# Rappresentazione degli algoritmi

Tullio Facchinetti

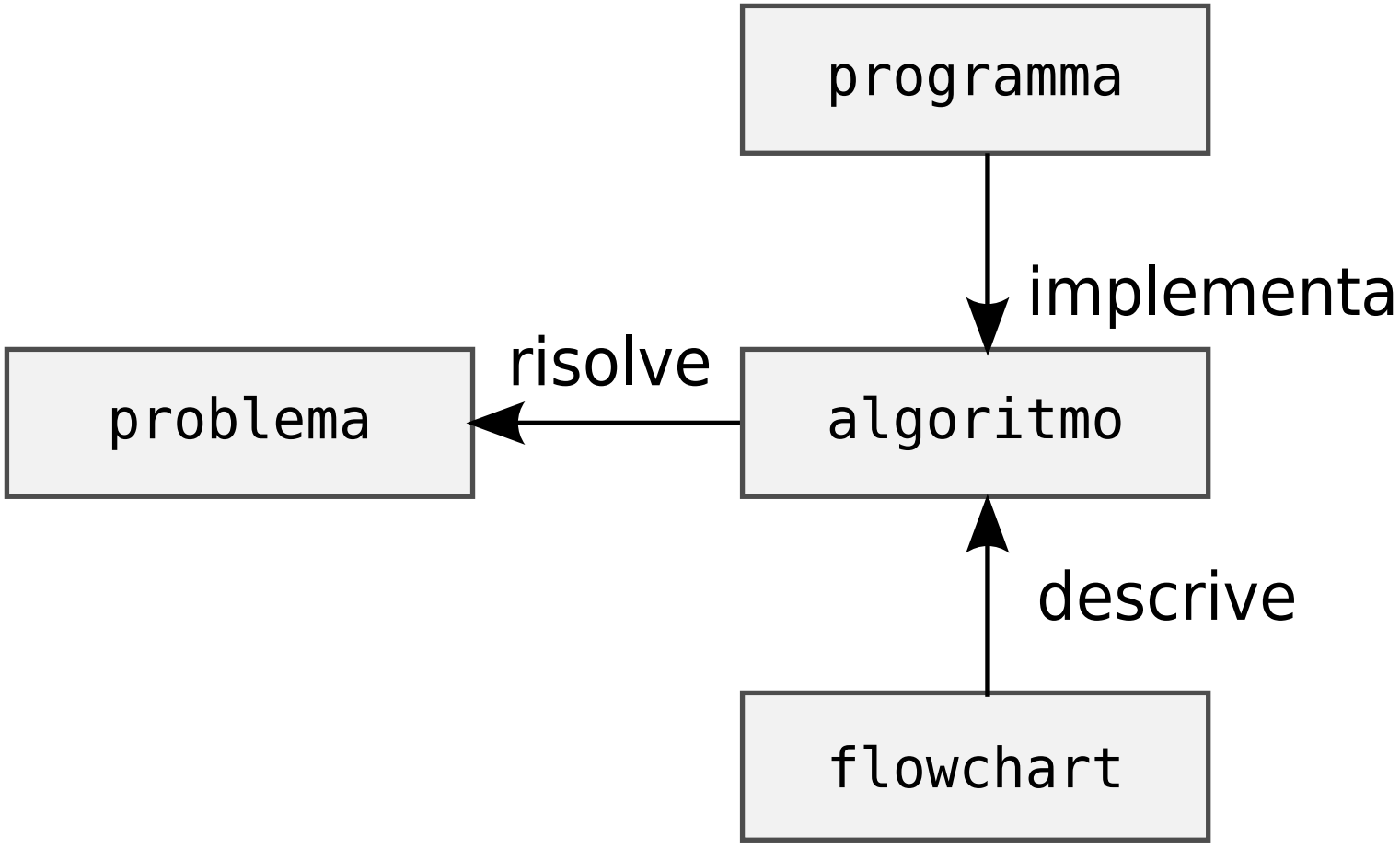
24 marzo 2023

# Rappresentazione degli algoritmi

- per rappresentare (descrivere) un algoritmo non è possibile utilizzare il linguaggio naturale in quanto questo può presentare ambiguità e causare false interpretazioni
- è necessario, pertanto, utilizzare linguaggi sintetici e standardizzati in modo da consentire all'esecutore una interpretazione univoca dei passi da svolgere

i formalismi che verranno trattati sono quelli dei diagrammi a blocchi e dello pseudo-codice

# Problemi, algoritmi, flowchart e programmi



# Operazioni fondamentali

le operazioni base per la realizzazione di un algoritmo sono 4:

1. trasferimento di informazioni: acquisizione dati, visualizzazione risultati intermedi, scrittura risultati finali
2. esecuzione di calcoli
3. assunzione di decisioni: scelta della successiva operazione da compiere sulla base di risultati intermedi
4. esecuzione di iterazioni: ripetizione di sequenze di operazioni

# Programmazione strutturata

è una tecnica di programmazione che ha lo scopo di semplificare la struttura di un algoritmo disciplinando l'uso delle strutture di controllo utilizzabili all'interno di uno schema blocchi

prevede l'uso di un numero limitato di strutture di controllo fondamentali, con un ingresso ed una uscita:

1. sequenza
2. selezione
3. iterazione

# Programmazione strutturata

la programmazione strutturata vincola quindi l'utilizzo delle strutture di controllo, ma offre i seguenti vantaggi:

- permette la definizione di algoritmi più leggibili, essendo più facile individuare i moduli corrispondenti alle varie parti di cui si compone l'algoritmo
- test, correzione e manutenzione del programma sono perciò più semplici, anche se per il test del sistema completo bisogna attendere di assemblare tutti i componenti
- rende possibile una progettazione di tipo top-down

# Progettazione top-down

si parte dall'obiettivo finale del problema e si esplicitano e raffinano ricorsivamente le parti che lo compongono

- la strategia per risolvere il problema viene originata dall'obiettivo del problema stesso
- ad ogni passo vengono identificati dei sotto-blocchi logici correlati che vengono rifiniti sempre più (decomposizione, specializzazione, specificazione)
- il processo di rifinizione termina quando si raggiunge il livello di dettaglio sufficiente per l'applicazione da risolvere
- tecnica adatta a problemi complicati di tipologie differenti
- contrapposta alla progettazione bottom-up

# Rappresentazione degli algoritmi mediante i diagrammi di flusso

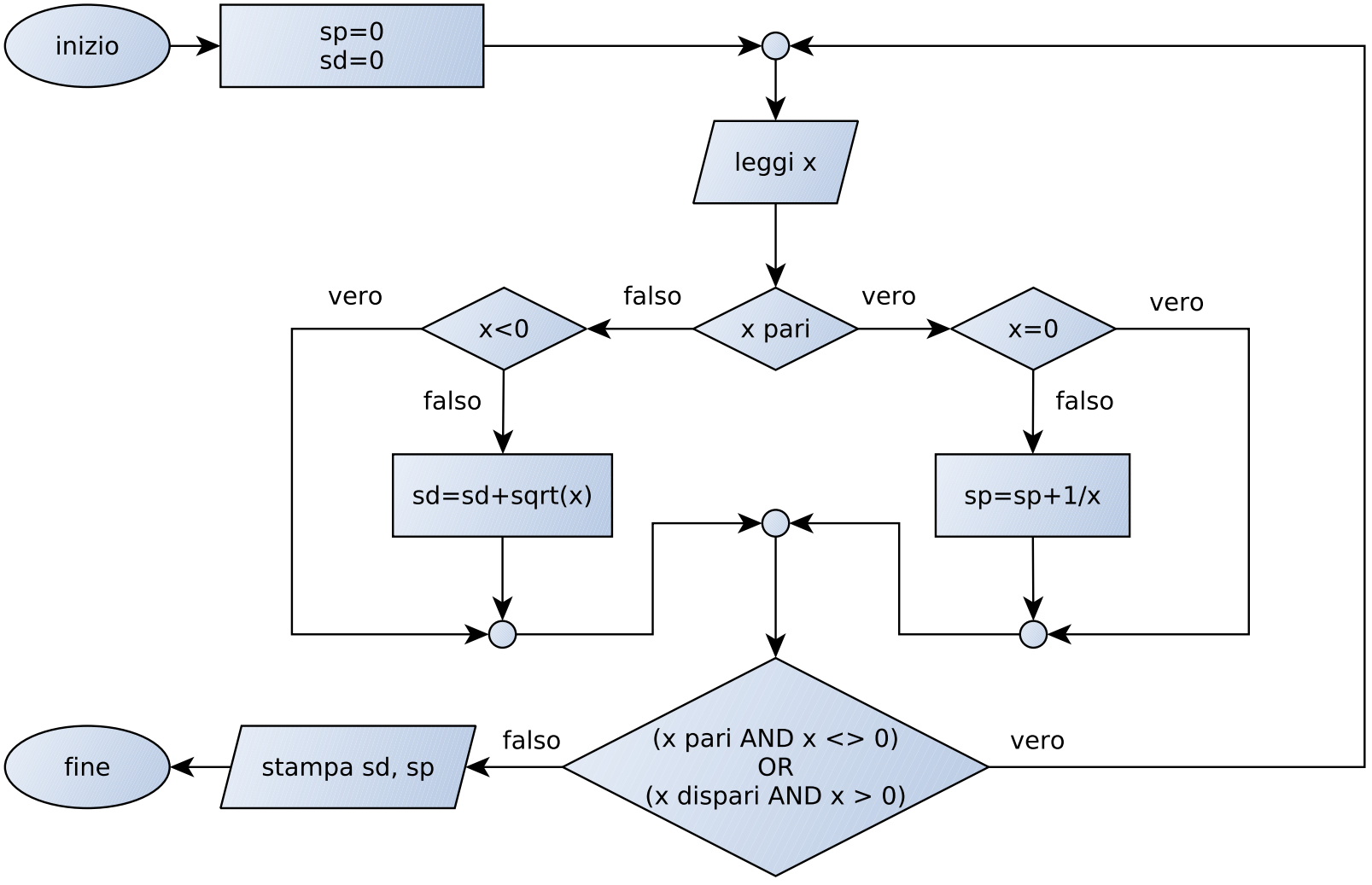
il flowchart è un formalismo che consente di rappresentare graficamente gli algoritmi

- i flowchart sono anche detti diagrammi di flusso o schemi/diagrammi a blocchi
- questa descrizione costituisce un efficace strumento per la descrizione degli algoritmi, più valido di una esposizione di tipo discorsivo (generica e ambigua)
- qualsiasi algoritmo può essere decomposto in poche strutture elementari

# Rappresentazione degli algoritmi: diagrammi di flusso

- un diagramma di flusso (o flowchart) descrive le azioni da eseguire ed il loro ordine di esecuzione
- ogni azione elementare corrisponde ad un simbolo grafico (blocco) diverso (sono convenzioni non universali)
- i blocchi sono collegati da rami o archi orientati (frecce) che determinano la sequenza dei blocchi
- un diagramma di flusso appare, quindi, come un insieme di blocchi di forme diverse che contengono le istruzioni da eseguire, collegati fra loro da linee orientate che specificano la sequenza in cui i blocchi devono essere eseguiti (flusso del controllo di esecuzione o sequenza di computazione)

# Esempio di rappresentazione



# Alcune definizioni

1. flusso di controllo: ordine di percorrenza dei blocchi individuato da un flowchart
2. struttura di controllo: flowchart parziale da assumere come modello di computazione, con un ingresso e un'uscita
3. sequenza di computazione: successione di blocchi operativi e decisionali prodotta dall'esecuzione di un flowchart per un certo insieme di dati in ingresso

# Lo pseudo-codice

descrizione informale di alto livello adatta a rappresentare un algoritmo o programma che usa le strutture di un linguaggio di programmazione

- adatto ad essere letto dall'uomo
- non adatto per la specifica formale di programmi (non direttamente comprensibile ad una macchina)
- omette dettagli non essenziali per la comprensibilità (es. dichiarazione di variabili)

# Caratteristiche dello pseudo-codice

- metodo alternativo a flowchart e UML (un altro metodo grafico di rappresentazione)
- più compatto di questi ultimi
- non esiste uno standard di rappresentazione
- la descrizione può essere completata da testo in linguaggio naturale
- tipicamente utilizzato in libri di testo, in articoli scientifici, nella pianificazione dello sviluppo di programmi

# Esempio di pseudo-codice

$sd \leftarrow 0$
$sp \leftarrow 0$
**repeat**
    leggi val
    **if** val pari **then**
        **if** val $\neq 0$ **then**
            $sp \leftarrow sp + (1/val)$
        **end if**
    **else**
        **if** val $> 0$ **then**
            $sd \leftarrow sd + \sqrt{val}$
        **end if**
    **end if**
**until** (val pari e val $\neq 0$) oppure (val dispari e val $> 0$)
stampa $sd$ e $sp$

## Valori e grandezze

### valori:

- numerici: interi (-1, 0, 42, ...), reali (3.14, 1.72, ...), o complessi (1+2i, ...)
- logici: Vero e Falso
- alfanumerici, detti anche stringhe (es. “AAA”, “C.Colombo”)

### grandezze:

- variabili: rappresentate da un nome simbolico cui è assegnato un valore che può cambiare durante l’esecuzione dell’algoritmo
- costanti: quantità note a priori, il cui valore non cambia durante l’esecuzione

# Le variabili

sono entità che permettono di memorizzare dei valori di vario tipo durante lo svolgimento dell'algoritmo

- utilizzate per memorizzare i valori di ingresso all'algoritmo, i risultati finali ed eventuali risultati parziali
- le variabili sono associate a nomi, anche detti identificatori, che ne rappresentano il valore
- il valore memorizzato può variare durante lo svolgimento dell'algoritmo

esempi: a, b, c, somma, delta, somma_parziale, ...

# Espressioni

**Espressione**

sequenza di variabili e costanti combinate fra loro mediante operatori

- espressioni aritmetiche: combinano valori numerici e generano un risultato di tipo numerico:
  - operatori impiegati: $+$, -, *, $/$, ...
  - funzioni: `sqrt()`, `sin()`, `cos()`, `exp()`, `log()`
- espressioni relazionali e logiche: forniscono un risultato di tipo logico (vero o falso)
  - operatori impiegati: $>$, $<$, $=$, $\geq$, $\leq$, $\neq$, AND, OR, NOT

# Esempi di espressioni

## espressioni aritmetiche:

A

100

2 * (s + r)

sqrt(x^2 + y^2)

## espressioni logiche:

somma $\leq$ 1000

a $\neq$ b

(A < -10) OR (B > 10)

# Valutazione di espressioni

## Valutazione di una espressione

consiste nella sostituzione di ogni variabile col relativo valore attuale e dell'esecuzione delle operazioni secondo un ordine prestabilito da regole di precedenza (possono comparire parentesi)

l'espressione **sqrt(x^2 + y^2)**

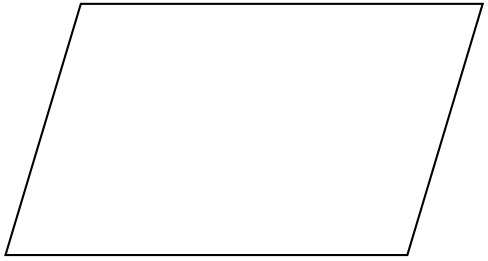
- assume il valore 5 se le variabili valgono $x = 3$ e $y = 4$
- assume il valore 10.8166 se vale $x = 6$ e $y = 9$

# Tipologie di blocchi

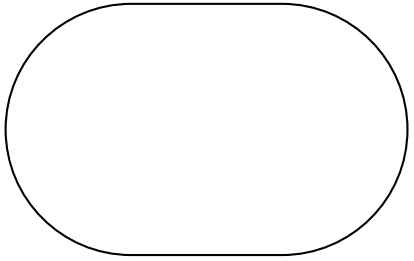
ogni azione è rappresentata in un flowchart da un blocco grafico

- blocchi semplici: esecuzione di operazioni sui dati
- blocco condizione: in base al verificarsi di una condizione, permette di differenziare il comportamento dell'algoritmo, mediante la scelta tra due strade alternative

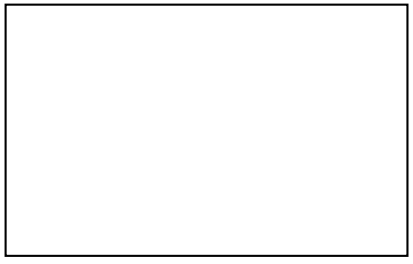
# I blocchi più comuni



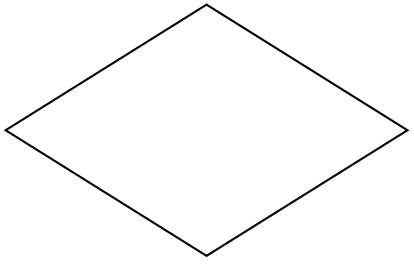
ingresso/uscita



inizio/fine

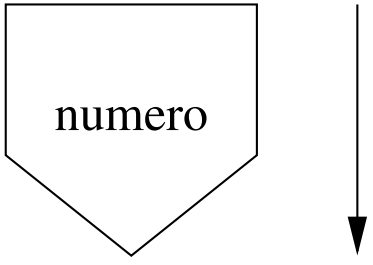


elaborazione/calcolo



decisione

elaborazione predefinita

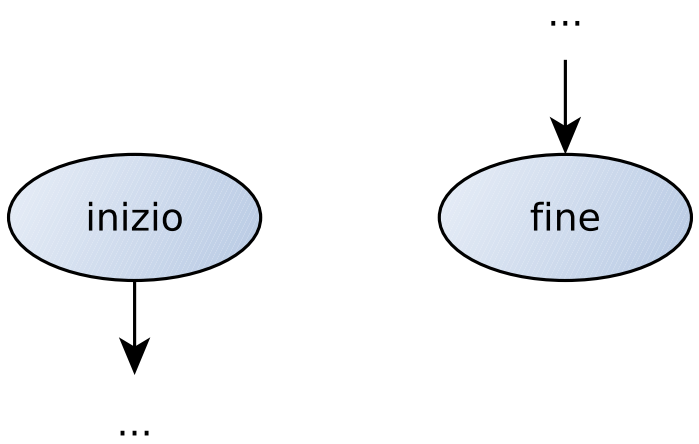



connessioni

# Inizio/fine

inizio e fine di un algoritmo

- inizio è il blocco da cui deve iniziare l'esecuzione (uno e uno solo)
- il blocco fine fa terminare l'esecuzione dell'algoritmo (almeno uno)

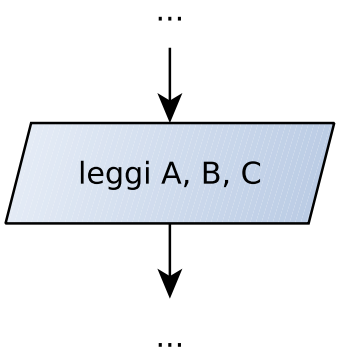


# Ingresso/lettura

esecuzione dell'istruzione:

- si ricevono dall'unità di ingresso (per esempio, la tastiera) tanti valori quante sono le variabili specificate all'interno del blocco, e si assegnano nello stesso ordine alle variabili
- A, B, C sono nomi di variabili

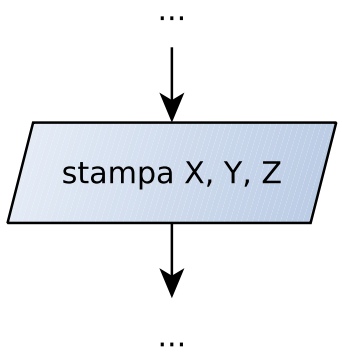
es. "Leggi i tre valori dati in ingresso, ed assegnali rispettivamente alle variabili A, B, e C"



# Uscita/stampa

si calcolano i valori delle espressioni e si trasmettono all'unità di uscita (ad esempio, il video)

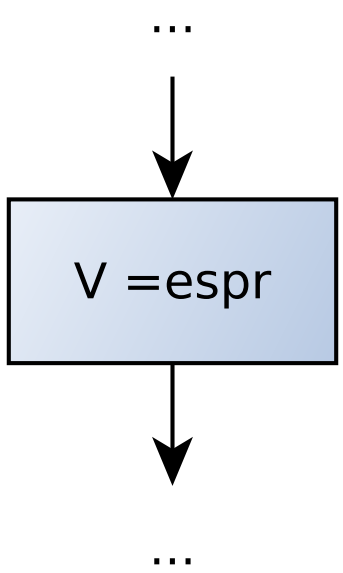
- X, Y, Z possono essere variabili
- se X, Y, Z sono espressioni $\rightarrow$ "calcola i valori delle espressioni X, Y e Z, e trasmettili in uscita"




N.B.: i valori di X, Y, Z non vengono comunque alterati dall'esecuzione del blocco

# Blocco di calcolo

contiene espressioni da valutare
ed assegnare a variabili



## Assegnamento

- si calcola il valore dell'espressione a destra del simbolo "="
- lo si assegna alla variabile indicata a sinistra del simbolo "="
- il valore precedente di V viene perduto

si può scrivere in generale

$$V = E$$

dove

- V è il nome di una variabile
- E è una espressione

# Assegnamento

$$V = E$$

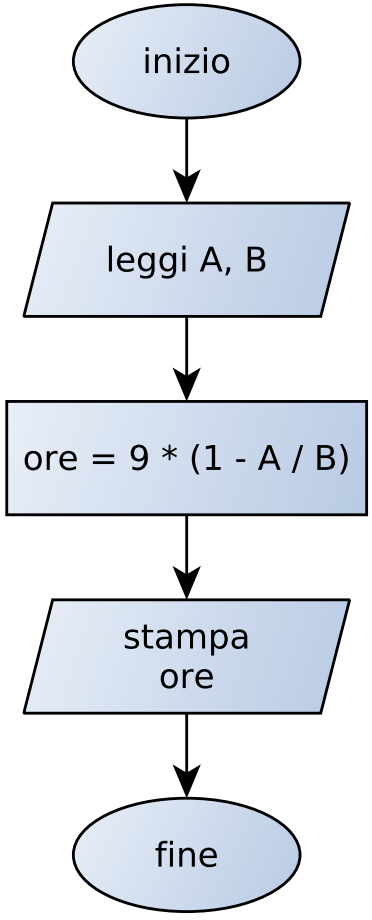
si interpreta come: *"valuta l'espressione $E$ e assegna il risultato alla variabile $V$"*

i due termini non sono scambiabili
($E = V$ non è un assegnamento valido)
a sinistra deve comparire una entità "assegnabile"

- una variabile è una entità assegnabile
- le stesse questioni si ripresentano nei linguaggi di programmazione
- deve esistere una locazione di memoria nella quale memorizzare il risultato dell'espressione

# Esempio: calcolo di una frazione

- Luca e Paola sono colleghi
- devono garantire ogni giorno esattamente 9 ore di lavoro complessive
- decidono di dividersi il lavoro in modo che Luca lavori una frazione A/B delle ore totali
- dati A e B, calcolare le ore lavorate da Paola giornalmente



# Esempio: calcolo di una frazione (pseudo-codice)

- Luca e Paola sono colleghi
- devono garantire ogni giorno esattamente 9 ore di lavoro complessive
- decidono di dividersi il lavoro in modo che Luca lavori una frazione A/B delle ore totali
- dati A e B, calcolare le ore lavorate da Paola giornalmente

**leggi** A, B
ore $\leftarrow$ 9 * (1 - A / B)
**stampa** ore

## Strutture di controllo

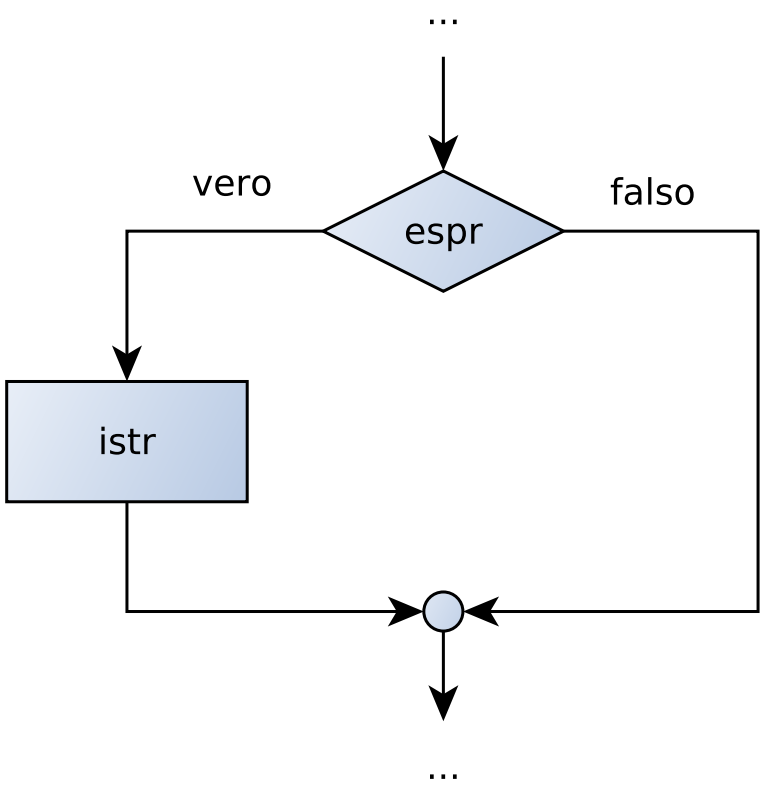
mediante i blocchi fondamentali, è possibile costruire delle strutture standard da utilizzare per il controllo del flusso di esecuzione dell'algoritmo

le strutture di controllo sono:

- selezione
- iterazione

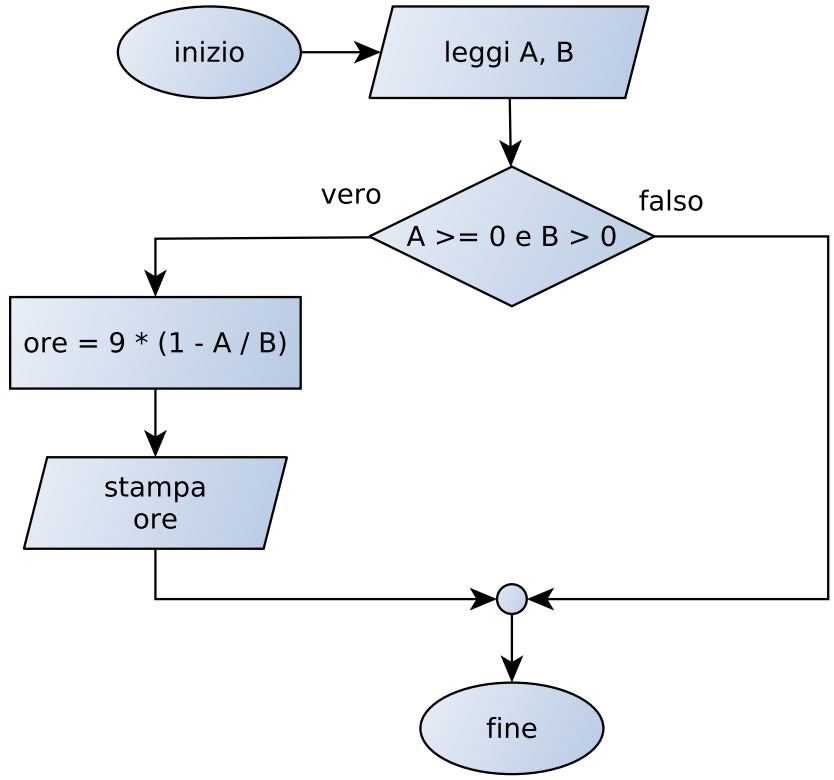
# Selezione: costrutto `if`

permette di eseguire un'istruzione, o blocco di istruzioni,
al verificarsi di una condizione



# Calcolo di una frazione

- Luca e Paola sono colleghi
- devono garantire ogni giorno esattamente 9 ore di lavoro complessive
- decidono di dividersi il lavoro in modo che Luca lavori una frazione A/B delle ore totali
- dati A e B, calcolare il totale di ore lavorate da Paola
- si verifichi che i dati inseriti permettano il calcolo corretto



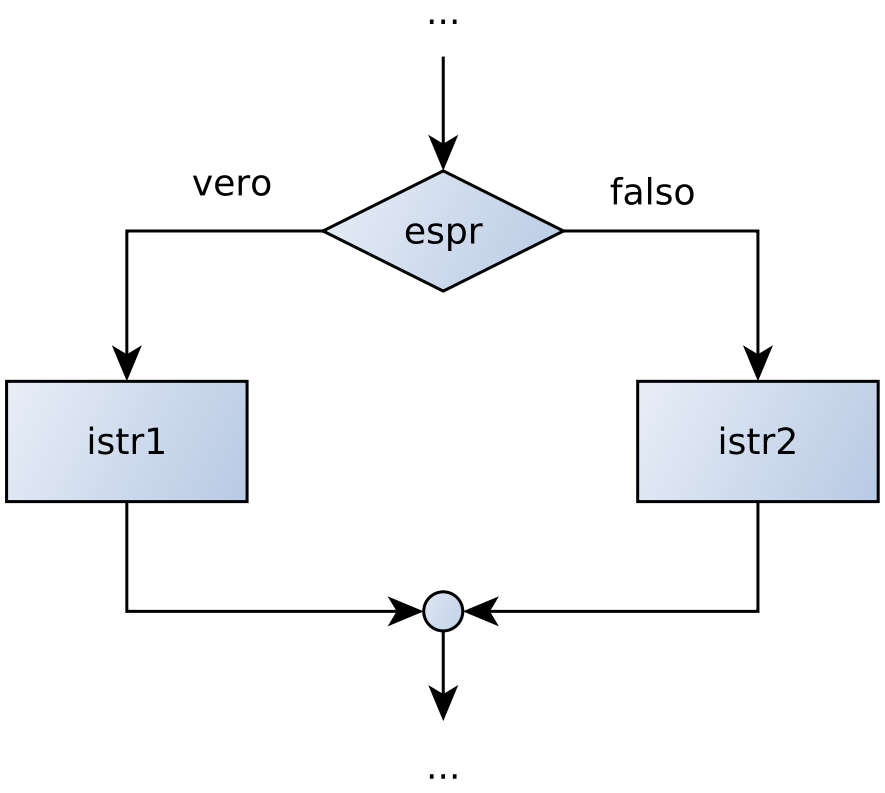
# Calcolo di una frazione (pseudo-codice)

- Luca e Paola sono colleghi
- devono garantire ogni giorno esattamente 9 ore di lavoro complessive
- decidono di dividersi il lavoro in modo che Luca lavori una frazione A/B delle ore totali
- dati A e B, calcolare il totale di ore lavorate da Paola
- si verifichi che i dati inseriti permettano il calcolo corretto

**leggi** A, B
**if** $A \geq 0$ e $B > 0$ **then**
    ore $\leftarrow$ 9 * (1 - A / B)
    **stampa** ore
**end if**

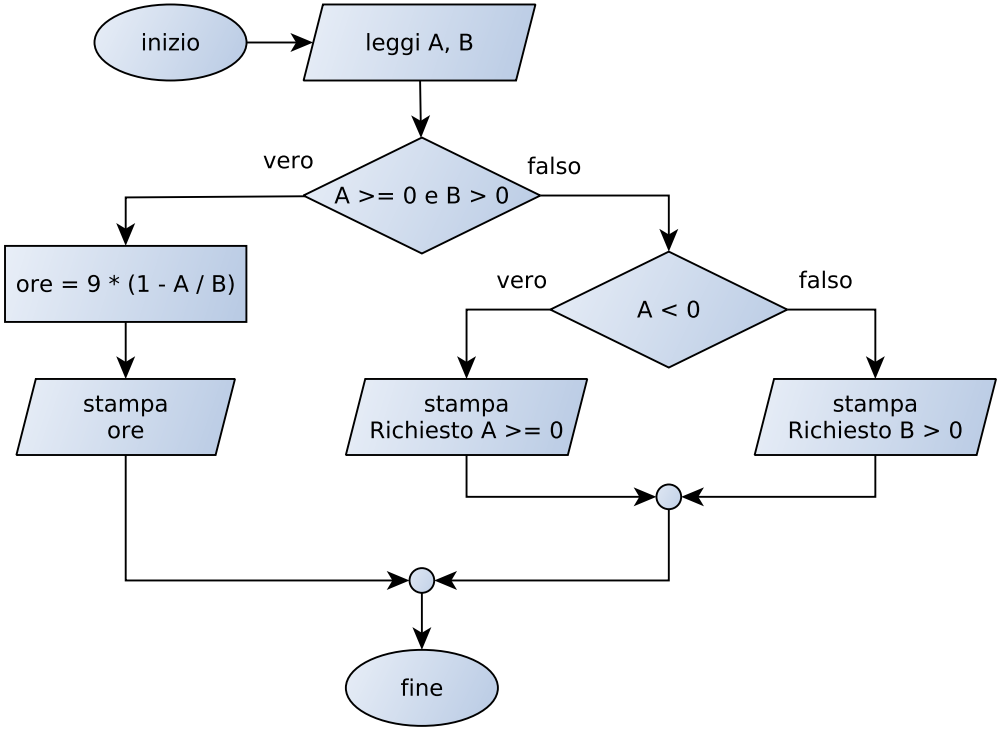
# Selezione: costrutto `if-else`

permette di scegliere tra due possibili azioni, o sequenze di azioni, mutuamente esclusive



# Calcolo di una frazione

- Luca e Paola sono colleghi
- devono garantire ogni giorno esattamente 9 ore di lavoro complessive
- decidono di dividersi il lavoro in modo che Luca lavori una frazione A/B delle ore totali
- dati A e B, calcolare il totale di ore lavorate da Paola
- si verifichi che i dati inseriti permettano il calcolo corretto
- si informi l'utente in caso di problemi

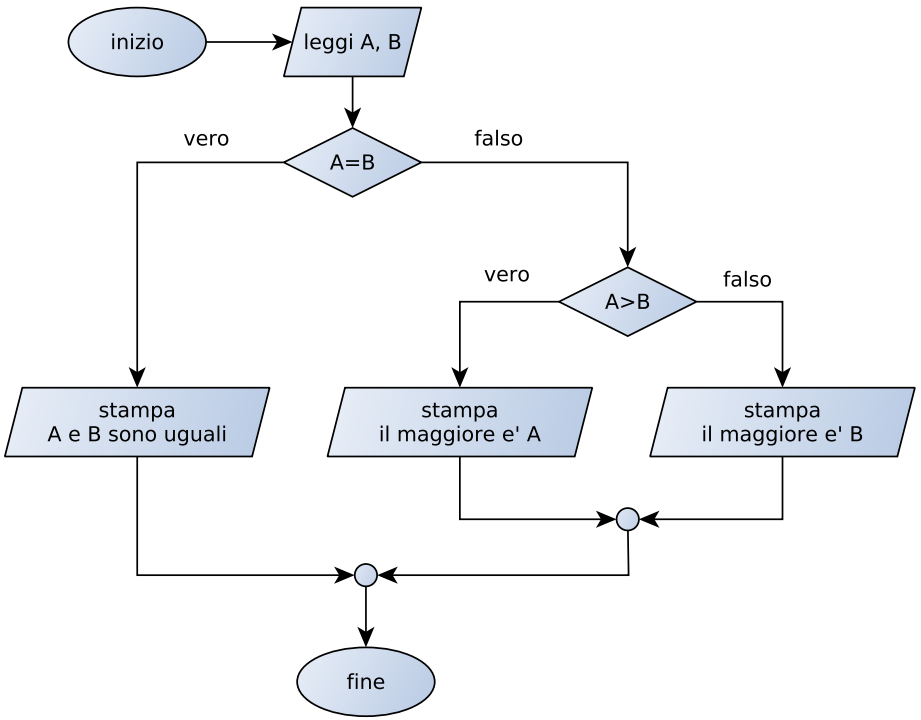


# Calcolo di una frazione (pseudo-codice)

- Luca e Paola sono colleghi
- devono garantire ogni giorno esattamente 9 ore di lavoro complessive
- decidono di dividersi il lavoro in modo che Luca lavori una frazione A/B delle ore totali
- dati A e B, calcolare il totale di ore lavorate da Paola
- si verifichi che i dati inseriti permettano il calcolo corretto
- si informi l'utente in caso di problemi

```
leggi A, B
if A ≥ 0 e B > 0 then
    ore ← 9 * (1 - A / B)
    stampa ore
else
    if A < 0 then
        stampa Richiesto A ≥ 0
    else
        stampa Richiesto B > 0
    end if
end if
```

# Confronto tra due numeri




**leggi** A, B
**if** $A = B$ **then**
    **stampa** "A e B sono uguali"
**else**
    **if** $A > B$ **then**
        **stampa** "Il maggiore è A"
    **else**
        **stampa** "Il maggiore è B"
    **end if**
**end if**

# Strutture di controllo: iterazione

permette la ripetizione di una sequenza di istruzioni

## nel caso più generale, è costituita da:

- inizializzazione: assegnazione dei valori iniziali alle variabili caratteristiche del ciclo (viene eseguita una sola volta)
- corpo: esecuzione delle istruzioni fondamentali del ciclo che devono essere eseguite in modo ripetitivo
- modifica: modifica dei valori delle variabili che controllano l'esecuzione del ciclo (eseguito ad ogni iterazione)
- controllo: determina, in base al valore delle variabili che controllano l'esecuzione del ciclo se il ciclo deve essere ripetuto o meno.

# Strutture di controllo: iterazione

Sono possibili tre configurazioni:

- do-while: **svolgi** le istruzioni **mentre** la condizione è vera
- while-do: **mentre** la condizione è vera **svolgi** le istruzioni
- for: **per** un valore che varia da un valore iniziale a uno finale **svolgi** le istruzioni **e modifica** il valore corrente

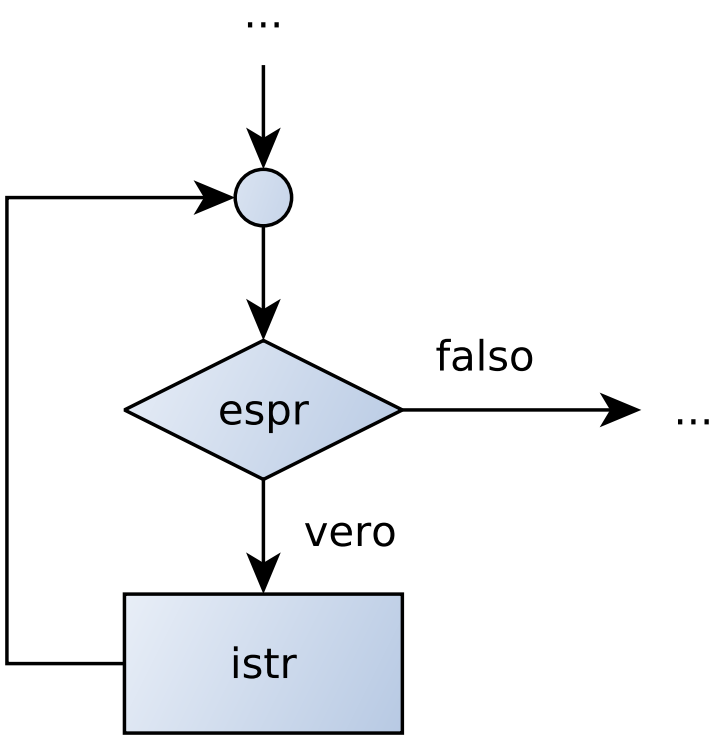
in alternativa al costrutto do-while è disponibile il costrutto repeat-until:

- repeat-until: **ripeti** le istruzioni **fino a quando** la condizione non diventa vera

I costrutti iterativi sono tutti intercambiabili (sono **equivalenti**).

# Ciclo while-do

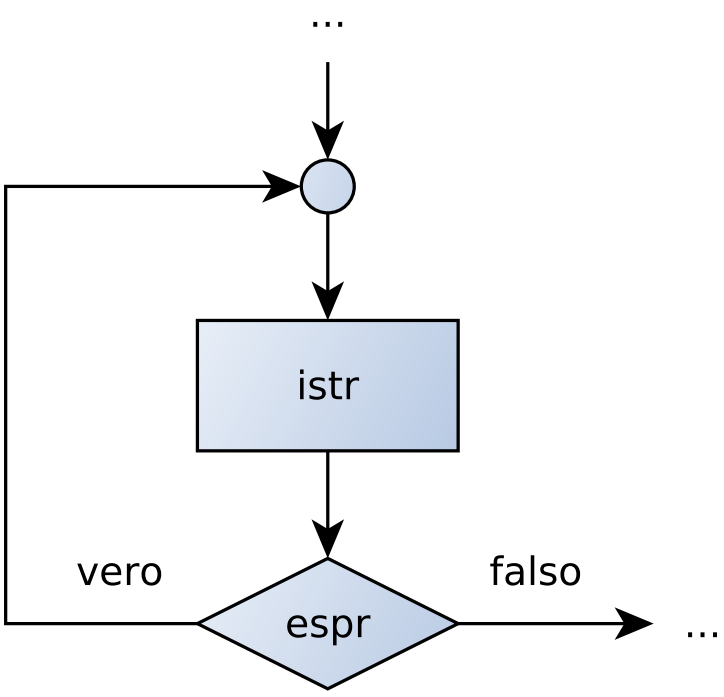
while-do: **mentre** la condizione è vera **svolgi** le istruzioni




**while** espr è vera **do**
    istr
**end while**

# Ciclo do-while

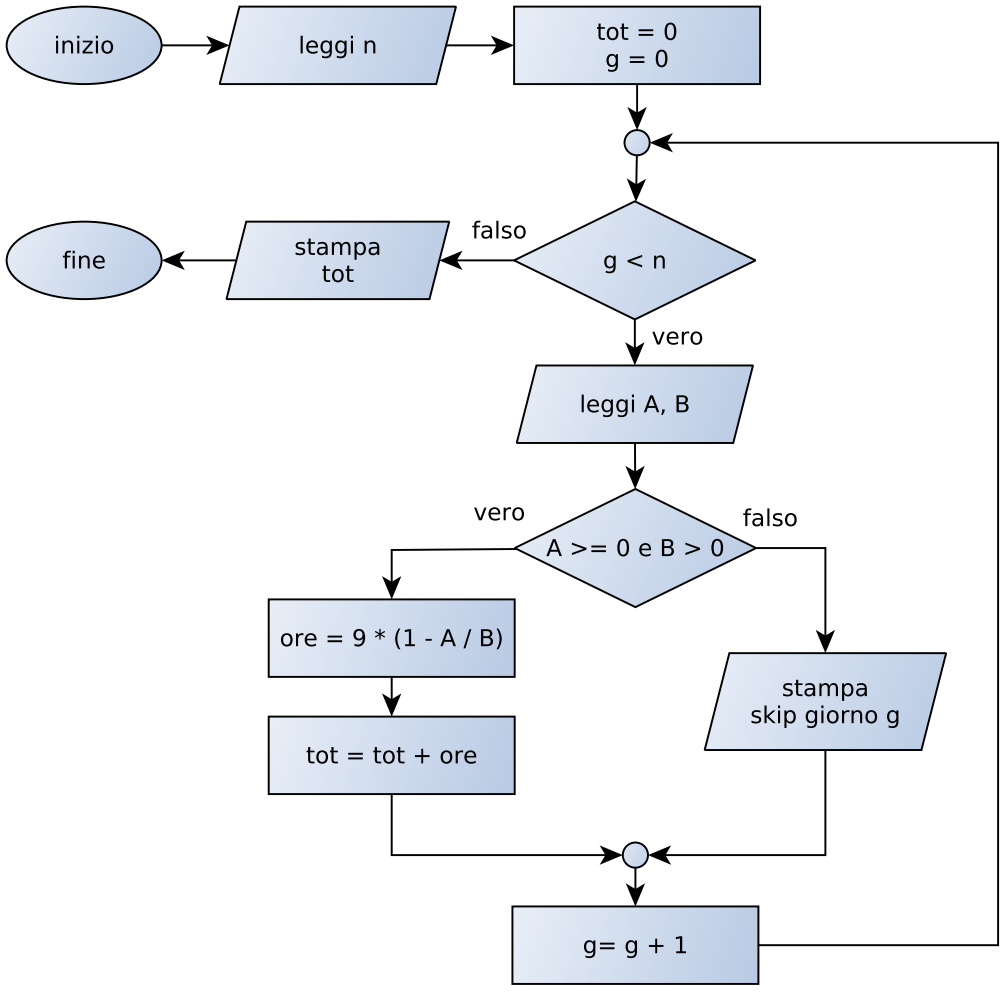
do-while: **svolgi** le istruzioni **mentre** la condizione è vera




**do**
    istr
**while** espr è vera

# Calcolo di una somma di frazioni

- Luca e Paola sono colleghi
- devono garantire ogni giorno esattamente 9 ore di lavoro complessive
- decidono di dividersi il lavoro in modo che Luca lavori una frazione A/B delle ore totali, variabile ogni giorno
- dati A e B, calcolare il totale di ore lavorate da Paola in n giorni
- si verifichi che i dati inseriti permettano il calcolo corretto



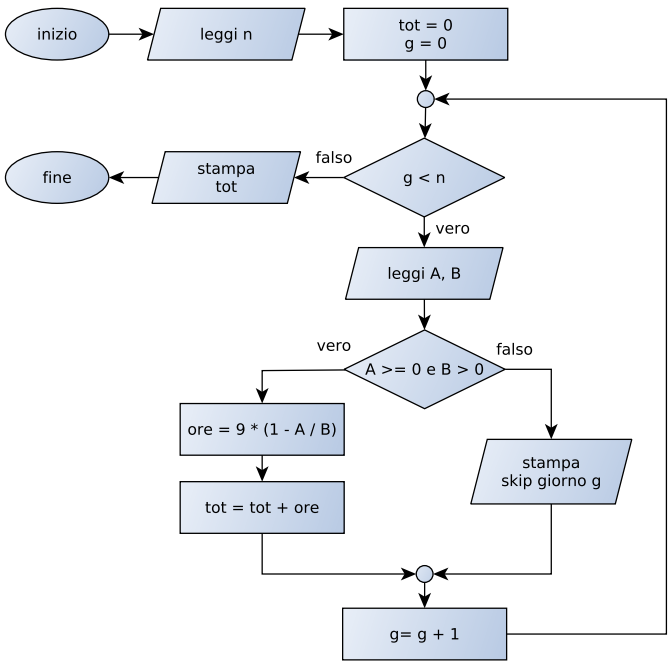
# Calcolo di una somma di frazioni (pseudo-codice)

- Luca e Paola sono colleghi
- devono garantire ogni giorno esattamente 9 ore di lavoro complessive
- decidono di dividersi il lavoro in modo che Luca lavori una frazione A/B delle ore totali, variabile ogni giorno
- dati A e B, calcolare il totale di ore lavorate da Paola in n giorni
- si verifichi che i dati inseriti permettano il calcolo corretto

```
leggi n
tot ← 0
g ← 0
while g < n do
    leggi A, B
    if A ≥ 0 e B > 0 then
        ore ← 9 * (1 - A / B)
        tot ← tot + ore
    else
        stampa Skip giorno g
    end if
    g ← g + 1
end while
stampa tot
```

# Calcolo di una somma di frazioni: esercizio

Nella soluzione presentata, nel caso in cui non si possono svolgere i calcoli in un determinato giorno, si passa al giorno seguente, saltando il giorno "problematico".

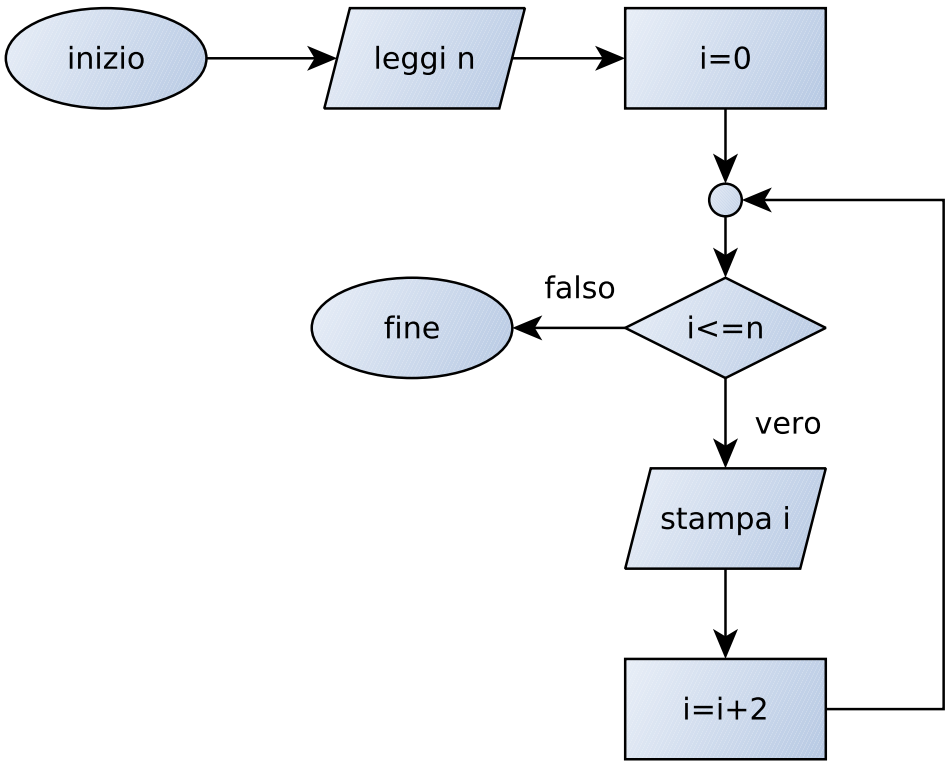



Modificare il flowchart per risolvere il problema in modo che

"In caso i valori dei parametri non permettano il calcolo per un determinato giorno, continuare a richiedere i valori fino a quando non vengono inseriti dei valori validi".

Ci sono possibili diverse soluzioni; cercare quella che richiede la minor quantità di modifiche al flowchart esistente.

# Esempio: stampa dei numeri pari minori o uguali a $n$




**leggi** n
$i \leftarrow 0$
**while** $i \leq n$ **do**
    **stampa** $i$
    $i \leftarrow i + 2$
**end while**

# Stampa dei primi n numeri pari positivi

Stampare i numeri pari positivi minori o uguali ad n

Stampare i primi n numeri pari positivi

- Piccola variante rispetto al problema precedente.
- La differenza di formulazione è piccola ma significativa, in quanto il risultato cambia in modo non indifferente

PRIMA

- n = 2 : stampa 0, 2
- n = 5 : stampa 0, 2, 4
- n = 100 : stampa 0, 2, . . . , 100

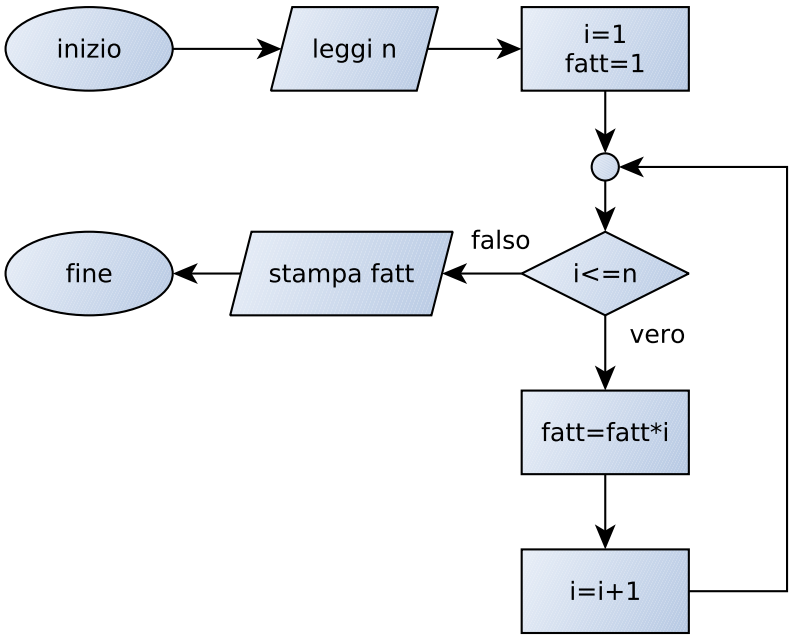
POI

- n = 2 : stampa 0, 2
- n = 5 : stampa 0, 2, 4, 6, 8
- n = 100 : stampa 0, 2, . . . , 198

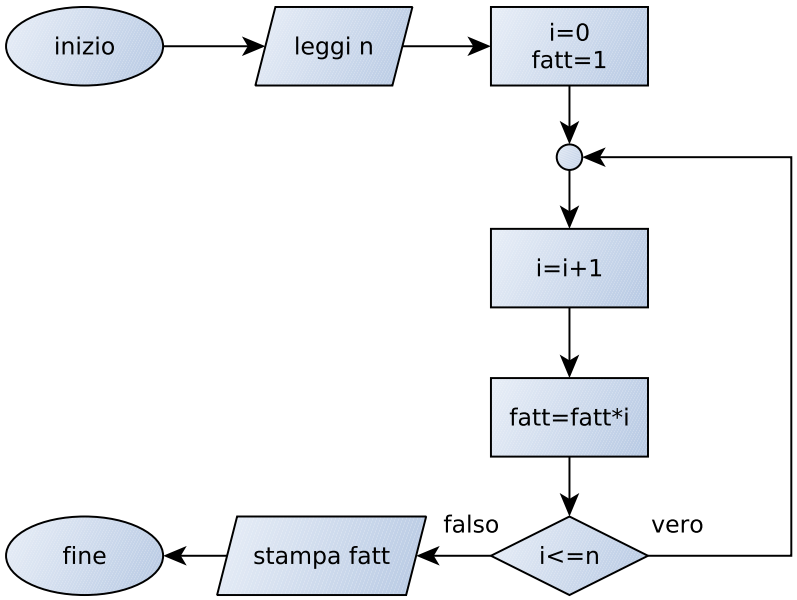
Cercare una soluzione che richieda il minor numero di modifiche rispetto al flowchart precedente.

# Esempio: fattoriale di $n$

$$n! = n \cdot (n-1) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$$




ciclo while-do




ciclo do-while

# Esempio: fattoriale di $n$ (pseudo-codice)

$$n! = n \cdot (n-1) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$$

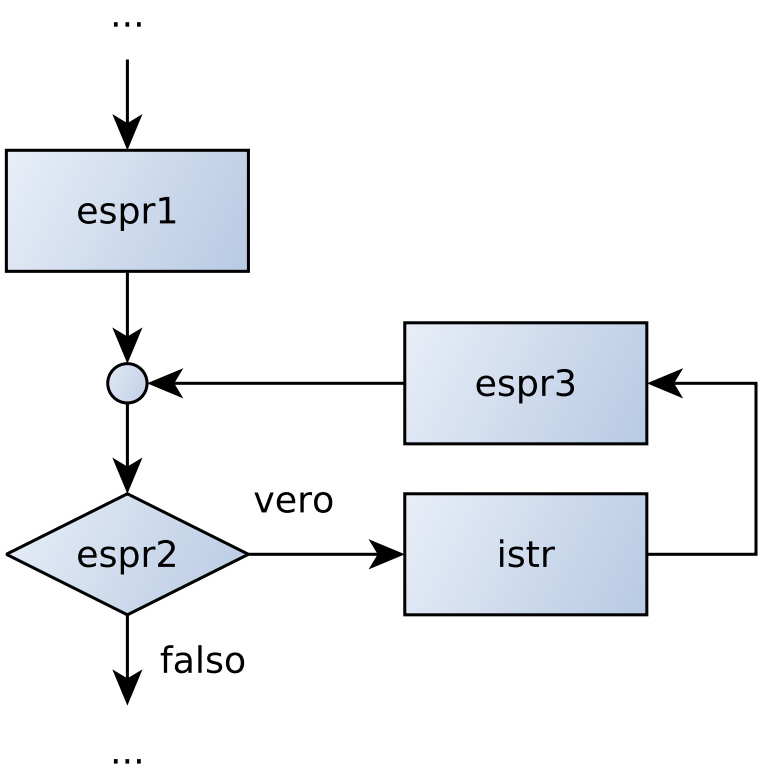
ciclo while-do

```
leggi n
i ← 1
fatt ← 1
while i ≤ n do
    fatt ← fatt * i
    i ← i + 1
end while
stampa fatt
```

ciclo do-while

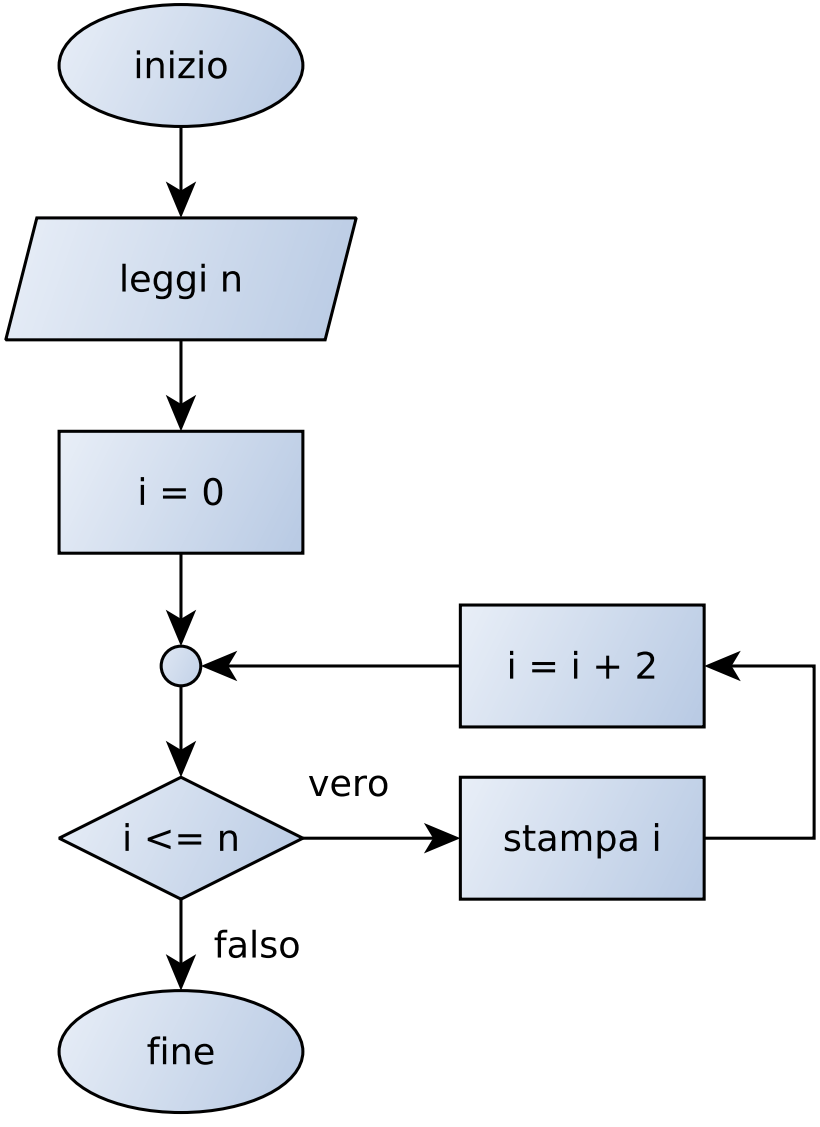
```
leggi n
i ← 0
fatt ← 1
do
    i ← i + 1
    fatt ← fatt * i
while i ≤ n
stampa fatt
```

# Ciclo for




**for** espr1, espr2, espr3 **do**
    istr
**end for**

# Stampa a video i numeri pari positivi minori o uguali a n




**leggi** $n$
**for** $i \leftarrow 0,\ i \leq n,\ i \leftarrow i + 2$ **do**
    **stampa** $i$
**end for**

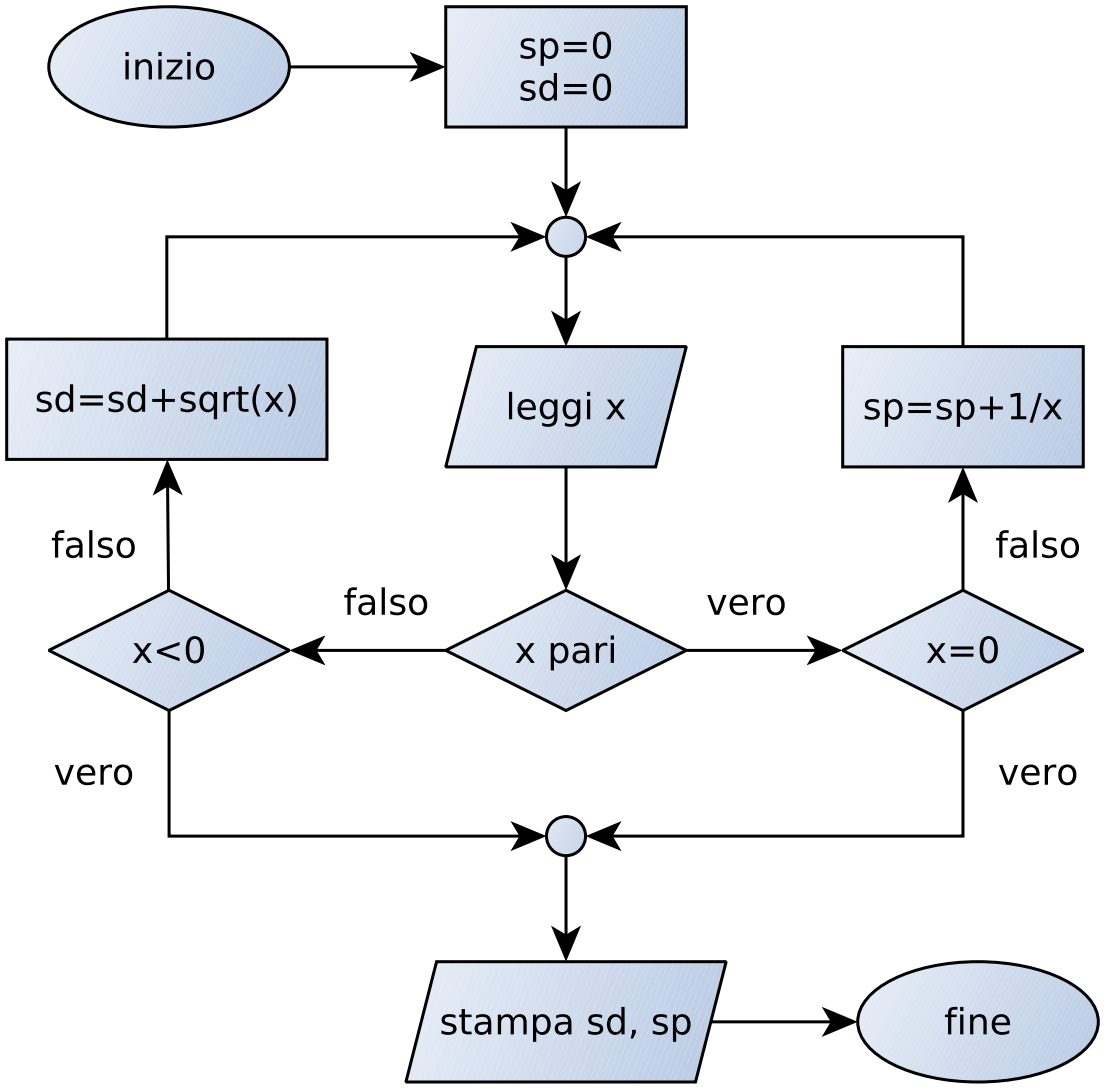
- espr1 : $i \leftarrow 0$
- espr2 : $i \leq n$
- espr3 : $i \leftarrow i + 2$
- istr : **stampa** $i$

# Problema

Realizzare diagramma di flusso che risolve il seguente problema:

- continui a leggere dei valori numerici
- effettui la somma delle radici quadrate di tutti i numeri dispari inseriti
- calcoli la somma dell'inverso di tutti i numeri pari
- il programma termini quando viene inserito un valore che non permette di effettuare correttamente il calcolo nel dominio dei numeri reali
- prima di terminare, si stampino i valori calcolati

# Soluzione: flowchart



# Soluzione: pseudo-codice

$sd \leftarrow 0$
$sp \leftarrow 0$
**repeat**
    leggi val
    **if** val pari **then**
        **if** val $\neq 0$ **then**
            $sp \leftarrow sp + (1/val)$
        **end if**
    **else**
        **if** val $> 0$ **then**
            $sd \leftarrow sd + \sqrt{val}$
        **end if**
    **end if**
**until** (val pari e val $\neq 0$) oppure (val dispari e val $> 0$)
stampa $sd$ e $sp$

# Concetti di equivalenza

## Equivalenza debole

due flowchart (algoritmi) sono debolmente equivalenti se, per ogni insieme di dati in ingresso, generano gli stessi dati in uscita

## Equivalenza forte

due flowchart sono fortemente equivalenti se sono debolmente equivalenti e le rispettive sequenze di computazione sono uguali, per ogni insieme di dati in ingresso

## Equivalenza fortissima

due flowchart sono fortissimamente equivalenti se sono fortemente equivalenti ed inoltre in essi compaiono lo stesso numero di volte gli stessi blocchi elementari

# Problema di realizzabilità degli algoritmi

esiste un problema fondamentale:

> è sicuro che usando solo le strutture di controllo fondamentali non si limita la capacità di realizzare algoritmi?

che equivale a domandarsi

> esistono problemi non risolubili per mezzo delle sole strutture di controllo fondamentali?

la risposta è data dal teorema di Jacopini-Böhm che asserisce la possibilità di realizzare qualunque algoritmo con le sole strutture di controllo fondamentali

## Teorema di Jacopini-Böhm

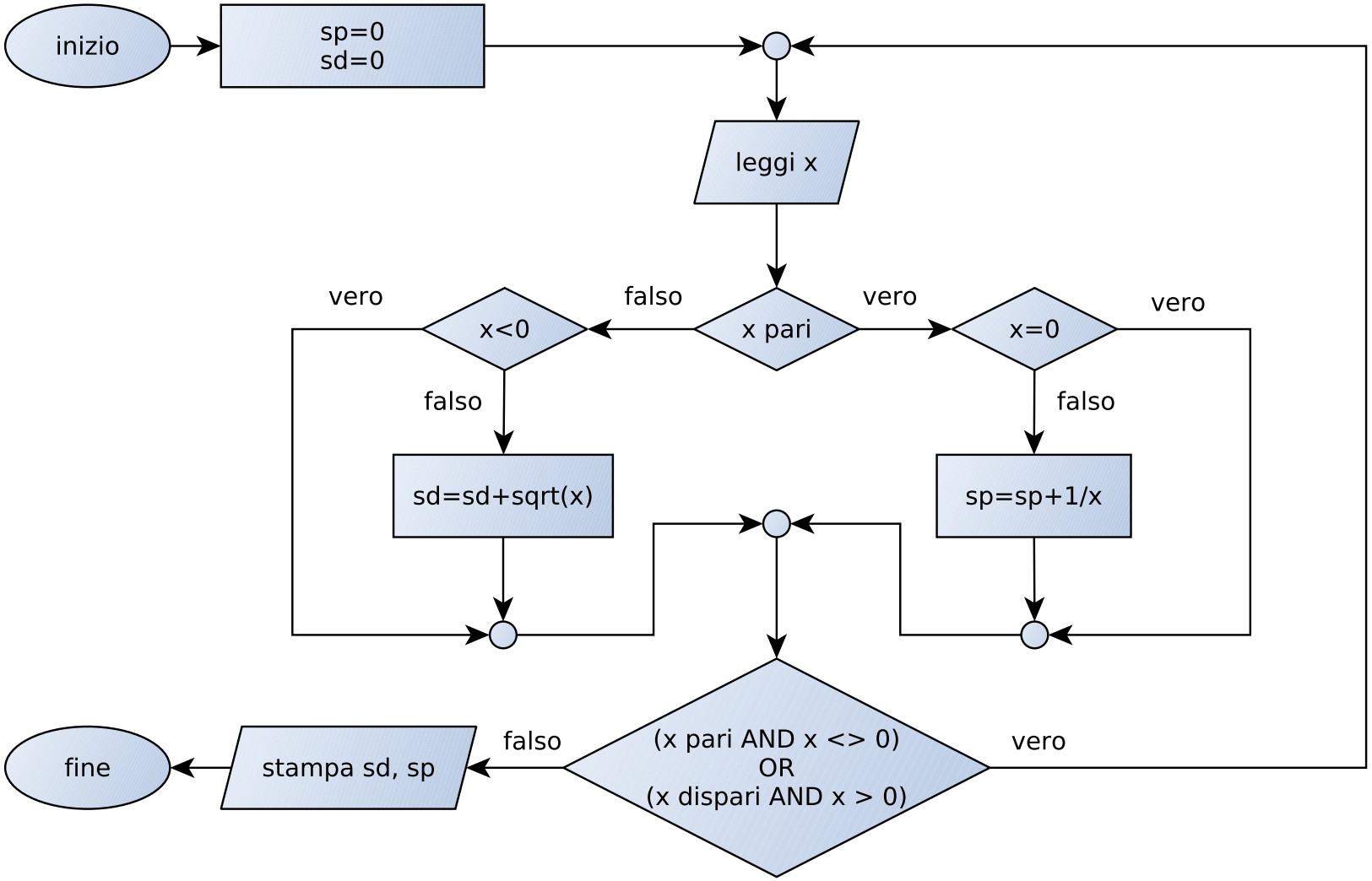
siano

- $\mathcal{P}$ l'insieme di tutti gli algoritmi realizzabili
- $\mathcal{D}$ l'insieme di tutti gli algoritmi realizzabili facendo uso esclusivo delle tre strutture di controllo fondamentali (sequenza, selezione e iterazione)

allora

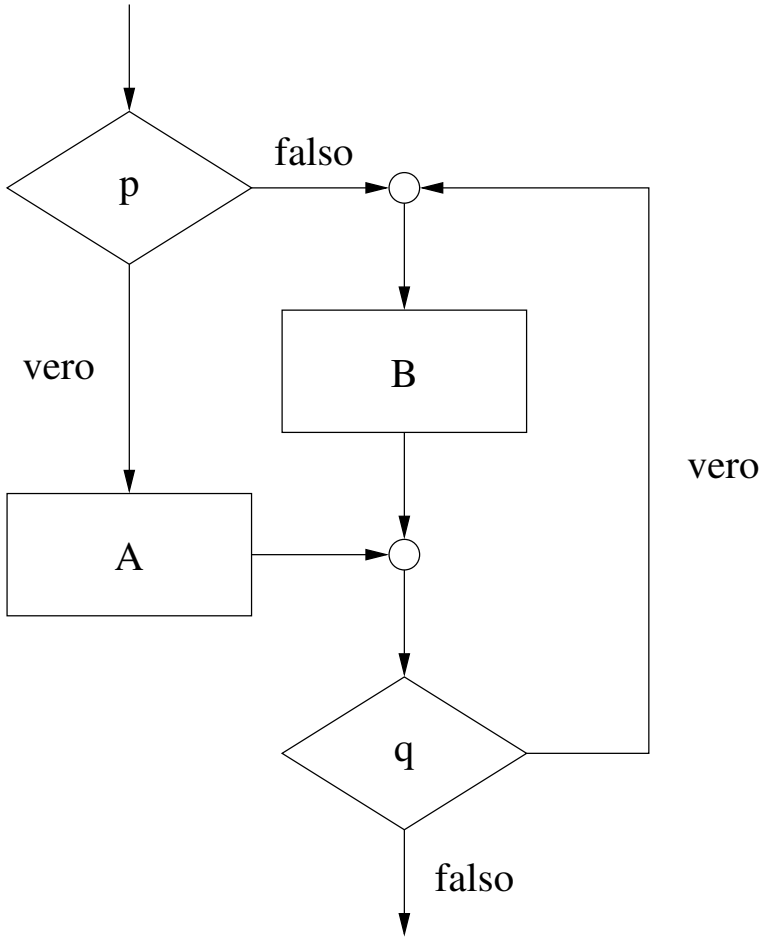
dato un programma $p \in \mathcal{P}$
esiste sempre un programma $d \in \mathcal{D}$
che risulta debolmente equivalente a $p$

# Esempio di flowchart che sfrutta la programmazione strutturata

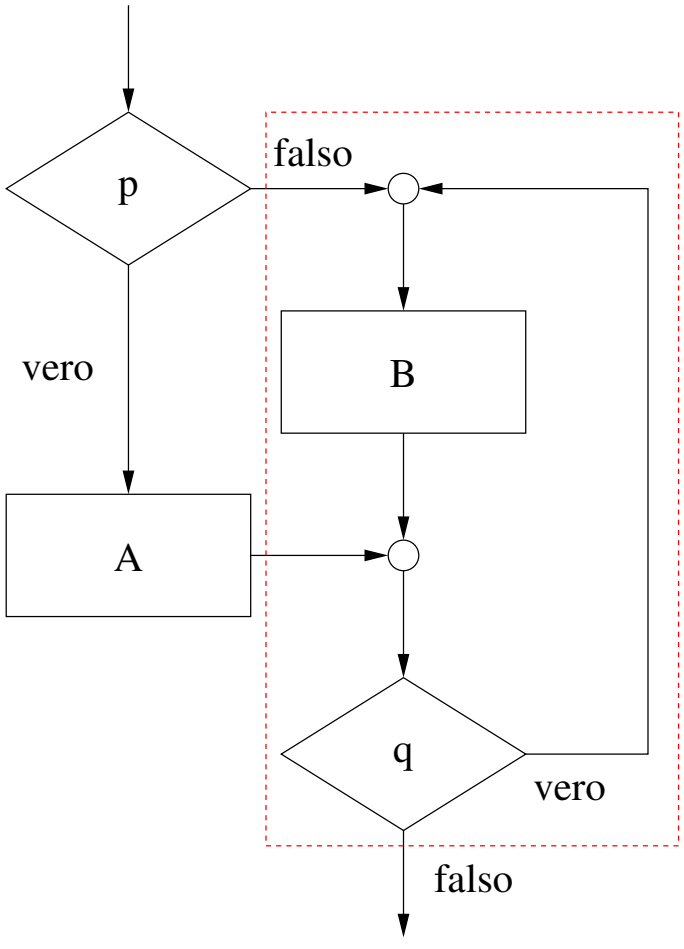


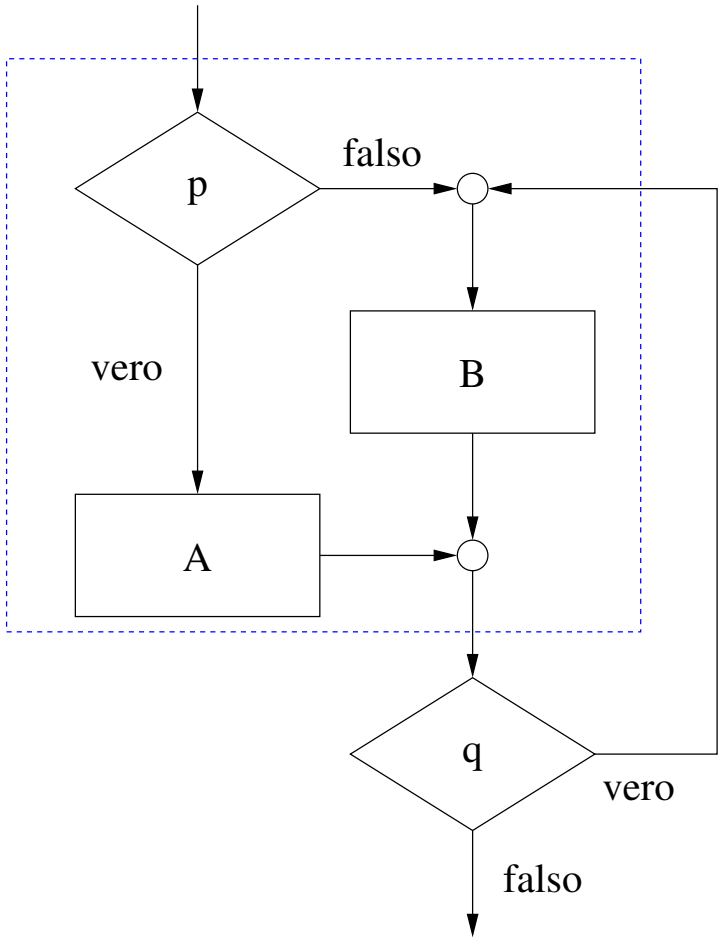
# Esempio di programmazione non strutturata

il flowchart presenta una struttura di controllo che non è realizzata mediante strutture fondamentali



# Esempio di programmazione non strutturata

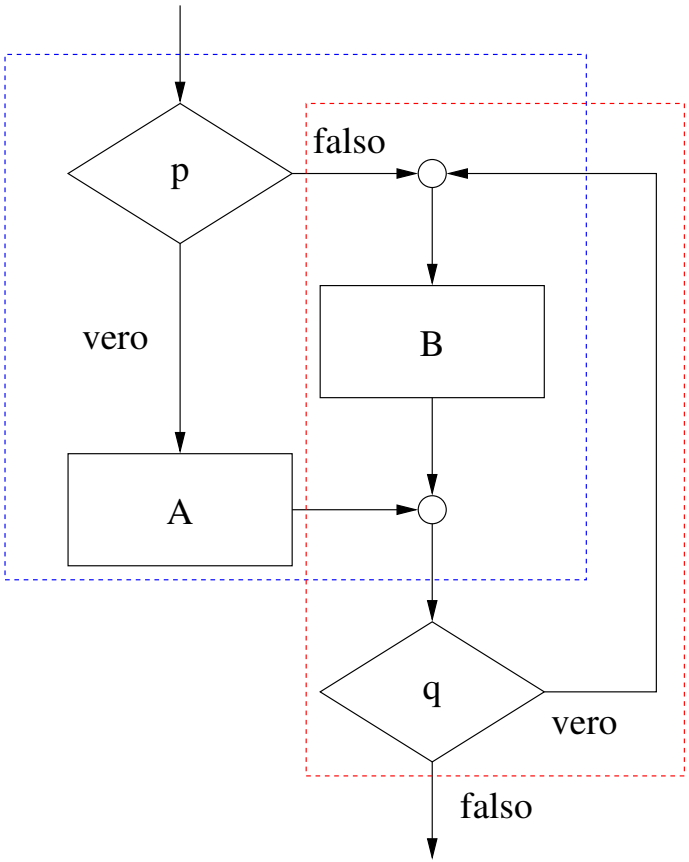


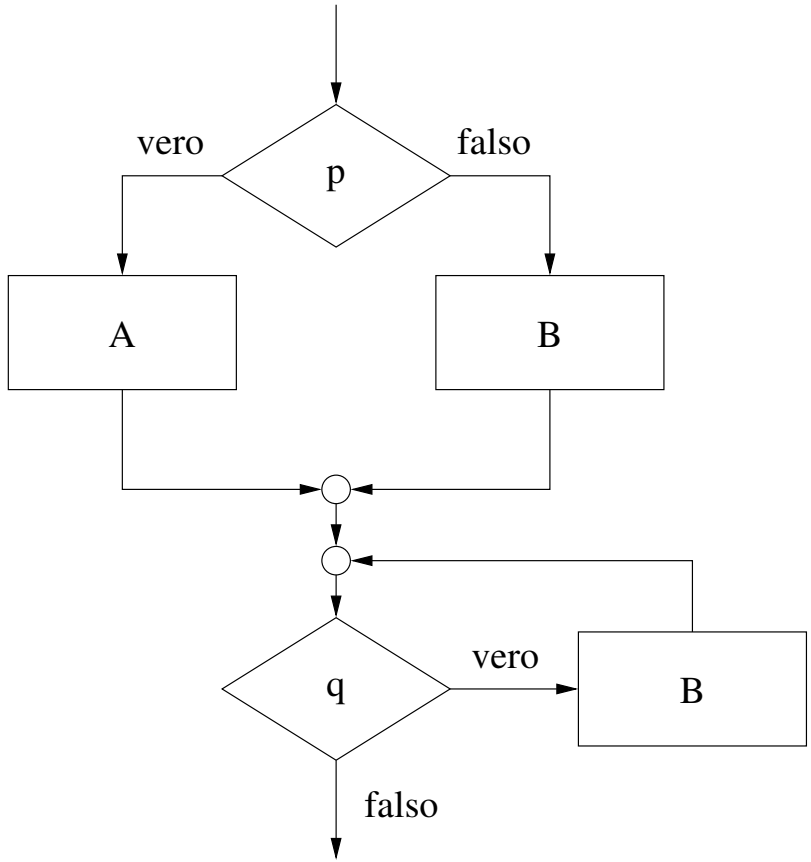




i blocchi evidenziati presentano due archi entranti

# Duplicazione dei blocchi

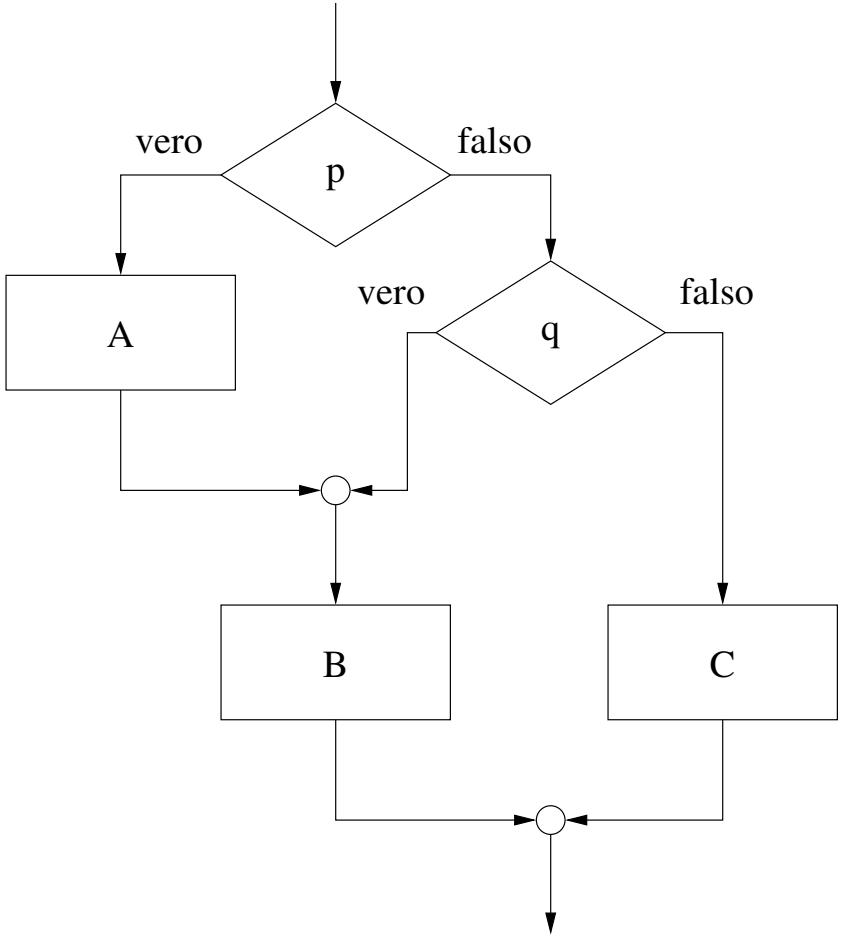
il passaggio da uno schema a blocchi non strutturato ad uno strutturato può avvenire attraverso la duplicazione di blocchi, ottenendo schemi fortemente equivalenti



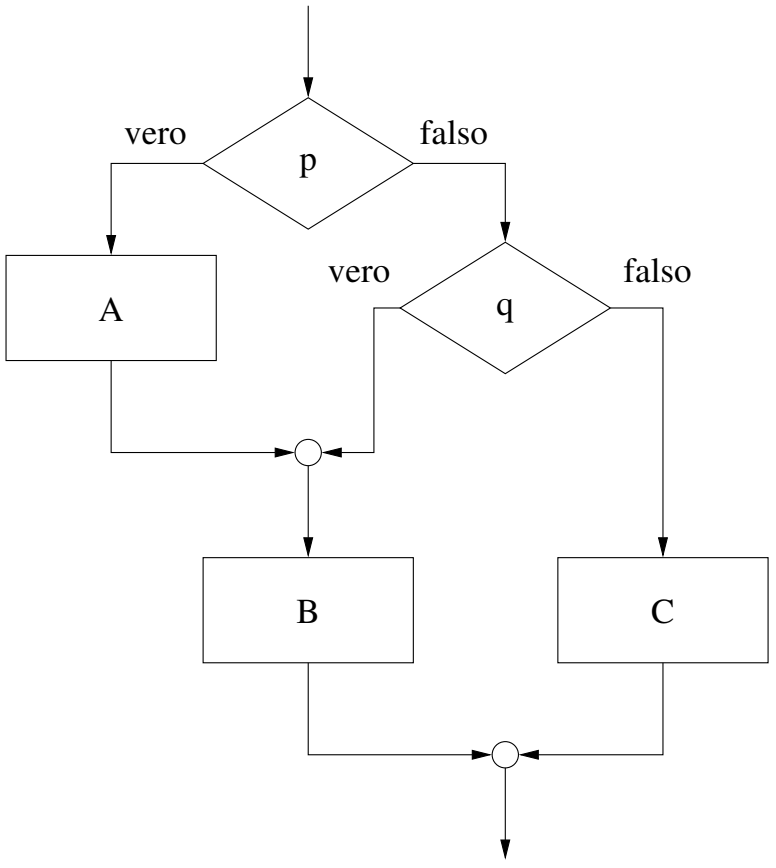



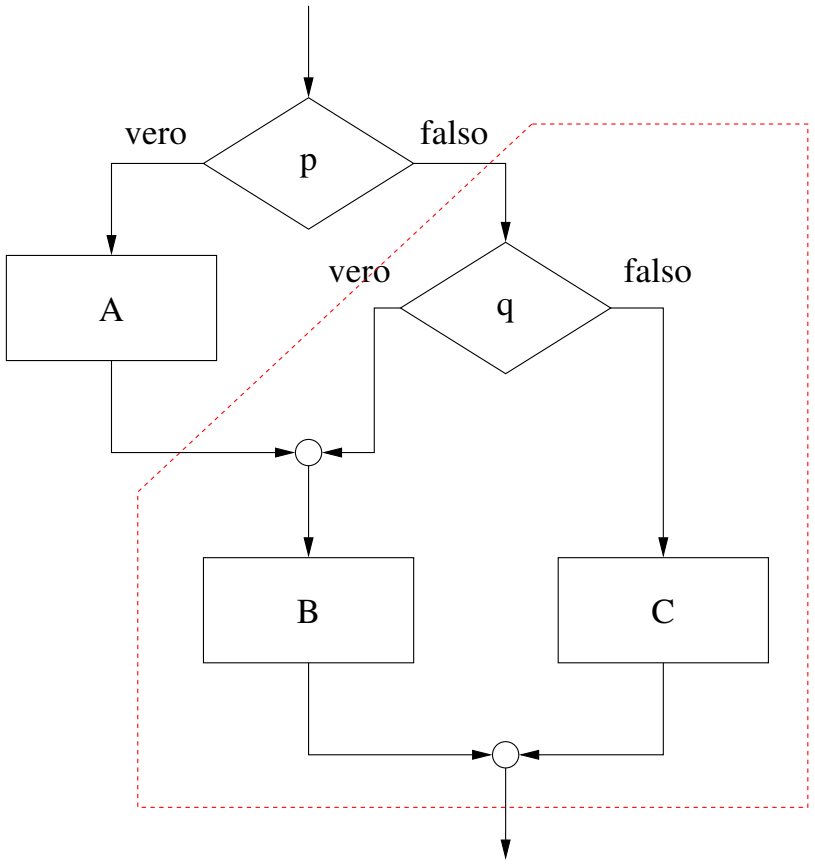
# Selezione anomala

esempio di due costrutti di selezione che si "intersecano" in modo anomalo



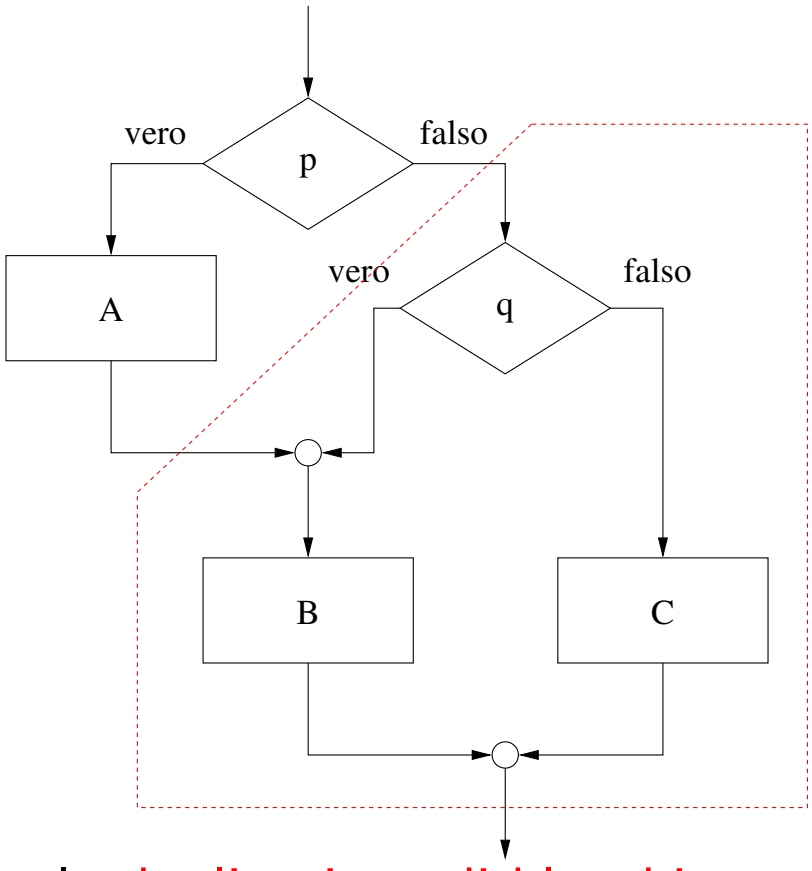
# Selezione anomala

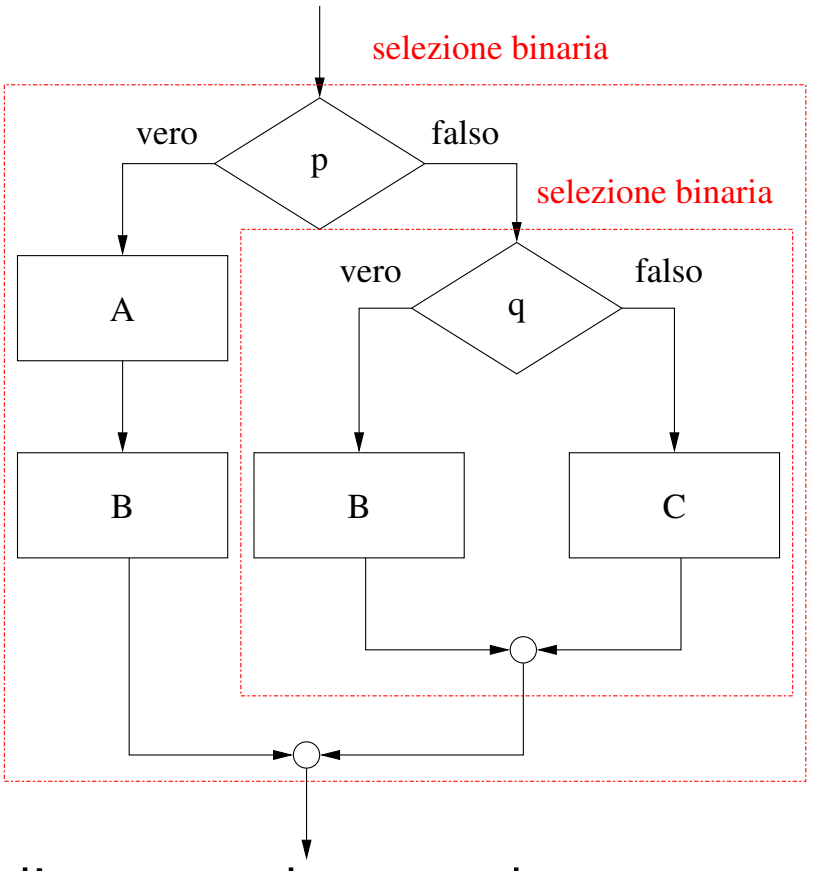



il blocco evidenziato presenta due archi entranti

# Selezione anomala




la duplicazione di blocchi permette di passare da uno schema a blocchi non strutturato ad uno strutturato ottenendo schemi fortemente equivalenti