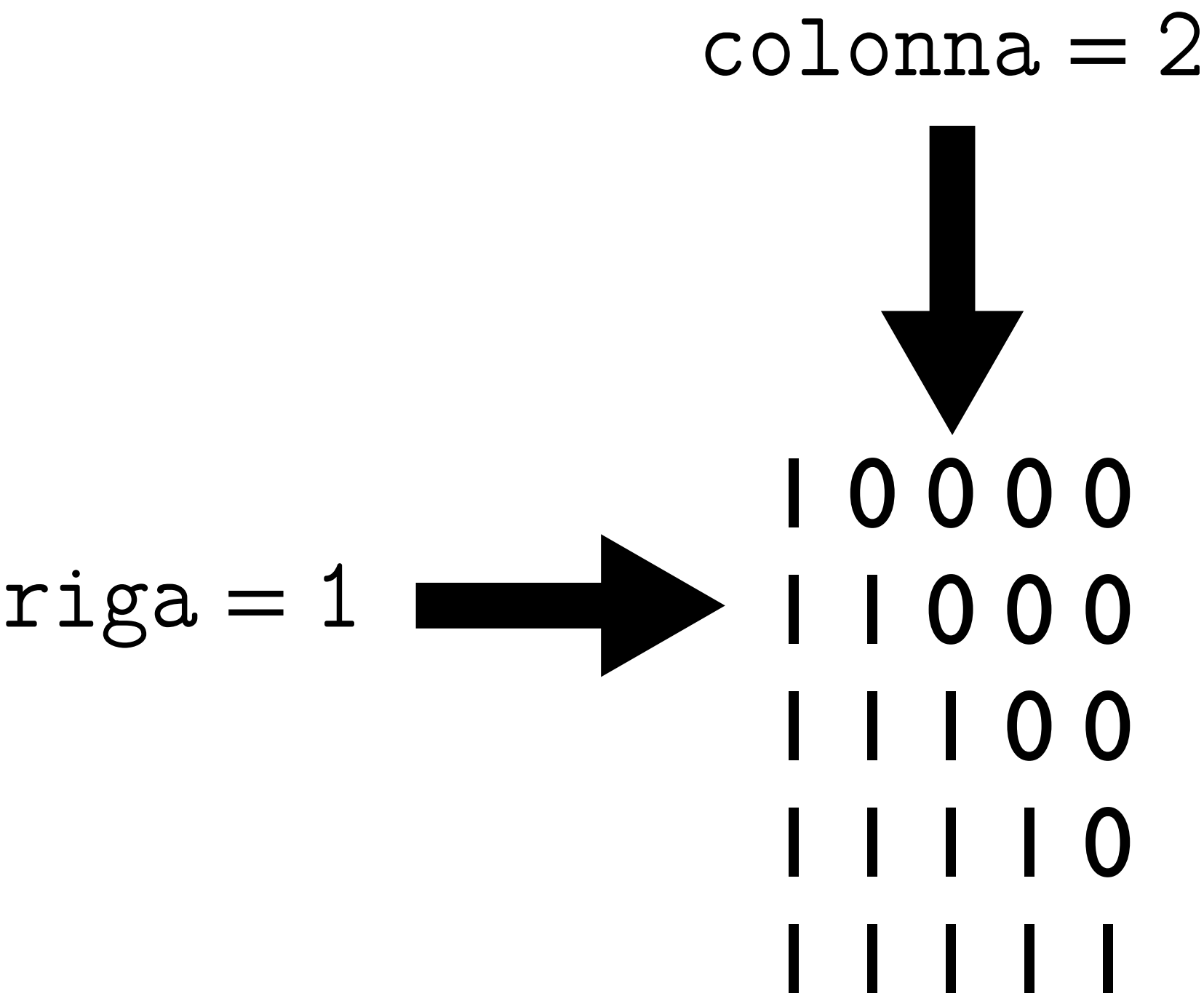


*stampa carattere “a capo”*

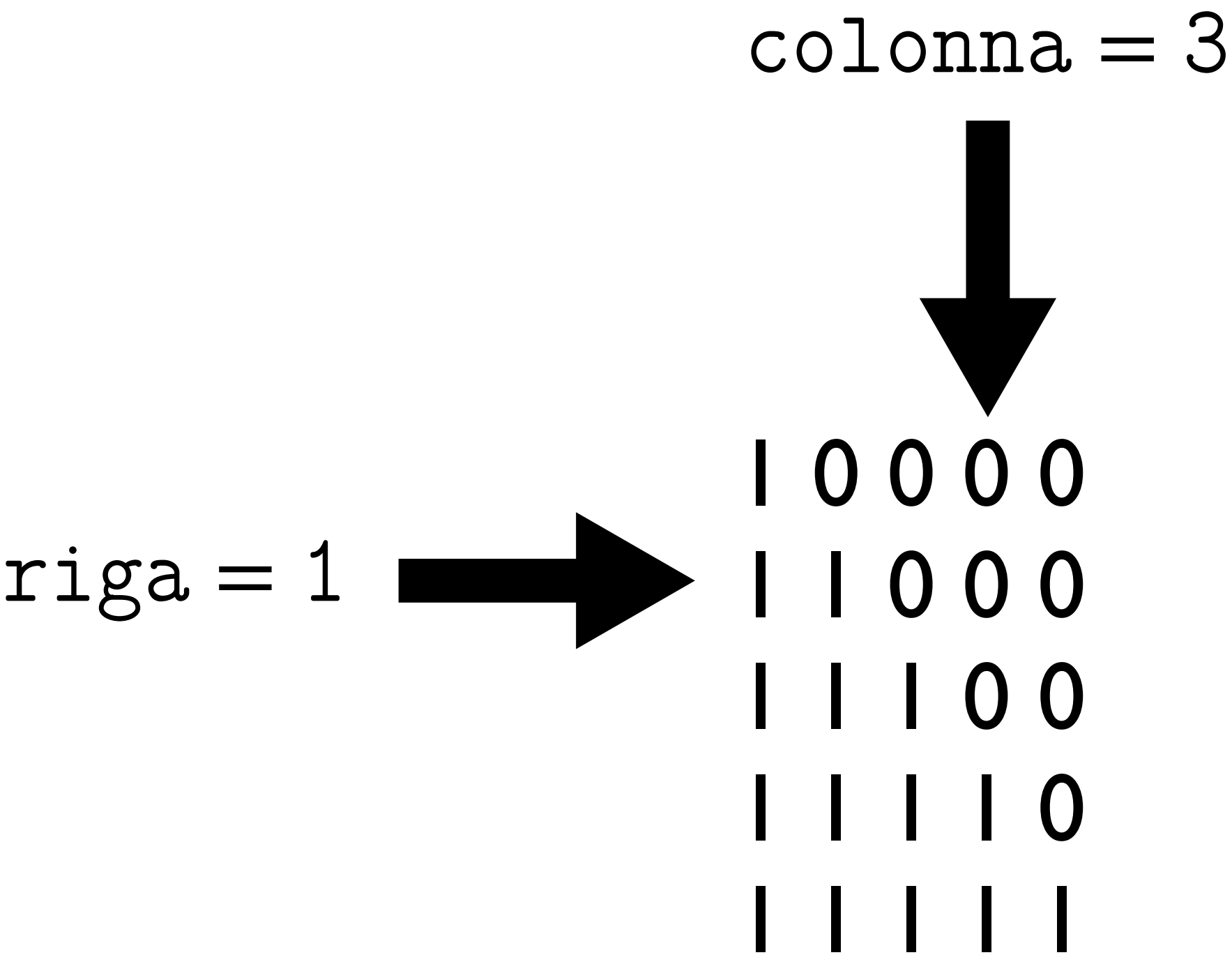
# Statement for Esercizio (per $n = 5$ )



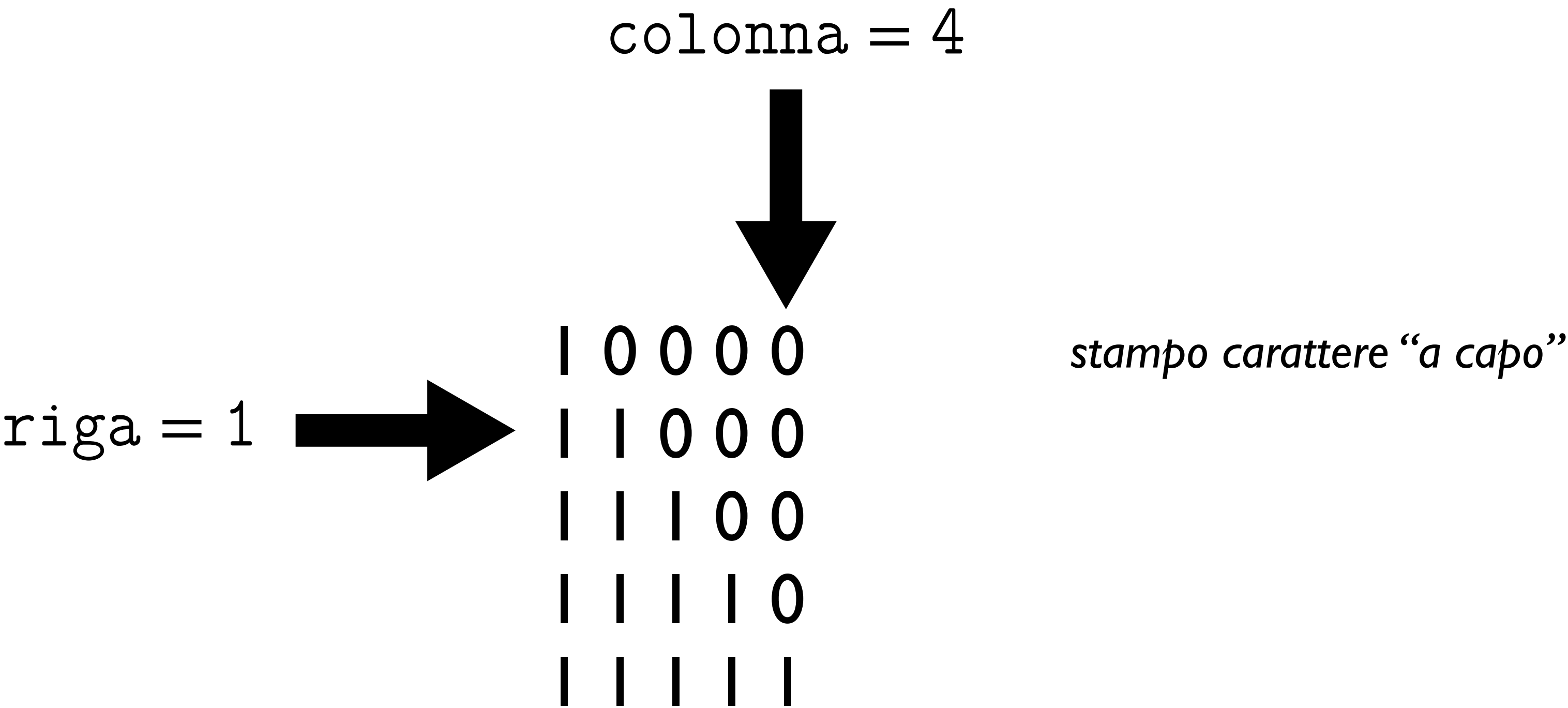
# Statement for Esercizio (per $n = 5$ )



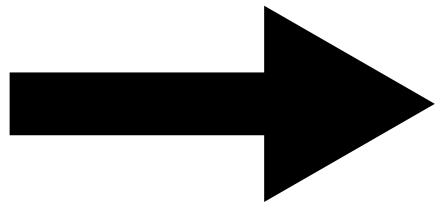
# Statement for Esercizio (per $n = 5$ )



# Statement for Esercizio (per $n = 5$ )



# Statement for Esercizio (per $n = 5$ )

riga = 2 

	0	0	0	0
		0	0	0
			0	0
				0

Quando stampare zero? Quando stampare uno? Quando andare a capo?

# Statement for

Esercizio (per  $n = 5$ )

	0	1	2	3	4
0		0	0	0	0
1			0	0	0
2				0	0
3					0
4					

# Statement `for`

Definizione *pura* dell'iterazione determinata (non quella implementata da C++)

`for` i = *inizio* to *fine* by *passo* do  
    *stmt*

variabile di controllo

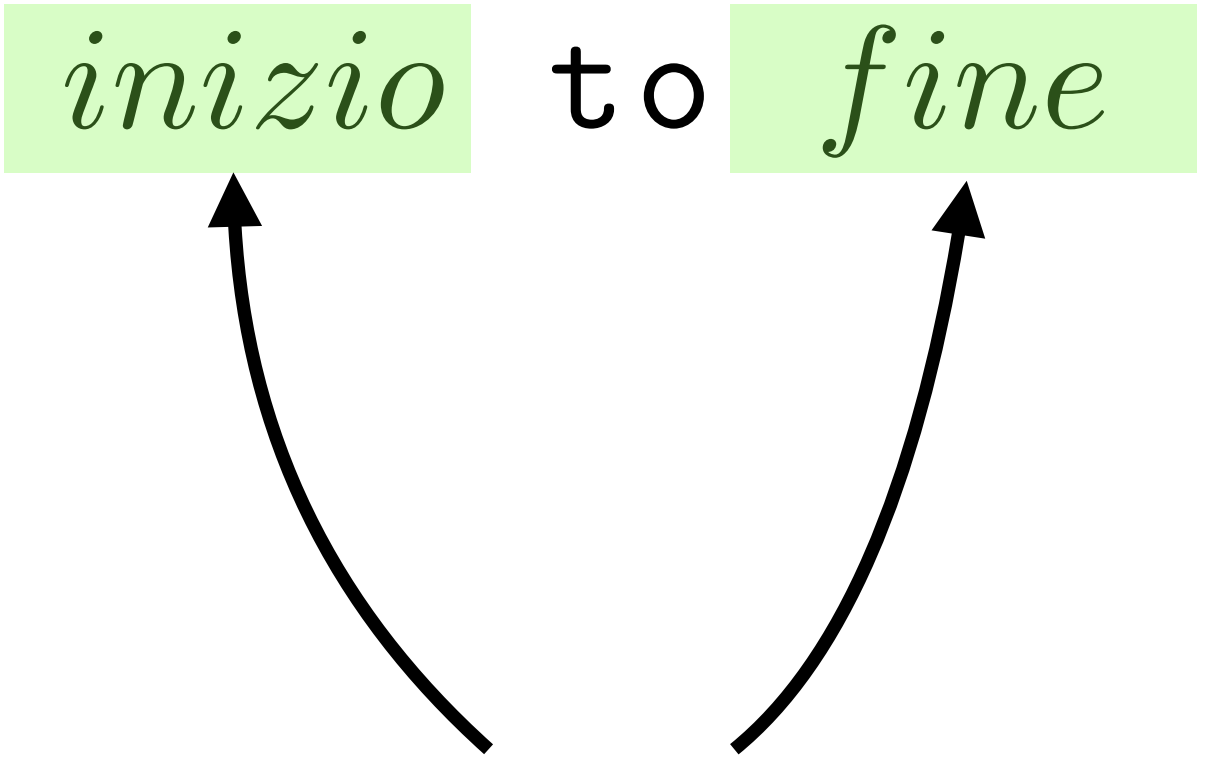




# Statement `for`

Definizione *pura* dell'iterazione determinata (non quella implementata da C++)

`for`  $i =$  *inizio* `to` *fine* `by` *passo* `do`  
*stmt*



espressioni

# Statement `for`

Definizione *pura* dell'iterazione determinata (non quella implementata da C++)

`for`  $i = \textit{inizio}$  `to`  $\textit{fine}$  `by` *passo* `do`  
 $\textit{stmt}$

costante

(senza perdita di generalità, supponiamo *passo* costante  $> 0$ )

# Statement `for`

Definizione *pura* dell'iterazione determinata (non quella implementata da C++)

`for` `i` = *inizio* `to` *fine* `by` *passo* `do`

*stmt*

corpo del ciclo

**che non può modificare/  
aggiornare il valore della  
variabile di controllo!**

# Statement `for`

Definizione *pura* dell'iterazione determinata (non quella implementata da C++)

`for`  $i =$  *inizio* `to` *fine* `by` *passo* `do`  
*stmt*

Prima dell'esecuzione del  
ciclo `for`, i valori di  
inizio e fine del ciclo  
vengono **congelati**

- Nella definizione pura dell'iterazione determinata il numero di iterazioni è stabilito **prima** dell'esecuzione del ciclo!
- Non è possibile creare cicli infiniti

# Statement `for`

Definizione *pura* dell'iterazione determinata (non quella implementata da C++)

`for`  $i =$  *inizio* `to` *fine* `by` *passo* `do`  
*stmt*

Prima dell'esecuzione del  
ciclo `for`, i valori di  
inizio e fine del ciclo  
vengono **congelati**

Iteration count è  $ic = \left\lfloor \frac{fine - inizio + passo}{passo} \right\rfloor$  se questa quantità è positiva, altrimenti 0

# Statement `for`

## In C++

espressione che viene eseguita  
**una volta** all'inizio della  
computazione del ciclo

Opzionale

`for` (*exp*<sub>1</sub> ; *exp*<sub>2</sub> ; *exp*<sub>3</sub> )  
*stmt*

condizione di terminazione

Opzionale

espressione che viene  
eseguita **al termine di**  
**ogni iterazione**

Opzionale

# Statement `for`

## In C++

espressione che viene eseguita  
**una volta** all'inizio della  
computazione del ciclo

Opzionale

```
for ( exp1 ; exp2 ; exp3 )  
    stmt
```

nessun vincolo su *stmt*  
(e.g., può modificare la  
variabile di controllo)

condizione di terminazione

Opzionale

espressione che viene  
eseguita **al termine di  
ogni iterazione**

Opzionale

# Statement for

## In C++

```
int i = 0;
for ( ; i < 5; ) {
    cout << i << ' ';
    i++;
}
```

equivalente a

```
int i = 0;
while (i < 5) {
    cout << i << ' ';
    i++;
}
```

---

```
int i = 0;
for ( ; ; ) {
    cout << i << ' ';
    i++;
}
```

non termina!



# Statement `for`

In C++

```
for (int i = 0; i < 5; i++)  
    cout << i * i << endl;
```

- C++ permette di dichiarare e inizializzare la variabile di controllo (nell'esempio `i`), **ma sarà visibile solamente dentro il ciclo `for`!**

# Statement for

## break e continue

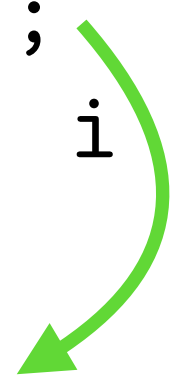
```
int i;  
for (i = 0; i < 5; i++) {  
    if (i == 3)  
        break;  
    cout << i << endl;  
}  
return 0;
```

```
int i;  
for (i = 0; i < 5; i++) {  
    if (i == 3)  
        continue;  
    cout << i << endl;  
}  
return 0;
```

# Statement for

## break e continue

```
int i;  
for (i = 0; i < 5; i++) {  
    if (i == 3)  
        break;  
    cout << i << endl;  
}  
return 0;
```



Output

0

1

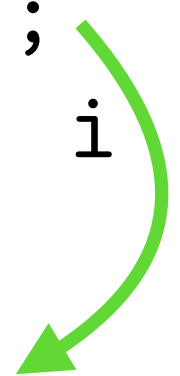
2

```
int i;  
for (i = 0; i < 5; i++) {  
    if (i == 3)  
        continue;  
    cout << i << endl;  
}  
return 0;
```

# Statement for

## break e continue


```
int i;  
for (i = 0; i < 5; i++) {  
    if (i == 3)  
        break;  
    cout << i << endl;  
}  
return 0;
```



Output

0  
1  
2

```
int i;  
for (i = 0; i < 5; i++) {  
    if (i == 3)  
        continue;  
    cout << i << endl;  
}  
return 0;
```



Output

0  
1  
2  
4

# Programmazione strutturata

- Anni 70/80: metodologia di programmazione con lo scopo di rendere la struttura dei programmi più facile da capire e mantenere
- Convenzioni sul formato del programma (indentazione del codice)
- Commenti e convenzioni lessicali (es.: nomi significativi per le variabili)
- Strutture di controllo (I-entry, I-exit):
  - Struttura sequenziale (concatenazione sequenziali di istruzioni)
  - Struttura di selezione/strutture condizionali (`if`)
  - Struttura iterativa/ciclica (`while`)