Algoritmi, linguaggi e programmi

Emilio Di Giacomo e Walter Didimo

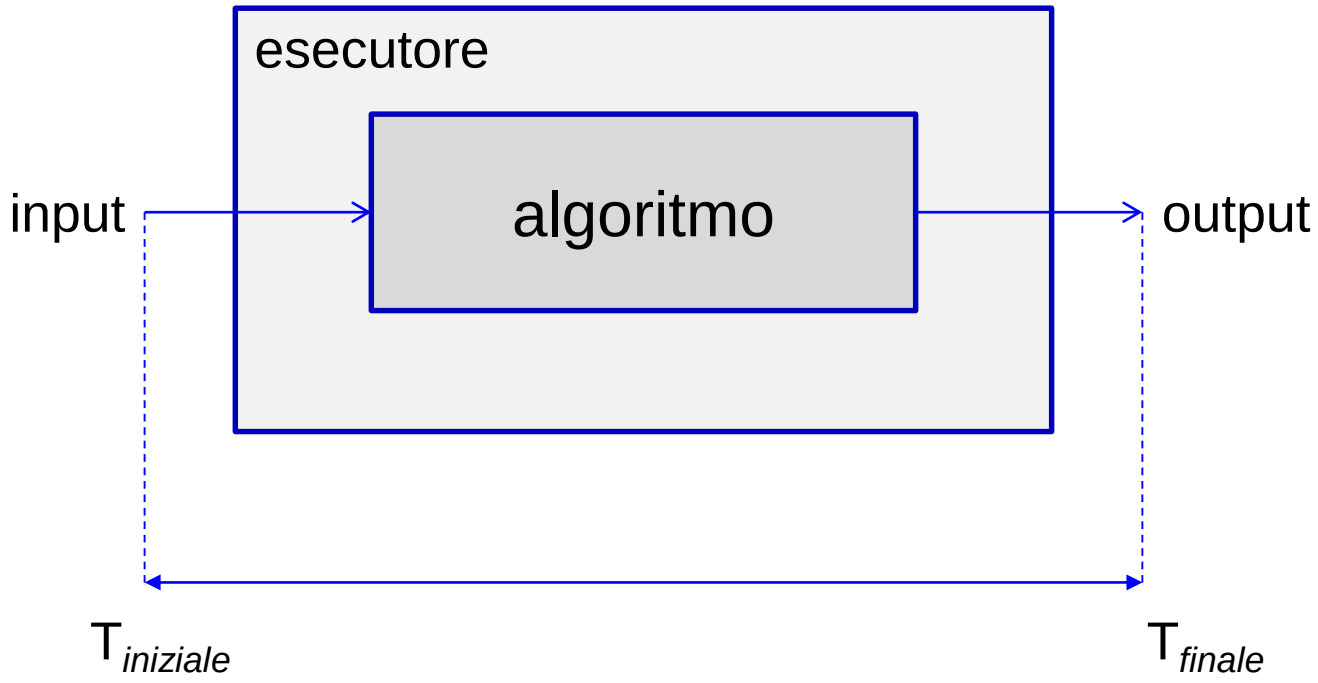
Problemi e algoritmi

- Il calcolatore permette di risolvere in maniera automatica diversi problemi "di calcolo".
  - Es: calcolo della media di *n* numeri
  - calcolo del prodotto di due matrici
  - calcolo del percorso più breve tra due città
  - ....e molti altri
- Per risolvere un problema dobbiamo innanzi tutto individuare un procedimento di soluzione
- Un algoritmo è un procedimento ben definito per risolvere un problema "di calcolo"
  - 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 prende in ingresso alcuni dati iniziali, detti input dell'algoritmo
  - l'algoritmo termina generando in uscita dei dati, detti output dell'algoritmo

Schema di algoritmo



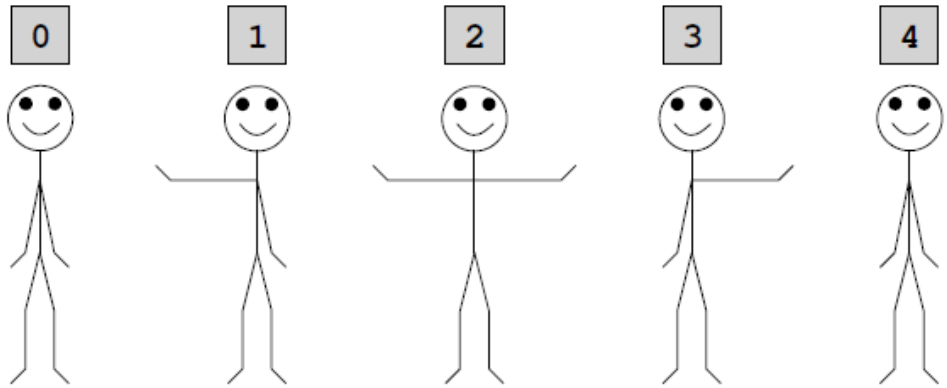
Ancora sugli esecutori

- 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 o in un registro;
  - leggi un numero da una cella di memoria o da un registro;
  - 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.
  - il concetto di cella di memoria è rimpiazzato dal concetto più astratto di variabile, contenitore che può memorizzare un dato di un certo di tipo
  - Il dato memorizzato da una variabile può variare nel tempo, proprio come il contenuto di una cella

Algoritmi: esempio

- Problema: calcolo della media aritmetica di un insieme $S=\{x_1,x_2,\ldots,x_n\}$ di numeri reali

Algoritmo: "Calcola-Media"

input: *insieme di numeri reali* $S = \{x_1, x_2, \ldots, x_n\}$
output: m_s, *media aritmetica dei numeri di S*

1. assegna a m_s il valore x_1
2. assegna a n il valore |S|
3. assegna a i il valore 2
4. se i > n salta all'istruzione 8
5. aggiungi a m_s il valore x_i
6. incrementa i di una unità
7. salta all'istruzione 4
8. dividi m_S per n
9. termina

Pseudo-codice

- Un algoritmo può essere scritto in maniera più compatta utilizzando uno pseudo-codice
- Lo pseudocodice è un formalismo per la descrizione di algoritmi che:
  - presenta dei costrutti strutturati come i linguaggi di programmazione
  - ma permette di semplificare il codice ignorando una serie di dettagli non essenziali
- Vantaggi dello pseudocodice:
  - Sufficientemente strutturato per descrivere gli algoritmi in maniera non ambigua
  - Ignora i dettagli tecnici non rilevanti alla fine della descrizione dell'algoritmo
  - Indipendente da ogni specifico linguaggio di programmazione

Esempio di pseudo-codice

Algoritmo: "Calcola-Media"

input: $S = \{x_1, x_2, \ldots, x_n\} \subset \boldsymbol{R}$ *(sequenza di numeri)*
output: $m_s \in \boldsymbol{R}$ *(media aritmetica dei numeri di S)*

1. $m_s \leftarrow x_1$
2. $n \leftarrow |S|$
3. $i \leftarrow 2$
4. if $(i > n)$ goto 8
5. $m_s \leftarrow m_s + x_i$
6. $i \leftarrow i + 1$
7. goto 4
8. $m_s \leftarrow m_s / n$
9. halt

pseudo-codice

Diagrammi di flusso

- Un altro formalismo a volte utilizzato per descrivere un algoritmo è quello dei diagrammi di flusso (flow chart)
- Un flow chart è un diagramma che descrive graficamente lo sviluppo sequenziale (flusso) di un algoritmo

Flow chart

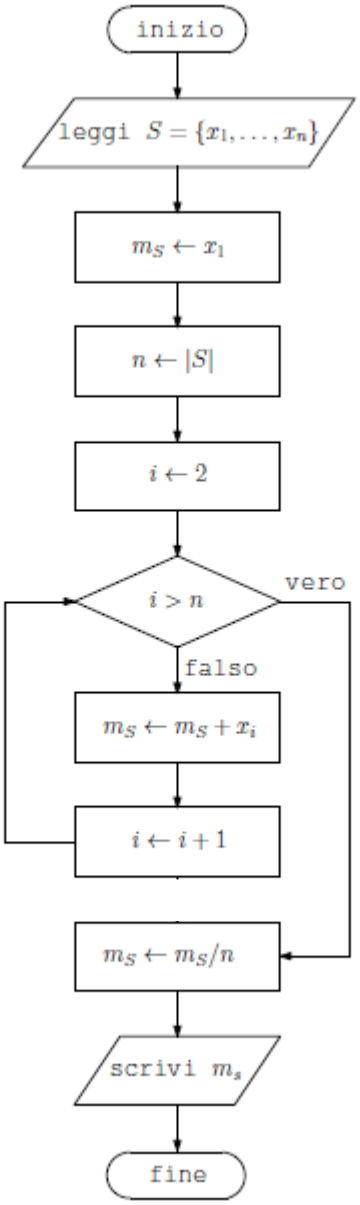
Algoritmo: “Calcola-Media”

input: $S = \{x_1, x_2, \ldots, x_n\} \subset \mathbf{R}$ *(sequenza di numeri)*
output: $m_s \in \mathbf{R}$ *(media aritmetica dei numeri di S)*

1. $m_s \leftarrow x_1$
2. $n \leftarrow |S|$
3. $i \leftarrow 2$
4. if $(i > n)$ goto 8
5. $m_s \leftarrow m_s + x_i$
6. $i \leftarrow i + 1$
7. goto 4
8. $m_s \leftarrow m_s / n$
9. halt

pseudo-codice

flow chart



Descrizioni semplificate

Algoritmo: "Massimo-Tra-Tre-Numeri"

input: a,b,c $\in$ ***R***
output: m = max {a, b, c}

1. m' $\leftarrow$ max {a, b}
2. m $\leftarrow$ max {m', c}
3. halt

non sono istruzioni elementari, ma sappiamo che sono scomponibili in istruzioni elementari

Algoritmo: "Massimo-Tra-Due-Numeri"

input: a,b $\in$ ***R***
output: m = max {a, b}

1. m $\leftarrow$ a
2. if (a > b) goto 4
3. m $\leftarrow$ b
4. halt

Linguaggi e programmi

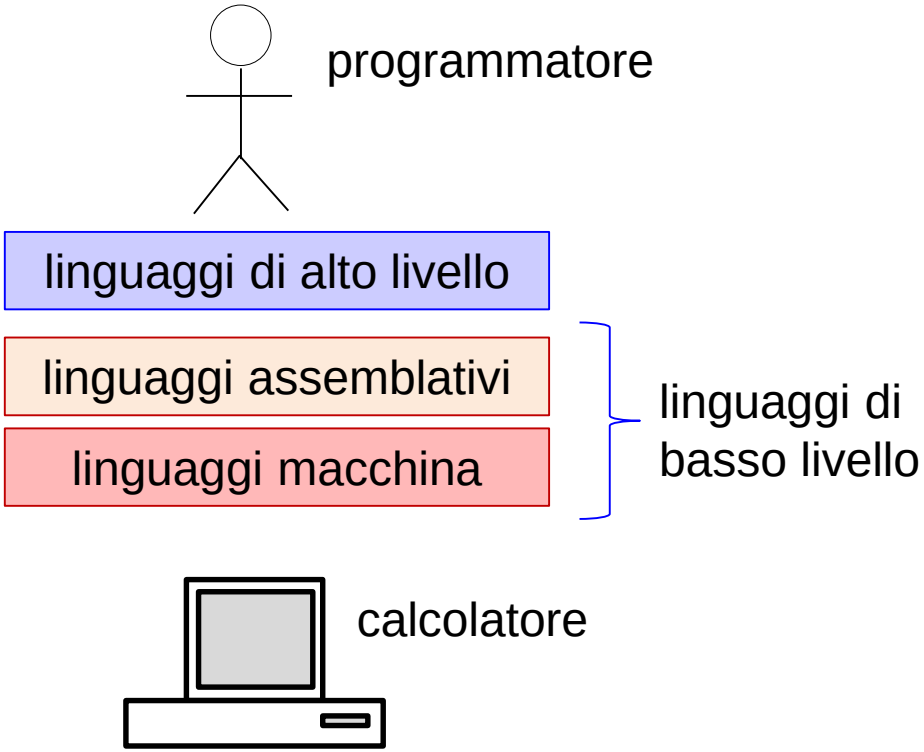
- Per poter essere eseguito da un calcolatore, un algoritmo deve essere trascritto in una forma che sia ad esso comprensibile (direttamente o indirettamente):
  - l'insieme delle regole di trascrizione (cioè il linguaggio utilizzato a tal fine) si chiama linguaggio di programmazione
  - Il risultato 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 che permettono di realizzare programmi direttamente comprensibili da un calcolatore
- Ogni microprocessore dispone di un insieme ristretto di istruzioni macchina molto semplici, eseguibili direttamente in hardware
  - le istruzioni macchina e gli eventuali parametri sono codificati con cifre binarie
  - architetture CISC: relativamente alto numero di istruzioni di elevata complessità (es. intel x86)
  - architetture RISC: insiemi ridotti di istruzioni di estrema semplicità (es. SPARC)

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 realizzato con linguaggio assemblativo deve essere tradotto in linguaggio macchina prima di poter essere eseguito
  - la traduzione avviene per opera di un software chiamato assemblatore (c’è tipicamente un rapporto uno a uno tra una istruzione assemblativa e una istruzione macchina)

Linguaggi assemblativi: esempio

```
Prog: Massimo-Tra-Due-Numeri
-----------------------------
1 LOAD    R1, 1500
2 LOAD    R2, 1501
3 STORE   R1, 2000
4 SUB     R1, R2
5 JGZERO  R1, 7
6 STORE   R2, 2000
7 HALT
-----------------------------
```

- I due numeri da confrontare (input) si trovano inizialmente nelle celle 1500 e 1501; l'output viene scritto nella cella 2000

Linguaggi di alto livello

- I linguaggi di alto livello permettono di scrivere programmi in modo molto più semplice
  - ogni istruzione può corrispondere a molte 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

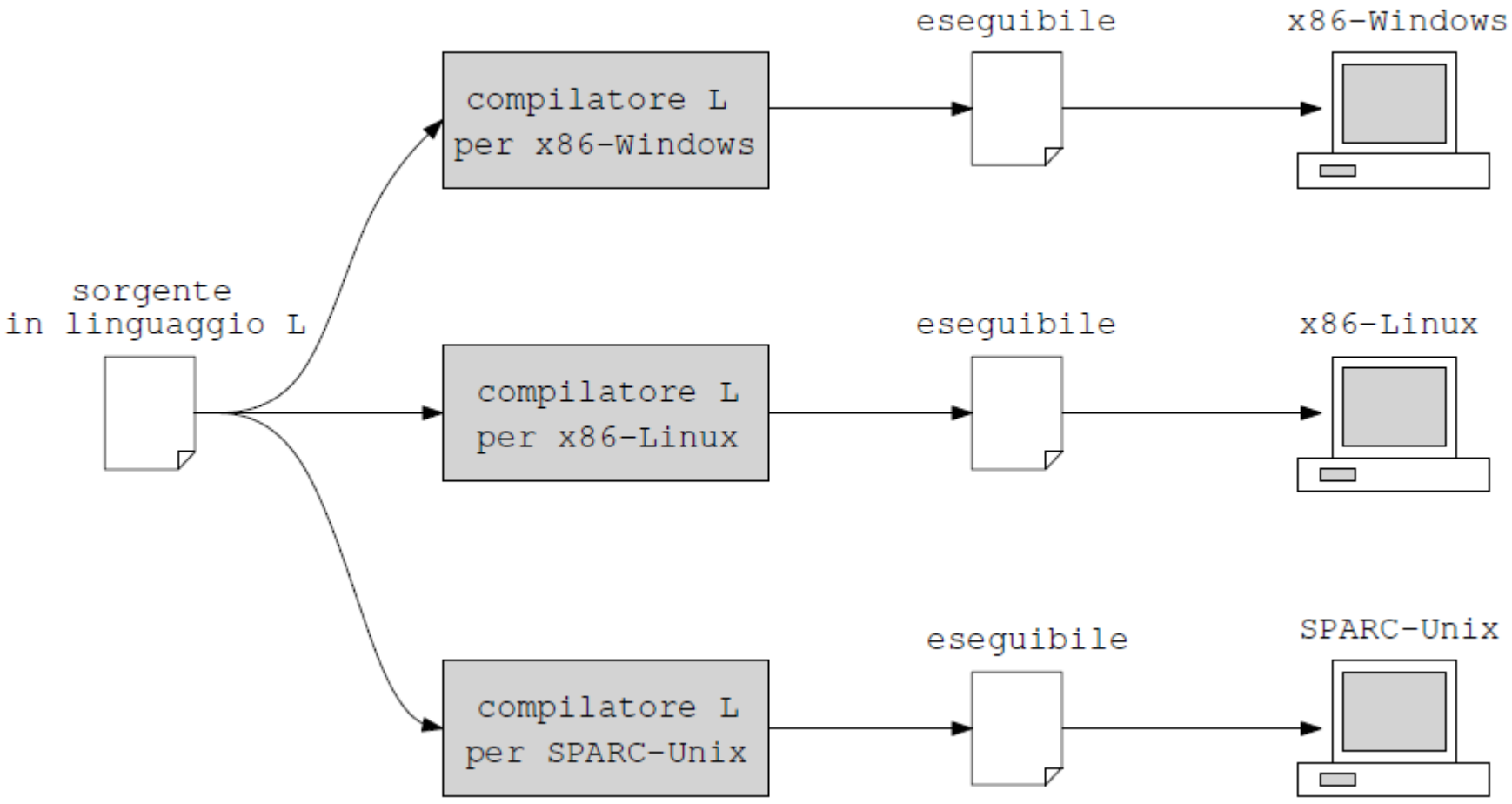
Traduttori per linguaggi di alto livello

- Per poter essere eseguito, un codice sorgente in L deve essere tradotto in un equivalente codice macchina, detto appunto 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 di 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 linguaggio di alto livello possono basarsi su diverse logiche di definizione dei programmi, dette paradigmi di programmazione
- Principali tipi di paradigmi:
  - imperativo (es: FORTAN, COBOL, C, Pascal, …): il programma è una sequenza di comandi per il calcolatore
  - paradigmi di ispirazione matematica: funzionale (es: LISP) o logica (es: PROLOG)
  - a oggetti (es. C++, C#, Delphi, Java, Python, …): il programma permette di definire oggetti virtuali e di farli interagire

Programmazione strutturata

- La programmazione strutturata è una metodologia di programmazione affermatasi a partire dagli anni ’70:
  - flusso di esecuzione più lineare possibile;
  - evitare l’uso di salti (goto)
  - unico punto di ingresso e unico punto di uscita per ogni sotto-procedura che logicamente rappresenti l’implementazione di un qualche algoritmo

Programmazione strutturata

- Tutti gli attuali linguaggi imperativi e ad oggetti supportano la programmazione strutturata, offrendo due tipologie di istruzioni di controllo:
  - istruzioni condizionali: eseguono il test di una condizione, permettendo di scegliere la prossima sequenza di istruzioni da eseguire (tra due possibili), a seconda che la condizione sia vera o falsa;
  - istruzioni iterative: ripetono una certa sequenza di istruzioni per un certo numero di volte (fissato a priori o dipendente dal perdurare o dal decadere di una condizione).

Programmazione strutturata: esempio

Algoritmo: “Massimo-Tra-Due-Numeri”

input: *a, b* $\in$ ***R***
output: *m = max{a,b}*

1. if (a > b) then
2. m $\leftarrow$ a
3. else
4. m $\leftarrow$ b
5. halt

istruzione condizionale

Programmazione strutturata: esempio

Algoritmo: "Calcola-Media"

input: $S = \{x_1, x_2, \ldots, x_n\} \subset \mathbf{R}$ *(sequenza di numeri)*
output: $m_s \in \mathbf{R}$ *(media aritmetica dei numeri di S)*

1. $m_s \leftarrow x_1$
2. $n \leftarrow |S|$
3. $i \leftarrow 2$
4. while $(i \leq n)$ do {
5. $\quad m_s \leftarrow m_s + x_i$
6. $\quad i \leftarrow i + 1$
7. }
8. $m_s \leftarrow m_s / n$
9. halt

istruzione iterativa

Il linguaggio C

- Il C è un linguaggio imperativo sviluppato da Dennis Ritchie pressi i Bell Labs negli anni 70:
  - prima implementazione nel 1972
  - standardizzato nel 1989 dall'ANSI (ANSI C o C standard)
  - standard aggiornato nel 1999 e nel 2011
  - è usato soprattutto per lo sviluppo di sistemi che richiedono prestazioni elevate
  - è alla base di molti altri linguaggi di programmazione: C++, Objective-C, C#, Java, PHP, JavaScript

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)

Schema di traduzione Java