

# Fondamenti di programmazione

---

## 4 | Programmazione Strutturata in C

# SOMMARIO

- Pseudocodice
- Diagrammi di flusso
- Strutture di controllo essenziali
- Strutture di controllo in C

# Strumenti di supporto alla programmazione

Prima di iniziare a scrivere un programma, è necessario **sapere cosa scrivere!**

Bisogna **capire il problema** e progettare una possibile soluzione.

Una soluzione, in genere, consiste di una sequenza **ordinata** e ben precisa di azioni (**algoritmo**).

Per la **progettazione** di un algoritmo/soluzione, vengono utilizzati strumenti di supporto quali:

- **Pseudocodice**
- **Diagrammi di flusso**

# Pseudocodice

Lo **pseudocodice** è un *linguaggio formale* che permette di definire un algoritmo.

Lo pseudocodice è **indipendente** da qualsiasi linguaggio di programmazione.

La traduzione di un algoritmo definito in pseudocodice in uno specifico linguaggio di programmazione è in genere semplice.

Inoltre:

- E' molto simile al linguaggio parlato
- Lo pseudocodice **non è direttamente eseguibile dal calcolatore**

# Diagrammi di flusso

I **diagrammi di flusso** rappresentano un ulteriore strumento per la descrizione e definizione di un algoritmo.

Differentemente dallo pseudocodice, i diagrammi di flusso sono una **rappresentazione grafica dell'algoritmo**.

Gli elementi alla base dei diagrammi di flusso sono 4:



Rettangolo

Rombo

Parallelogramma

Cerchio

# Diagrammi di flusso

I **diagrammi di flusso** rappresentano un ulteriore strumento per la descrizione e definizione di un algoritmo.

Differentemente dallo pseudocodice, i diagrammi di flusso sono una **rappresentazione grafica dell'algoritmo**.

Gli elementi alla base dei diagrammi di flusso sono 4:



**Rettangolo**

**Azioni  
generiche**

**Rombo**

**Condizioni**

**Parallelogramma**

**Azioni  
I/O**

**Cerchio**

**Start/End**

# Strutture di controllo

Le istruzioni di un programma scritto in linguaggio C vengono eseguite **in sequenza**, una dopo l'altra.

In genere, un programma reale, ha la necessità di modificare il flusso di esecuzione delle istruzioni.

Le **strutture di controllo** servono, appunto, per modificare l'ordine di esecuzione delle istruzioni, secondo determinate regole.

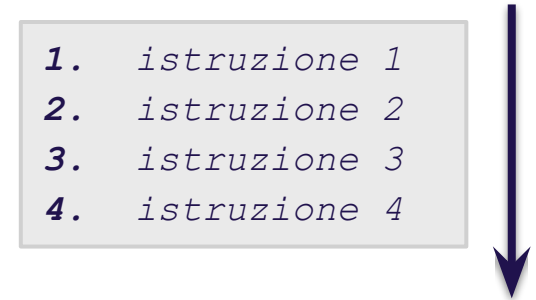
Si dimostra (**Teorema di Böhm-Jacopini**) che **qualunque** algoritmo può essere implementato utilizzando solamente **3 tipi** di strutture di controllo:

- Sequenza
- Selezione
- Iterazione

# Strutture di controllo

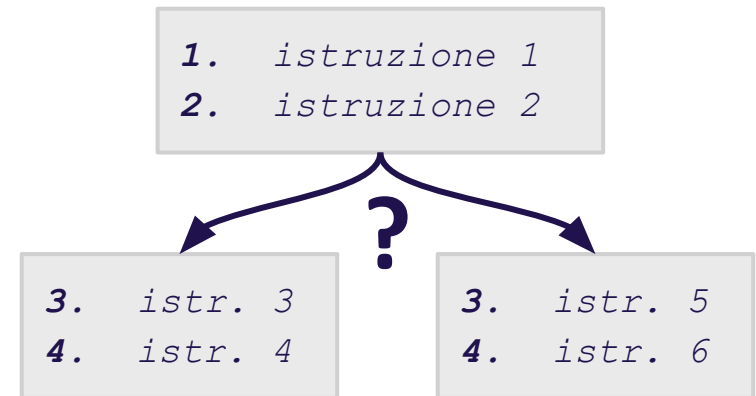
## Sequenza

Esecuzione sequenziale delle istruzioni. Le istruzioni vengono eseguite una di seguito all'altra nell'ordine in cui sono state scritte.



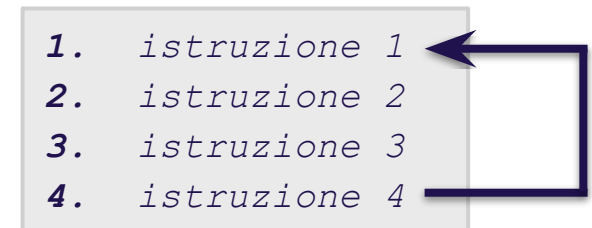
## Selezione

Scelta fra due **percorsi** che viene effettuata sulla base di una condizione (che può essere **vera** o **falsa**).



## Iterazione

**Ripetizione** di un blocco di istruzioni fino a quando non si verifica una condizione



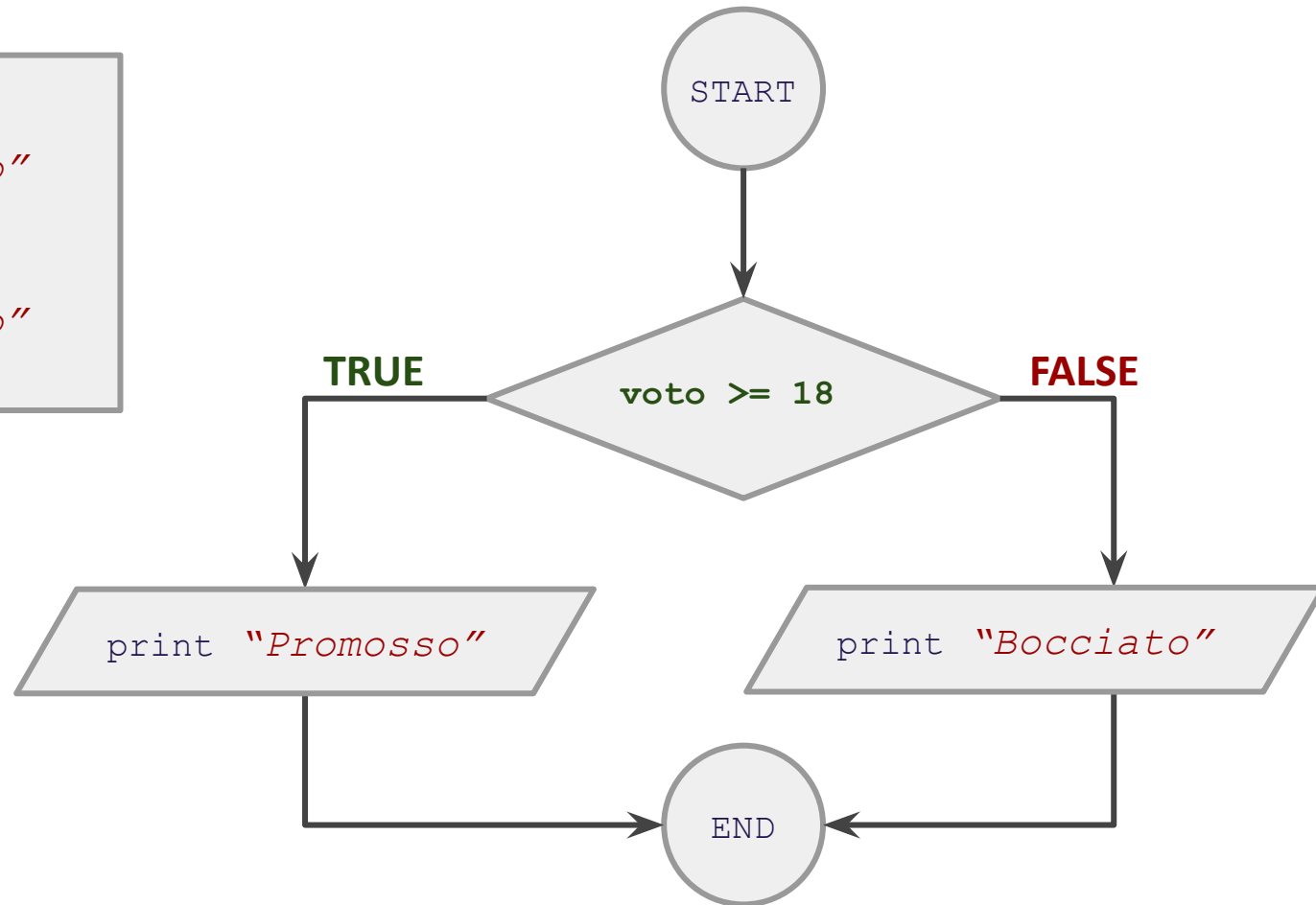


# Strutture di controllo - Selezione

## PSEUDOCODICE

```
if voto >= 18
    print "Promosso"
else
    print "Bocciato"
```

## DIAGRAMMA DI FLUSSO

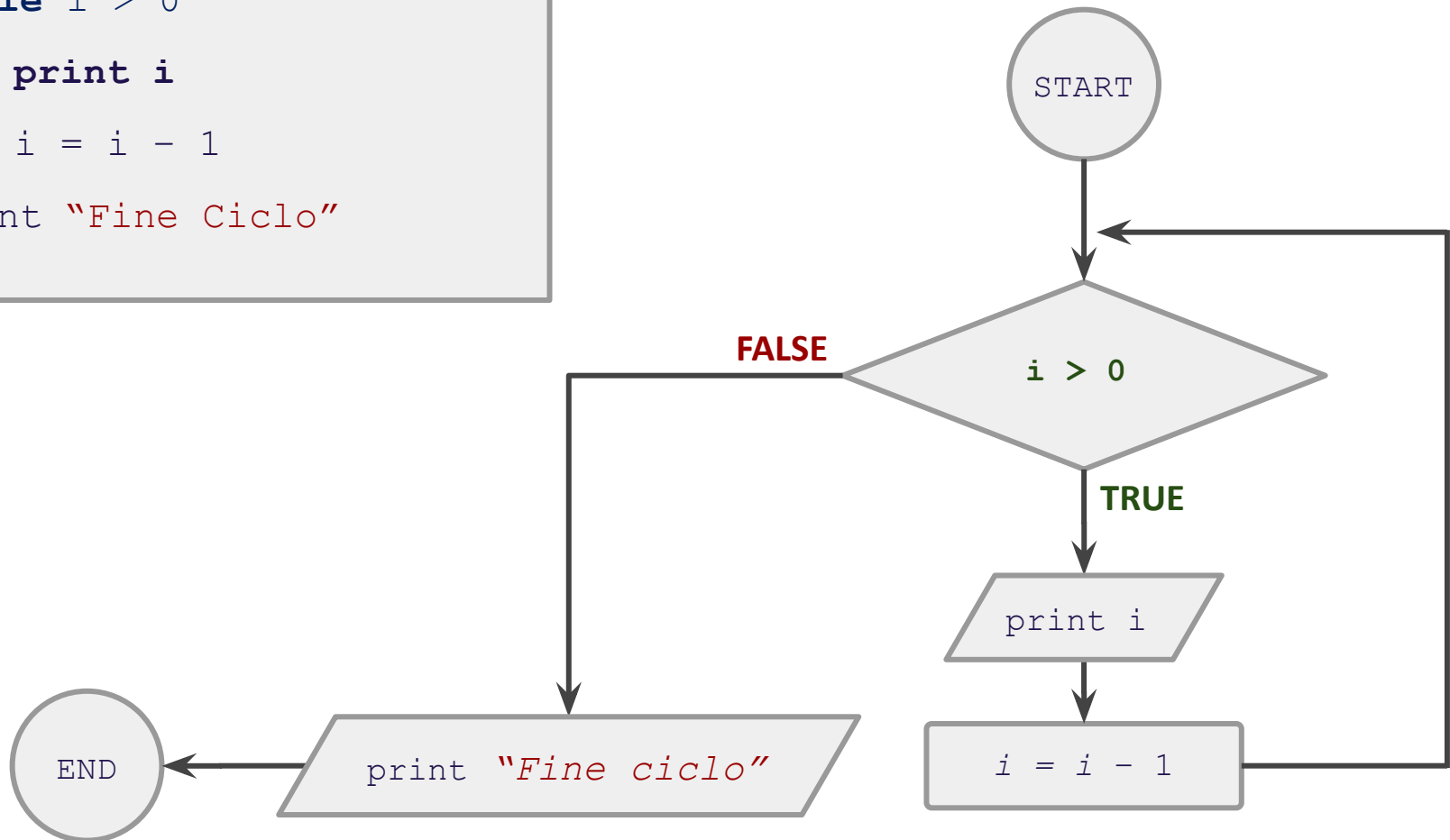


# Strutture di controllo - Iterazione

## PSEUDOCODICE

```
while i > 0
    print i
    i = i - 1
print "Fine Ciclo"
```

## DIAGRAMMA DI FLUSSO



# Strutture di controllo in C - Iterazione

I cicli **iterativi** che in genere vengono divisi in:

- Cicli **definiti** (*Counter-controlled repetition*)

Nei cicli **definiti** si conosce a monte il **numero di volte** che il ciclo dovrà essere eseguito.

In questo tipo di ciclo, in genere, si utilizza una **variabile contatore** che viene incrementata fino a quando non raggiunge un certo valore, oltre il quale il ciclo viene terminato

- Cicli **indefiniti** (*Sentinel-controlled repetition*)

Nei cicli **indefiniti** **NON** si conosce a monte il **numero di volte** che il ciclo dovrà essere eseguito.

Viene utilizzata, in genere, una **variabile sentinella** il cui valore verrà utilizzato nella **condizione di uscita del ciclo**.

# Strutture di controllo in C - Iterazione

Il linguaggio C definisce differenti strutture di controllo **iterative**:

- **for**
- **while**
- **do while**

Tali strutture di controllo sono del tutto **equivalenti** (nel senso che ciò che si può implementare con un ciclo **for**, si può anche implementare con un ciclo **while**).

Nonostante ciò, in genere, il costrutto iterativo **for** si presta bene per l'implementazione di cicli **definiti**, mentre i costrutti **while** e **do while** si prestano particolarmente bene per i cicli **indefiniti**.

# Strutture di controllo iterative in C - Costrutto for

Il costrutto **iterativo for** ha la seguente sintassi:

```
for(<init>; <cond>; <inc>){  
    //istruzioni da iterare  
}
```

- <init>

## INIZIALIZZAZIONE

Istruzione che viene eseguita **solo una volta** prima di *eseguire ciclo*

In genere si usa per inizializzare la **variabile contatore**

- <cond>

## CONDIZIONE

Condizione di **uscita dal ciclo**. Le istruzioni del ciclo **vengono ripetute** fino a quando la condizione è **vera**. Viene controllata **prima di ogni iterazione**.

- <inc>

## INCREMENTO

Istruzione che viene ripetuta alla **fine di ogni iterazione** e, in genere, serve per **incrementare la variabile contatore**.

# Strutture di controllo iterative in C - Costrutto for

## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int i;

    for(i=0; i<10; ++i){
        printf("%d\n", i);
    }
    return 0;
}
```



# Strutture di controllo iterative in C - Costrutto for

## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int i;

    for(i=0; i<10; ++i){
        printf("%d\n", i);
    }

    return 0;
}
```

### Inizializzazione:

Viene inizializzata la variabile **i** a 0 (che rappresenta la **variabile contatore**).

# Strutture di controllo iterative in C - Costrutto for

## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int i;

    for(i=0; i<10; ++i) {
        printf("%d\n", i);
    }

    return 0;
}
```

### Inizializzazione:

Viene inizializzata la variabile **i** a 0 (che rappresenta la **variabile contatore**).

### Condizione:

Ripeti le istruzioni del ciclo fintanto che il valore della variabile **i** è **minore** di 10.



# Strutture di controllo iterative in C - Costrutto for

## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int i;

    for(i=0; i<10; ++i) {
        printf("%d\n", i);
    }

    return 0;
}
```

### Inizializzazione:

Viene inizializzata la variabile **i** a 0 (che rappresenta la **variabile contatore**).

### Condizione:

Ripeti le istruzioni del ciclo fintanto che il valore della variabile **i** è **minore** di 10.

### Incremento:

Ad ogni iterazione, il valore della variabile **i** viene incrementato di 1.

# Strutture di controllo iterative in C - Costrutto for


## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int i;

    for(i=0; i<10; ++i){
        printf("%d\n", i);
    }

    return 0;
}
```



### Inizializzazione:

Viene inizializzata la variabile **i** a 0 (che rappresenta la **variabile contatore**).

### Condizione:

Ripeti le istruzioni del ciclo fintanto che il valore della variabile **i** è **minore** di 10.

### Incremento:

Ad ogni iterazione, il valore della variabile **i** viene incrementato di 1.

### Istruzioni da iterare:

Ad ogni iterazione, stampa il valore della variabile **i** su una nuova riga

# Strutture di controllo iterative in C - Costrutto for

## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int i;

    for(i=0; i<10; ++i){
        printf("%d\n", i);
    }

    return 0;
}
```

### Inizializzazione:

Viene inizializzata la variabile **i** a 0 (che rappresenta la **variabile contatore**).

### Condizione:

Ripeti le istruzioni del ciclo fintanto che il valore della variabile **i** è **minore** di 10.

### Incremento:

Ad ogni iterazione, il valore della variabile **i** viene incrementato di 1.

### Output del programma . . . . .



### Istruzioni da iterare:

Ad ogni iterazione, stampa il valore della variabile **i** su una nuova riga

# Strutture di controllo iterative in C - Costrutto for

## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int i;

    for(i=0; i<10; ++i){
        printf("%d\n", i);
    }

    return 0;
}
```

### Output del programma

Stampa tutti i valori da **0** a **9**

### Inizializzazione:

Viene inizializzata la variabile **i** a 0 (che rappresenta la **variabile contatore**).

### Condizione:

Ripeti le istruzioni del ciclo fintanto che il valore della variabile **i** è **minore** di 10.

### Incremento:

Ad ogni iterazione, il valore della variabile **i** viene incrementato di 1.

### Istruzioni da iterare:

Ad ogni iterazione, stampa il valore della variabile **i** su una nuova riga

# Strutture di controllo iterative in C - Costrutto for

## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int i;

    for(i=0; i<10; ++i){
        printf("%d\n", i);
    }

    return 0;
}
```

**PERCHE' IL VALORE 10  
NON VIENE STAMPATO?**

### Inizializzazione:

Viene inizializzata la variabile **i** a 0 (che rappresenta la **variabile contatore**).

### Condizione:

Ripeti le istruzioni del ciclo fintanto che il valore della variabile **i** è **minore** di 10.

### Incremento:

Ad ogni iterazione, il valore della variabile **i** viene incrementato di 1.

### Output del programma

Stampa tutti i valori da **0** a **9**

### Istruzioni da iterare:

Ad ogni iterazione, stampa il valore della variabile **i** su una nuova riga

# Strutture di controllo iterative in C - Costrutto for

## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int i;

    for(i=0; i<10; ++i){
        printf("%d\n", i);
    }

    return 0;
}
```

**PERCHE' IL VALORE 10  
NON VIENE STAMPATO?**

Nell'iterazione in cui *i* ha valore 9, succede che:

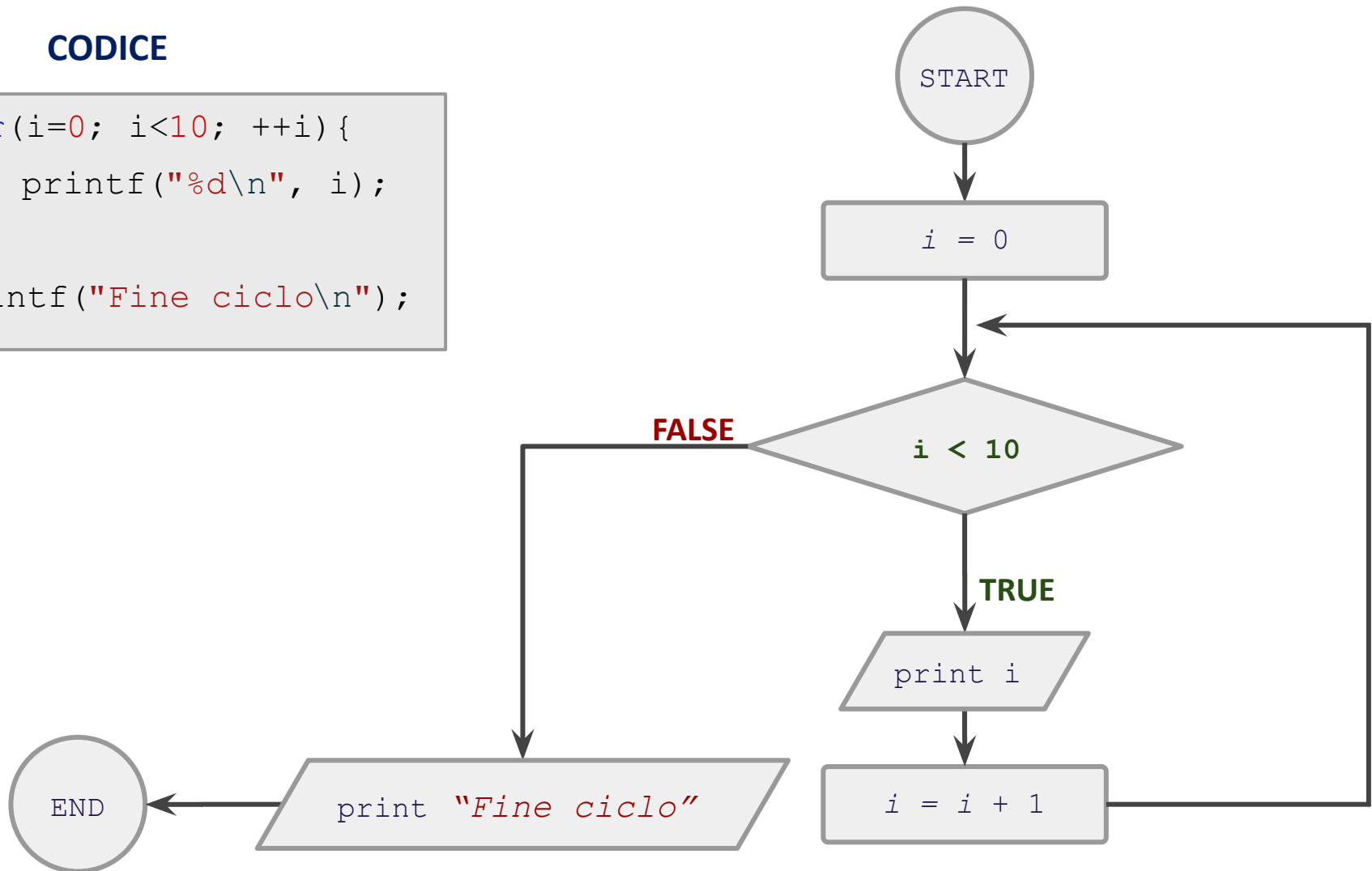
1. printf: viene stampato il valore **9**
2. Fine ciclo: viene incrementata la variabile *i*. Adesso, *i*, ha valore **10**
3. Inizio ciclo: prima di ripetere nuovamente il ciclo, viene controllata la condizione. Il controllo che viene fatto è *i* < 10 ma *i* vale 10, quindi l'espressione diventa 10 < 10, che viene valutata come **FALSA**.
4. La condizione viene valutata come **FALSA** e quindi l'iterazione per *i*=10 non viene effettuata

# Strutture di controllo - diagramma di flusso `for`

## CODICE

```
for(i=0; i<10; ++i){  
    printf("%d\n", i);  
}  
printf("Fine ciclo\n");
```

## DIAGRAMMA DI FLUSSO

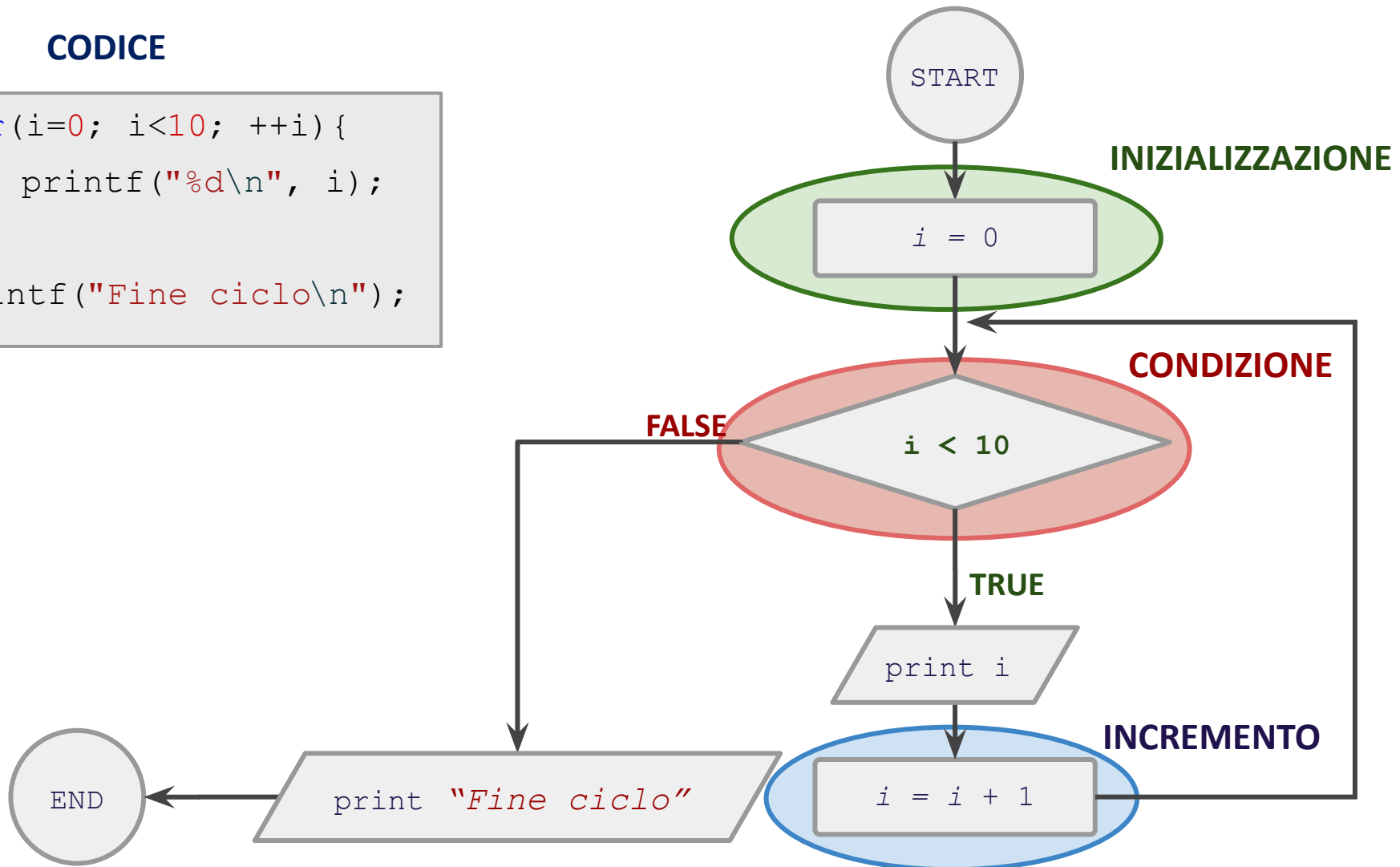


# Strutture di controllo - diagramma di flusso `for`

## CODICE

```
for(i=0; i<10; ++i){  
    printf("%d\n", i);  
}  
printf("Fine ciclo\n");
```

## DIAGRAMMA DI FLUSSO





# Strutture di controllo iterative in C - Costrutto while

Il costrutto **iterativo while** ha la seguente sintassi:

```
while (<cond>) {  
    //istruzioni da iterare  
}
```

- **<cond>**

## CONDIZIONE

Condizione di **uscita dal ciclo**. Le istruzioni del ciclo **vengono ripetute** fino a quando la condizione è **VERA**.

La condizione viene controllata **prima di ogni iterazione**.

# Strutture di controllo iterative in C - Costrutto while

## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int a = 100;

    while(a > 5){
        a /= 2;
        printf("a = %d\n", a);
    }
    printf("Fine: a = %d\n", a);
    return 0;
}
```

### Condizione:

Ripeti le istruzioni del ciclo fintanto che il valore della variabile **a** è **maggiore** di 5.

### Istruzioni da iterare:

Ad ogni iterazione, dimezza il valore della variabile **a** e ne stampa il valore a schermo

### Output del programma

```
a = 50
a = 25
a = 12
a = 6
a = 3
Fine: a = 3
```

# Strutture di controllo iterative in C - Costrutto while

## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int a = 3;

    while(a > 5){
        a /= 2;
        printf("a = %d\n", a);
    }
    printf("Fine: a = %d\n", a);
    return 0;
}
```

**CHE SUCCEDEREBBE SE, INIZIALMENTE,  
LA VARIABILE *a* HA UN VALORE  
MINORE DI 5?**

### Condizione:

Ripeti le istruzioni del ciclo fintanto  
che il valore della variabile **a** è  
**maggiore** di 5.

### Istruzioni da iterare:

Ad ogni iterazione, dimezza il valore  
della variabile **a** e ne stampa il valore  
a schermo

### Output del programma



# Strutture di controllo iterative in C - Costrutto while

## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int a = 3;

    while(a > 5){
        a /= 2;
        printf("a = %d\n", a);
    }
    printf("Fine: a = %d\n", a);
    return 0;
}
```

**CHE SUCCEDA SE, INIZIALMENTE,  
LA VARIABILE *a* HA UN VALORE  
MINORE DI 5?**

### Condizione:

Ripeti le istruzioni del ciclo fintanto  
che il valore della variabile **a** è  
**maggiore** di 5.

### Istruzioni da iterare:

Ad ogni iterazione, dimezza il valore  
della variabile **a** e ne stampa il valore  
a schermo

### Output del programma

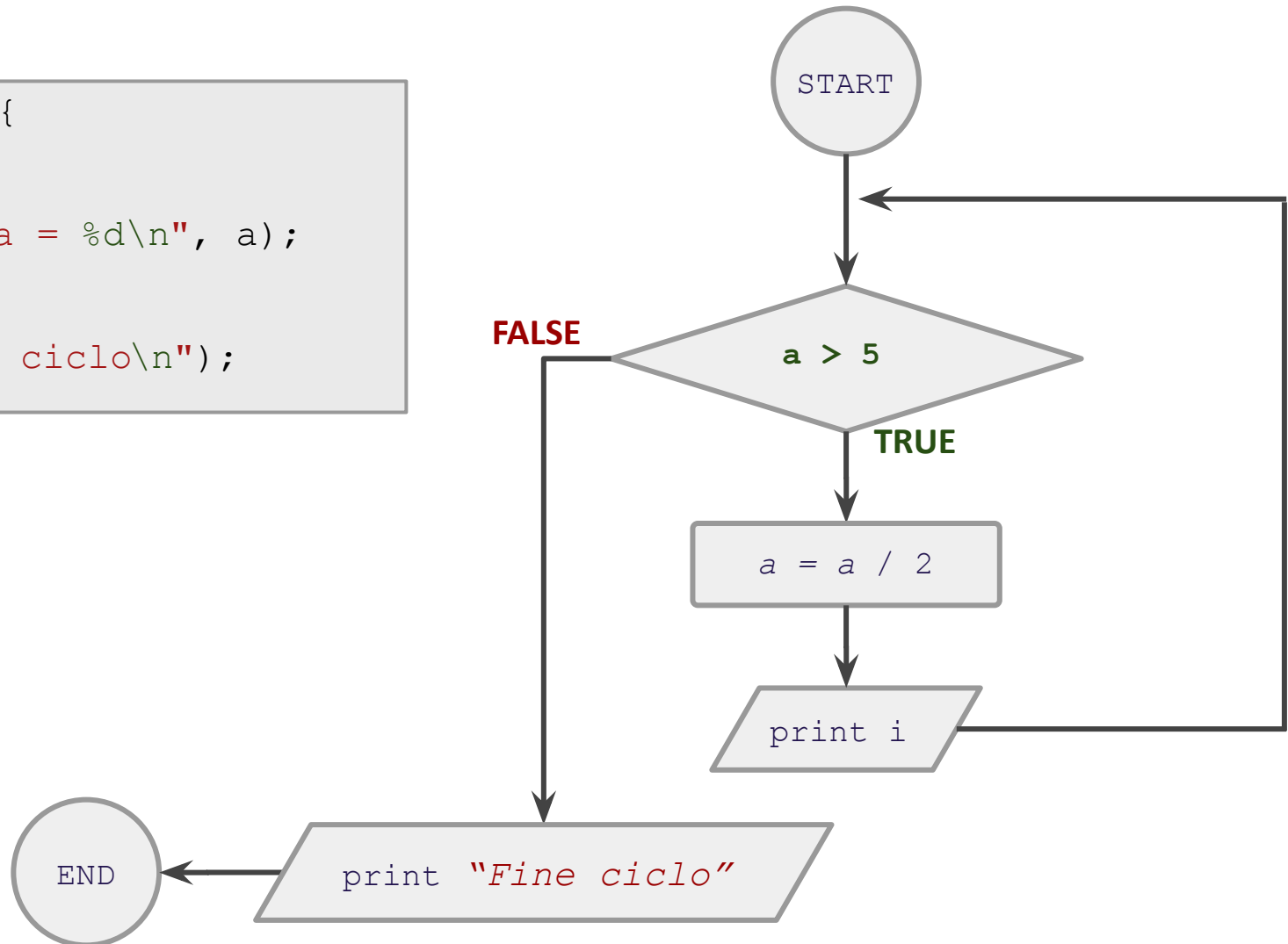
Fine: a = 3

# Strutture di controllo - diagramma di flusso `while`

## CODICE

```
while(a > 5){  
    a /= 2;  
    printf("a = %d\n", a);  
}  
printf("Fine ciclo\n");
```

## DIAGRAMMA DI FLUSSO



# Strutture di controllo - diagramma di flusso `while`

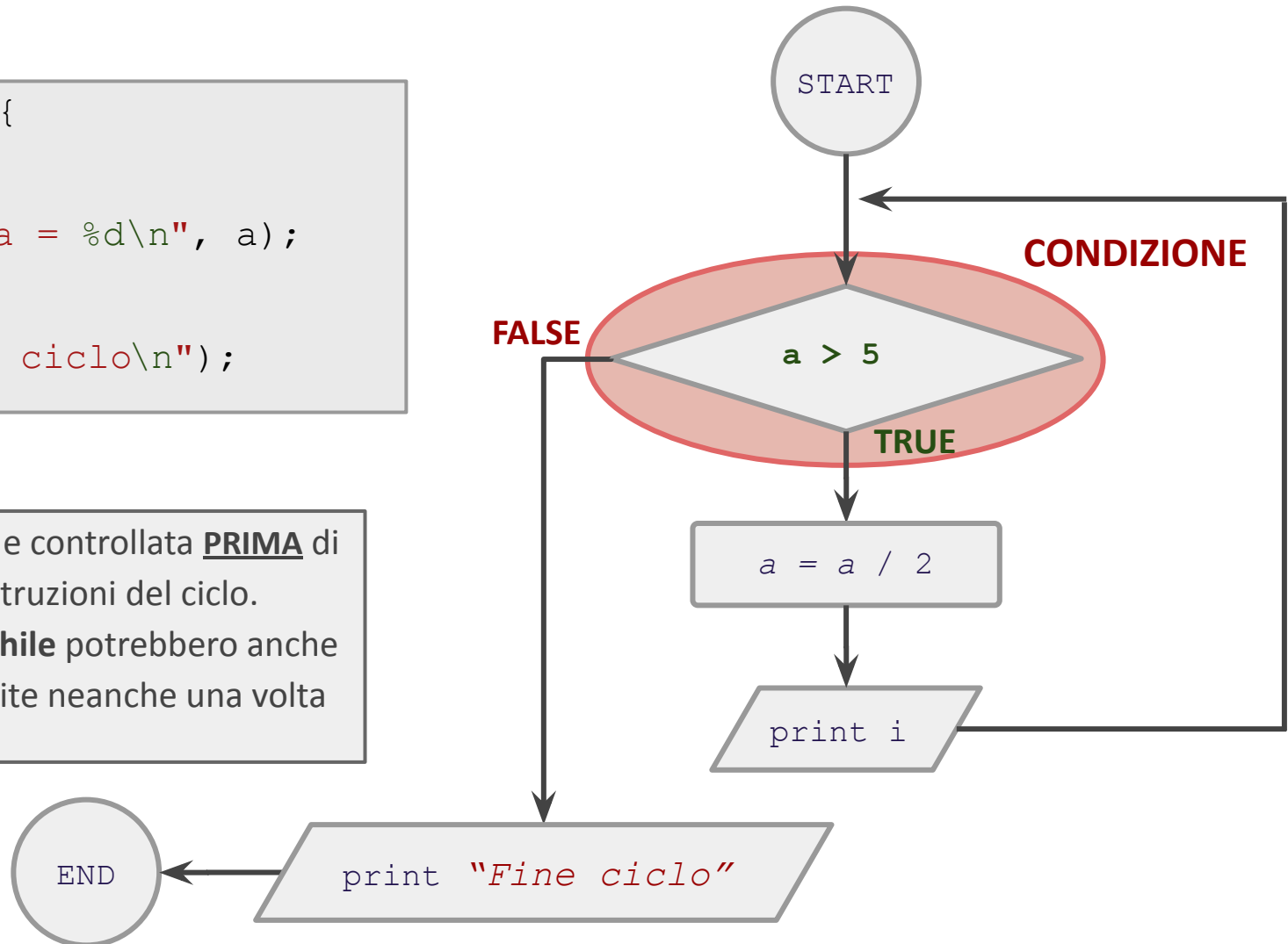
## CODICE

```
while(a > 5) {  
    a /= 2;  
    printf("a = %d\n", a);  
}  
printf("Fine ciclo\n");
```

## N.B.

La condizione viene controllata **PRIMA** di eseguire le istruzioni del ciclo.  
Le istruzioni del **while** potrebbero anche non essere eseguite neanche una volta

## DIAGRAMMA DI FLUSSO



# Strutture di controllo iterative in C - do while

Il costrutto **iterativo “do while”** ha la seguente sintassi:

```
do{  
    //istruzioni da iterare  
}while (<cond>) ;
```

- <cond>

## CONDIZIONE

Condizione di **uscita dal ciclo**. Le istruzioni del ciclo **vengono ripetute** fino a quando la condizione è **VERA**.

La condizione viene controllata **ALLA FINE** di ogni iterazione.

# Strutture di controllo iterative in C - do while

## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int a = 3;

    do{
        a /= 2;
        printf("a = %d\n", a);
    }while(a > 5);
    printf("Fine: a = %d\n", a);
    return 0;
}
```

**CHE SUCCEDA SE, INIZIALMENTE,  
LA VARIABILE *a* HA UN VALORE  
MINORE DI 5?**

### Condizione:

Ripeti le istruzioni del ciclo fintanto  
che il valore della variabile **a** è  
**maggiore** di 5.

### Istruzioni da iterare:

Ad ogni iterazione, stampa il valore  
della variabile **i** su una nuova riga

### Output del programma





# Strutture di controllo iterative in C - do while

## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int a = 3;

    do{
        a /= 2;
        printf("a = %d\n", a);
    }while(a > 5);
    printf("Fine: a = %d\n", a);
    return 0;
}
```

**CHE SUCCEDEREBBE SE, INIZIALMENTE,  
LA VARIABILE *a* HA UN VALORE  
MINORE DI 5?**

### Condizione:

Ripeti le istruzioni del ciclo fintanto  
che il valore della variabile **a** è  
**maggiore** di 5.

### Istruzioni da iterare:

Ad ogni iterazione, stampa il valore  
della variabile **i** su una nuova riga

### Output del programma

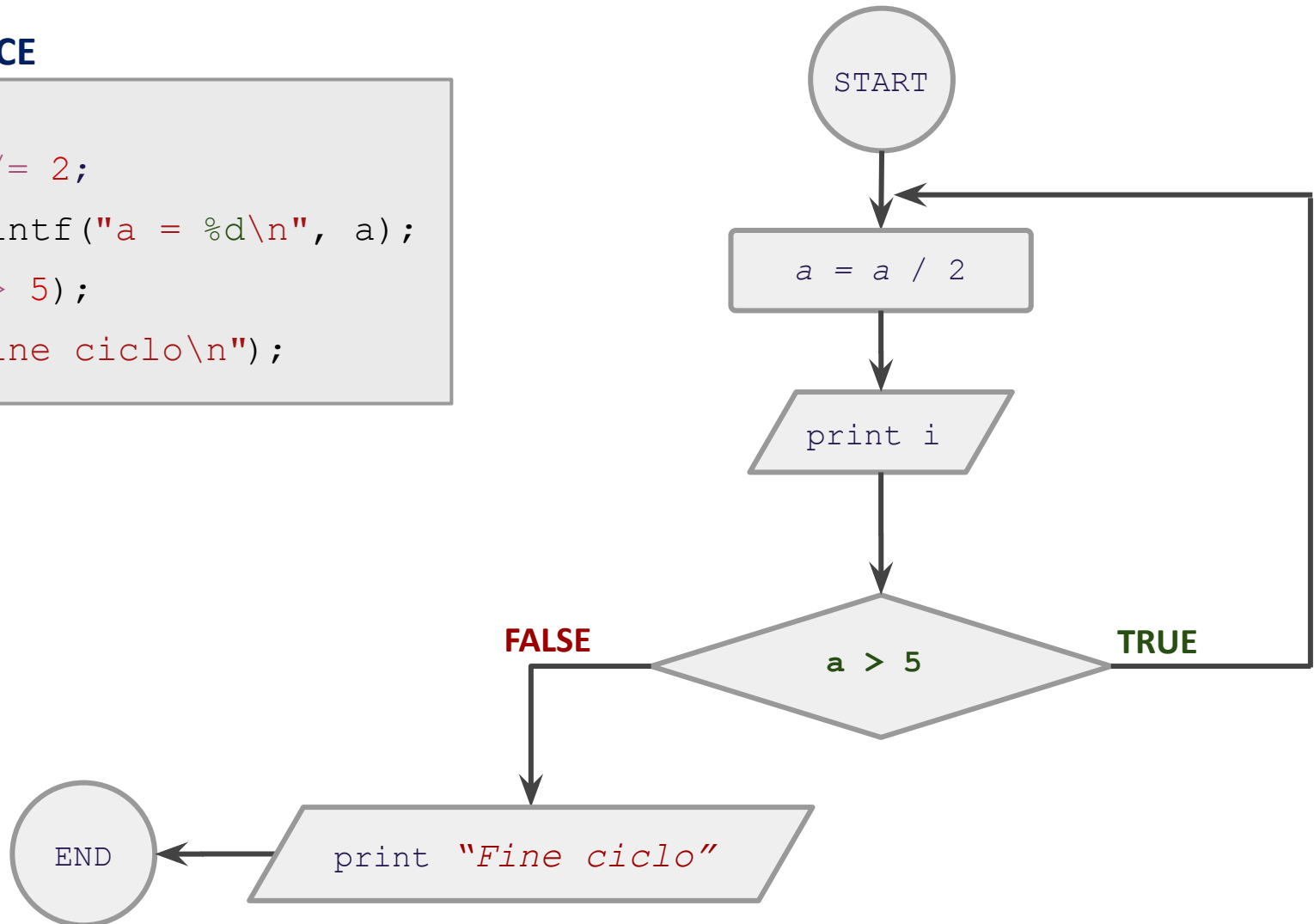
```
a = 1
Fine: a = 1
```

# Strutture di controllo - diagramma di flusso `do while`

## CODICE

```
do{  
    a /= 2;  
    printf("a = %d\n", a);  
}while(a > 5);  
printf("Fine ciclo\n");
```

## DIAGRAMMA DI FLUSSO



# Strutture di controllo - diagramma di flusso `do while`

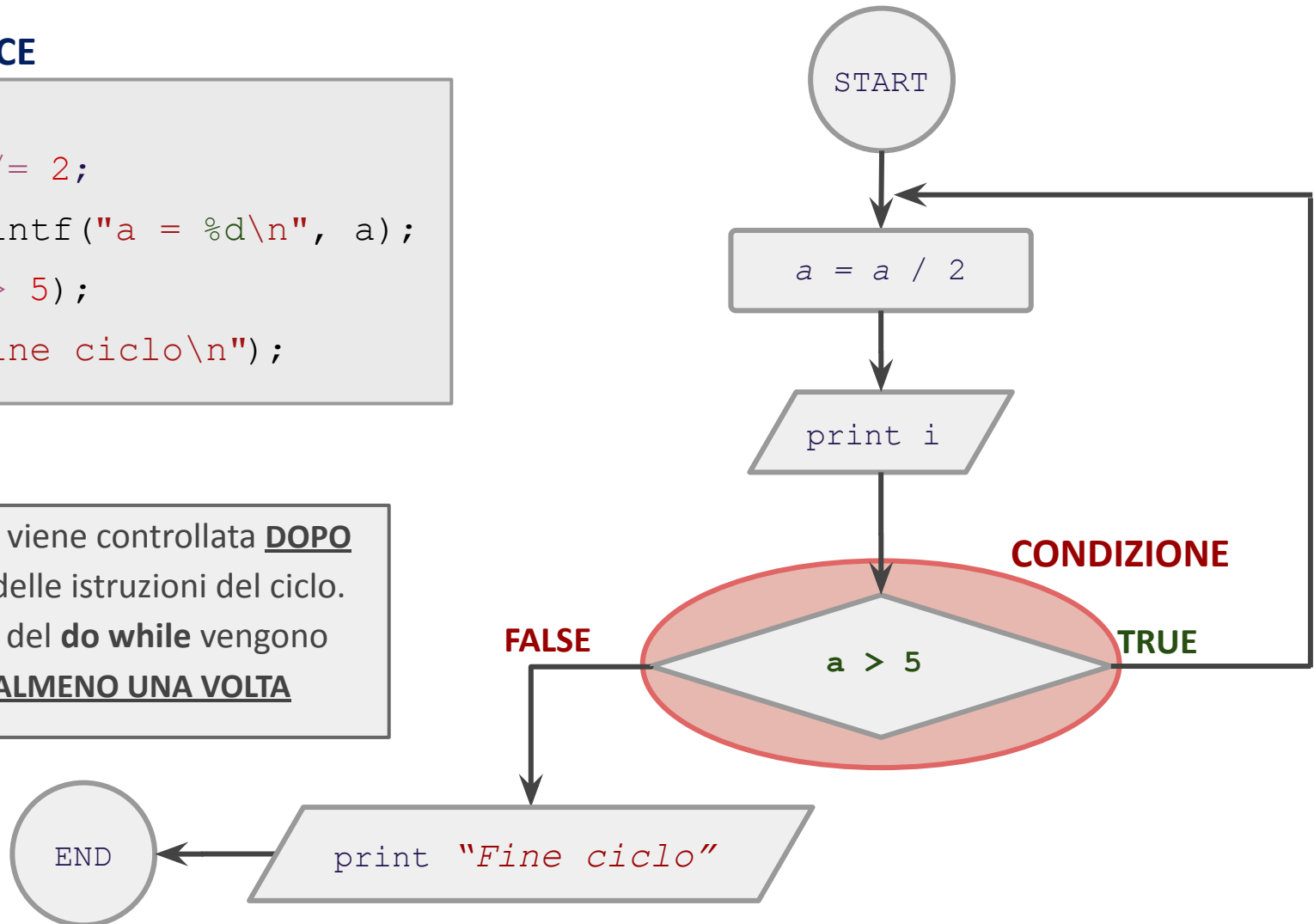
## CODICE

```
do{  
    a /= 2;  
    printf("a = %d\n", a);  
}while(a > 5);  
printf("Fine ciclo\n");
```

## N.B.

La condizione viene controllata **DOPO** l'esecuzione delle istruzioni del ciclo.  
Le istruzioni del **do while** vengono eseguite **ALMENO UNA VOLTA**

## DIAGRAMMA DI FLUSSO



# Strutture iterative: statement CONTINUE e BREAK

Gli statement **continue** e **break** vengono utilizzati nei costrutti iterativi (**for**, **while**, **do while**) e modificano il flusso di esecuzione del ciclo in cui vengono usati.

- Il **continue** termina immediatamente l'**iterazione corrente** ed esegue l'**iterazione successiva**.
- Il **break** termina immediatamente l'**iterazione corrente** ed **esce dal ciclo**.

# Strutture iterative: statement CONTINUE e BREAK

Gli statement **continue** e **break** vengono utilizzati nei costrutti iterativi (**for**, **while**, **do while**) e modificano il flusso di esecuzione del ciclo in cui vengono usati.

- Il **continue** termina immediatamente l'**iterazione corrente** ed esegue l'**iterazione successiva**.
- Il **break** termina immediatamente l'**iterazione corrente** ed **esce dal ciclo**.

```
int main(void) {  
    int i;  
    for(i=0; i<10; ++i) {  
        if(i == 5)  
            continue;  
        printf("%d ", i);  
    }  
    return 0;  
}
```

```
int main(void) {  
    int i;  
    for(i=0; i<10; ++i) {  
        if(i == 5)  
            break;  
        printf("%d ", i);  
    }  
    return 0;  
}
```

# Strutture iterative: statement CONTINUE e BREAK

Gli statement **continue** e **break** vengono utilizzati nei costrutti iterativi (**for**, **while**, **do while**) e modificano il flusso di esecuzione del ciclo in cui vengono usati.

- Il **continue** termina immediatamente l'**iterazione corrente** ed esegue l'**iterazione successiva**.
- Il **break** termina immediatamente l'**iterazione corrente** ed **esce dal ciclo**.

```
int main(void) {  
    int i;  
    for(i=0; i<10; ++i) {  
        if(i == 5)  
            continue;  
        printf("%d ", i);  
    }  
    return 0;  
}
```

**STAMPA:**

0 1 2 3 4 6 7 8 9

```
int main(void) {  
    int i;  
    for(i=0; i<10; ++i) {  
        if(i == 5)  
            break;  
        printf("%d ", i);  
    }  
    return 0;  
}
```

**STAMPA:**

0 1 2 3 4

# Strutture di selezione in C - if

Il costrutto **di selezione** **if** ha la seguente sintassi:

```
if(<cond>) {  
    //istruzioni da eseguire se <cond> =  
    TRUE  
}
```

- **<cond>**

## CONDIZIONE

Le istruzioni del **corpo dell'if** (racchiuso fra parentesi graffe) vengono eseguite se la condizione è **VERA**.

La condizione viene in genere espressa mediante un'espressione relazionale (operatori relazionali ...)

# Strutture di selezione in C - if

## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int a = 3;

    if(a > 0 && a < 10) {
        ++a;
    }
    printf("a = %d\n", a);
    return 0;
}
```

## Condizione:

Se il valore della variabile **a** è  
(strettamente) maggiore di 0 e  
(strettamente) minore di 10, esegui  
le istruzioni contenute nel corpo  
dell'**if**

## Istruzioni da iterare:

(pre)incremento della variabile **a**



# Strutture di selezione in C - if

## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int a = 3;

    if(a > 0 && a < 10) {
        ++a;
    }
    printf("a = %d\n", a);
    return 0;
}
```

### Condizione:

Se il valore della variabile **a** è  
(strettamente) maggiore di 0 e  
(strettamente) minore di 10, esegui  
le istruzioni contenute nel corpo  
dell'**if**

### Istruzioni da iterare:

(pre)incremento della variabile **a**

### Output del programma

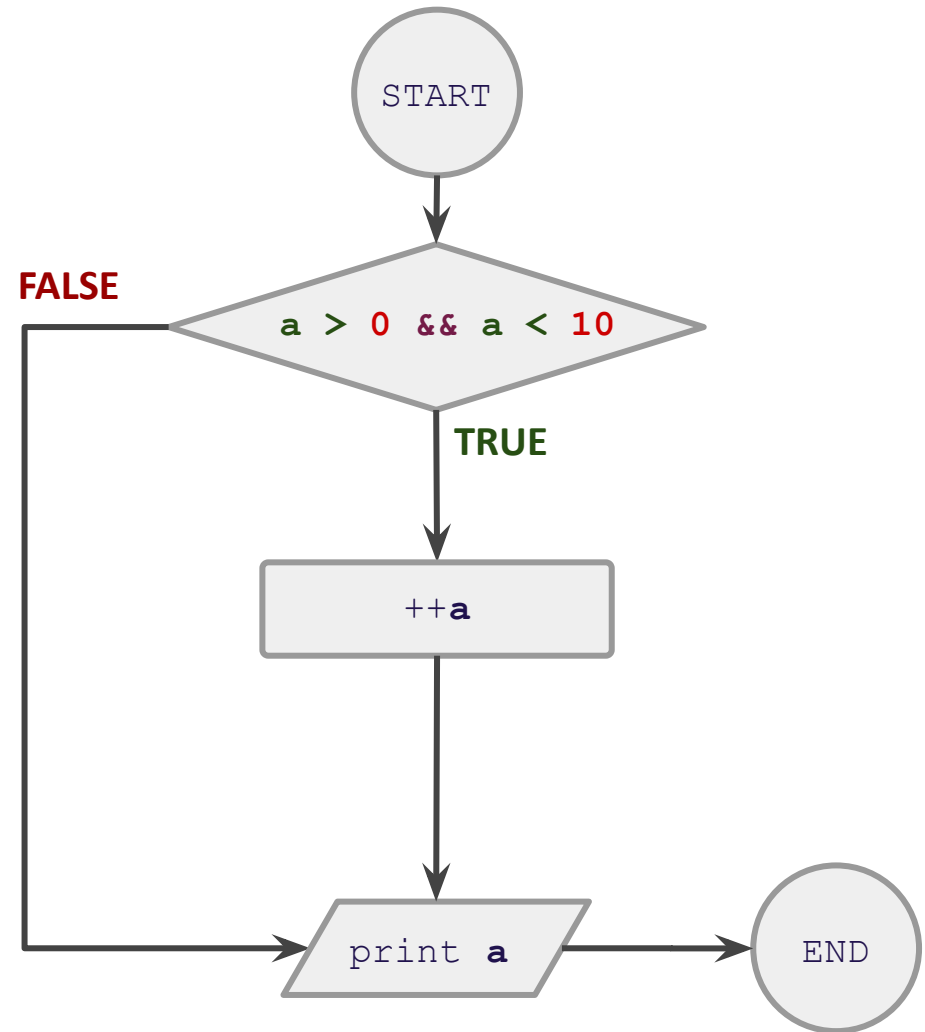
a = 4

# Strutture di selezione in C - if

## CODICE

```
if(a > 0 && a < 10) {  
    ++a;  
}  
printf("a = %d\n", a);  
return 0;
```

## DIAGRAMMA DI FLUSSO



# Strutture di selezione in C - if

Il costrutto di selezione **if ... else** ha la seguente sintassi:

```
if(<cond_1>){  
    //istruzioni da eseguire se <cond_1> = TRUE  
} else if(<cond_2>){  
    //istruzioni da eseguire se  
    <cond_1> = FALSE e <cond_2> = TRUE  
} else {  
    //istruzioni da eseguire se  
    <cond_1> = <cond_2> = FALSE  
}
```

- **<cond>**

## CONDIZIONE

Le istruzioni del **corpo dell'if** (racchiuso fra parentesi graffe) vengono eseguite se la rispettiva condizione è **VERA**.

Non appena viene soddisfatta una condizione, le successive non vengono più “controllate”.

# Strutture di selezione in C - if

## ESEMPIO:

```
#include <stdio.h>

int main(void) {
    int a = 35;

    if(a >= 0 && a <= 10) {
        printf("Tra 0 e 10\n");
    } else if(a >= 11 && a <= 20) {
        printf("Tra 11 e 20\n");
    } else {
        printf("Maggiore di 20\n");
    }
    return 0;
}
```

### Condizione 1:

Se il valore della variabile **a** è maggiore o uguale a 0 e minore o uguale a 10, esegui le istruzioni contenute nel corpo dell'`if`

### Condizione 2:

Se il valore della variabile **a** è maggiore o uguale a 11 e minore o uguale a 20, esegui le istruzioni contenute nel corpo dell'`if`

### Output del programma

Maggiore di 20

# Strutture di selezione in C - switch

Il costrutto di selezione **switch** ha la seguente sintassi:

```
switch(variable) {  
    case value1:  
        //istruzioni da eseguire se variable == value1  
        break;  
    case value2:  
        //istruzioni da eseguire se variable == value2  
        break;  
    default:  
        //se nessun caso precedente è soddisfatto  
        break;  
}
```

Lo **switch** viene utilizzato per effettuare delle “decisioni” in base al **valore** di **una sola variabile**.

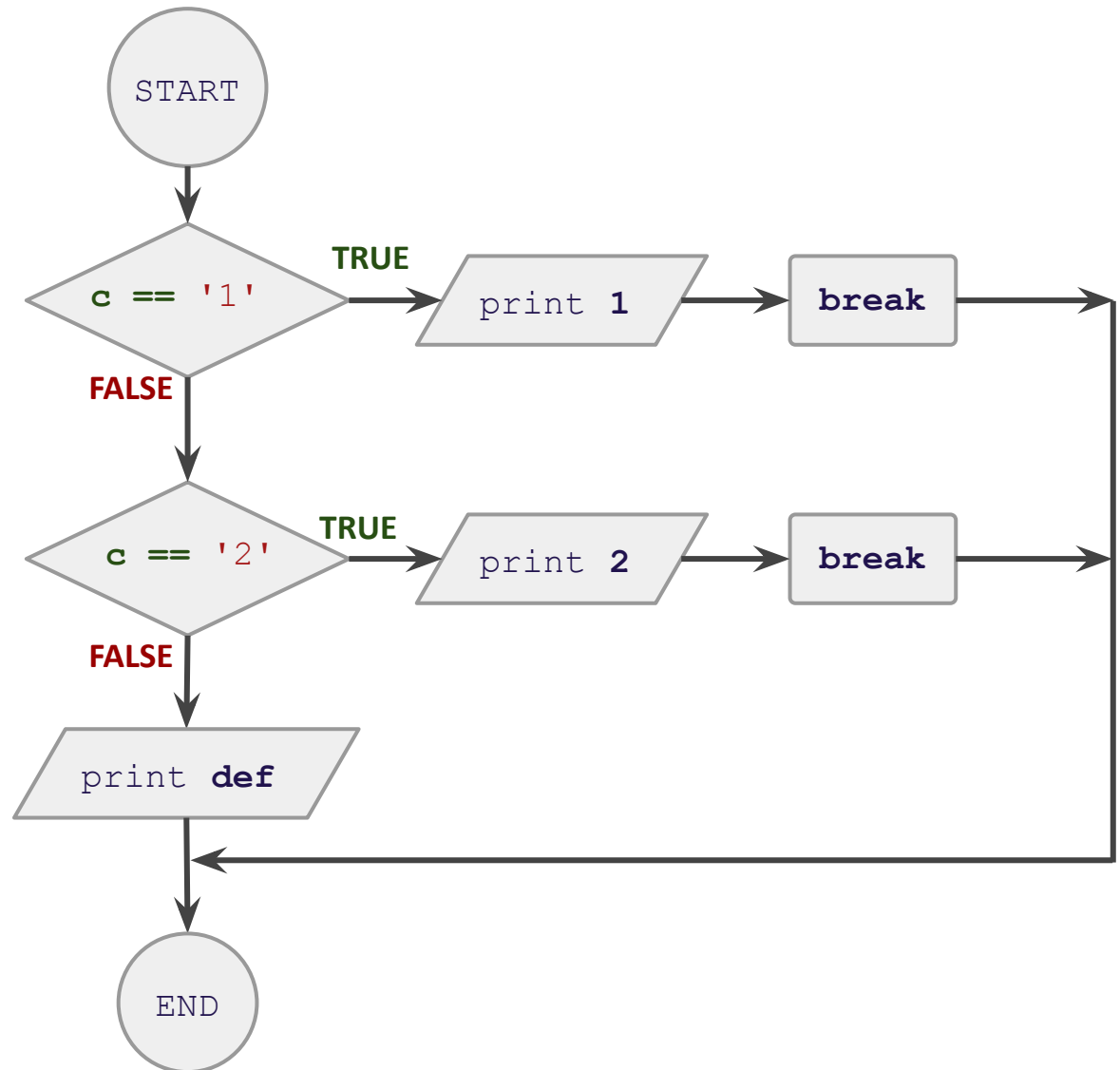
Se viene omesso il **break**, una volta eseguito il blocco di istruzioni appartenente ad un **case**, verrà eseguito anche il blocco di istruzioni del **case** successivo.

Viceversa, inserendo il **break** ci si assicura che verrà eseguito soltanto un **case**.

# Strutture di selezione in C - switch

```
char c = '1';

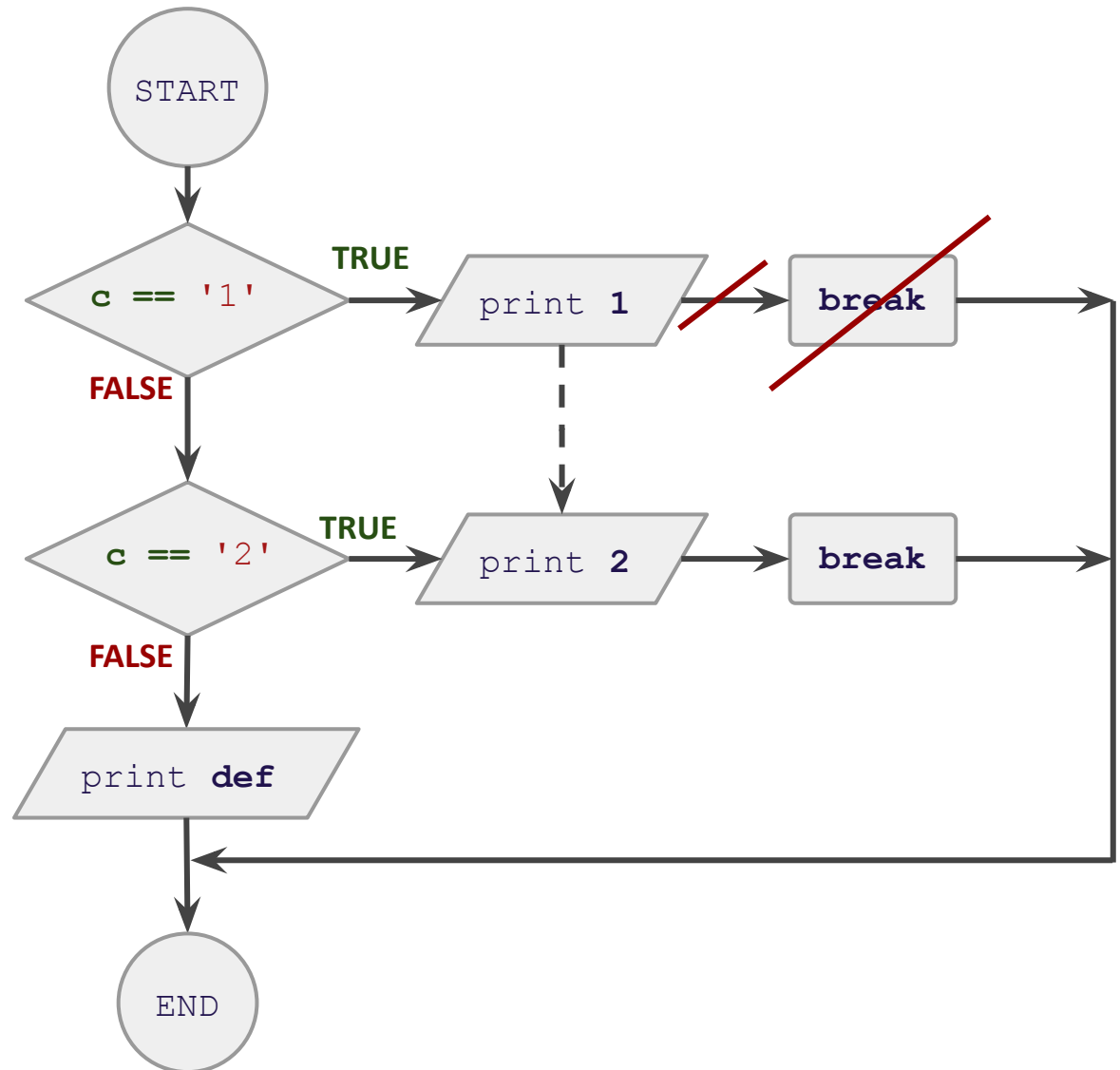
switch( c ){
    case '1':
        printf("1 \n");
        break;
    case '2':
        printf("2 \n");
        break;
    default:
        printf("def \n");
        break;
}
```



# Strutture di selezione in C - switch

```
char c = '1';

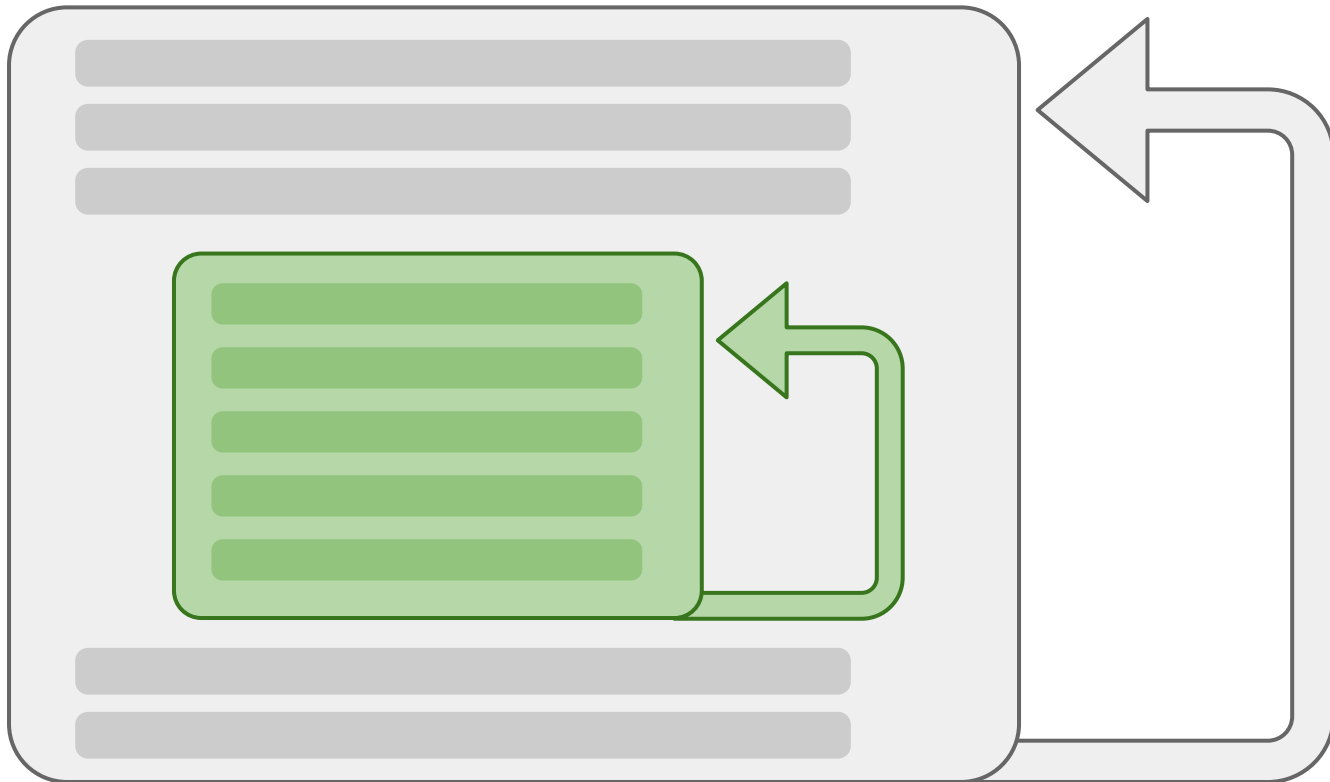
switch( c ){
  case '1':
    printf("1 \n");
    break;
  case '2':
    printf("2 \n");
    break;
  default:
    printf("def \n");
    break;
}
```



# Strutture di iterazione in C - cicli annidati

Molte volte è necessario inserire un **ciclo** all'**interno** di un altro **ciclo**.

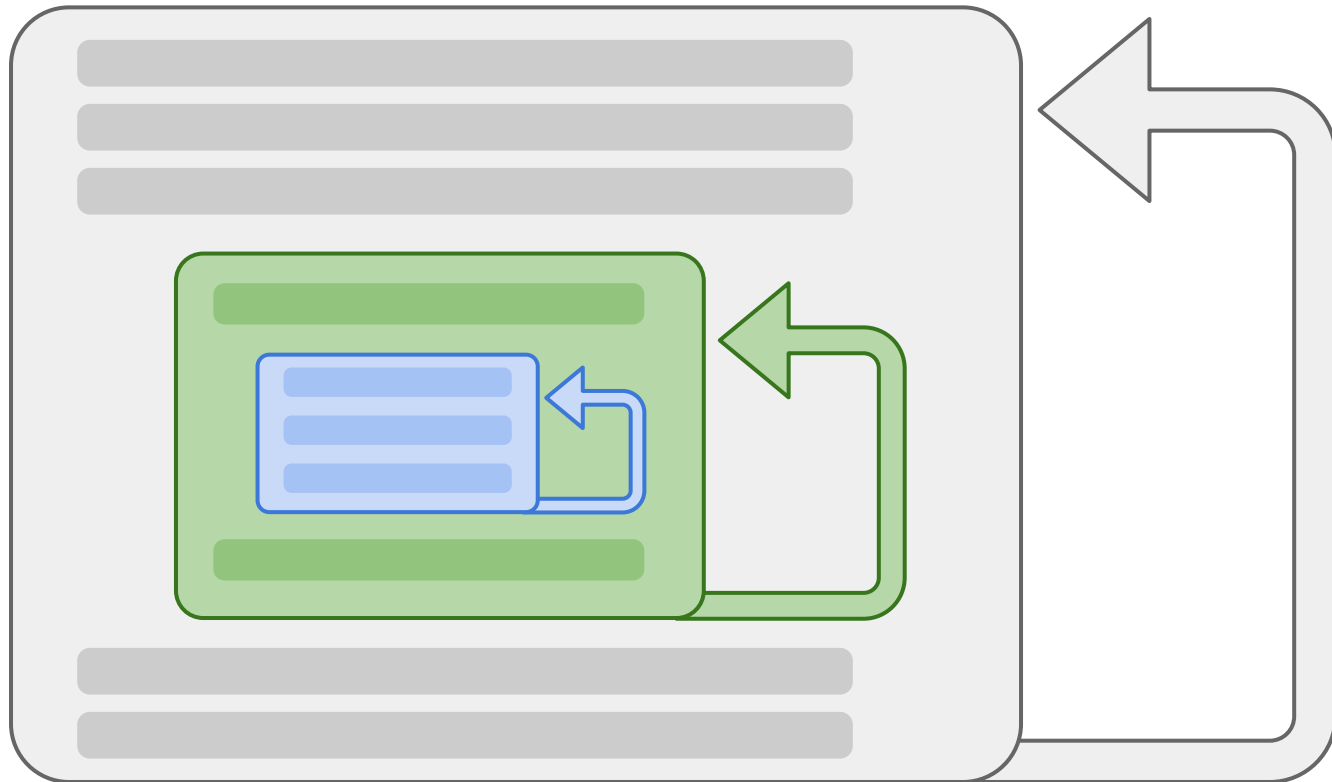
In questo caso si parla di **cicli annidati** facendo riferimento proprio al fatto che un ciclo è contenuto in un altro.





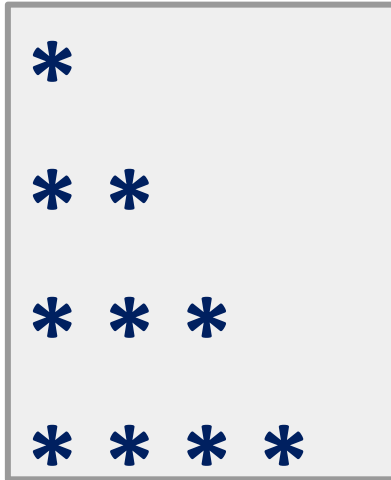
# Strutture di iterazione in C - cicli annidati

E' possibile annidare un numero indefinito di cicli, ma in genere è buona norma (leggibilità del codice, manutenibilità, debug...) non superare mai i 2 cicli.



# Strutture di selezione in C - cicli annidati

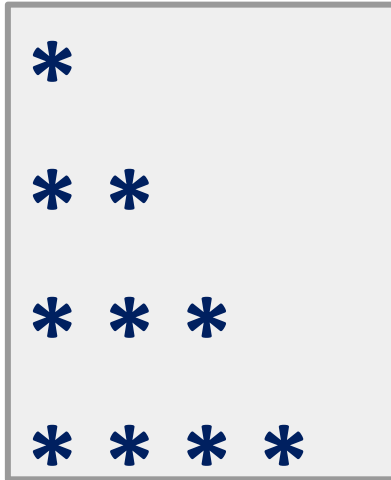
Si supponga di dover stampare a schermo la seguente struttura di caratteri



**Come si potrebbe procedere??**

# Strutture di selezione in C - cicli annidati

Si supponga di dover stampare a schermo la seguente struttura di caratteri



```
#include <stdio.h>

int main(void) {
    int n = 4;
    int i;

    for(i=0; i<n; ++i){

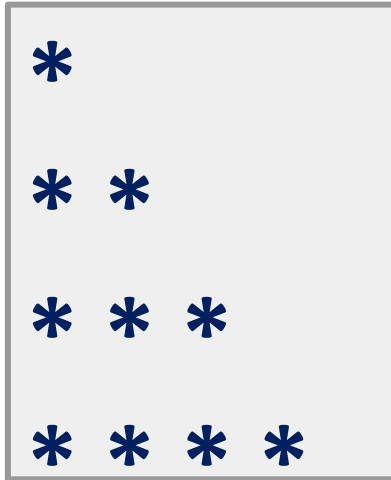
        // Stampa una riga

    }

    return 0;
}
```

# Strutture di selezione in C - cicli annidati

Si supponga di dover stampare a schermo la seguente struttura di caratteri



```
#include <stdio.h>

int main(void) {
    int n = 4;
    int i;

    for(i=0; i<n; ++i){

        // Stampa una riga

    }

    return 0;
}
```

# Strutture di selezione in C - cicli annidati

Si supponga di dover stampare a schermo la seguente struttura di caratteri

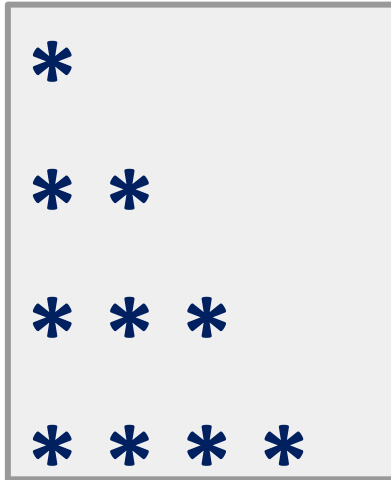
```
*  
  
* *  
  
* * *  
  
* * * *
```

```
int j;  
int num = 10;  
for(j=0; j<num; ++j){  
    printf("* ");  
}  
printf("\n");
```

```
#include <stdio.h>  
  
int main(void) {  
    int n = 4;  
    int i;  
  
    for(i=0; i<n; ++i){  
  
        // Stampa una riga  
  
    }  
  
    return 0;  
}
```

# Strutture di selezione in C - cicli annidati

Si supponga di dover stampare a schermo la seguente struttura di caratteri



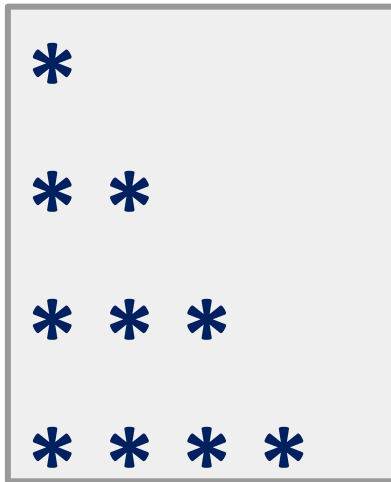
```
#include <stdio.h>

int main(void) {
    int n = 5, num = 10;
    int i, j;

    for(i=0; i<n; ++i){
        for(j=0; j<num; ++j){
            printf("* ");
        }
        printf("\n");
    }
    return 0;
}
```

# Strutture di selezione in C - cicli annidati

Si supponga di dover stampare a schermo la seguente struttura di caratteri



Questa versione non è ancora corretta.  
Infatti, ogni riga viene stampata con lo stesso numero di “\*”.

**Come si risolve?**

```
#include <stdio.h>

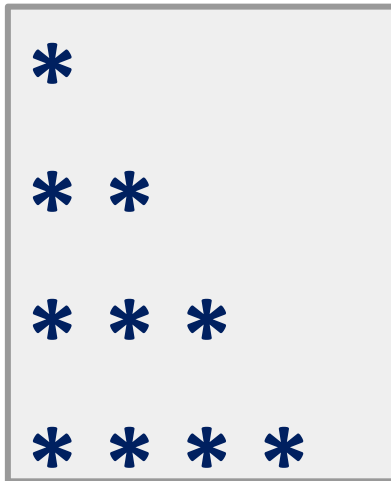
int main(void) {
    int n = 5, num = 10;
    int i, j;

    for(i=0; i<n; ++i){
        for(j=0; j<num; ++j){
            printf("* ");
        }
        printf("\n");
    }

    return 0;
}
```

# Strutture di selezione in C - cicli annidati

Si supponga di dover stampare a schermo la seguente struttura di caratteri



```
#include <stdio.h>

int main(void) {
    int n = 5;
    int i, j;

    for(i=0; i<n; ++i){
        for(j=0; j<i; ++j){
            printf("* ");
        }
        printf("\n");
    }
    return 0;
}
```