

Programmazione

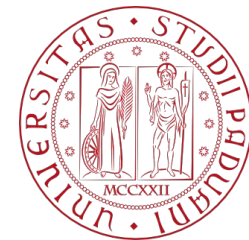
Giovanni Da San Martino

Dipartimento of Matematica, Università degli Studi di Padova
giovanni.dasanmartino@unipd.it



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Pseudocodice e Problem Solving



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

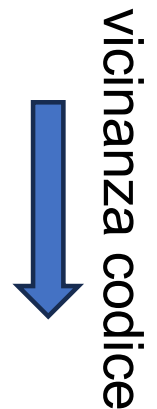
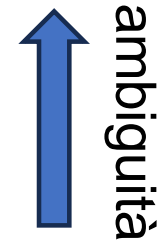
L' "Algoritmo" per Creare Algoritmi



L' "Algoritmo" per Creare Algoritmi



- Creiamo un linguaggio di programmazione “semplificato”
 - che ha molte meno complicazioni rispetto al C
 - per concentrarci sulla strategia risolutiva
- Lo useremo inizialmente per progettare la bozza di soluzione, con l’obiettivo di poterla comunicare ad un nostro collega
- In realtà possiamo usare più alternative con diversi livelli di strutturazione/formalizzazione:
 - Testo libero
 - pseudocodice / diagrammi di flusso
 - diagrammi di flusso eseguibili (sintassi più rigida, ma come il C permettono di testare i nostri algoritmi)



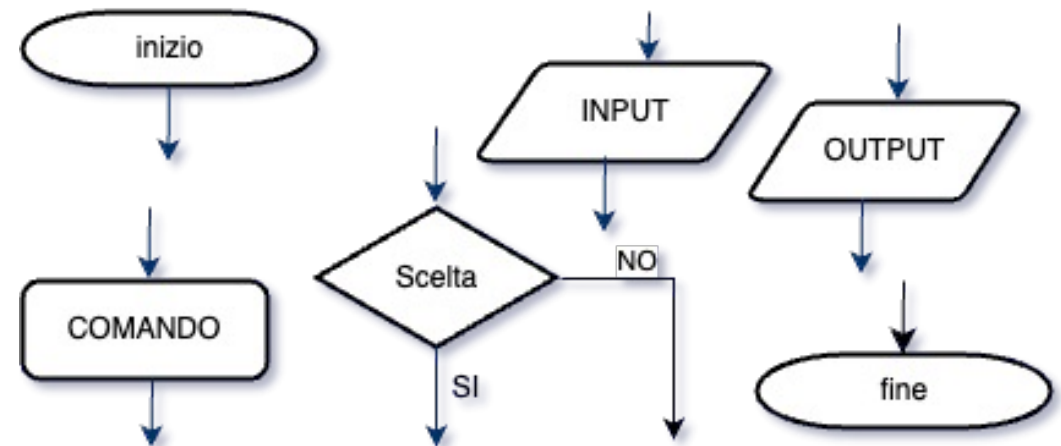
Diagrammi di Flusso (eseguibile)



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Un diagramma di flusso ha:

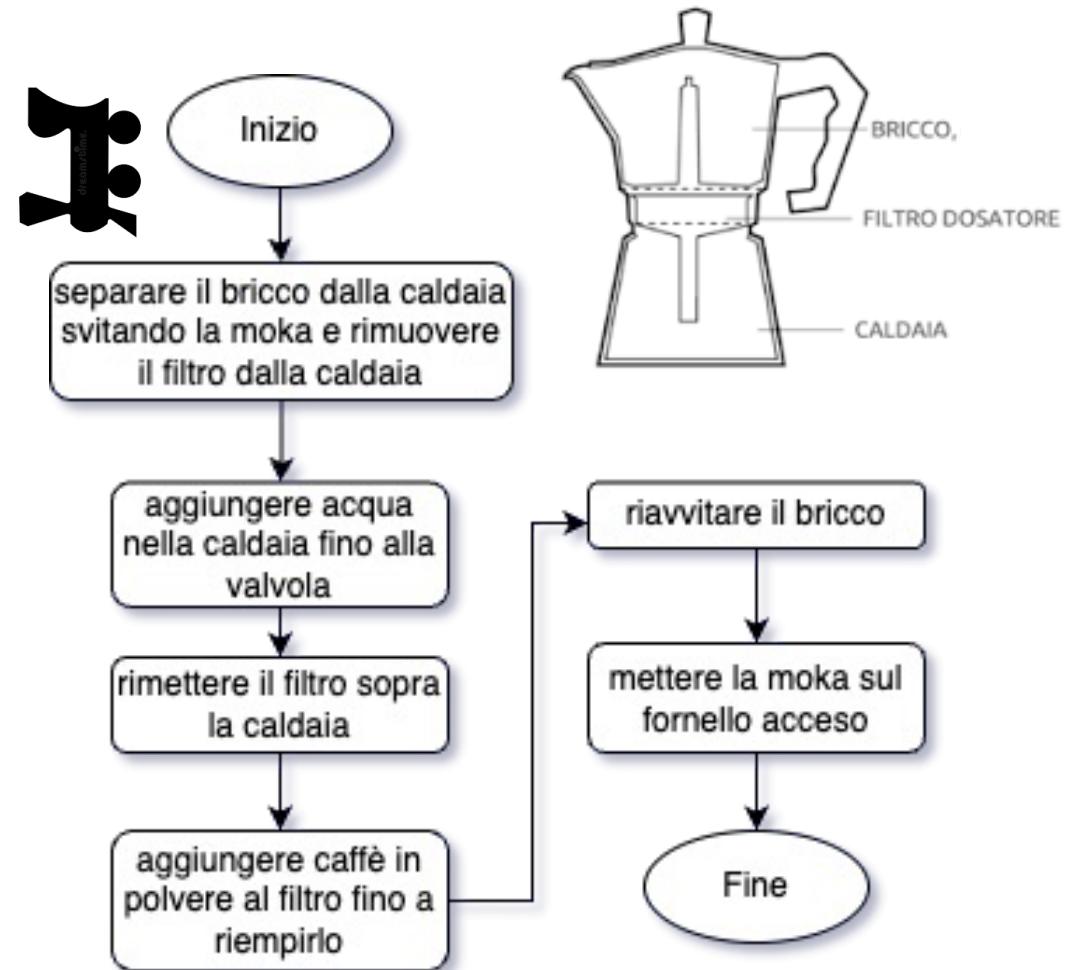
- un solo inizio, una sola fine (come un treno che va da A a B)
- i blocchi Input, Output, Comando hanno esattamente un arco entrante ed uno uscente
- il rombo ha un'entrata e due uscite: l'uscita da percorrere si determina in base ad una condizione che deve essere valutabile come vera/falsa (si/no) in modo non ambiguo
- l'esecuzione inizia dal nodo *inizio* e prosegue fino a quello *fine*, con i blocchi *Input* (*leggi*), *Output* (*scrivi*) e *Comando* che modificano lo stato del mondo circostante



Diagrammi di Flusso



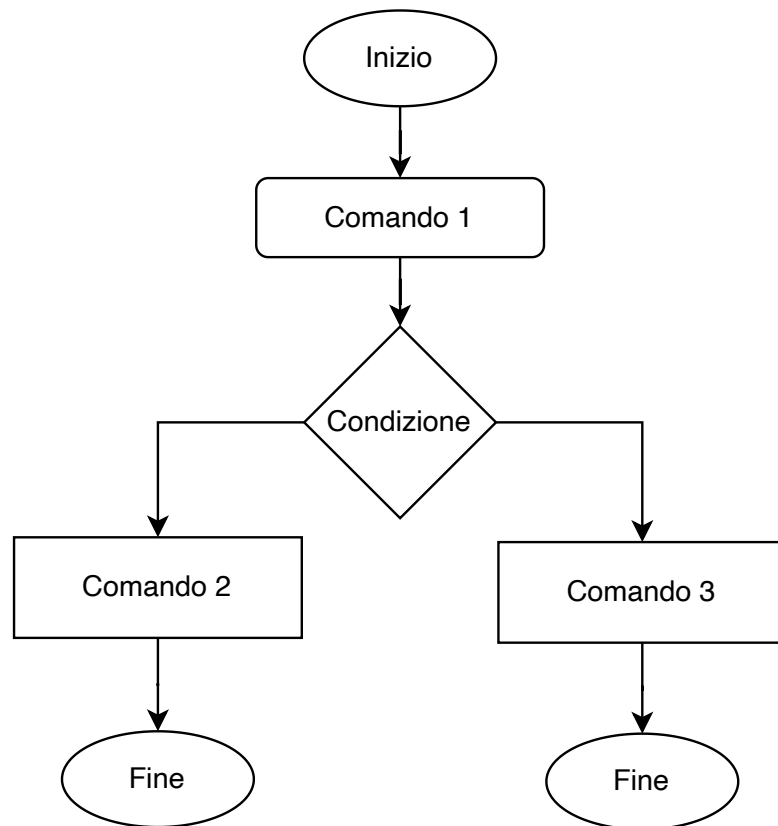
- L'esecuzione inizia dal nodo *Inizio* e prosegue fino a quello *Fine*
- Si deve poter eseguire il contenuto di un blocco senza ambiguità ed in tempo finito
- I blocchi *Input*, *Output* e *Comando* modificano lo stato del mondo circostante (la moka nell'esempio)



Potenza Diagrammi di Flusso: blocco Fine



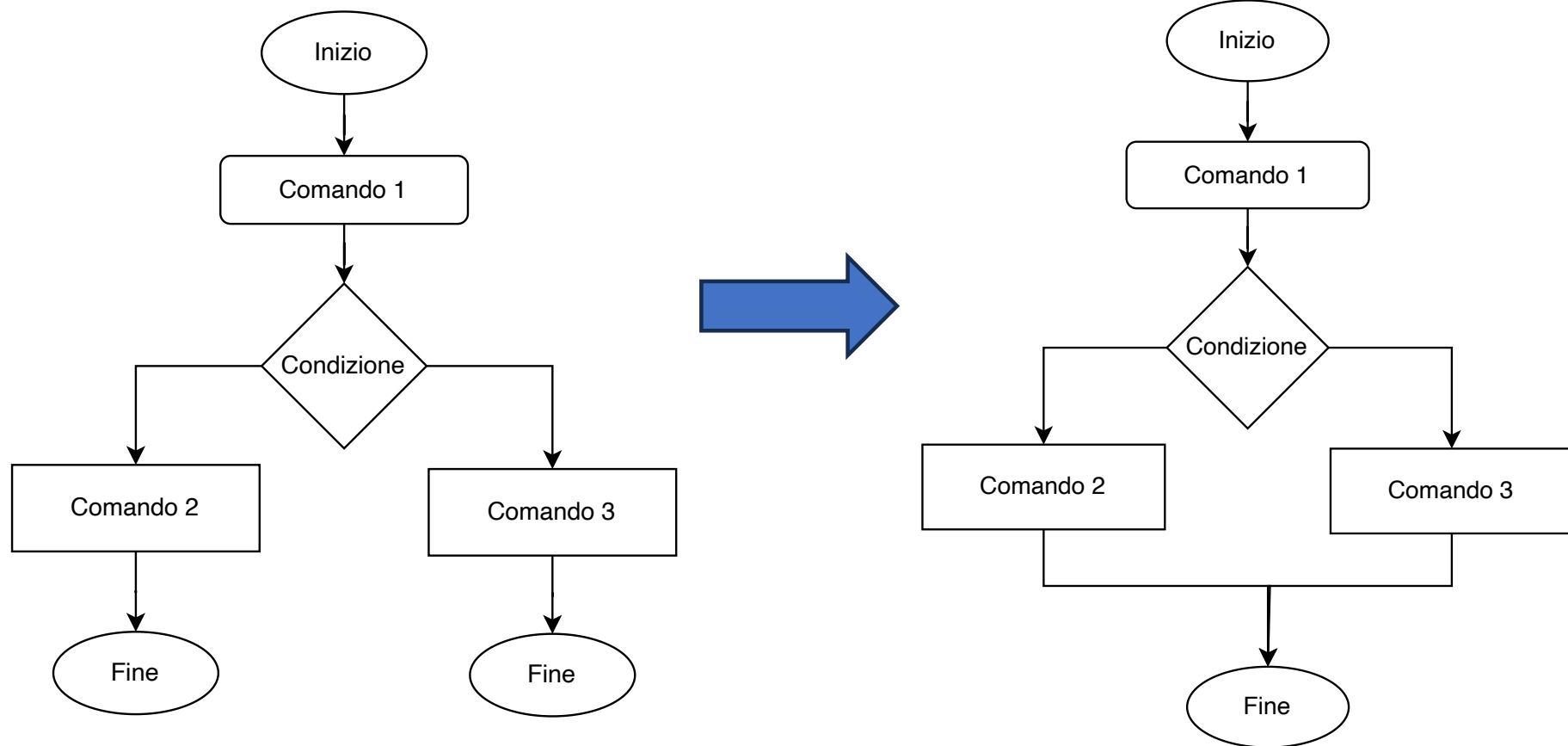
Un diagramma di flusso ha una sola fine → è una limitazione?



Potenza Diagrammi di Flusso: blocco Fine



Un diagramma di flusso ha una sola fine → è una limitazione? NO



Blocco Comando – Espressioni Aritmetiche





UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- Cosa possiamo mettere all'interno di un blocco Comando?
 - un'istruzione elementare \rightarrow ovvero lo decidiamo noi (basta non essere ambigui per l'esecutore e ricordarsi che si esegue un comando alla volta)
- Possiamo assumere di avere a disposizione tutti gli operatori aritmetici per formare espressioni che usiamo normalmente.
- Ecco un esempio di sintassi:
 - $(2+5)*7-10/3+2^3$
 - Quoziente della divisione tra a e b: $a \text{ div } b$
 - $7 \text{ div } 2 = 3$
 - Resto della divisione tra a e b: $a \% b$ ($a \text{ mod } b$)
 - $7 \% 2 = 1$

- I dati di ingresso e i risultati vengono rappresentati attraverso dei nomi simbolici, detti **variabili**, ad esempio x
- Possiamo utilizzare simboli nelle espressioni aritmetiche (e logiche):
 - $3x+5$
- A differenza dell'uso che facciamo in matematica, dove x può rappresentare un valore indefinito,
- A tutte le variabili che compaiono in un'espressione deve essere stato associato un valore prima della valutazione dell'espressione
- altrimenti violo il principio che il diagramma non sia ambiguo
- Come assegnamo o modifichiamo il valore ad una variabile?

Pseudocodice - Variabili



- Come assegniamo un valore ad una variabile?
 - $x=2$ // cambiamo il valore di x in 2 \rightarrow d'ora in poi ogni volta che x appare in un'espressione possiamo sostituirla con 2
-  • $y=4x$ // OK perché $x=2$ e quindi $y=8$
-  • $y=4+z$ // NO perché non sappiamo che valore abbia z (e quindi y)
- $x=2; x=x+1$ // $x=3$, la stessa variabile può comparire a sinistra e alla destra in un assegnamento
- Chiamiamo x e y variabili intere (perché contengono un numero)
- Possiamo avere variabili che rappresentano valori booleani (vero, falso)
- Non le combiniamo assieme (sarebbe come sommare una velocità e un peso)

- Una variabile può rappresentare una lista ordinata di elementi
 - $L = [1,4,5,6]$
- Ogni elemento della lista è identificato da un indice, un numero
 - $L[1] \rightarrow 1$; $L[2] \rightarrow 4$; $L[4] \rightarrow 6$
 - es. l'elemento di indice 2 di L ha valore 4
- $|L| \rightarrow$ numero di elementi della lista L
- $L = [1,4,7,6,8,12]$; $L[2:4] \rightarrow [4,7,6]$ sottolista di L dall'indice 2 all'indice 4 compresi

- Con le variabili abbiamo usato simboli per rappresentare valori interi o booleani
- Allo stesso modo, se una parte di un diagramma di flusso risolve un (sotto)problema, possiamo dargli un nome e riusarlo in seguito

Fai Nodo alla Cravatta



1. Posiziona la cravatta con il lato destro in alto sotto il colletto. L'estremità grande dovrebbe essere a sinistra e l'estremità piccola a destra. Assicurati di lasciare più lunghezza per la tua estremità grande.
2. Incrocia il lato grande su quello piccolo.
3. Fai scorrere l'estremità grande attorno all'estremità piccola della cravatta.
4. Porta l'estremità più grande della cravatta sull'estremità più piccola.
5. Passa il lato più grande sotto la cravatta posizionando il dito indice nel nodo che si forma.
6. Togli l'indice e fai scorrere l'estremità grande attraverso il passante.
7. Regola la cravatta. Deve fermarsi prima della cintura dei pantaloni e il lato piccolo non deve essere visibile.



Vestirsi per Matrimonio



Pseudocodice

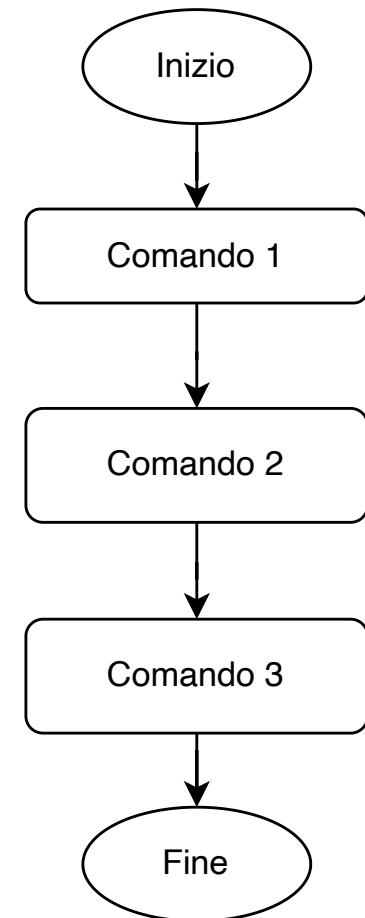


- Lo pseudocodice è la versione testuale di un diagramma di flusso
- I nodi inizio e fine non vengono tradotti nello pseudocodice
- I nodi Input/Output diventano i comandi leggi/scrivi
- il contenuto di ogni blocco comando viene copiato letteralmente: un comando per riga o separando due comandi con ;

Es.

comando 1

comando 2; comando 3



- Proprio come accade in matematica, una funzione può calcolare e restituire un valore e/o avere parametri su cui operare
- In pseudocodice possiamo indicare una funzione con: funzione + nome seguito da una lista di parametri su cui opera tra parentesi
 - funzione somma(x,y)
- se la funzione calcola un valore, indichiamo tale valore precedendolo dalla parola return/restituisce (l'esecuzione della funzione termina non appena si è eseguita una restituisci – anche se ci sono altri comandi dopo)

Es.

```
funzione somma(x,y) {  
    restituisci x+y  
}  
x = somma(3, 7) // x=10
```

- Possiamo associare dei comandi alle liste tramite la definizione di funzioni:
- Possiamo rimuovere un elemento da una lista indicando il suo indice
 - $L = [1,4,5,6]; \text{rimuovi}(L, 2) \rightarrow L=[1,5,6]$
 - $L = [1,4,5,6]; L.\text{rimuovi}(2) \rightarrow L=[1,5,6]$
- Possiamo aggiungere un elemento ad una lista
 - $L = [1,4,5,6]; L.\text{appendi}(3) \rightarrow [1,4,5,6,3]$ // aggiunto alla fine della lista
 - $L = [1,4,5,6]; L.\text{aggiungi}(3, 2) \rightarrow [1,3,4,5,6]$ // aggiunto 3 in posizione 2

Pseudocodice - Esempio

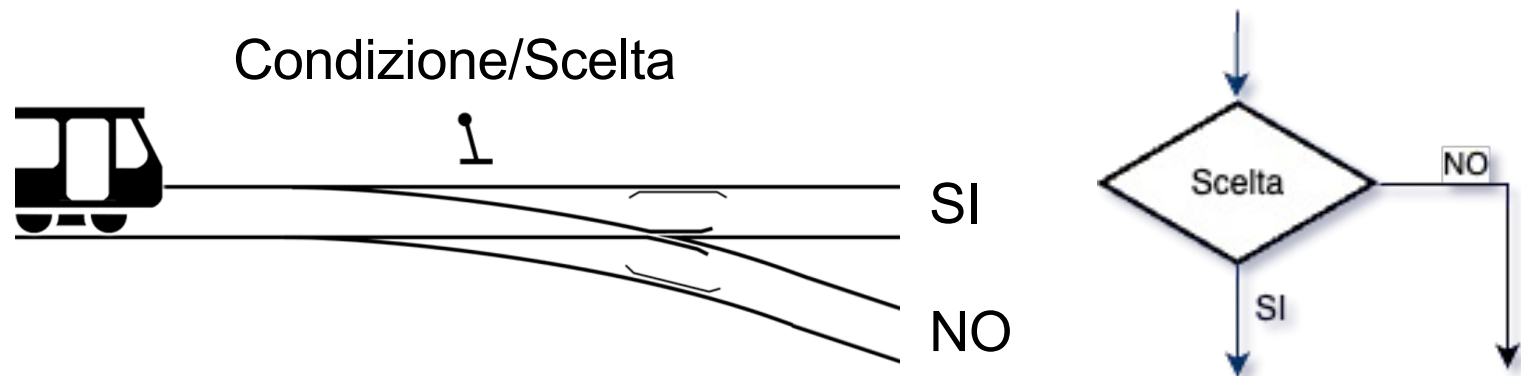


Trasformare il valore in gradi fahrenheit della variabile fahrenheit (X) nel corrispondente valore celsius (Y) e stampare
"X gradi fahrenheit corrispondono a Y gradi celsius"

leggi x

$$y = 5/9(x - 32)$$

scrivi x "gradi fahrenheit corrispondono a" y "gradi celsius"



- La strada che l'esecuzione percorre dipende dalla Scelta o Condizione:
 - ovvero una domanda la cui risposta si calcola in un tempo finito la cui valutazione può essere solamente sì o no (vero/falso)
 - chiamiamo queste Condizioni espressioni booleane
 - Esempi: $5 > 3$? $5 \leq 2$? $4 = 4$? $x > 7$?
 - Se usciamo dal rombo dal SI, la condizione è vera, se usciamo dal NO, l'opposto della condizione è vero

Operatori Booleani



- Gli operatori di confronto possono essere combinati tra loro tramite NOT, AND, OR per ottenere espressioni complesse.
- Poiché i valori di output di un'espressione booleana sono 2, si possono elencare tutti i possibili input/output tramite una tabella (detta tabella di verità):

a	NOT a
vero	falso
falso	vero

a	b	a AND b
vero	vero	vero
vero	falso	falso
falso	vero	falso
falso	falso	falso

a	b	a OR b
vero	vero	vero
vero	falso	vero
falso	vero	vero
falso	falso	falso

Risoluzione di Espressioni Booleane



- Con lo stesso metodo, è possibile calcolare funzioni booleane complesse come $(a \text{ OR } b) \text{ OR NOT}(a \text{ AND } b)$:

a	b	a OR b	a AND b	NOT (a AND b)	(a OR b) OR NOT(a AND b)
vero	vero	vero	vero	falso	
vero	falso	vero	falso	vero	
falso	vero	vero	falso	vero	
falso	falso	falso	falso	vero	

Risoluzione di Espressioni Booleane



- $(a \text{ OR } b) \text{ OR } \text{NOT}(a \text{ AND } b)$ è un OR tra la colonna 3 “ $(a \text{ OR } b)$ ” e la colonna 5 “ $\text{NOT}(a \text{ AND } b)$ ”

a	b	a OR b	a AND b	NOT (a AND b)	(a OR b) OR NOT(a AND b)
vero	vero	vero	vero	falso	vero
vero	falso	vero	falso	vero	
falso	vero	vero	falso	vero	
falso	falso	falso	falso	vero	

Risoluzione di Espressioni Booleane



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

a	b	a OR b	a AND b	NOT (a AND b)	(a OR b) OR NOT(a AND b)
vero	vero	vero	vero	falso	vero
vero	falso	vero	falso	vero	vero
falso	vero	vero	falso	vero	
falso	falso	falso	falso	vero	

Risoluzione di Espressioni Booleane



a	b	a OR b	a AND b	NOT (a AND b)	(a OR b) OR NOT(a AND b)
vero	vero	vero	vero	falso	vero
vero	falso	vero	falso	vero	vero
falso	vero	vero	falso	vero	vero
falso	falso	falso	falso	vero	

Risoluzione di Espressioni Booleane



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

a	b	a OR b	a AND b	NOT (a AND b)	(a OR b) OR NOT(a AND b)
vero	vero	vero	vero	falso	vero
vero	falso	vero	falso	vero	vero
falso	vero	vero	falso	vero	vero
falso	falso	falso	falso	vero	vero

Risoluzione di Espressioni Booleane



- Per determinare se due funzioni sono uguali, si mostra che tutte le coppie input/output sono identiche \rightarrow due formule logiche sono equivalenti se le tabelle di verità sono identiche
- Es. “ $(a \text{ OR } b) \text{ OR NOT}(a \text{ AND } b)$ ” equivale a “ $b \text{ OR NOT}(b)$ ”?

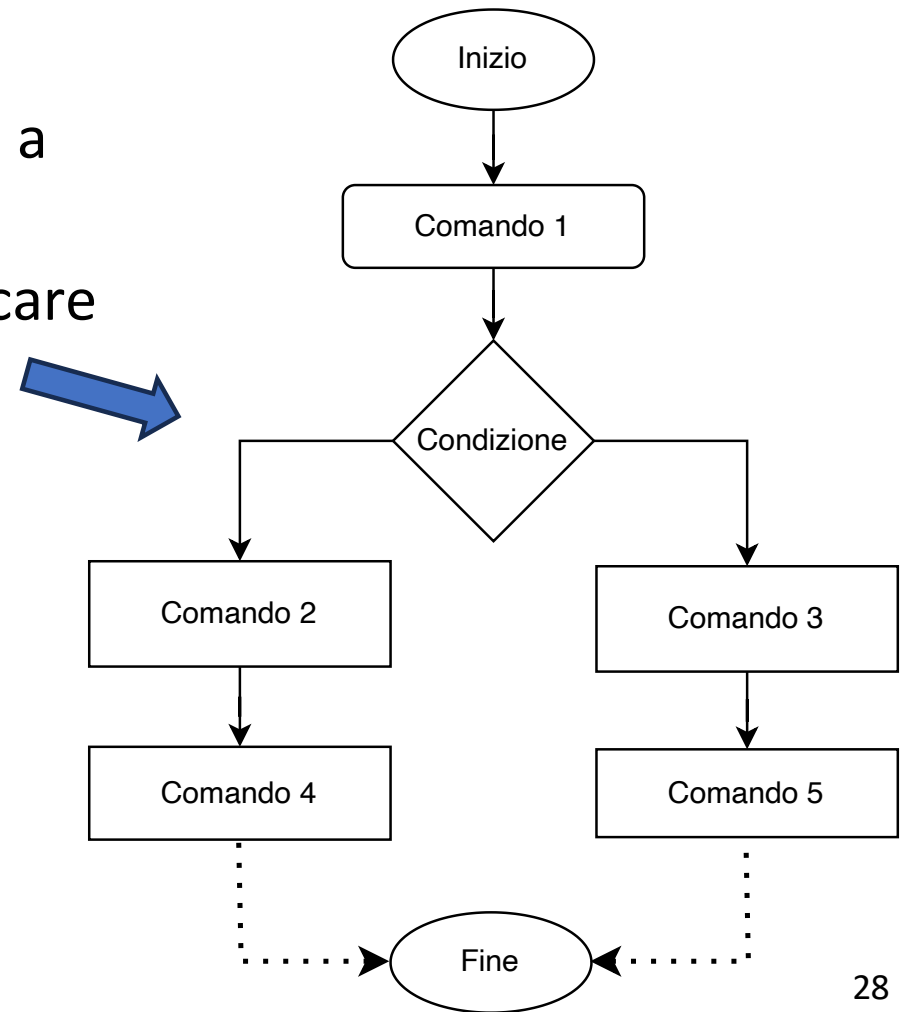
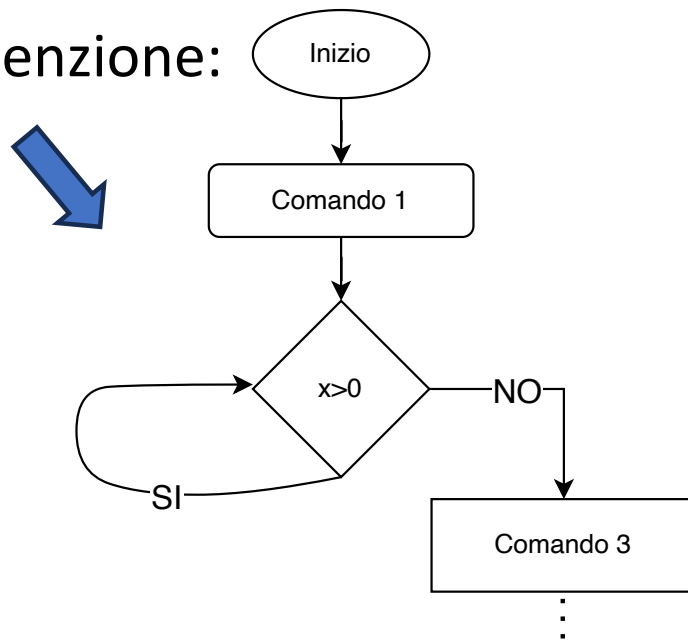
a	b	NOT b	b OR (NOT b)	$(a \text{ OR } b) \text{ OR NOT}(a \text{ AND } b)$
vero	vero	falso	vero	vero
vero	falso	vero	vero	vero
falso	vero	falso	vero	vero
falso	falso	vero	vero	vero

- Si perché, per ogni coppia (a,b) il risultato delle due formule è lo stesso (sempre vero)

Selezione



- I programmi generalmente permettono di eseguire funzionalità diverse (per esempio a seconda delle interazioni con l'utente)
- La selezione (rombo) permette di diversificare l'esecuzione a seconda della condizione
- Ma attenzione:



Pseudocodice per Selezione



- Se [condizione] allora {
 lista comandi se [condizione] è vera
} altrimenti {
 lista comandi se [condizione è falsa]
} //possiamo evitare {} se indentiamo il codice

Esempio:

Comando 1

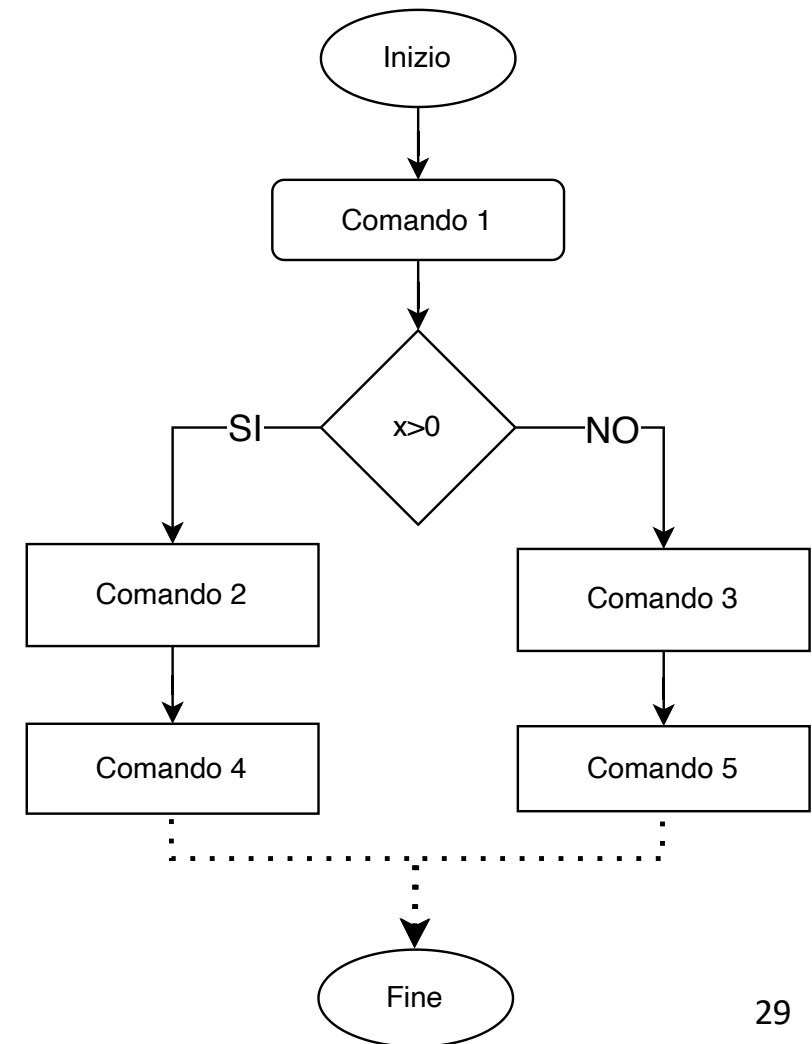
Se $x > 0$ {

 comando 2

 comando 4

} altrimenti

 comando 3; comando 4



Pseudocodice per Selezione



- Se [condizione] allora {
 lista comandi se [condizione] è vera
} altrimenti {
 lista comandi se [condizione è falsa]
} //possiamo evitare {} se indentiamo il codice

Esempio:

Comando 1

Se $x > 0$ {

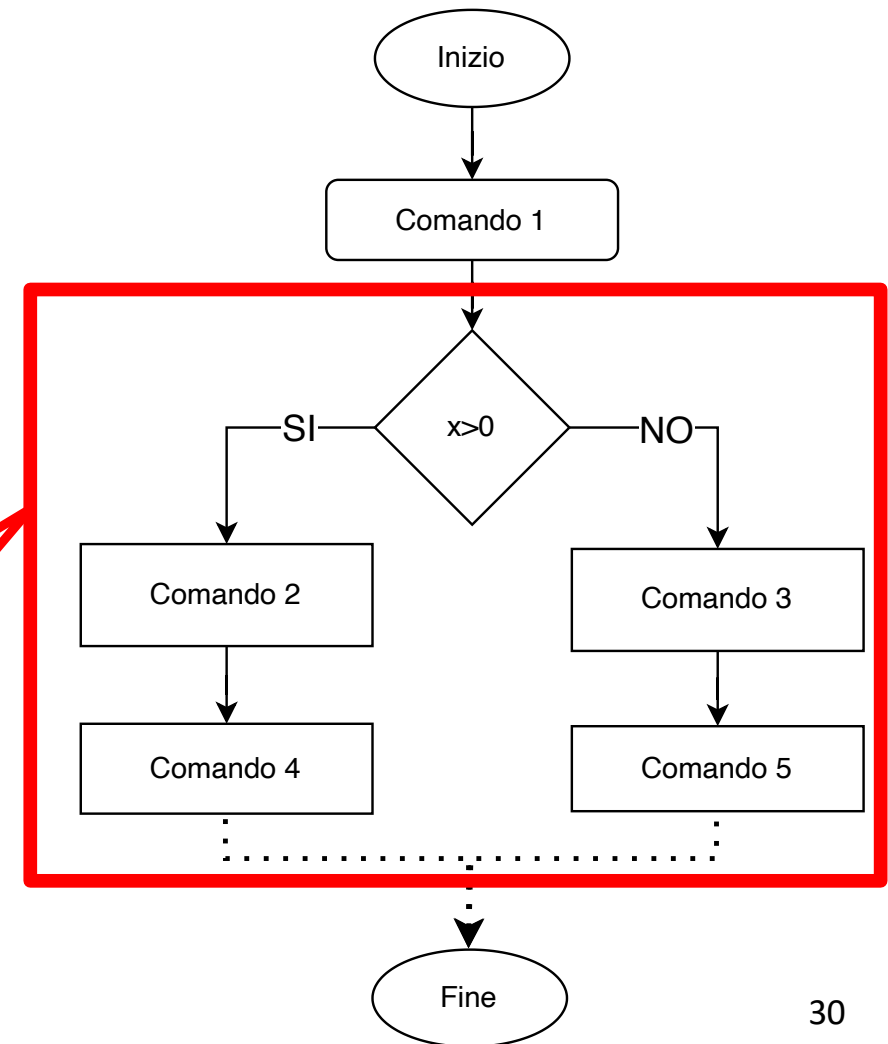
 comando 2

 comando 4

} altrimenti

 comando 3; comando 4

il Se termina
quando i due rami
si ricongiungono



Selezione: Varianti e Sintassi

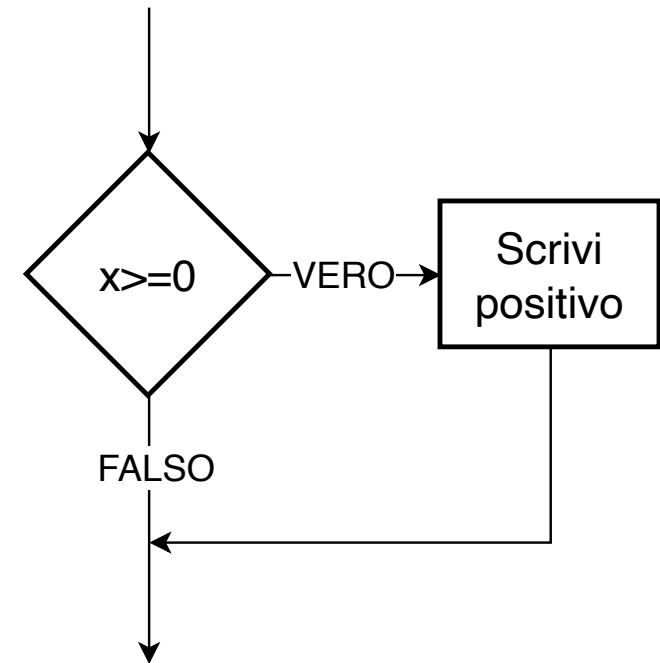


```
Se [condizione] {  
    //comandi da eseguire se la condizione è vera  
}  
//comando2
```

Esempio:

```
Se  $x \geq 0$  {  
    scrivi "positivo"  
}  
//comando2
```

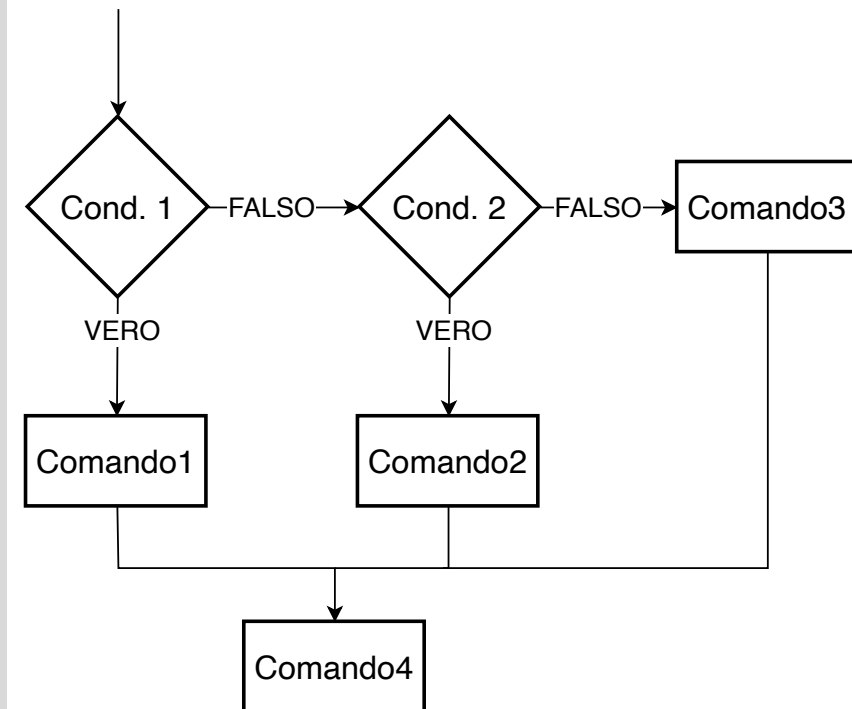
Uno dei due rami può essere “vuoto”



IF: Varianti e Sintassi



```
Se [condizione1] {  
    Comando1      // cond.1 è vera  
} altrimenti  
    se [condizione2] {  
        Comando2 // cond.1 falsa, cond.2 vera  
    } altrimenti {  
        Comando3      // condizion1-2 false  
    }  
Comando4
```

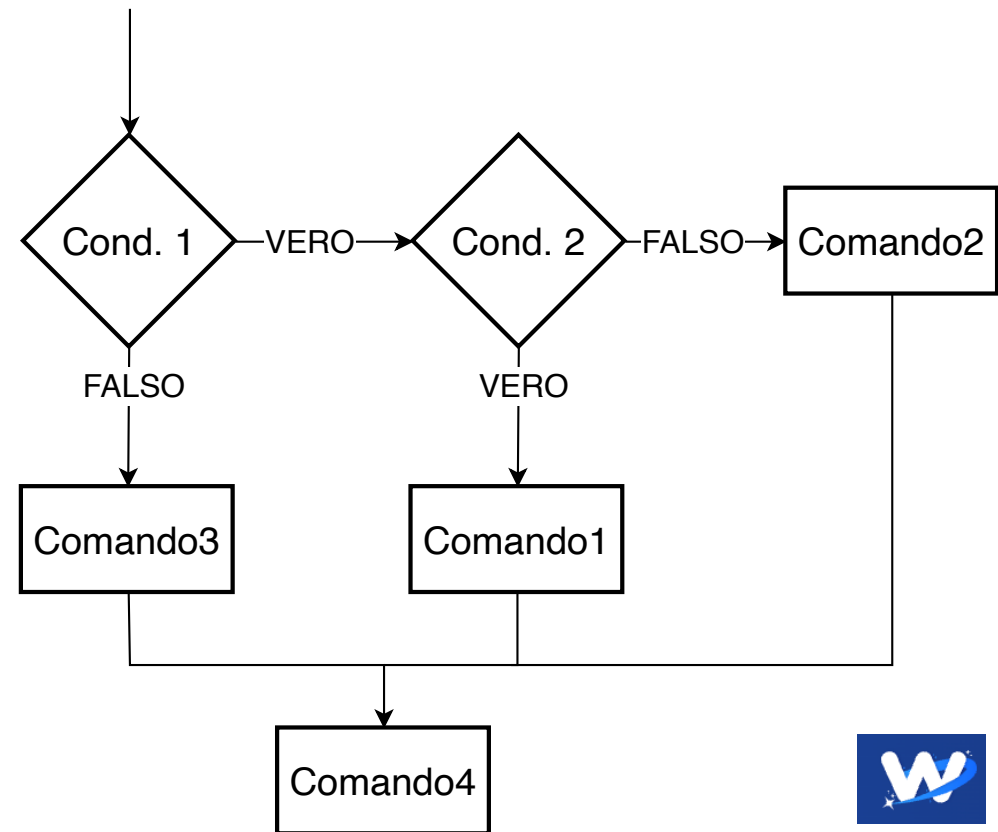


È possibile comporre comandi Se, notate che solo uno tra Comando1,...,Comando3 viene eseguito.

IF: Varianti e Sintassi



```
Se [condizione1] {  
    Se [condizione2] {  
        Com1 // cond.1 e cond.2 vere  
    } altrimenti {  
        Com 2 // cond.1 vera,  
              // cond.2 falsa  
    }  
} altrimenti {  
    Com3 // cond.1 falsa  
}
```



È possibile utilizzare più Se uno dentro l'altro

Esercizio



Date 3 variabili intere: x, y, z , stampare un numero che è Il valore minimo tra le 3 variabili.

Es. se $x=5, y=2, z=7$ stampa

"Il minore dei tre valori è 2 "

leggi x, y, z

scrivi "Il minore dei tre valori è "

...

Esercizio



Date 3 variabili intere: x, y, z , stampare un numero che è il valore minimo tra le 3 variabili

Es. se $x=5, y=2, z=7$ stampa

"Il minore dei tre valori è 2 "

$x=5, y=2, z=7 \rightarrow$ Il minore dei tre valori è 2

$x=1, y=2, z=7 \rightarrow$ " 1

$x=5, y=2, z=1 \rightarrow$ " 1

PRE: –

POST: stampato valore minore tra x, y, z (var. numeriche)

Esercizio



Date 3 variabili intere: x, y, z , stampare un numero che è il valore minimo tra le 3 variabili.

Es. se $x=5, y=2, z=7$ stampa "Il minore dei tre valori è 2"

Esempi di input \rightarrow output

$x=5, y=2, z=7 \rightarrow$ Il minore dei tre valori è 2

$x=1, y=2, z=7 \rightarrow$ " 1

$x=5, y=2, z=1 \rightarrow$ " 1

$x=1, y=2, z=1 \rightarrow$ " 1

$x=2, y=1, z=1 \rightarrow$ " 1

$x=1, y=1, z=2 \rightarrow$ " 1

$x=1, y=1, z=1 \rightarrow$ " 1

Esercizio – Una Soluzione



```
leggi x, y, z
scrivi "Il minore dei tre valori è "
Se x < y {
    Se x < z
        scrivi x
    altrimenti
        scrivi z
} altrimenti {
    Se y < z
        scrivi y
    altrimenti
        scrivi z
}
```

Esercizio: Test



Input \rightarrow output aspettato – output ottenuto (per ragioni di spazio non ripeto la stringa “Il minore dei tre valori è”):

$x=5, y=2, z=7 \rightarrow$	Il minore dei tre valori è	2 – 2
$x=1, y=2, z=7 \rightarrow$	“	1 – 1
$x=5, y=2, z=1 \rightarrow$	“	1 – 1
$x=1, y=2, z=1 \rightarrow$	“	1 – 1
$x=2, y=1, z=1 \rightarrow$	“	1 – 1
$x=1, y=1, z=2 \rightarrow$	“	1 – 1
$x=1, y=1, z=1 \rightarrow$	“	1 – 1

- La nostra soluzione non ha fallito alcun test, proviamo a convincerci che è corretta per ogni possibile input

Esercizio - Soluzione



```
leggi x, y, z
scrivi "Il minore dei tre valori è "
Se x < y {
    Se x < z
        scrivi x // x<y, x<z → ok scrivere x
    altrimenti
        scrivi z // x<y, z<=x → ok scrivere z
} altrimenti {
    Se y < z
        scrivi y // y<=x, y<z → ok scrivere y
    altrimenti
        scrivi z // z<=y<=x → ok scrivere z
}
```

Cosa rimane chiederci per verificare la POST?

Esercizio - Soluzione



```
leggi x, y, z
scrivi "Il minore dei tre valori è "
Se x < y {
    Se x < z
        scrivi x // x<y, x<z → ok scrivere x
    altrimenti
        scrivi z // x<y, z<=x → ok scrivere z
} altrimenti {
    Se y < z
        scrivi y // y<=x, y<z → ok scrivere y
    altrimenti
        scrivi z // z<=y<=x → ok scrivere z
}
```

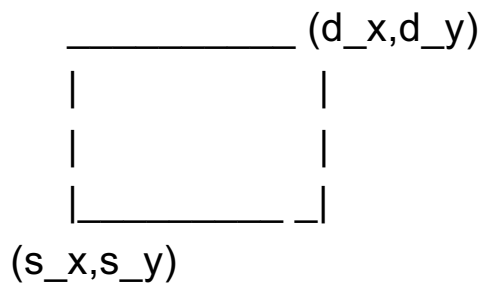
- Succede mai che non stampiamo nessuna tra x,y o z? No, ogni Se ha un ramo *altrimenti*, per cui almeno una *scrivi* viene sempre eseguita
- Succede mai che stampiamo più di una tra x,y,z? No, per come sono disposti i Se, se ne esegue solo uno e quindi una sola *scrivi* (verificate che dopo ogni scrivi non si esegue alcun comando)

Esercizio



Dato un piano cartesiano nel quale è disegnato un rettangolo, identificato dalle coordinate del punto più in basso a sinistra $s=(s_x,s_y)$ e dal punto in alto a destra $d=(d_x,d_y)$, e un punto di coordinate (p_x,p_y) , stampare

- “ (p_x,p_y) interno al rettangolo” se (p_x,p_y) è all'interno del rettangolo (i punti sul bordo non fanno parte dell'interno del rettangolo)
- “ (p_x,p_y) esterno al rettangolo” altrimenti



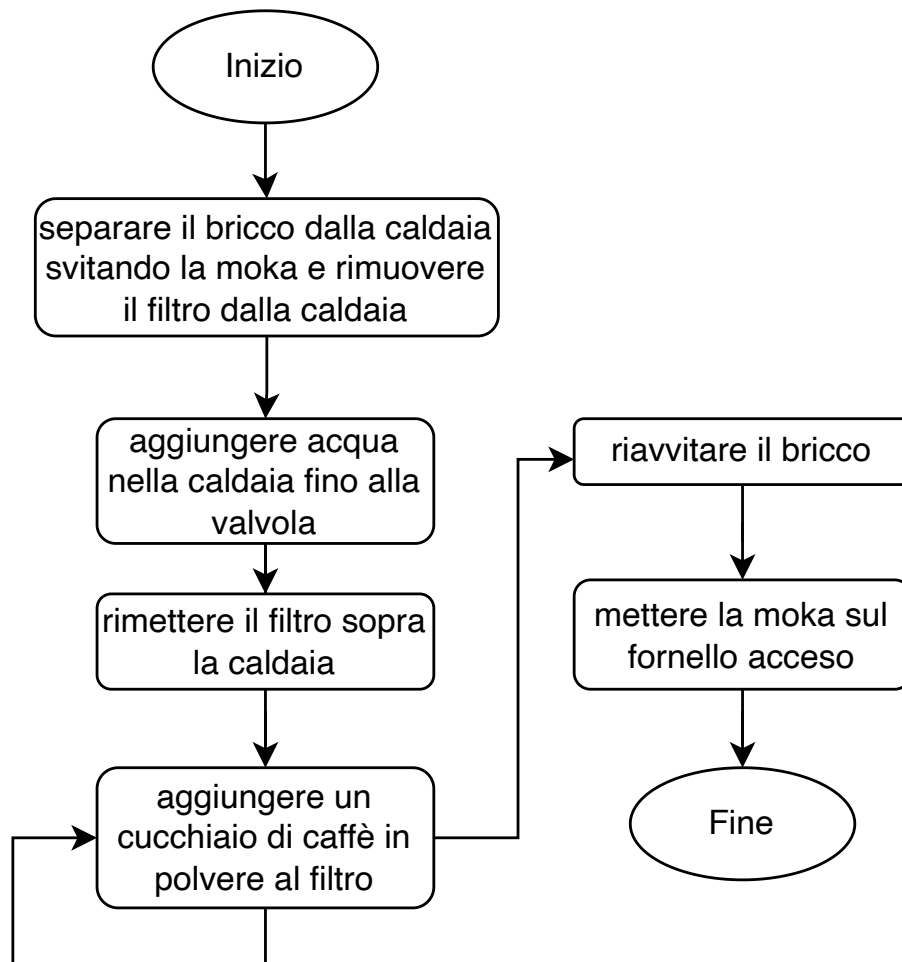
Esempi: se $s=(1,1)$, $d=(4,2)$ e $p=(3,1.5)$, stampa “ $(3, 1.5)$ interno al rettangolo”
se $s=(1,1)$, $d=(4,2)$ e $p=(3,2)$, stampa “ $(3, 2)$ esterno al rettangolo”

Esercizio - Soluzione



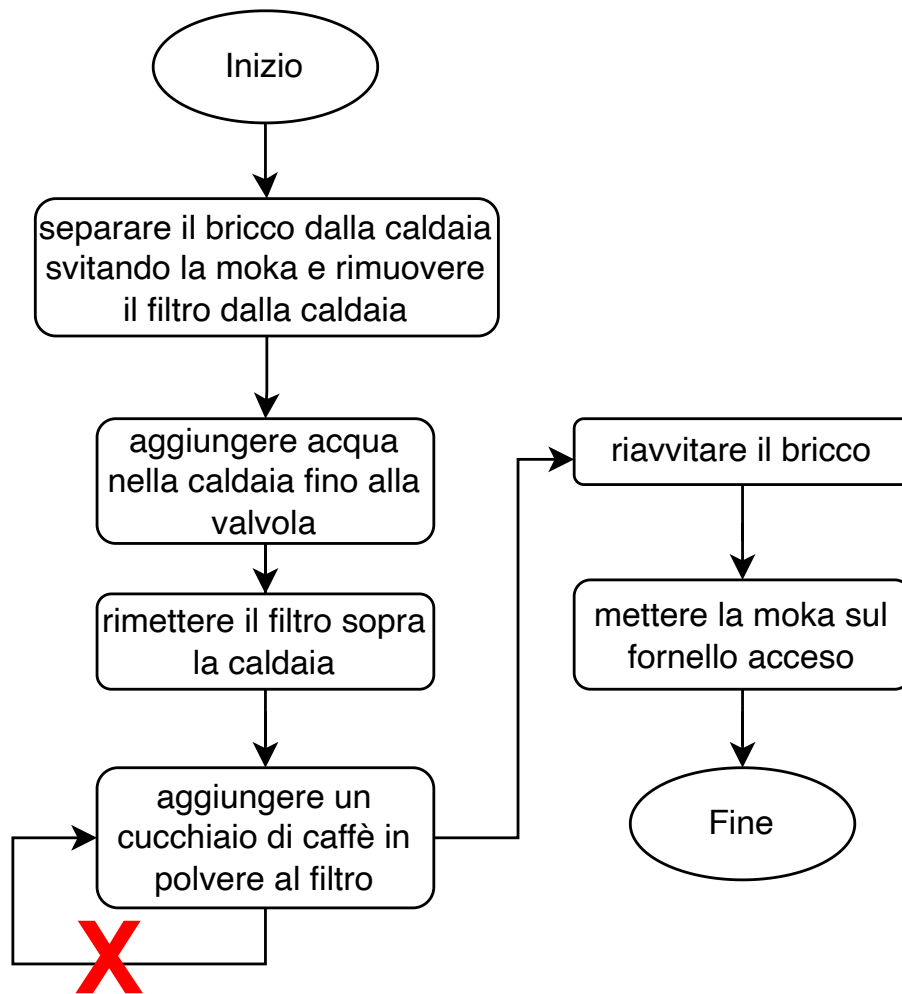
```
/*  
    PRE: s_x < d_x, s_y < d_y  
    POST: (s_x, s_y), (d_x, d_y) coordinate del vertice in basso  
a sinistra / in alto a destra di un rettangolo, stampa (p_x,  
p_y) interno/esterno al rettangolo a seconda che (p_x, p_y) sia  
interno/esterno al rettangolo.  
*/  
funzione interno_rettangolo(s_x, s_y, d_x, d_y, p_x, p_y) {  
    Se (p_x > s_x and p_x < d_x) and  
        (p_y > s_y and p_y < d_y)  
        scrivi (p_x, p_y) interno al rettangolo  
    altrimenti  
        scrivi (p_x, p_y) esterno al rettangolo  
}
```

Diagrammi di Flusso



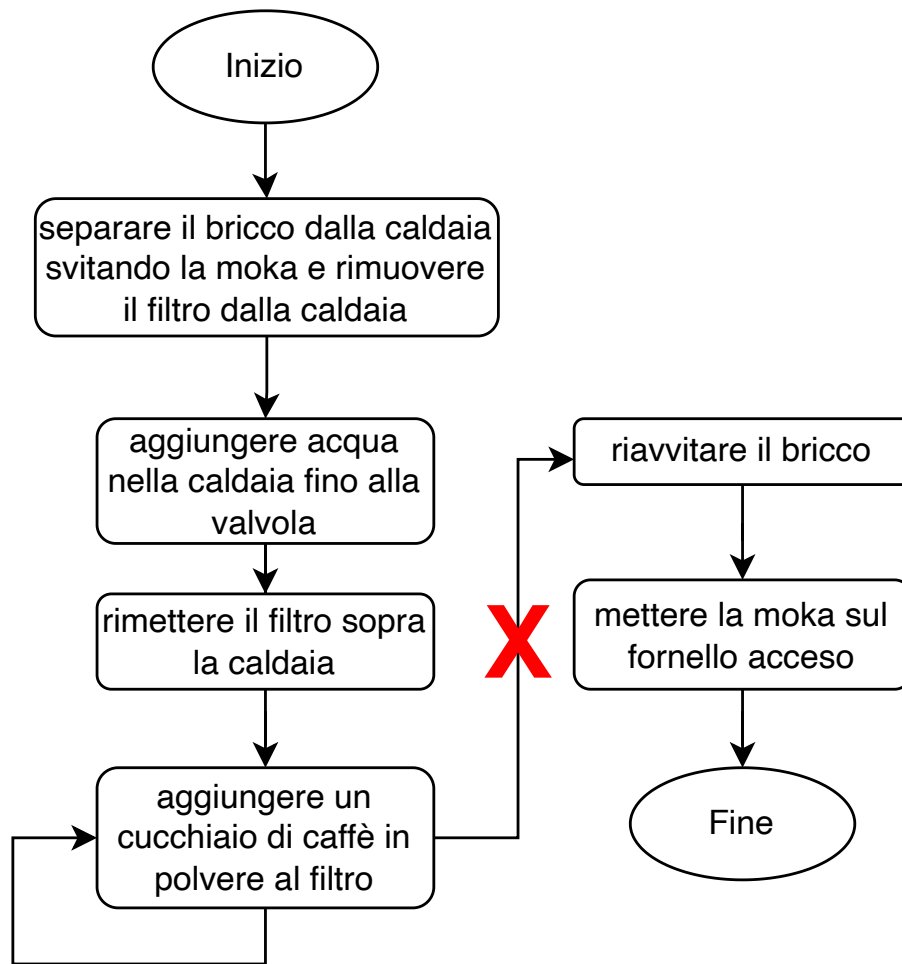
- Abbiamo già visto che il rombo permette di creare due percorsi diversi
- Potremmo pensare che possiamo collegare a piacimento ogni blocco con qualsiasi altro che abbiamo già definito
- Ma l'esempio non è corretto, Il blocco in basso ha due uscite!
- L'esecutore non sa che strada prendere

Diagrammi di Flusso



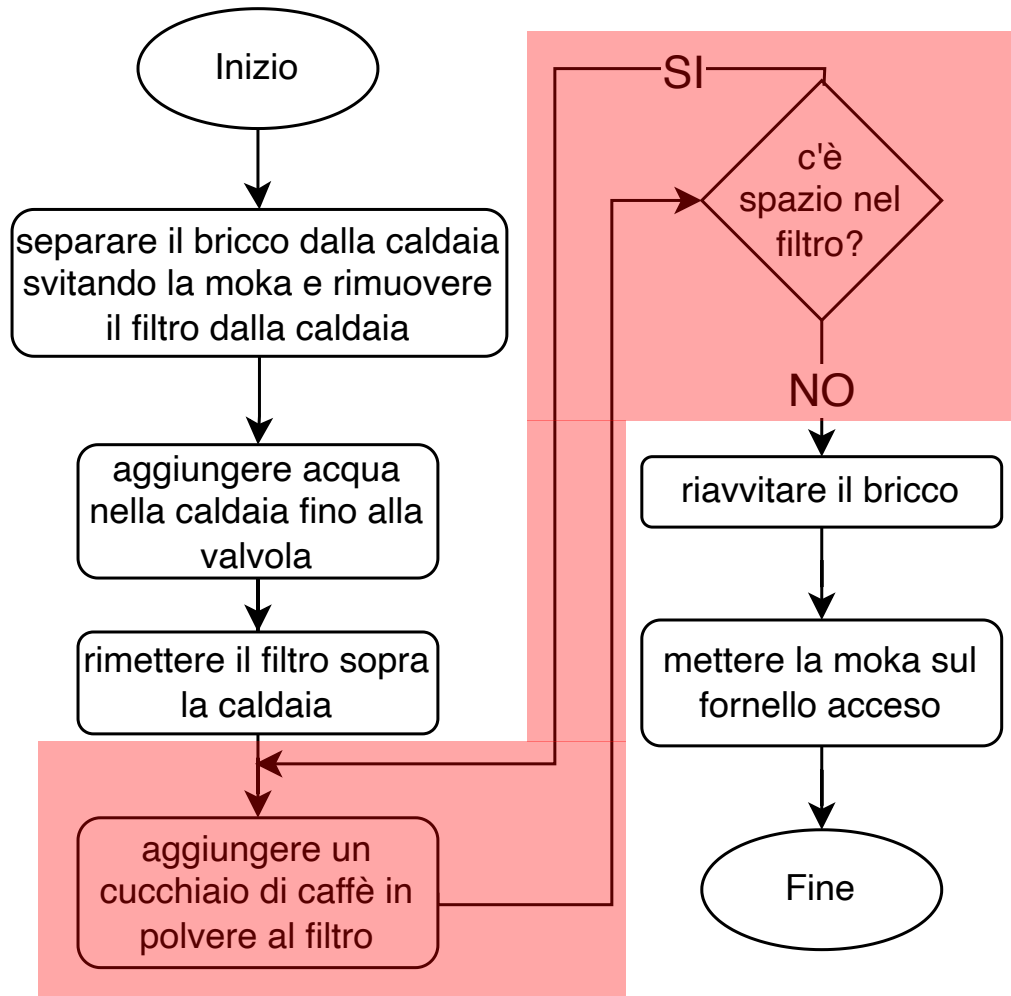
- Corretto ma abbiamo un percorso lineare
- non più due percorsi nel diagramma di flusso

Diagrammi di Flusso



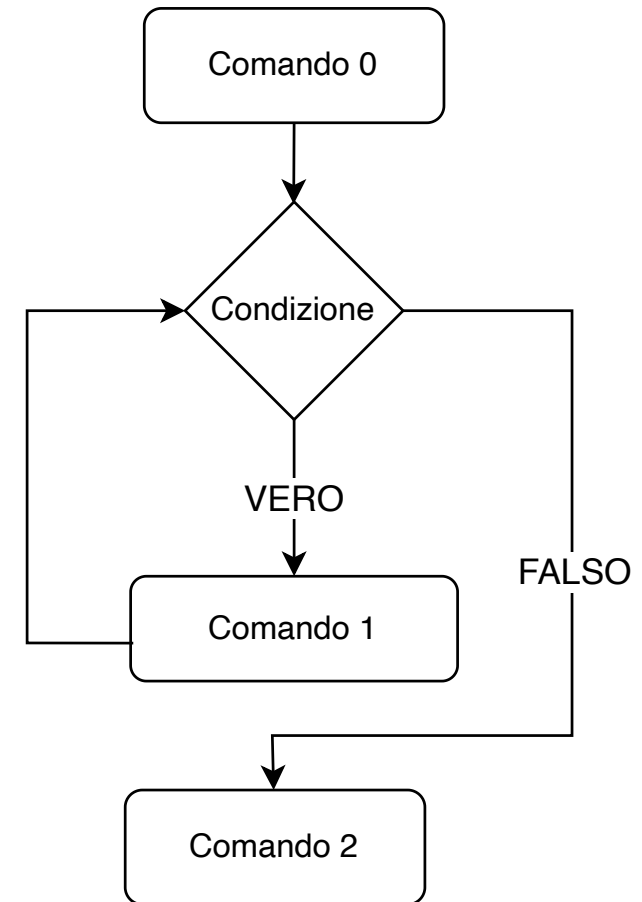
- Non corretto, non si raggiunge mai il nodo fine
- l'unico modo per collegare un nodo con uno precedente
- Cosa succede se invece usiamo il rombo e colleghiamo una (sola) delle uscite ad un nodo già definito?

Diagrammi di Flusso



- Il diagramma è valido (rispetta le nostre regole)
- Cosa calcola la parte evidenziata?
- “aggiungere caffè finché non si riempie il filtro”
- più in generale si ripetono uno o più comandi finché la condizione è vera

- In molti casi si presenta la necessità di ripetere più volte una serie di istruzioni
 - es. Stampare "Ciao Mondo!" molte volte
 - Spesso non sappiamo a priori quante volte dobbiamo ripetere una sequenza di istruzioni!
- Lo schema a fianco è talmente frequente che ha associato un proprio comando in pseudocodice:
- ```
ripeti finché [condizione] {
 Comando 1
}
Comando 2
```





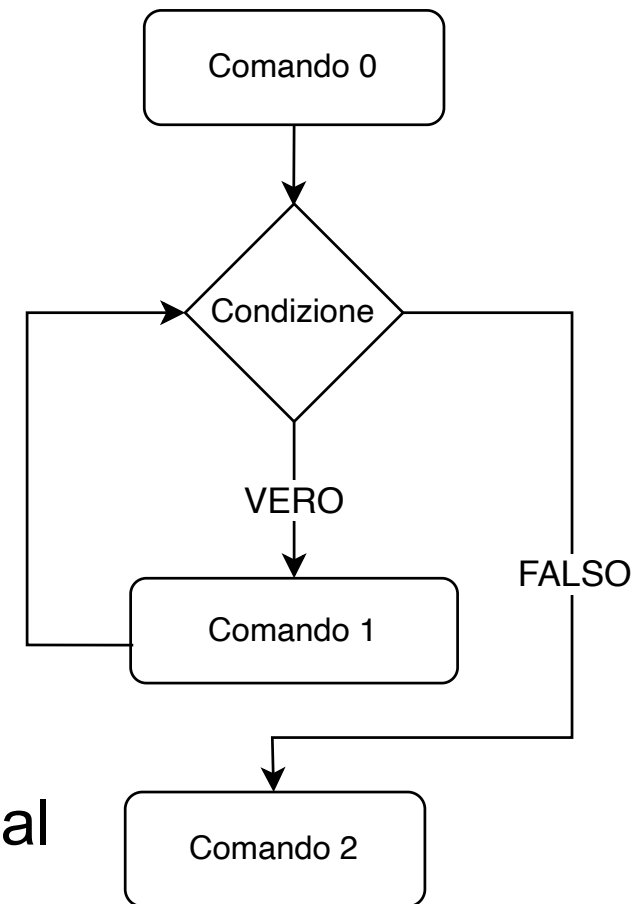
# Iterazione: while



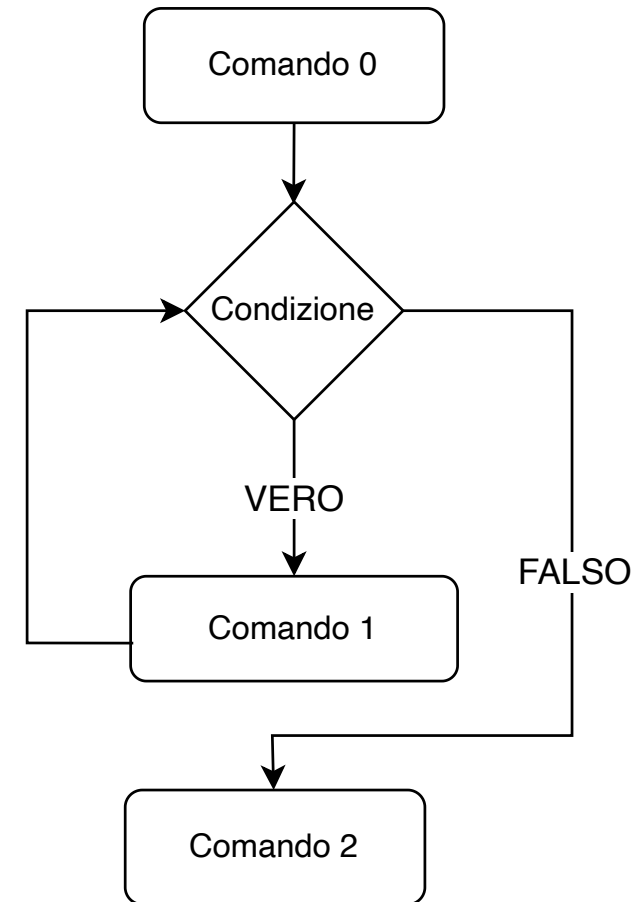
```
ripeti finché [condizione] {
 Comando 1 //anche più di un comando
}
Comando 2
```

Il comando ripeti:

1. se *condizione* è falsa, non esegue i comandi all'interno del blocco e passa a comando2
2. se *condizione* è vera, esegue i comandi all'interno del blocco {}
3. Una volta eseguiti i comandi del blocco, ritorna al punto 1



- I costrutti iterativi consistono dei seguenti elementi
- un comando di inizializzazione (Comando 0): indica le condizioni iniziali delle variabili coinvolte nel ciclo
- una condizione di continuazione che indica se l'esecuzione del ciclo debba continuare
- il corpo del ciclo: la sequenza di istruzioni da ripetere potenzialmente più volte, tra cui
- istruzione iterativa: fa sì che il ciclo prosegua verso la terminazione

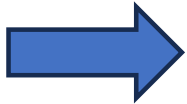


# Iterazione: Esempio



- Stampare i numeri da 1 a 10

scrivi 1  
scrivi 2  
scrivi 3  
...  
scrivi 10



```
i=1; scrivi i
i=2; scrivi i
i=3; scrivi i
...
i=10; scrivi i
```

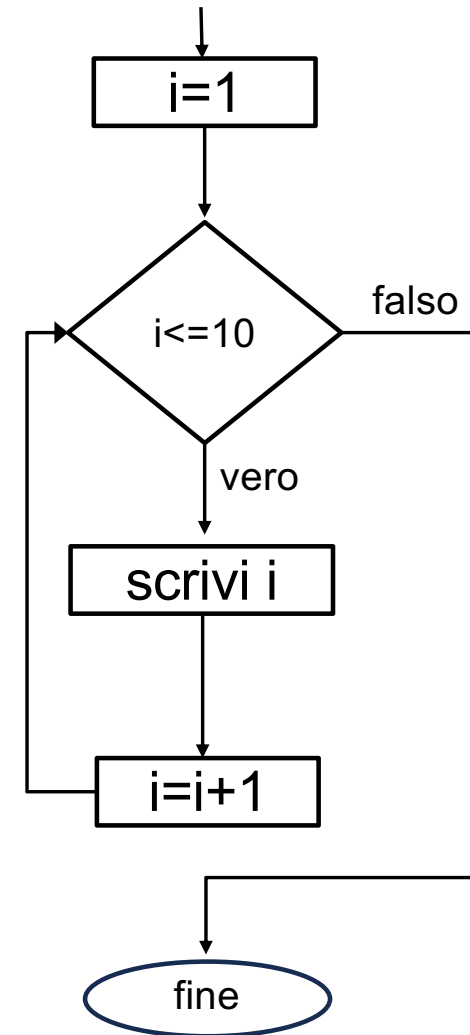


abbiamo espresso la nostra serie di comandi iniziale tramite lo stesso comando: scrivi i; tutto quello che dobbiamo fare è var variare la variabile i da 1 a 10 e richiamare il comando che si ripete.

# Iterazione: Esempio



- Stampare i numeri da 1 a 10
1. inizializzare una variabile, es.  $i=1$ .
  2. finché  $i$  è minore o uguale a 10
    - a) stampa  $i$
    - b) incrementa  $i$  di 1
    - c) ritorna al punto 2



# Iterazione: Esempio



- Stampare i numeri da 1 a 10
- 1. inizializzare una variabile, es.  $i=1$
- 2. finché  $i$  è minore o uguale a 10
  - 1. stampa  $i$
  - 2. incrementa  $i$  di 1
  - 3. ritorna al punto 2

```
i=1
ripeti finché i<=10 {
 scrivi i
 i=i+1
}
```

# Iterazione: Esempio



- Se si rimuove l'istruzione  $i=i+1$ ; cosa succede?

?

```
i=1
ripeti finché i<=10 {
 scrivi i
 i=i+1
}
```

# Iterazione: Esempio



- Se si rimuove l'istruzione  $i=i+1$ ;
- l'esecuzione del ciclo non termina mai perché  $i$  è sempre uguale a 1 e perciò la condizione  $i \leq 10$  è sempre vera

```
i=1
ripeti finché i<=10 {
 scrivi i
 i=i+1
}
```

- È utile chiedersi:
  - Quante volte viene eseguito il corpo di un ciclo?
  - Quale valore ha la variabile di iterazione (a) la prima volta che viene usata nel corpo del ciclo?
  - Quale valore ha l'ultima volta?
  - Quale valore assume dopo l'uscita del ciclo?

```
a=0;
ripeti finché a<5 {
 scrivi a
 a = a+2;
}
```

// subito dopo il ciclo sappiamo che:  $a \geq 5$



- È utile chiedersi:
  - Quante volte viene eseguito il corpo di un ciclo? 3
  - Quale valore ha la variabile di iterazione (a) la prima volta che viene usata nel corpo del ciclo? 0
  - Quale valore ha l'ultima volta? 4
  - Quale valore assume dopo l'uscita del ciclo? 6

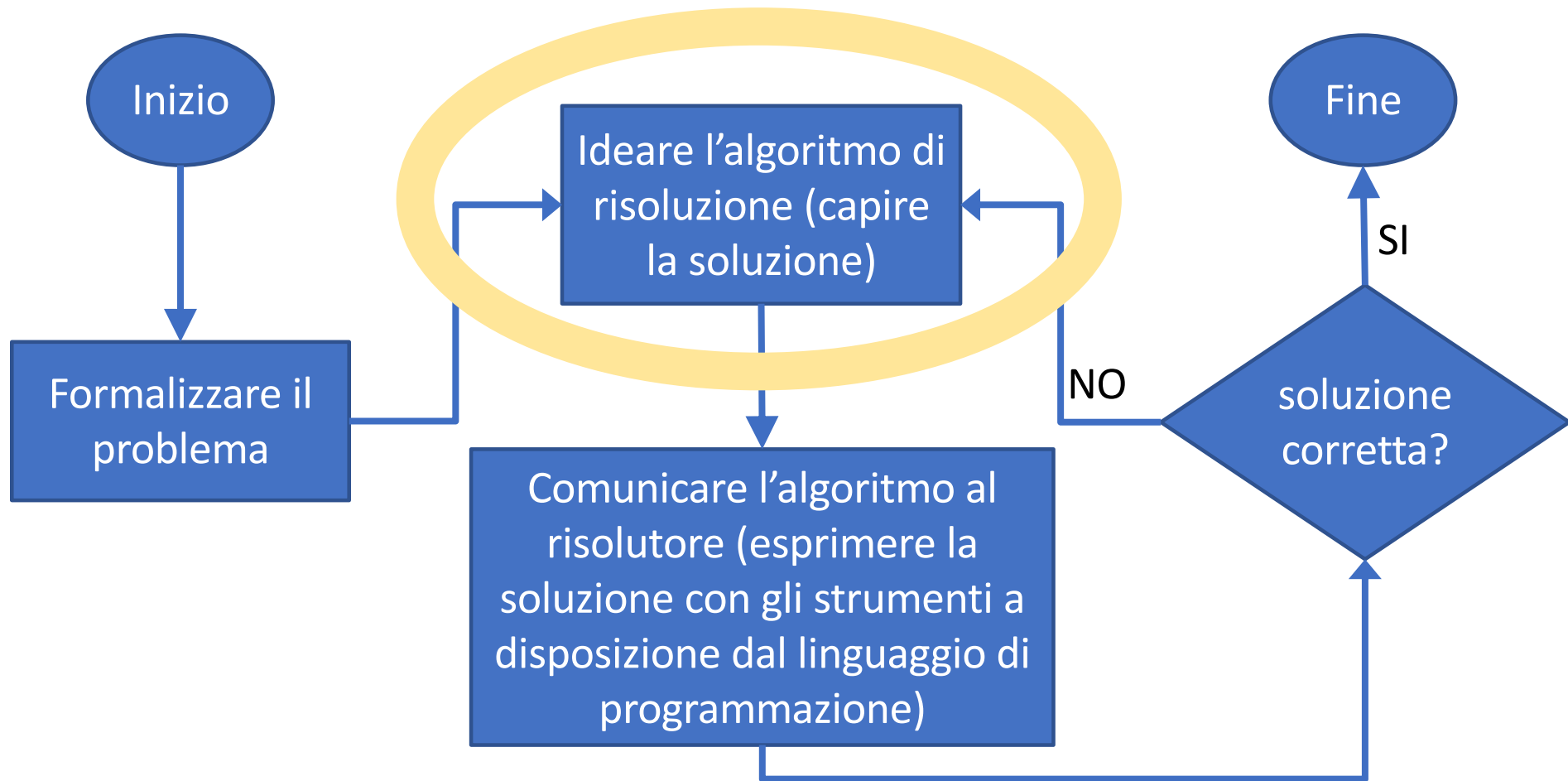
```
a=0;
ripeti finché a<5 {
 scrivi a
 a = a+2;
}
```

// subito dopo il ciclo sappiamo che:  $a \geq 5$

# Strategie di Risoluzione di Problemi



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA



# Scambio di Variabili



- Per risolvere un problema, possiamo dover creare nuove variabili oltre a quelle per rappresentare l'input.
- Consegna: date due variabili  $x, y$ , scambiare i valori
  - Es.  $x=5, y=7 \rightarrow x=7, y=5$

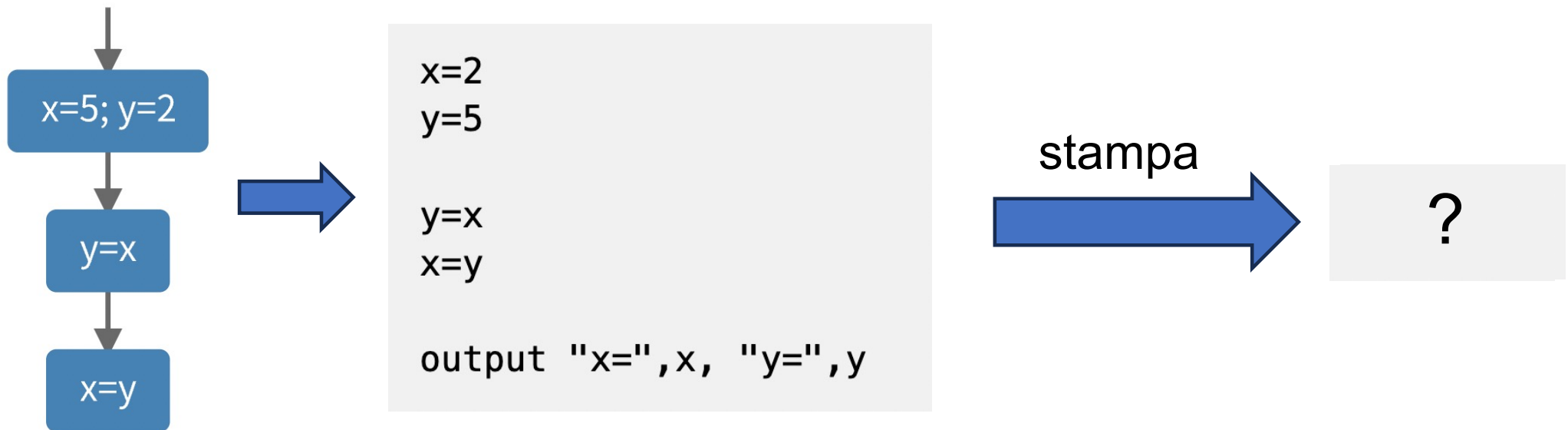
|           | Input: x | Input: y | → | Output:x | Output:y |
|-----------|----------|----------|---|----------|----------|
| esempio 1 | 5        | 7        | → | 7        | 5        |
| esempio 2 | 3        | 1        | → | 1        | 3        |

- PRE:
- POST: se  $x, y$  sono i valori all'inizio dell'esecuzione e  $x', y'$  alla fine, allora  $x'=y, y'=x$

# Esempio: Scambio di Variabili



- POST: se  $x, y$  sono i valori all'inizio dell'esecuzione e  $x', y'$  alla fine, allora  $x' = y$ ,  $y' = x$



# Esempio: Scambio di Variabili



- POST: se  $x, y$  sono i valori all'inizio dell'esecuzione e  $x', y'$  alla fine, allora  $x' = y$ ,  $y' = x$
- la freccia rappresenta l'istruzione che abbiamo appena eseguito, a destra una fotografia della memoria in quel momento



`x=2`

`y=5`

`y=x`

`x=y`

`output "x=", x, "y=", y`

| Variabile | Valore |
|-----------|--------|
| x         | 2      |
| y         | 5      |

# Esempio: Scambio di Variabili



- POST: se  $x, y$  sono i valori all'inizio dell'esecuzione e  $x', y'$  alla fine, allora  $x' = y$ ,  $y' = x$
- la freccia rappresenta l'istruzione che abbiamo appena eseguito, a destra una fotografia della memoria in quel momento



```
x=2
y=5
```

```
y=x
x=y
```

```
output "x=", x, "y=", y
```

| Variabile | Valore |
|-----------|--------|
| x         | 2      |
| y         | 2      |

# Esempio: Scambio di Variabili



- POST: se  $x, y$  sono i valori all'inizio dell'esecuzione e  $x', y'$  alla fine, allora  $x' = y$ ,  $y' = x$
- la freccia rappresenta l'istruzione che abbiamo appena eseguito, a destra una fotografia della memoria in quel momento

`x=2`

`y=5`

`y=x`

`x=y`

`output "x=", x, "y=", y`

| Variabile | Valore |
|-----------|--------|
| x         | 2      |
| y         | 2      |

# Esempio: Scambio di Variabili



- POST: se  $x, y$  sono i valori all'inizio dell'esecuzione e  $x', y'$  alla fine, allora  $x' = y$ ,  $y' = x$

```
x=2
```

```
y=5
```

```
y=x
```

```
x=y
```

```
output "x=", x, "y=", y
```

stampa



```
x=2 y=2
```

- Poiché i comandi (quindi anche gli assegnamenti) vengono eseguiti uno alla volta, quando si esegue  $y=x$  si perde il valore di  $y$ .
- Idea: dobbiamo utilizzare una variabile per ricordarci il valore di  $y$  prima di eseguire  $y=x$



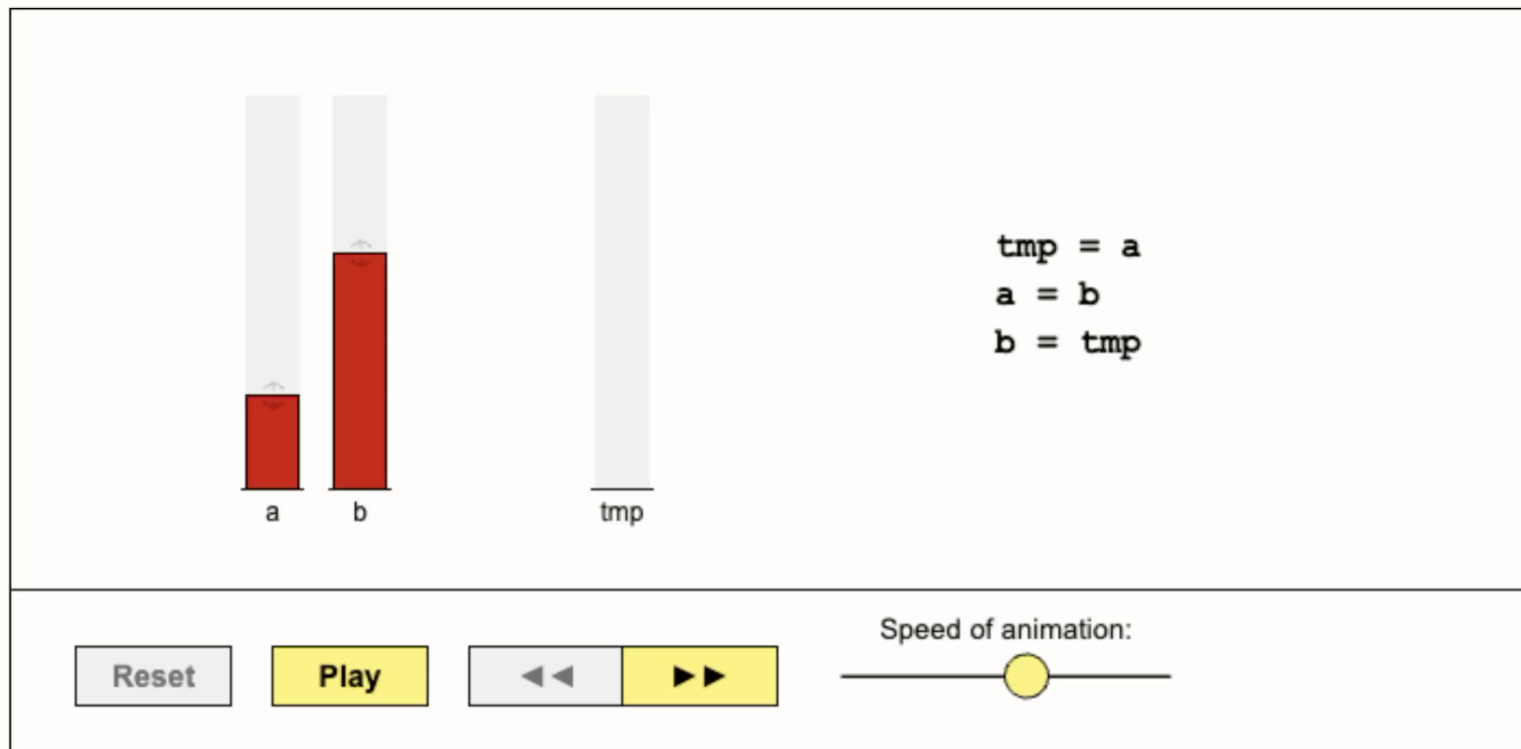
# Esempio: Scambio di Variabili



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA

## Swapping two variables

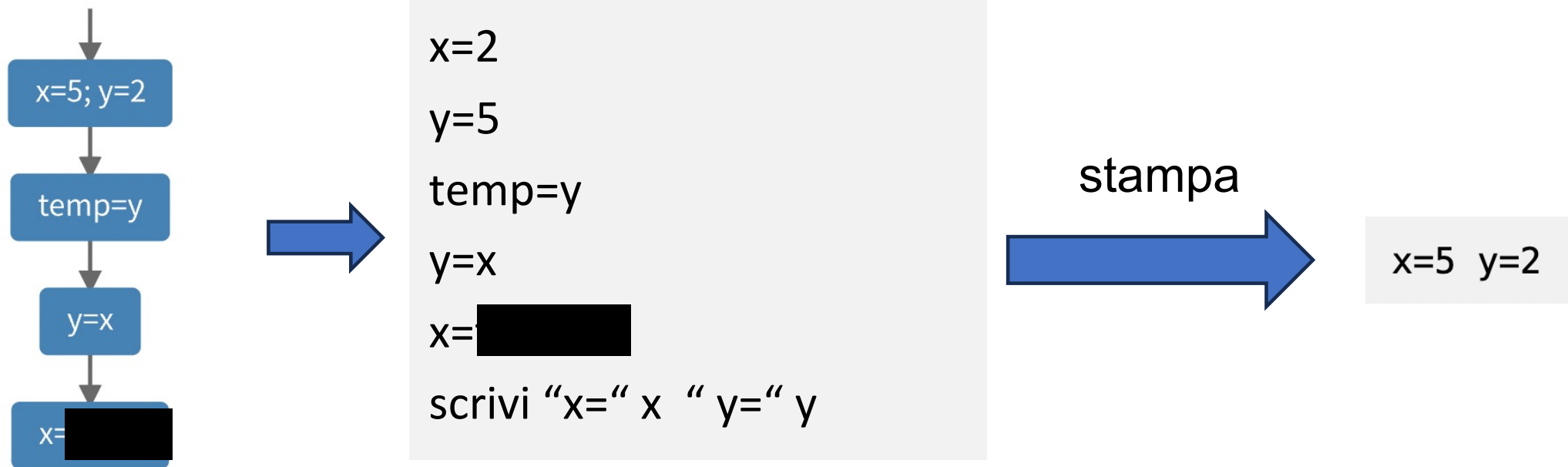
This animation shows how to swap the value of two variables (a,b) using a third variable (tmp).



# Esempio: Scambio di Variabili



- POST: se  $x, y$  sono i valori all'inizio dell'esecuzione e  $x', y'$  alla fine, allora  $x' = y$ ,  $y' = x$

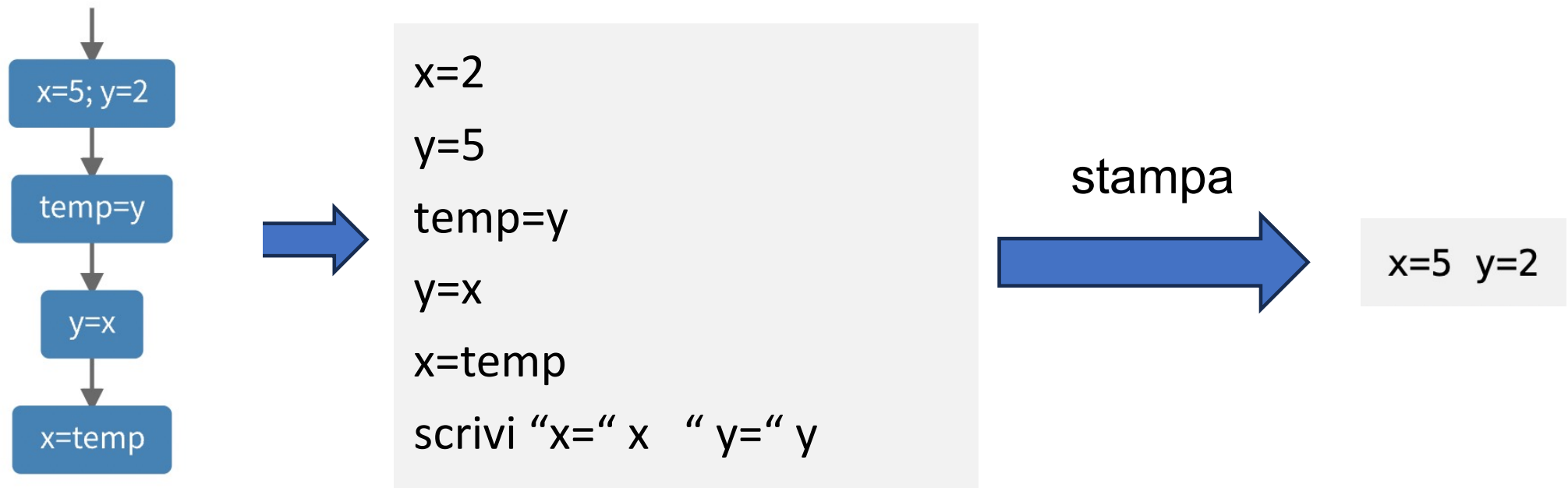


Qual è l'istruzione mancante?

# Esempio: Scambio di Variabili



- POST: se  $x, y$  sono i valori all'inizio dell'esecuzione e  $x', y'$  alla fine, allora  $x' = y$ ,  $y' = x$



# Esempio: Scambio di Variabili



- POST: se  $x, y$  sono i valori all'inizio dell'esecuzione e  $x', y'$  alla fine, allora  $x' = y$ ,  $y' = x$
- la freccia rappresenta l'istruzione che abbiamo appena eseguito, a destra una fotografia della memoria in quel momento



```
x=2
y=5
temp=y
y=x
x=temp
scrivi "x=" x " y=" y
```

| Variabile | Valore |
|-----------|--------|
| x         | 2      |
| y         | 5      |

# Esempio: Scambio di Variabili



- POST: se  $x, y$  sono i valori all'inizio dell'esecuzione e  $x', y'$  alla fine, allora  $x' = y$ ,  $y' = x$
- la freccia rappresenta l'istruzione che abbiamo appena eseguito, a destra una fotografia della memoria in quel momento



```
x=2
y=5
temp=y
y=x
x=temp
scrivi "x=" x " y=" y
```

| Variabile | Valore |
|-----------|--------|
| x         | 2      |
| y         | 5      |
| temp      | 5      |

# Esempio: Scambio di Variabili



- POST: se  $x, y$  sono i valori all'inizio dell'esecuzione e  $x', y'$  alla fine, allora  $x' = y$ ,  $y' = x$
- la freccia rappresenta l'istruzione che abbiamo appena eseguito, a destra una fotografia della memoria in quel momento

```
x=2
y=5
temp=y
→ y=x
x=temp
scrivi "x=" x " y=" y
```

| Variabile | Valore |
|-----------|--------|
| x         | 2      |
| y         | 2      |
| temp      | 5      |

# Esempio: Scambio di Variabili




- POST: se  $x, y$  sono i valori all'inizio dell'esecuzione e  $x', y'$  alla fine, allora  $x' = y$ ,  $y' = x$
- la freccia rappresenta l'istruzione che abbiamo appena eseguito, a destra una fotografia della memoria in quel momento

`x=2`

`y=5`

`temp=y`

`y=x`

 `x=temp`

`scrivi "x=" x " y=" y`

| Variabile | Valore |
|-----------|--------|
| x         | 5      |
| y         | 2      |
| temp      | 5      |

# Esempio: Somma numeri da 1 a N



- Dato un valore  $n$ , scrivere in output la somma dei numeri da 1 ad  $n$
- PRE:  $n \geq 1$
- POST: restituisce  $1+2+\dots+n$
- Possiamo eseguire una operazione alla volta:

$$1+2+3+4+5 = (((1+2)+3)+4)+5$$

$sum = 1+2$

$sum = sum + 3$

$sum = sum + 4$

$sum = sum + 5$



Variabili

$sum$  // contiene la somma degli  $n$  numeri

| n | somma |
|---|-------|
| 0 | 0     |
| 1 | 1     |
| 2 | 3     |
| 5 | 15    |



# Esempio: Somma numeri da 1 a N



- Dato un valore  $n$ , scrivere in output la somma dei numeri da 1 ad  $n$
- PRE:  $n \geq 1$
- POST: restituisce  $1+2+\dots+n$
- Possiamo eseguire una operazione alla volta:

$$1+2+3+4+5 = (((1+2)+3)+4)+5$$

sum=0

sum=sum+1

sum=sum+2

sum=sum+3

sum=sum+4

sum = sum +5



Variabili

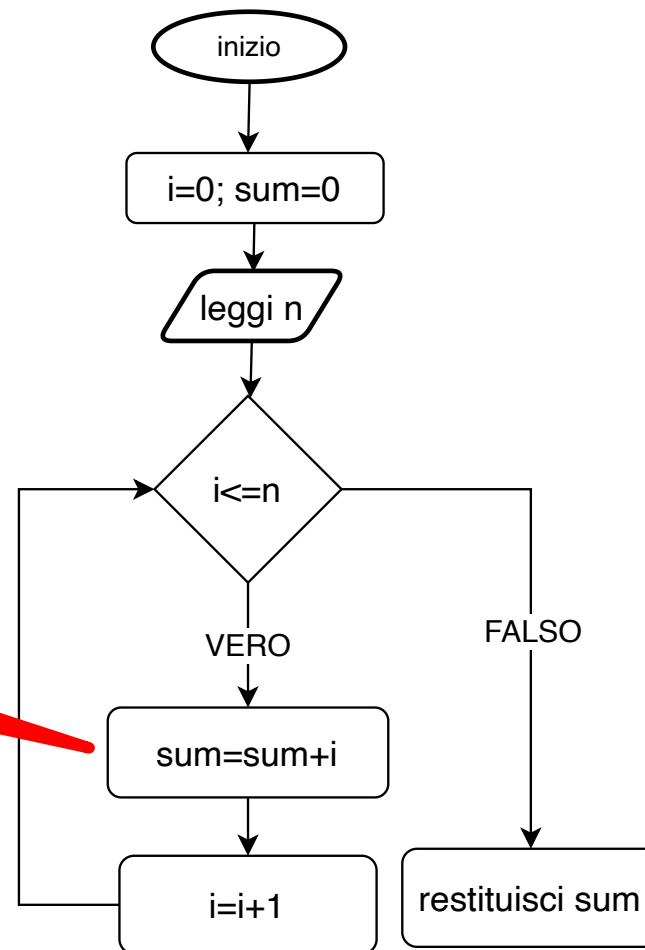
sum // contiene la somma degli  $n$  numeri

| n | somma |
|---|-------|
| 0 | 0     |
| 1 | 1     |
| 2 | 3     |
| 5 | 15    |

# Esempio: Somma numeri da 1 a N



Funzione Somma1n



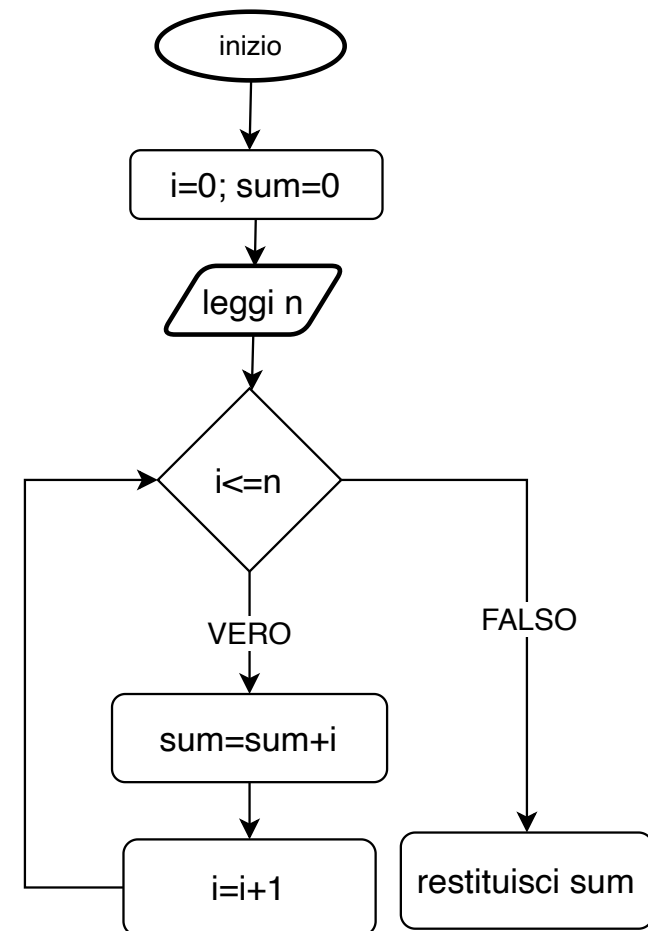
Contiene la somma  
dei numeri tra 0 e i  
incluso

# Esempio: Somma numeri da 1 a N



```
i=0
sum=0 // somma da 0 a i (incluso)
leggi n
ripeti finché i<=n {
 sum = sum + i
 i = i + 1
}
```

Funzione Somma1n



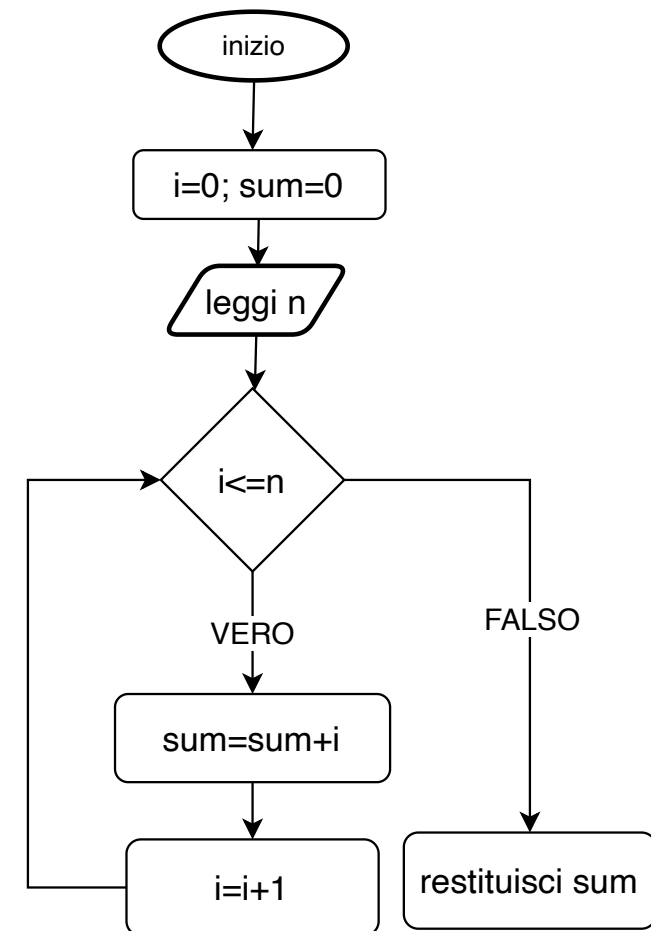
# Esempio: Somma numeri da 1 a N



```
i=0
sum=0 // somma da 0 a i (incluso)
leggi n
ripeti finché i<=n {
 sum = sum + i
 i = i + 1
}
```

Si può scrivere un programma  
equivalente senza la variabile i ?

Funzione Somma1n



# Esempio: Somma numeri da 1 a N

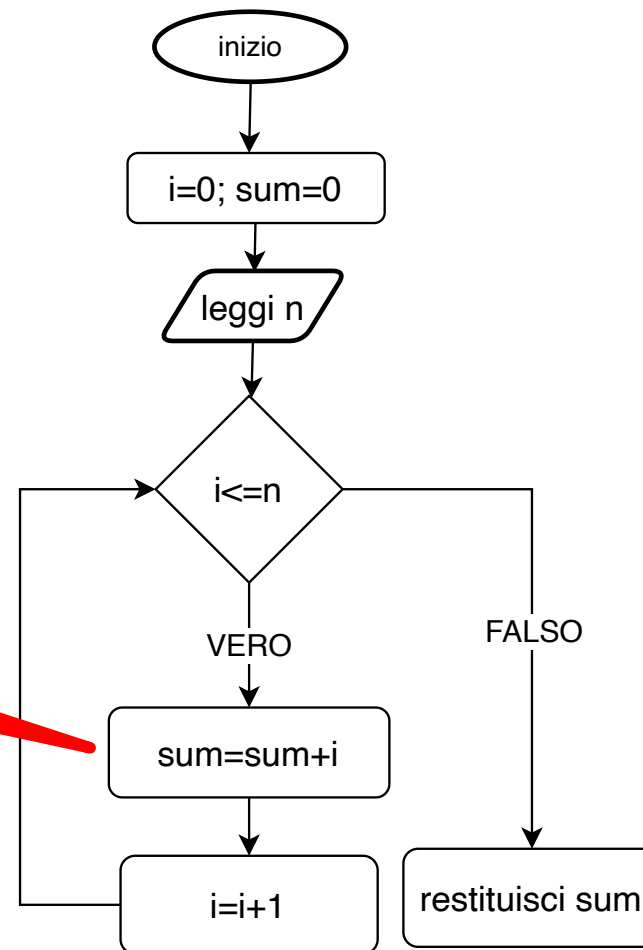


```
/*
 PRE: $n \geq 0$
 POST: stampato $1+2+\dots+n$
*/
leggi n
sum=n // somma di tutti i valori che n assume
ripeti finché $n > 1$ {
 $n = n - 1$
 sum = sum + n
}
scrivi sum
```

# Esempio: Somma numeri da 1 a N



Funzione Somma1n



Contiene la somma  
dei numeri tra 0 e i  
incluso

Schema di risoluzione:

1. Ad ogni iterazione del ciclo, calcoliamo la soluzione del problema di “dimensione i” (la somma dei numeri da 1 a i nel nostro esempio)
2. Facciamo variare i da 1 a n per ottenere la soluzione del problema di “dimensione n”

# Esempio: Numeri di Pell



- La sequenza di Pell e' cosi' definita:
  - $P(0) = 0$
  - $P(1) = 1$
  - $P(n) = 2*P(n-1)+P(n-2)$  per  $n \geq 2$
- Scrivere una funzione che, dato  $n$ , restituisca  $P(n)$ .
- PRE:  $n \geq 0$
- POST: restituisce l' $n$ -esimo numero della sequenza di Pell:  
$$P(n) = \begin{cases} 2*P(n-1)+P(n-2) & \text{se } n \geq 2 \\ n & \text{se } 0 \leq n \leq 1 \end{cases}$$

| n | Pell(n) |
|---|---------|
| 0 | 0       |
| 1 | 1       |
| 2 | 2       |
| 3 | 5       |
| 4 | 12      |

Nota: possiamo usare solamente gli strumenti visti fino ad ora a lezione (no ricorsione)

# Esempio: Numeri di Pell

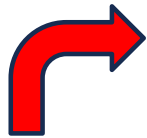


$$P(n) = P(n-2) + 2 * P(n-1)$$

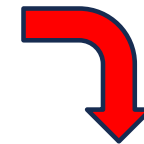
se  $n \geq 2$

$n$

se  $0 \leq n \leq 1$



Usiamo 3 variabili:  
 $p'' = P(n-2)$ ;  $p' = P(n-1)$ ;  
 $p = P(n)$



| n | Pell(n) |
|---|---------|
| 0 | 0       |
| 1 | 1       |
| 2 | 2       |
| 3 | 5       |
| 4 | 12      |

$P(0)$

$P(1)$   $P(1)$

$P(2)$   $P(2)$

$P(3)$

| n | Pell(n) |
|---|---------|
| 0 | 0       |
| 1 | 1       |
| 2 | 2       |
| 3 | 5       |
| 4 | 12      |

$p'' = P(0)$

$p' = P(1)$   $p'' = P(1)$

$p = P(2)$   $p' = P(2)$

$p = P(3)$



# Esempio: Numeri di Pell



```
/* PRE: n>=0
```

```
POST: restituisce l'n-esimo numero della sequenza di Pell: P(n)=2*P(n-1)+P(n-2) se n>=2
 n se 0<=n<=1 */
```

```
funzione pell(n) {
```

```
 Se n<2
```

```
 restituisci n
```

```
 p''=0; p'=1; p=2; /* valori P(i-2), P(i-1), P(i) per i=2*/
```

```
 i=3; // prossimo valore di pell da calcolare
```

```
 ripeti finché i<=n
```

```
 p''=p'; p'=p
```

```
 p=2*p'+p''
```

```
 i=i+1
```

```
 restituisci p
```

```
}
```

# Esercizio: Massimo di Lista Positivi



- Dato una lista L di interi positivi, scrivere una funzione `massimo_lista()` che restituisca il valore più grande in L.
- Es. `L=[4,8,9,12,6,2,5]`; `massimo_lista(L) → 12`
- `massimo_lista([]) → -1` //per la lista vuota si restituisce -1

```
funzione massimo_lista(L) {
 ...
}

leggi L
scrivi "massimo lista = " massimo_lista(L)
```

# Esercizio: Massimo di Lista Positivi



```
/*
 PRE: per ogni i. $1 \leq i \leq |L|$, $L[i] \geq 0$
 POST: restituisce
 valore massimo in L se $|L| > 0$
 -1 altrimenti
*/
funzione massimo_lista(L) {
 i=1; max=-1
 ripeti finché $i \leq |L|$
 Se $L[i] > \text{max}$
 max=L[i]
 restituisci max
}
```

# Esercizio: Minimo di Lista Positivi



- Dato una lista L di interi positivi, scrivere una funzione `minimo_lista()` che restituisca il valore minore in L.
- Es. `L=[4,8,9,5,6,2,12]`; `minimo_lista(L) → 2`
- `minimo_lista([]) → -1` //per la lista vuota si restituisce -1

```
funzione minimo_lista(L) {
 ...
}

leggi L
scrivi "minimo lista = " minimo_lista(L)
```

# Esercizio: Minimo di Lista Positivi



```
/* PRE: per ogni i. $1 \leq i \leq |L|$ $L[i] \geq 0$
 POST: restituisce valore minimo in L se $|L| > 0$
 -1 altrimenti */

funzione minimo_lista(L) {
 Se $|L| = 0$
 restituisci -1
 altrimenti
 min=L[1]; i=2;
 ripeti finché $i \leq |L|$
 Se $L[i] < \text{min}$
 min=L[i]
 restituisci min
}
```

# Esercizio: Minimo di una Lista



```
/* PRE: per ogni i. $1 \leq i \leq |L|$ $L[i] \geq 0$
 POST: restituisce 0,0 se $|L|=0$
 1,x se $|L|>0$ dove x è il valore minore in L */

funzione minimo_lista2(L) {
 Se $|L|=0$
 restituisci 0,0
 altrimenti
 min=L[1]; i=2;
 ripeti finché $i \leq |L|$
 Se $L[i] < \text{min}$
 min=L[i]
 restituisci 1,min
}
```

# Esercizio: Indice del Minimo di una Lista



```
/* PRE: per ogni i. $1 \leq i \leq |L|$, $|L| > 0$
 POST: restituisce i. $\forall j. 1 \leq j \leq |L|. L[i] \leq L[j]$
funzione indice_minimo_lista(L) {
 min=1; i=2;
 ripeti finché $i \leq |L|$
 Se $L[i] < L[\text{min}]$
 min=i
 restituisci min
}
```

# Esercizio: Ordinamento



Scrivere una funzione che ordini una lista  $L$  di interi positivi in modo crescente, utilizzando l'algoritmo visto nell'introduzione al corso, ma senza utilizzare una seconda lista: la  $i$ -esima volta che selezioniamo il minore da  $L$ , lo inseriamo in posizione  $i$  in  $L$ .

Oltre a `minimo_lista()`, abbiamo a disposizione la seguente funzione che possiamo usare all'interno di `ordina()`

Ho aggiunto la seguente notazione alle liste:  $L[i:j]$  restituisce la sottolista di  $L$  dall'elemento  $i$  a quello  $j$  inclusi. Ad es.  $L=[3,6,5,7]$ ;  $L[2:3]$  corrisponde a  $[6,5]$

```
/*
 PRE: $1 \leq i, j \leq |L|$
 POST: $L'[i]=L[j]$, $L'[j]=L[i]$ dove L' è L al termine dell'esecuzione
*/
funzione scambia(L, i, j)
```



# Esempio: Ricerca Certa in Lista



- Scrivere una funzione `trova_certo()` che, data una lista  $L$  di valori ed un valore  $x \in L$ , restituisce l'indice di  $x$  in  $L$ . Es.  $L=[4,6,8,7]$ ,  $x=8 \rightarrow$   
`trova_certo(L, x)  $\rightarrow$  3`

/\*

PRE:  $x \in L$

POST: restituisce  $i$ .  $x=L[i]$

\*/

funzione `trova_certo(L, x)` { //L: Lista di interi, x intero

}

Se esiste più di un valore in  $L$  che è uguale ad  $x$ ?

- POST: restituisce  $\min(i)$ .  $x=L[i]$

# Esempio: Ricerca Certa in Lista



```
/*
 PRE: $x \in L$
 POST: restituisce min(i). $x=L[i]$
*/
funzione trova_certo(L, x) { //L: lista di interi, x intero
 i=1
 ripeti finché NOT($x=L[i]$)
 i=i+1
 restituisci i
}
```

# Esercizio: Ricerca Certa in Lista 2



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA

```
/*
```

```
PRE: $x \in L$
```

```
POST: restituisce max(i). $x=L[i]$
```

```
*/
```

```
funzione trova_certo(L, x) { //L: lista di interi, x intero
```

```
 i=1
```

```
 ripeti finché NOT($x=L[i]$)
```

```
 i=i+1
```

```
 restituisci i
```

```
}
```

Come dobbiamo cambiare  
la funzione perché  
restituisca l'elemento di  
indice massimo uguale a x?

versione  
precedente del  
codice

# Esercizio: Ricerca Certa in Lista 2



```
/*
 PRE: $x \in L$
 POST: restituisce max(i). $x=L[i]$
*/
funzione trova_certo(L, x) { //L: lista di interi, x intero
 i=|L|
 ripeti finché NOT($x=L[i]$)
 i=i-1
 restituisci i
}
```

# Esempio: Ricerca in Lista



- Scrivere una funzione trova() che, data una lista L di valori ed un valore x in L, trovi la prima occorrenza di x in L.

Se  $x \in L$  trova() restituisce min(i).  $x=L[i]$ , altrimenti restituisce -1

- PRE:
- POST: restituisce

|                    |                 |
|--------------------|-----------------|
| min(i). $x = L[i]$ | se $x \in L$    |
| -1                 | se $x \notin L$ |

# Esempio: Ricerca in Lista



```
/*
 PRE:
 POST: restituisce min(i). x =L[i] se x in L
 -1 altrimenti
*/
funzione trova (L, x) {
 ripeti finché [Condizione] {
 ...
 }
 // usciamo dal ciclo se abbiamo trovato x oppure se abbiamo
 // esaminato tutti gli elementi di L \rightarrow $x=L[i]$ OR $i>|L|$, quindi
}
```

# Esempio: Ricerca in Lista



```
/* PRE:
 POST: restituisce min(i). x = L[i] se x in L
 -1 altrimenti */

funzione trova (L, x) {
 ripeti finché i <= |L| AND NOT(x = L[i])
 i = i + 1
 Se i = |L|
 restituisci -1
 altrimenti
 restituisci i
}
```

# Esempio: Ricerca in Lista



```
/* PRE:
 POST: restituisce min(i). x =L[i] se x in L
 -1 altrimenti */

funzione trova (L, x) {
 trovato = falso
 ripeti finché i<= |L| AND NOT(trovato)
 Se x=L[i]
 trovato = vero
 i=i+1
 Se trovato
 restituisci i
 altrimenti
 restituisci -1
}
```



# Ricerca in Lista (Nota per L'esame)



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA

```
/* PRE:
 POST: restituisce min(i). x =L[i] se x in L
 -1 altrimenti */

funzione trova (L, x) {
 ripeti finché i<= |L|
 Se x=L[i]
 restituisci i
 i=i+1
 restituisci -1
}
```

Almeno in questa parte del corso evitiamo di usare restituisci all'interno di cicli (perché vogliamo imparare a gestire condizioni complicate)

# Esempio: Maggiore di Lista



- Scrivere una funzione che, dato  $x$  intero ed una lista  $L$  di interi, determinare se  $x$  è più grande di ciascun elemento di  $L$ . Restituire vero se  $x$  è maggiore di ogni elemento di  $L$ , falso altrimenti
- PRE:
- POST: stampato
  - "x maggiore di L" se  $\forall l \in L. x > l$ ;
  - "x non maggiore di L" altrimenti
- Nota: se  $L$  è vuota restituisce vero (in generale una proprietà vale sempre per l'insieme vuoto)

# Esempio: Maggiore di Lista



```
/* PRE:
 POST: restituisce vero se $\forall l \in L. x > l$;
 falso altrimenti. */

funzione maggioreDiLista(L, x) {
 maggiore = vero; i=1
 ripeti finché $i \leq |L|$
 Se $x < L[i]$
 maggiore = falso
 i=i+1
 restituisci maggiore
}
```

# Esempio: Maggiore di Lista



```
/*
 PRE:
 POST: restituisce vero se $\forall l \in L. x > l$;
 falso altrimenti.
*/
funzione maggioreDiLista(L, x) {
 maggiore = vero; i=1
 ripeti finché $i \leq |L|$
 maggiore = maggiore AND $x > L[i]$
 i=i+1
 restituisci maggiore
}
```