Algoritmi, linguaggi e programmi

Emilio Di Giacomo e Walter Didimo

Risolvere i problemi

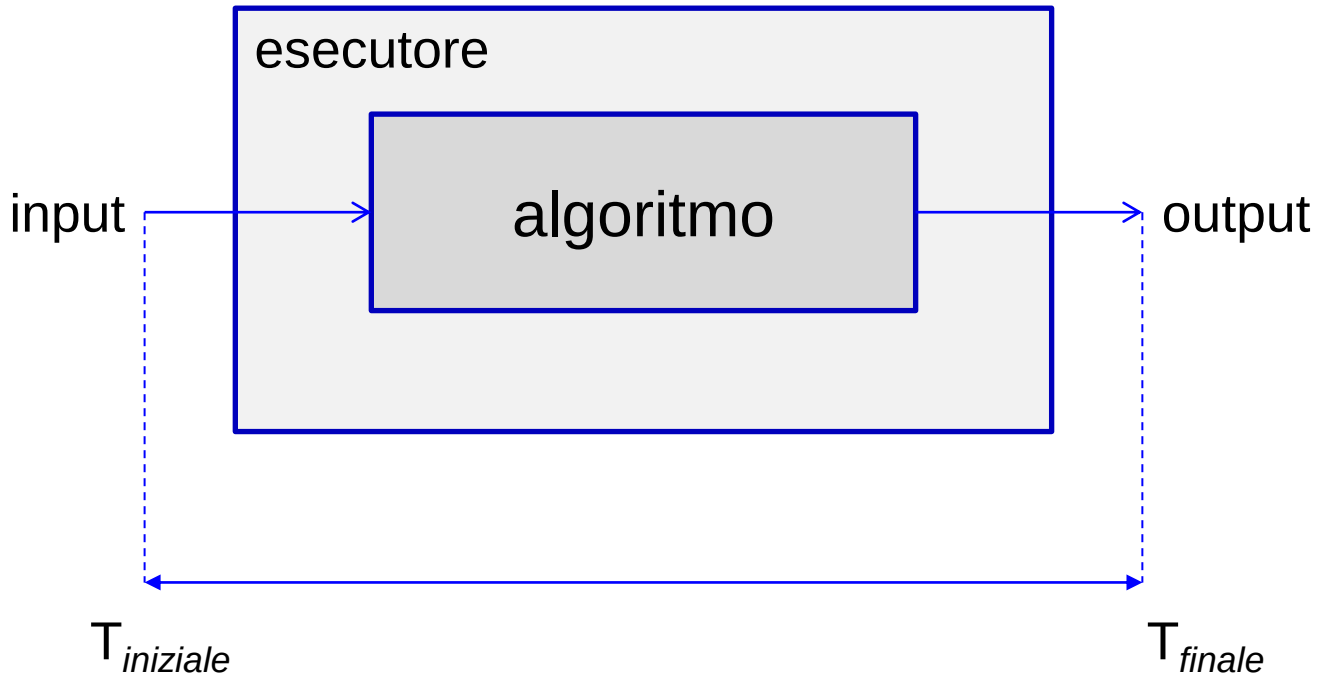
- Il calcolatore permette di risolvere molti problemi, … ma sfortunatamente non tutti

- Il concetto di “algoritmo” consente di stabilire quali problemi sono risolvibili con un calcolatore
  - nel seguito definiremo il concetto di “algoritmo”

Esecutore

- Supponiamo dato un qualche esecutore
  - "dispositivo" in grado di eseguire un insieme predefinito di azioni elementari, dette istruzioni
  - ogni istruzione è eseguita in un *tempo finito*
- Un algoritmo per tale esecutore è una *sequenza finita di istruzioni* del suo insieme:
  - l'algoritmo può prendere in ingresso dei dati iniziali, detti input dell'algoritmo
  - l'algoritmo può generare in uscita dei dati, detti output dell'algoritmo

Schema di algoritmo



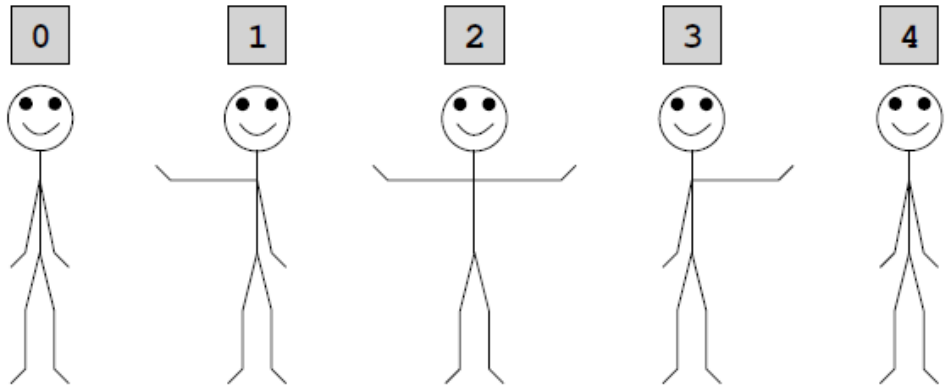
Ancora sugli esecutori

- Il concetto di “esecutore” è generalizzabile
  - ad esempio, l’esecutore può essere un ginnasta che sa eseguire semplici esercizi

Algoritmo: “Esercizio-Ginnico”

input: *<posizione iniziale>*
output: *<posizione finale>*

1. solleva braccio destro di 90°
2. solleva braccio sinistro di 90°
3. abbassa braccio destro di 90°
4. abbassa braccio sinistro di 90°



  - o ancora un cuoco, che sa eseguire un procedimento (algoritmo) per preparare una pietanza (input = ingredienti, output = pietanza)

Algoritmo: proprietà

- *non ambiguità*: ogni istruzione deve essere univocamente interpretabile dall'esecutore
- *eseguibilità*: ogni istruzione deve essere effettivamente tra quelle eseguibili dall'esecutore
- *terminazione*: l'algoritmo deve terminare in un tempo finito
- *univocità*: il risultato finale dell'algoritmo (output) deve essere univocamente determinabile (in funzione dell'input)

Problemi e Algoritmi

- Indichiamo con P un problema esprimibile in termini di input ed output
  - esempio: dato un intero positivo *n* (input), stabilire se *n* è un numero primo (l'output è una risposta del tipo vero/falso)

- P è risolvibile tramite un esecutore E se esiste un algoritmo A per E in grado di fornire un output di P corretto per ogni possibile input di P
  - input ed output di A coincidono con quelli di P

Il calcolatore come esecutore

- Un calcolatore è un esecutore che sa svolgere istruzioni molto elementari:
  - scrivi un numero (binario) in una cella di memoria;
  - leggi un numero da una cella di memoria;
  - confronta due numeri;
  - somma/sottrai/moltiplica/dividi due numeri;
  - salta ad una certa istruzione,
  - ecc.

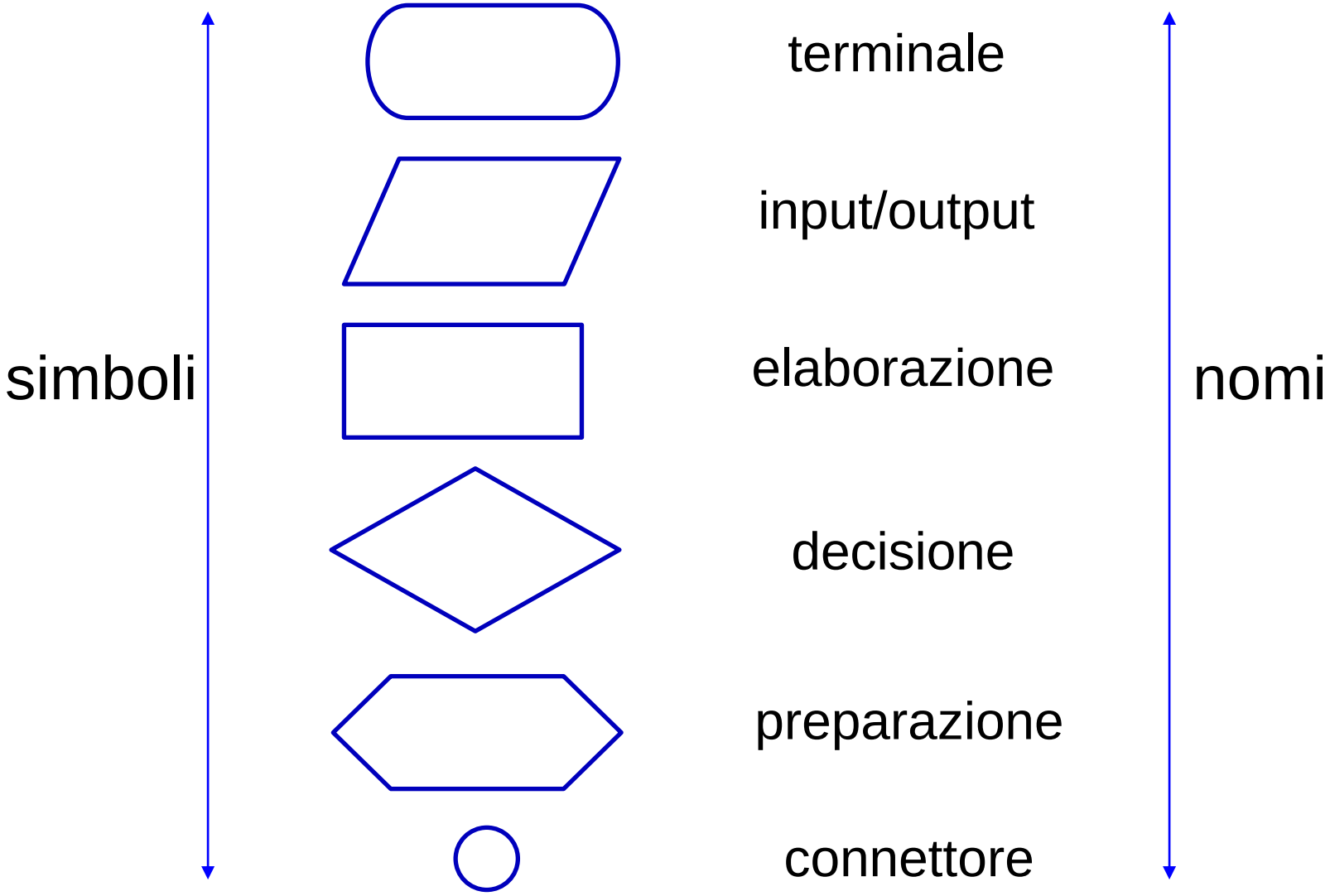
Algoritmi per calcolatore

- Un algoritmo per un calcolatore è dunque una sequenza delle sue istruzioni elementari
- Nella descrizione di un algoritmo, si usa il concetto di variabile in luogo di «cella di memoria»
  - una variabile ha un nome simbolico (come nella matematica)
  - una variabile può memorizzare un dato che varia nel tempo; quando memorizzo (cioè scrivo) un valore in una variabile, sovrascrivo il precedente valore
  - Se **x** è una variabile ed **a** è un valore, **x ← a** indica che vogliamo memorizzare **a** in **x**

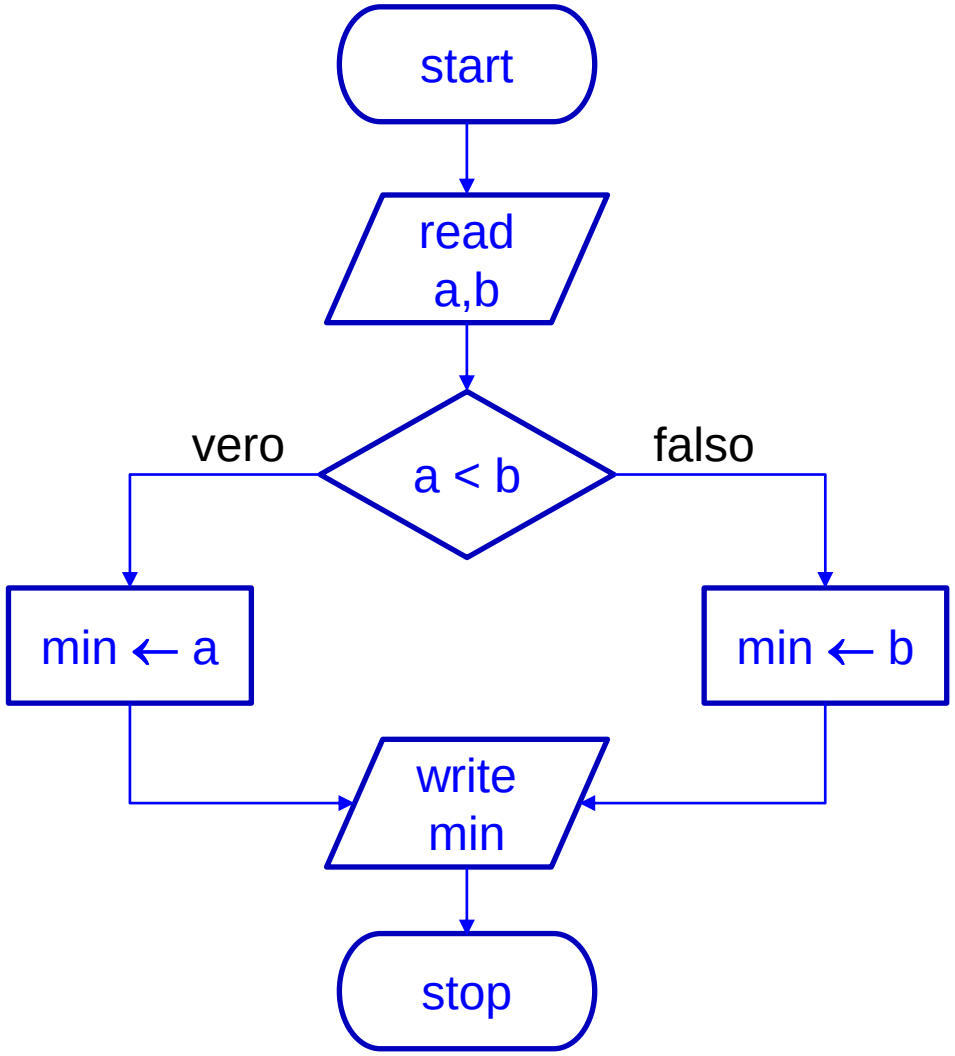
Descrizione di algoritmi

- Come descriviamo un algoritmo?
- Ci sono due metodi alternativi molto diffusi:
  - tramite un diagramma (linguaggio grafico), detto diagramma di flusso, o flow-chart
  - tramite uno pseudo-codice, formalismo testuale simile alle istruzioni di un «linguaggio di programmazione», ma più generico

Flow-chart


simboli
terminale
input/output
elaborazione
decisione
preparazione
connettore
nomi

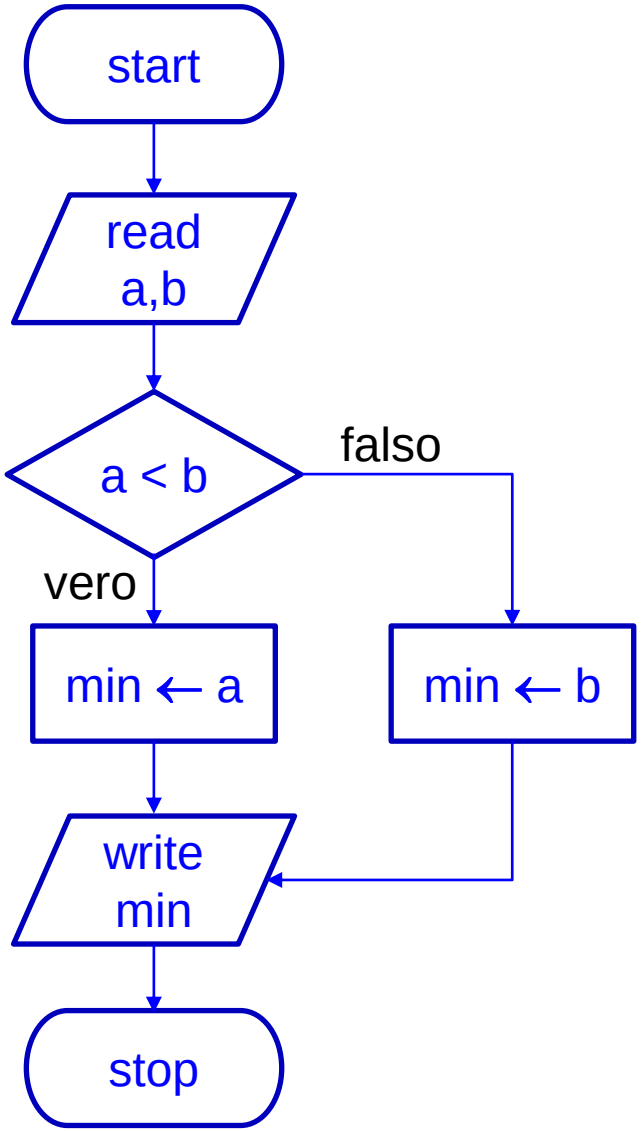
Flow-chart: un esempio



algoritmo che riceve in input due numeri a e b e che stampa in output il minimo tra i due

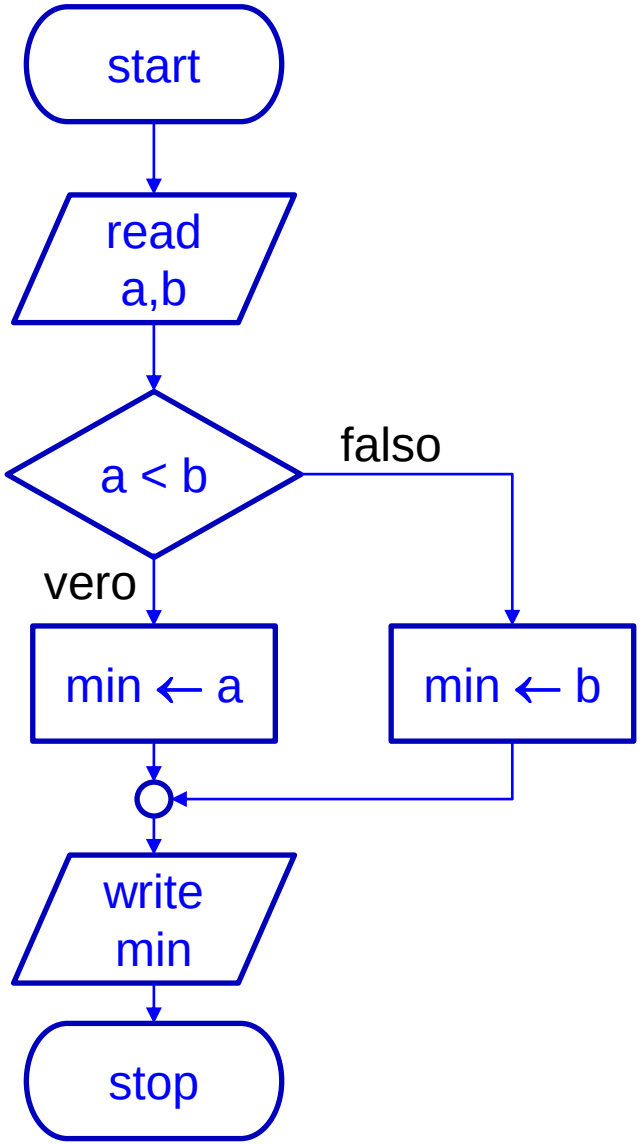
a, **b**, e **min** sono variabili dell'algoritmo

Flow-chart: un altro diagramma



Questa visualizzazione risulta più lineare, sebbene l'algoritmo sia lo stesso

Flow-chart: o ancora



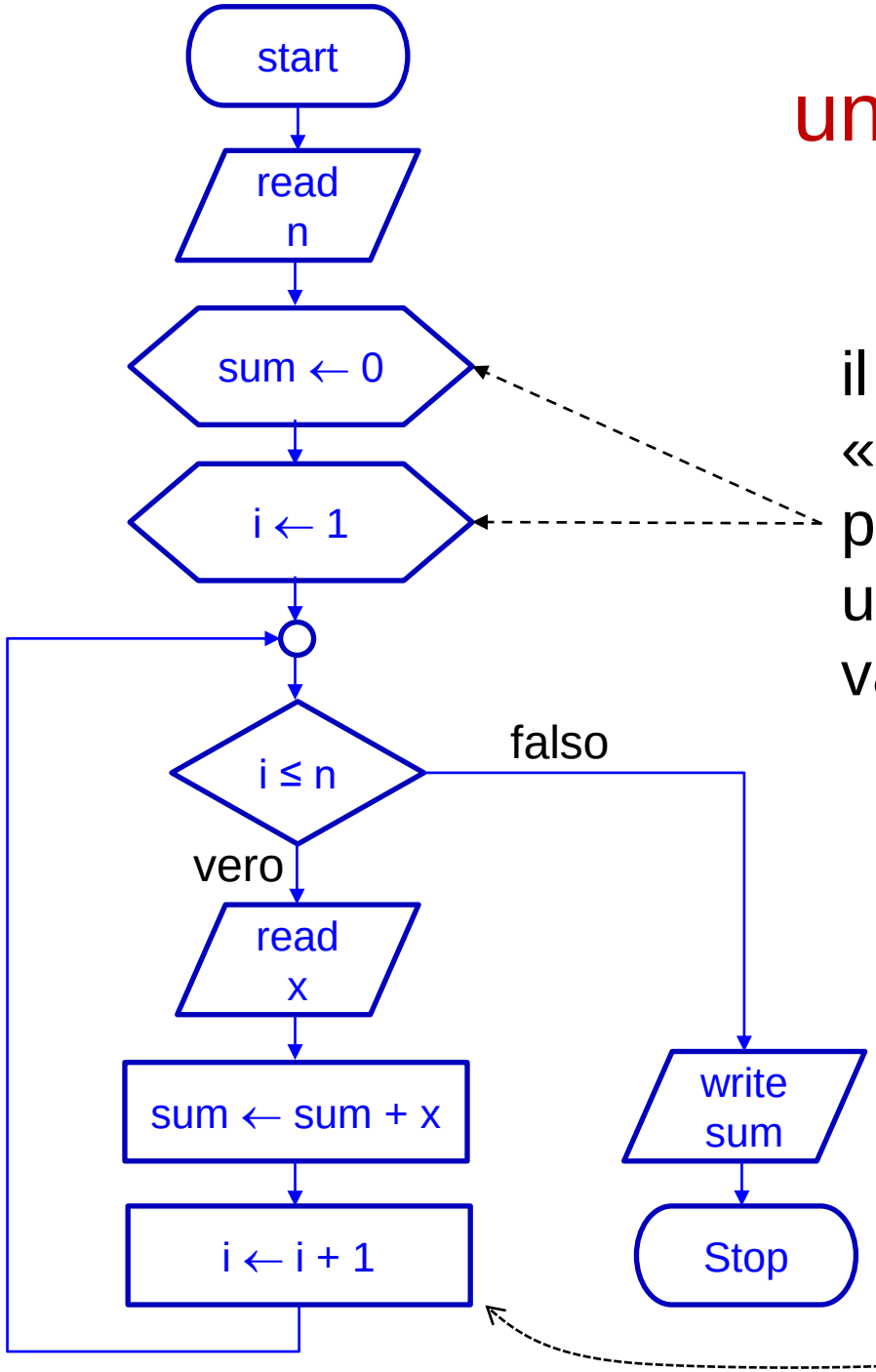
Questa visualizzazione utilizza il simbolo «connettore»

Un algoritmo più complesso

- Realizziamo il diagramma di flusso di un algoritmo che:
  - riceve in input **n** numeri: **x1**, **x2**, .., **xn**
  - stampa in output la somma **sum** di tali numeri

- Strategia:
  - leggi **n** (cioè quanti sono i numeri da sommare)
  - inizializza **sum** con il valore 0
  - leggi i numeri **x1**, …, **xn**, uno alla volta
    - ogni volta che leggi un numero incrementa **sum** con il valore del numero letto
  - stampa **sum**

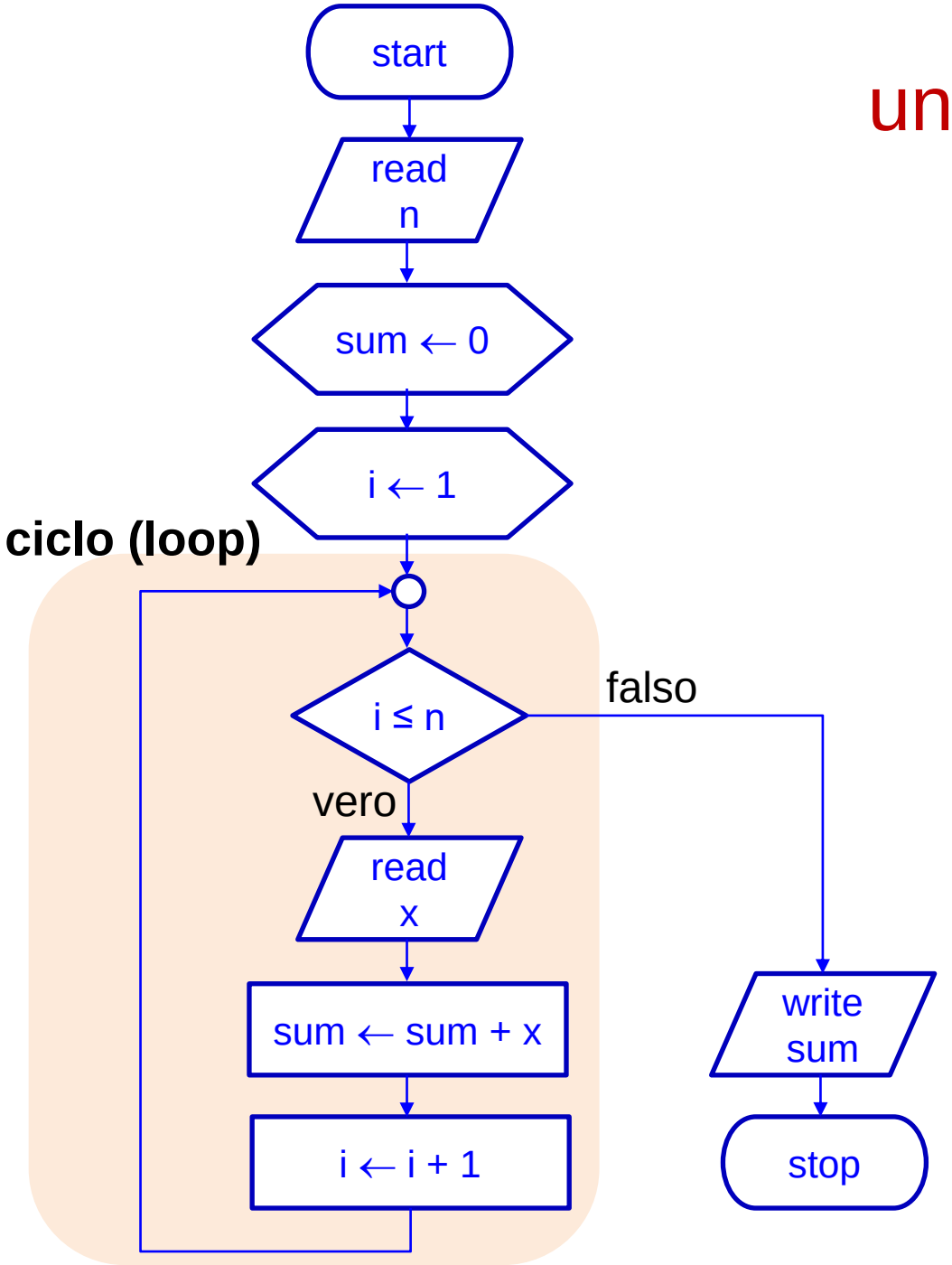
un possibile flow-chart



il simbolo di «preparazione» è utilizzato per inizializzare (cioè dare un primo valore ad) una variabile

La variabile **i** è usata per «contare» quanti numeri leggo; per questo si chiama anche contatore

un possibile flow-chart



Pseudo-codice

- Un diagramma di flusso richiede una certa occupazione di spazio
  - può essere dispendioso in termini di spazio rappresentare un algoritmo complesso tramite un flow-chart

- L'utilizzo di uno pseudo-codice fornisce una descrizione equivalente, ma più compatta

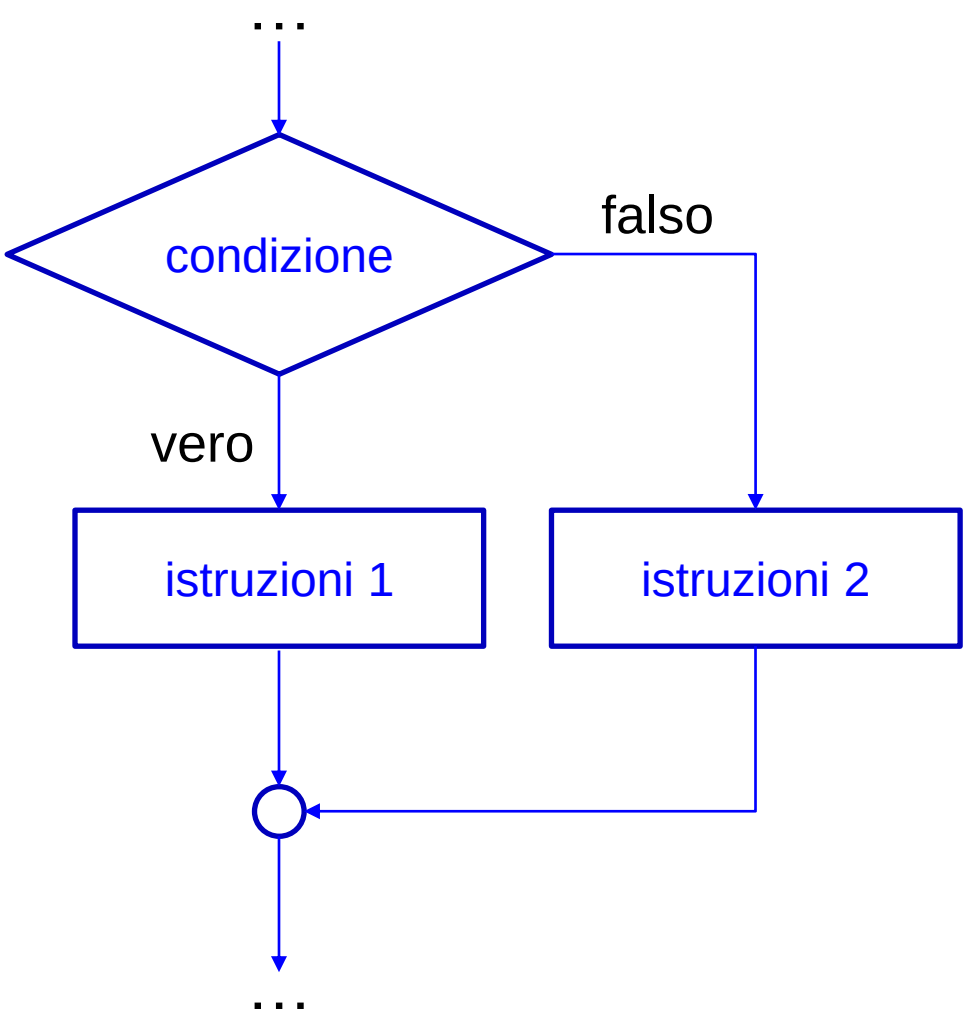
Un possibile pseudo-linguaggio

- Esistono diversi pseudo-linguaggi per scrivere pseudo-codici, ma sono tutti molto simili tra loro
- Definiamo un nostro pseudo-linguaggio, usando un numero minimale di istruzioni di controllo

L’istruzione «if .. else»

```
…

if (condizione)
        istruzioni 1
else
        istruzioni 2
…
```

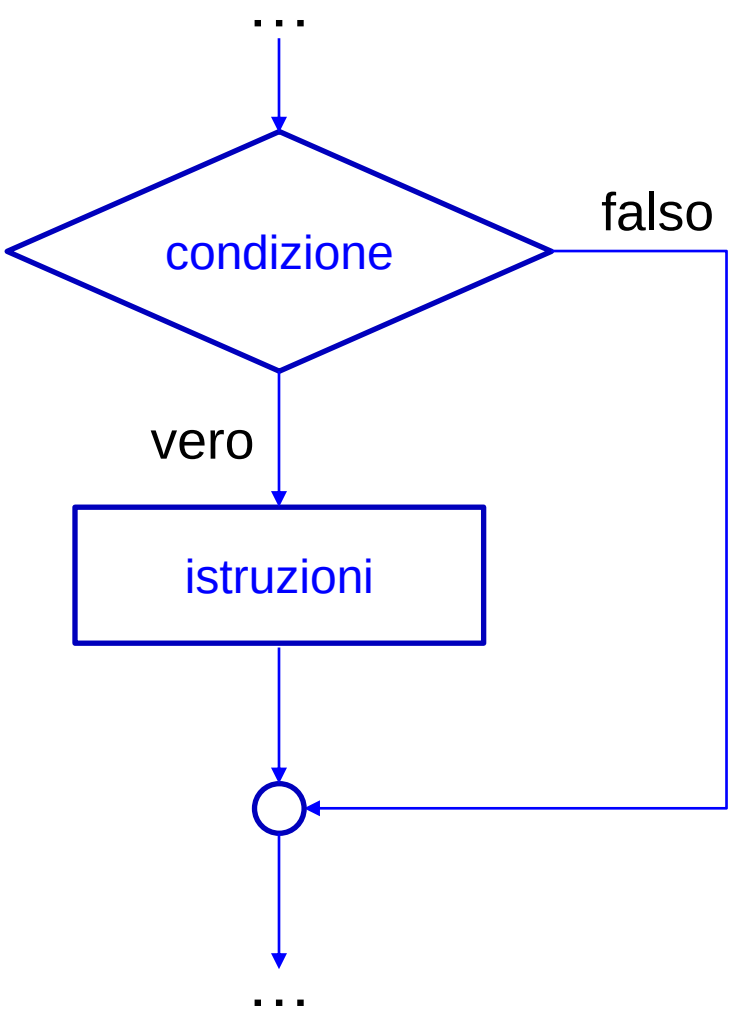


L’istruzione «if»

…

if (condizione)
istruzioni

…

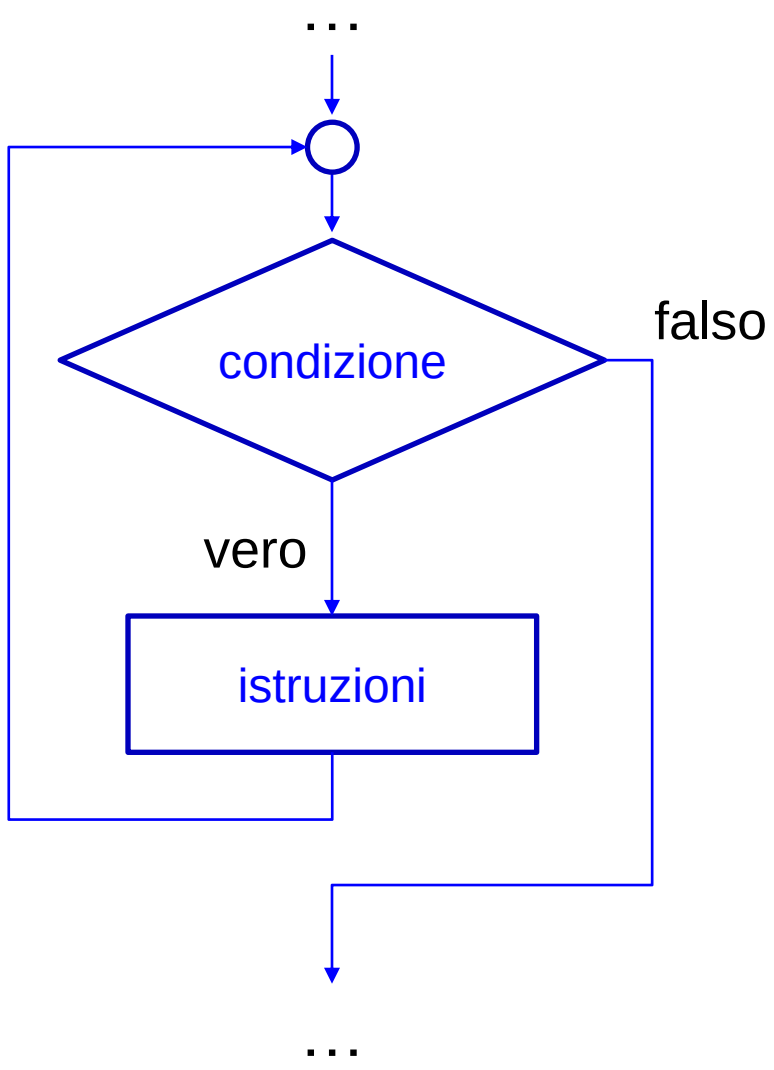


L’istruzione «while»

…

while (condizione)
istruzioni

…

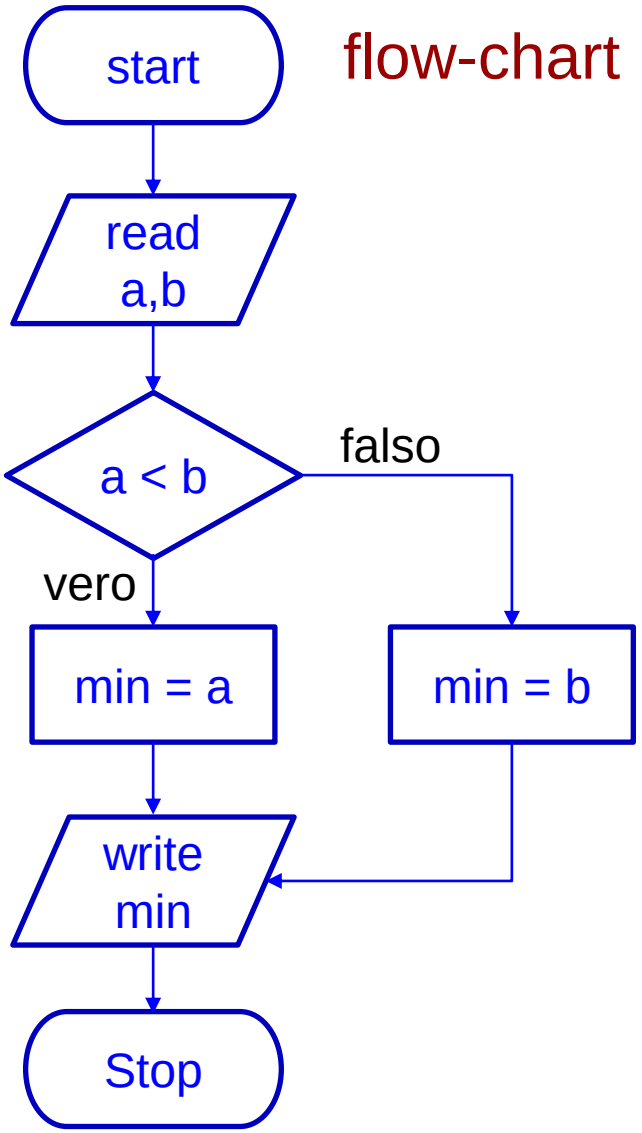


Pseudo-codice: un esempio

algoritmo che riceve in input due numeri **a** e **b** e che stampa in output il minimo tra i due

pseudo-codice

```
read a, b;
if (a < b)
        min ← a;
else
        min ← b;
write min;
```



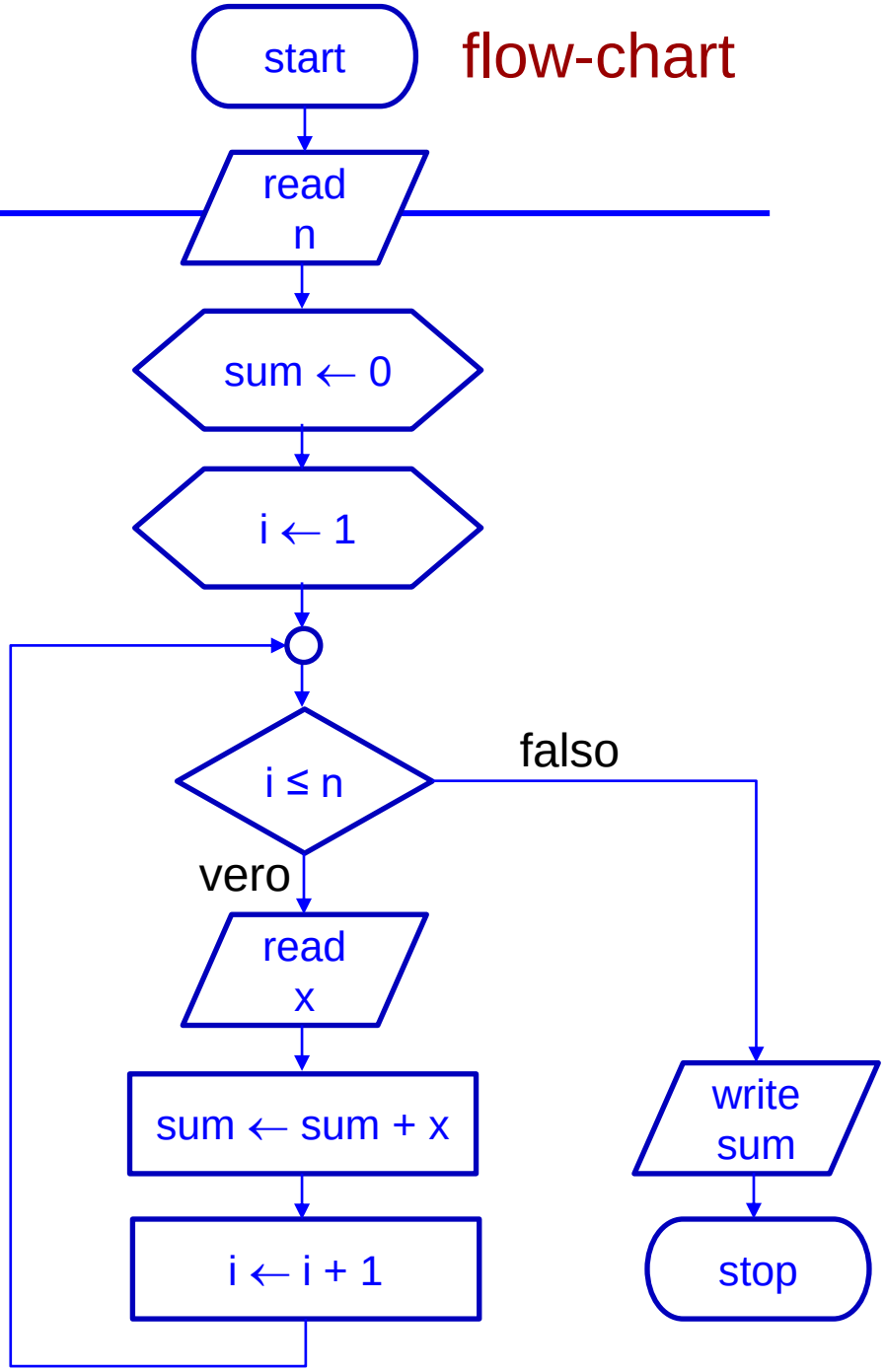
flow-chart

Altro esempio

algoritmo che riceve in input **n** numeri e che stampa in output la somma **sum** di tali numeri

pseudo-codice

```
read n;
sum ← 0;
i ← 1;
while (i ≤ n)
        read x;
        sum ← sum + x;
        i ← i + 1;
write sum;
```



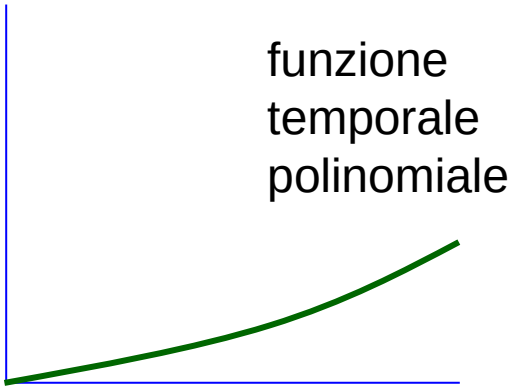
Efficienza di un algoritmo

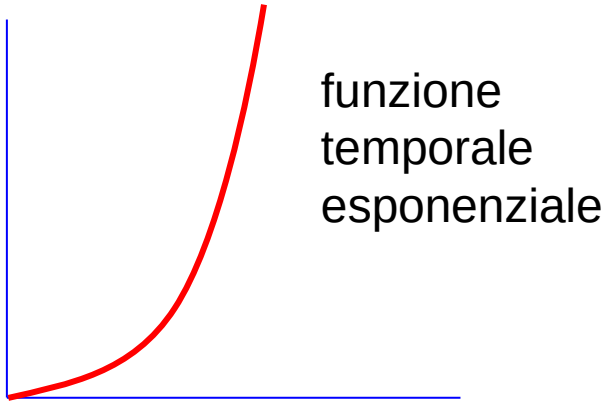
- Il tempo di esecuzione effettivo di un algoritmo da parte di un calcolatore dipende fortemente dalle caratteristiche fisiche del calcolatore

- Il tempo di calcolo di un algoritmo si esprime attraverso il numero di istruzioni elementari richieste da una sua esecuzione
  - questo numero va espresso come funzione della dimensione dell'input
  - il tempo di calcolo fornisce una stima teorica dell'efficienza dell'algoritmo

Problemi trattabili e non

- I problemi risolvibili tramite un calcolatore si classificano in:
  - problemi trattabili: se è noto un algoritmo con tempo di calcolo polinomiale nella dimensione dell'input
  - problemi non trattabili: se sono noti solo algoritmi con tempo di calcolo esponenziale nella dimensione dell'input





Linguaggi e programmi

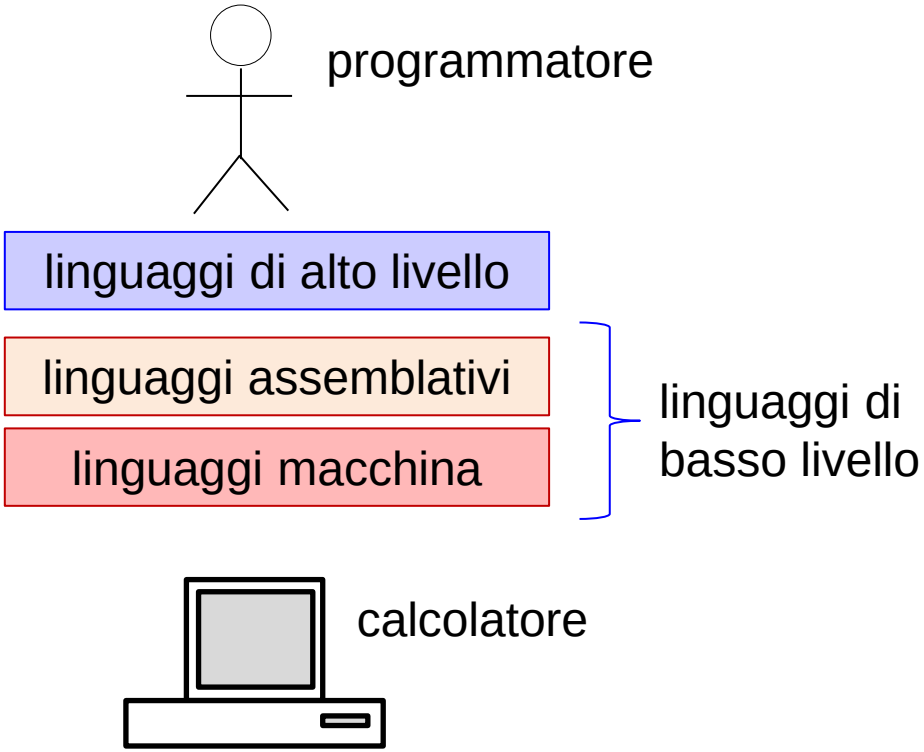
- Per poter essere eseguito da un calcolatore, un algoritmo deve essere trascritto in una forma che sia ad esso comprensibile (direttamente o indirettamente):
  - l'insieme delle regole di trascrizione (cioè il linguaggio utilizzato a tal fine) si chiama linguaggio di programmazione
  - Il risultato della trascrizione si chiama programma
  - Il procedimento di trascrizione si chiama implementazione
  - colui che scrive il programma si chiama programmatore

Linguaggi di programmazione

- Un linguaggio di programmazione definisce due tipi di regole:
  - regole sintattiche: stabiliscono in che modo si possono scrivere programmi sintatticamente (cioè “grammaticalmente”) corretti, ossia quali sono i costrutti sintatticamente validi
  - regole semantiche: stabiliscono quale sia l’effetto di un costrutto sintatticamente valido; permettono dunque di capire quale sarà il comportamento del calcolatore quando eseguirà un programma sintatticamente corretto.

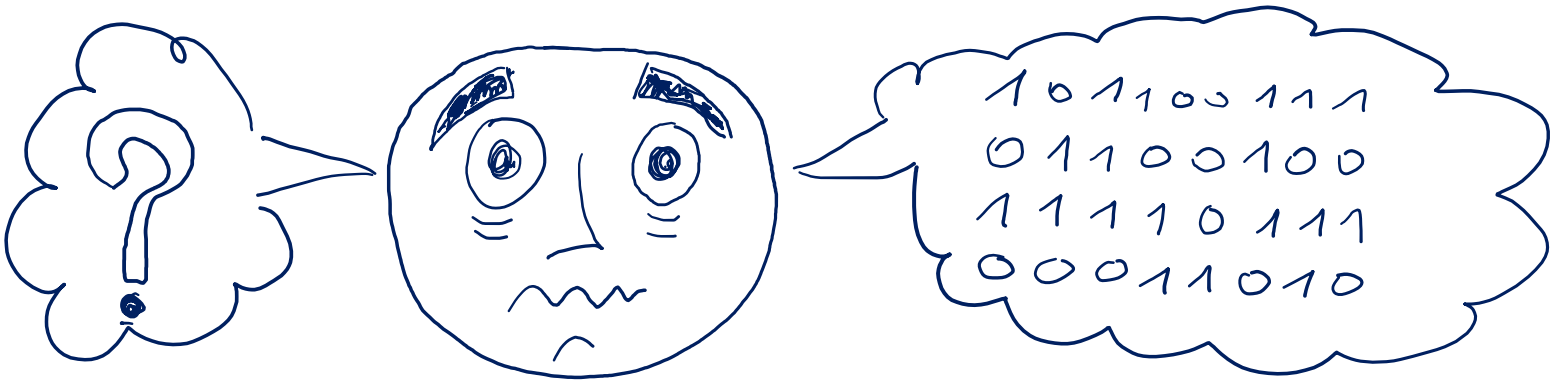
Classificazione dei linguaggi

- Esistono molti linguaggi di programmazione, che è possibile classificare in tre categorie principali, in base al livello di astrazione dal calcolatore



Linguaggi macchina

- I linguaggi macchina sono gli unici direttamente comprensibili da un calcolatore
- Ogni microprocessore conosce alcune istruzioni macchina molto semplici, eseguibili direttamente in hardware
  - le istruzioni macchina e gli eventuali parametri sono codificati con cifre binarie

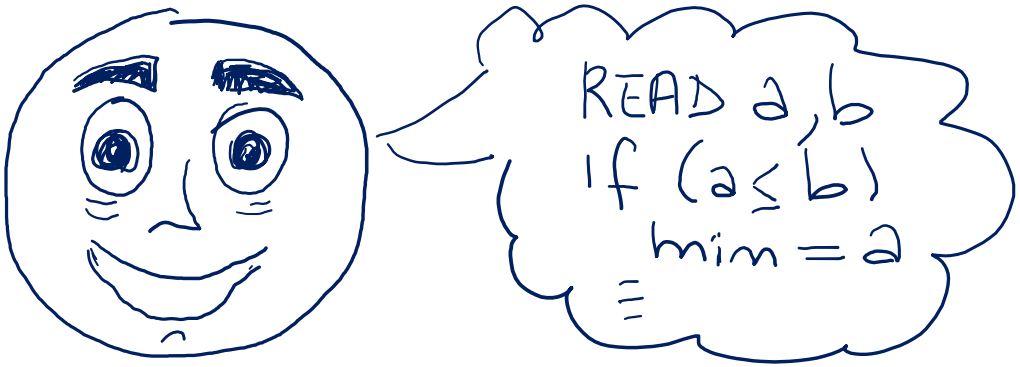


Linguaggi assemblativi

- I linguaggi assemblativi hanno un livello di astrazione simile a quello dei linguaggi macchina, ma sono linguaggi *mnemonici*
  - ogni istruzione è codificata con un “nome” invece che con una sequenza di cifre binarie
  - ciò aiuta il programmatore nel ricordare le tipologie di istruzioni
- Un programma in linguaggio assemblativo deve essere tradotto in linguaggio macchina
  - la traduzione avviene per opera di un software chiamato assemblatore

Linguaggi di alto livello

- I linguaggi di alto livello permettono di scrivere programmi in modo molto più semplice
  - ogni istruzione corrisponde a più istruzioni macchina
  - sono più adatti alla scrittura di programmi complessi



- Un programma scritto con un linguaggio L di alto livello si chiama codice sorgente in L

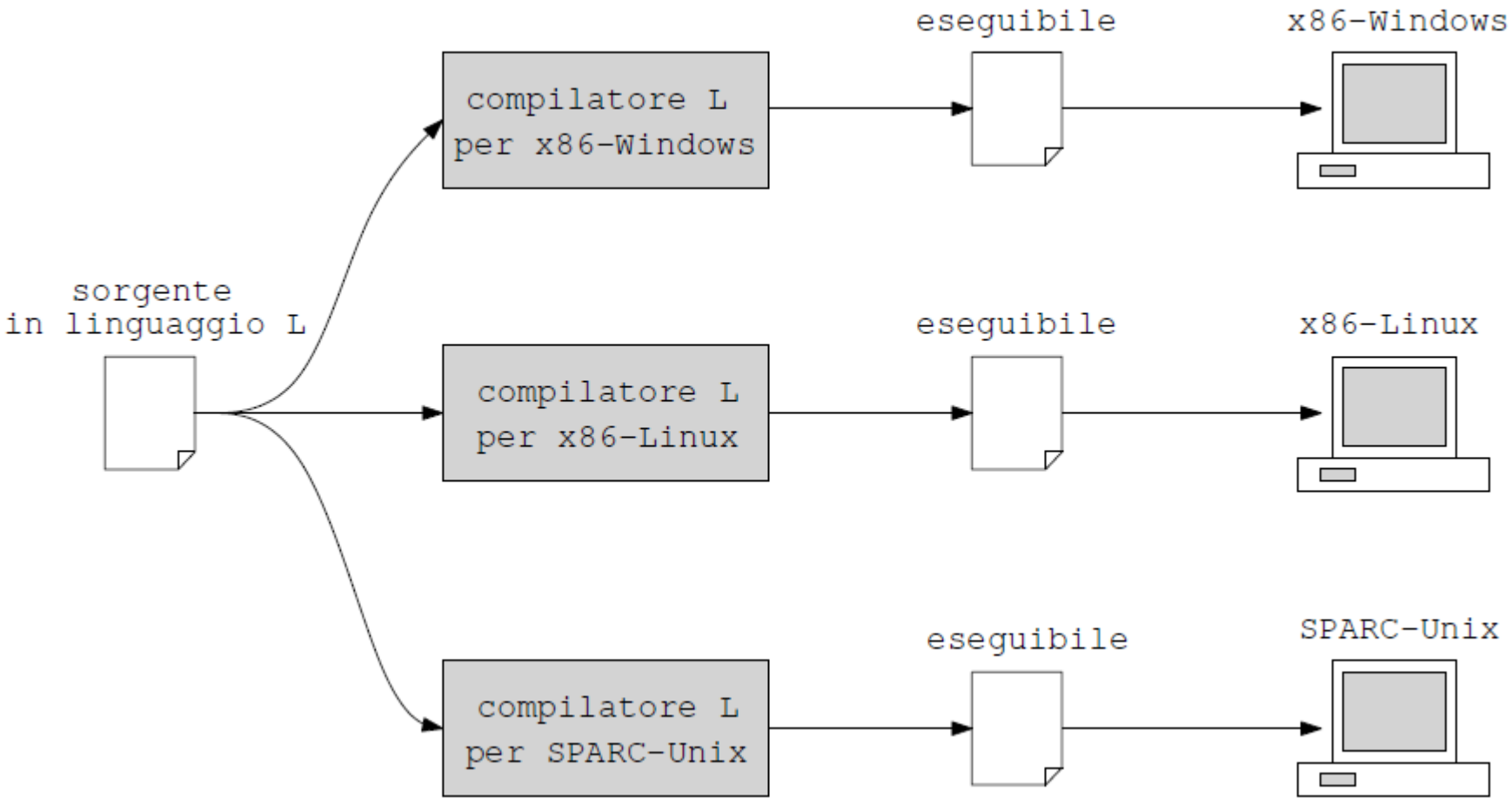
Traduttori per linguaggi di alto livello

- Per poter essere eseguito, un codice sorgente in L deve essere tradotto in un equivalente codice macchina, detto codice eseguibile
- Un traduttore per L è un software che genera automaticamente il codice eseguibile a partire dal codice sorgente in L
- Esistono due principali categorie di traduttori:
  - compilatori
  - interpreti

Compilatori

- I compilatori effettuano *l'intera* traduzione del codice sorgente prima di iniziare l'esecuzione:
  - la fase di traduzione si chiama compilazione
  - durante la compilazione vengono rilevati e segnalati eventuali errori sintattici
  - il risultato della compilazione dipende dalla specifica piattaforma (linguaggio macchina + sistema operativo)
- Una volta tradotto, il programma può essere eseguito più e più volte (alle massime prestazioni possibili per lo specifico calcolatore)

Schema sulla compilazione



Interpreti

- Gli interpreti traducono una istruzione di alto livello alla volta e subito la eseguono
  - la fase di traduzione e di esecuzione si alternano
  - non occorre attendere per iniziare ad eseguire il programma, ma l'esecuzione risulta complessivamente meno efficiente

Paradigmi di programmazione

- I linguaggi di alto livello possono basarsi su diverse logiche di definizione dei programmi, dette paradigmi di programmazione
- Principali paradigmi:
  - imperativo-procedurale (es: FORTRAN, COBOL, C, Pascal, …): il programma è una sequenza di comandi per il calcolatore
  - paradigmi di ispirazione matematica: funzionale (es: LISP) o logica (es: PROLOG)
  - imperativo-a oggetti (es. C++, C#, Delphi, Java, …): il programma permette di definire oggetti virtuali e di farli interagire

Il linguaggio Java

- Java è un linguaggio orientato agli oggetti creato agli inizi degli anni ‘90 dalla Sun Microsystem
  - presentato ufficialmente nel 1995
  - diffusosi rapidamente in ambito industriale e accademico
- Introduce la Java Virtual Machine (JVM):
  - macchina virtuale che simula un calcolatore reale
  - un codice sorgente in Java viene compilato nel codice macchina per la JVM (bytecode)
  - il bytecode non dipende dalla piattaforma;
  - il bytecode viene eseguito da un emulatore della JVM (che effettua la traduzione in linguaggio macchina)