

# Fondamenti di Informatica

Allievi Automatici

A.A. 2015-16

Algoritmi e Programmi

La catena di programmazione

# Algoritmi

- **Informatica: gestione dell'informazione**
  - uso e trasformazione dell'informazione in modo funzionale agli obiettivi
- Le informazioni sono usate e trasformate attraverso **algoritmi**
  - Il concetto di algoritmo è **fondamentale**
- **Algoritmo: specifica di una sequenza finita di passi eseguibili senza ambiguità**
  - affinché la sequenza sia automatizzabile

# Algoritmo

(definizione informale)

Una sequenza **finita** di operazioni **elementari**,  
comprensibili da un **esecutore**,  
che portano alla realizzazione di un **compito**

- Esecutore: chiunque sappia comprendere la specifica delle operazioni
  - tipicamente uno strumento automatico
- Compito: la risoluzione di un problema

# Osservazioni sulla definizione

- Mette in luce gli aspetti **progettuali** e **realizzativi** dell'attività dell'informatico
- Dice che si può svolgere attività informatica ***senza usare un calcolatore elettronico***
  - Esempio: progettare/applicare regole precise per le operazioni aritmetiche su numeri grandi usando solo carta e matita (*chi è il tipico esecutore?*)
  - Il calcolatore elettronico è solo un esecutore **potente** (preciso e veloce), che gestisce quantità di informazioni difficilmente trattabili altrimenti

# Un esempio di algoritmo

(scritto in linguaggio naturale)

## **Ricetta di cucina** (*uovo al tegamino*):

1. Metti un ricciolo di burro in una padella
2. Metti la padella sul fuoco
3. Aspetta due minuti
4. Rompi un uovo     (*è un'istruzione "elementare"??*)
5. Versa il tuorlo e l'albume nella padella
6. Aggiungi un pizzico di sale     (*quanto sale??*)
7. Quando l'albume si è rappreso, togli dal fuoco

# Altri esempi

- Istruzioni di montaggio di un elettrodomestico (*comprensibile ?*)
- Uso di un terminale Bancomat
- Calcolo del massimo comune divisore di due numeri naturali
  - È essenziale che un algoritmo sia ***comprensibile*** al suo esecutore

# Problemi ed esecutori

- Ogni algoritmo risolve **un solo problema**
  - Meglio: una sola *classe* di problemi
- Ogni algoritmo dipende strettamente dall'**esecutore** per cui è formalizzato
  - Operazioni “elementari” per un esecutore possono non esserlo affatto per un altro

# Revisione: definizione di algoritmo

- Dati un **problema** specifico e un **esecutore** specifico, un algoritmo è:
  - una sequenza *finita* di passi *elementari* tale che:
    - i passi sono effettuabili *senza ambiguità* da parte dell'esecutore
    - la successione *risolve* il problema dato
- Nel nostro caso: algoritmi *sequenziali*
  - i passi si eseguono in ordine, uno alla volta



# Risoluzione automatica di problemi

- Le attitudini umane si adattano tipicamente a **individuare metodi** per ottenere le soluzioni
- I calcolatori elettronici, invece, eccellono in:
  - Ripetizione di un grande numero di operazioni di per sé relativamente semplici
  - Capacità di trattare grandi quantità di dati senza errori (*trattare: leggere, scrivere, trasferire*)
  - Rapidità e precisione nell'esecuzione

**Ma non trovano da soli i metodi di soluzione**


# Esempio 1: l'algoritmo del risveglio

1. Alzarsi dal letto
2. Togliersi il pigiama
3. Fare la doccia
4. Vestirsi
5. Fare colazione
6. Prendere il bus per andare a scuola

NB: I passi sono eseguiti in sequenza e l'ordine delle istruzioni è essenziale per la correttezza dell'algoritmo! (2,3 → 3,2 ... !!!)

# Non basta che i passi siano in sequenza

1. Alzarsi dal letto
2. Togliersi il pigiama
3. Fare la doccia
4. Vestirsi
5. Fare colazione
- 6. Se piove allora**  
    prendere ombrello
7. Prendere il bus per andare a scuola



Controllo del flusso  
se ... allora ...

# Altra forma di controllo del flusso

(se ... allora ... altrimenti ...)

1. Alzarsi dal letto
2. Togliersi il pigiama
3. Fare la doccia
4. Vestirsi
5. Fare colazione
- 6. Se piove allora**  
    prendere la macchina  
**altrimenti**  
    prendere il bus

# Ulteriore forma di controllo del flusso (ciclo “fintantoché”)

1. Alzarsi dal letto
2. Togliersi il pigiama
3. Fare la doccia
4. Vestirsi
5. Fare colazione
- 6. Fintantoché piove**  
restare in casa
7. Prendere il bus per andare a scuola

## Esempio 2: gestione biblioteca

- Libri disposti sugli scaffali
- Ogni libro si trova in una precisa, invariabile **posizione** con due coordinate  $\langle \mathbf{s}, \mathbf{p} \rangle$ 
  - *scaffale* e *posizione* nello scaffale
- C'è uno schedario, **ordinato** per autore e titolo
  - Ogni scheda contiene, nell'ordine:
    - cognome e nome dell'autore
    - titolo del libro, editore e data di pubblicazione
    - numero dello scaffale in cui si trova (**s**)
    - numero d'ordine della posizione nello scaffale (**p**)

# Esempio di scheda

Autore/i: Atzeni, Paolo  
Ceri, Stefano  
Paraboschi, Stefano  
Torlone, Riccardo  
Titolo: Database Systems,  
McGraw-Hill, 1999  
Scaffale: 35  
Posizione: 21

# Formulazione dell'algoritmo


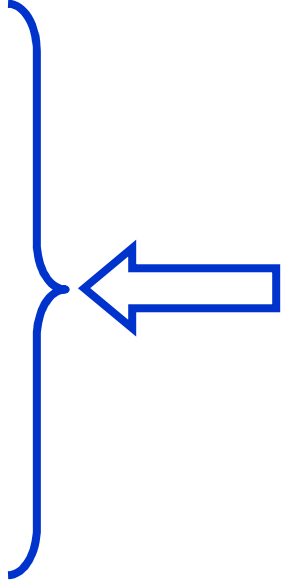
1. Decidi il libro da richiedere
2. Preleva il libro richiesto ←

Se un passo dell'algoritmo non è direttamente comprensibile ed eseguibile dall'esecutore, occorre dettagliarlo a sua volta (mediante un algoritmo!)

Tale procedimento incrementale si dice *top-down* o anche procedimento per raffinamenti successivi (*stepwise refinement*)



# Un algoritmo per il prelievo


- 
1. Decidi il libro da richiedere
  2. Cerca la scheda del libro richiesto
  3. Segnati scaffale e posizione  $\langle \mathbf{s}, \mathbf{p} \rangle$
  4. Cerca lo scaffale  $\mathbf{s}$
  5. Preleva il libro alla posizione  $\mathbf{p}$
  6. Compila la “scheda prestito”
- 

# Il “sotto-algoritmo” di ricerca

1. Prendi la prima scheda
2. Titolo e autore/i sono quelli cercati?
  - 2.1 Se sì,  
la ricerca è termina con successo,  
altrimenti  
prendi la scheda successiva
  - 2.2 Se le schede sono esaurite  
il libro cercato non esiste (in biblioteca)  
altrimenti  
ricomincia dal punto 2.

*Che cosa succede se l'autore cercato è “Manzoni, A.”  
o, peggio, “Zola, E.” ?*

# Un “sotto-algoritmo” migliore

1. Esamina la scheda centrale dello schedario
2.  Se la scheda **centrale** è quella cercata, termina
3. Se non corrisponde, prosegui nello stesso modo nella metà superiore (inferiore) dello schedario se il libro cercato segue (precede) quello indicato sulla scheda

L'algoritmo è incompleto

*Esiste un'altra condizione di terminazione:  
quando il libro non esiste*

## Revisione del passo 2

2. Se la scheda centrale corrisponde al libro  
cercato oppure se la parte di schedario da  
consultare è vuota, termina



Libro trovato



Libro inesistente

# Qualità degli algoritmi

## Criteri di valutazione di un algoritmo:

- **Correttezza**: capacità di pervenire alla soluzione in tutti i casi significativi possibili
- **Efficienza**: proprietà strettamente correlata al tempo di esecuzione e alla memoria occupata

La correttezza è imprescindibile

L'efficienza è auspicabile

# Il problema e la soluzione

- Prima di formulare la soluzione occorre capire esattamente il problema
- **Non serve risolvere il problema sbagliato**
  - In questo corso supporremo che il problema sia ben noto e chiaramente formulato e ci concentreremo sulla progettazione delle soluzioni
  - Spesso, in pratica, è più difficile capire esattamente la natura del problema che non trovare una soluzione!
    - Requirements analysis in Ingegneria del Software

# Dal problema alla soluzione automatica

- **Specifiche dei requisiti:**  
descrizione **precisa** e **corretta** dei requisiti (*verificabilità*)  $\Rightarrow$  **che cosa?**
- **Progetto:** procedimento con cui si individua la soluzione  $\Rightarrow$  **come?**
- **Soluzione:** **un algoritmo**

## Esempio 3: prodotto di interi positivi

- Leggi il numero  $X$  da terminale
- Leggi il numero  $Y$  da terminale
- **Prendi 0 e sommagli  $X$  per  $Y$  volte**
- Scrivi il risultato  $Z$  sul terminale



# Prodotto di due interi positivi

- 1 **Leggi X**
- 2 **Leggi Y**
- 3 **SP = 0**
- 4 **NS = Y**
- 5 **SP = SP + X**
- 6 **NS = NS - 1**
- 7 **NS è uguale a 0 ?**  
    **Se no: torna al passo 5**
- 8 **Z = SP**
- 9 **Scrivi Z**

- Procedimento **sequenziale**
- **Non ambiguo**
- Formulazione **generale**
- Prevede **tutti i casi** ?

**SP e NS sono VARIABILI,**  
**introdotte come ausilio alla**  
**scrittura dell'algoritmo**

**SP: SommaParziale**

**NS: NumeroSomme**

# Sintassi e Semantica

Interpretiamo correttamente le istruzioni 3,4,5,6,8

- **Sintassi** [*come si scrivono: forma e struttura*]

*<variabile> = <espressione>*

- **Semantica** [*come si interpretano: significato*]

- Interpretazione: “calcola il valore dell’espressione e assegna al contenuto della variabile il valore calcolato”

- Si perde il valore precedentemente contenuto nella variabile

**NON** è la semantica delle equazioni !!

- $SP = SP + X$  se e solo se  $X=0$
- $NS = NS - 1$  è una contraddizione  $\forall$  valore di NS
- Sono **ASSEGNAMENTI** di valori
- Le istruzioni di assegnamento **modificano** i valori

1 Leggi X  
2 Leggi Y  
3  $SP = 0$   
4  $NS = Y$   
5  $SP = SP + X$   
6  $NS = NS - 1$   
7 Se NS è diverso da 0, torna a 5  
8  $Z = SP$   
9 Scrivi Z

# Evoluzione dello stato

- Durante la computazione evolve lo **stato** del sistema
  - *stato*: il complesso dei valori contenuti nelle variabili (*informale!*)
- Ad ogni ripetizione delle istruzioni 5-6 l'evoluzione dello stato può essere tale da cambiare l'esito dell'istruzione 7
  - Se questo non accadesse mai?
    - l'algoritmo entrerebbe in un ciclo infinito (loop)
  - Tuttavia in questo caso è impossibile:
    - Si ricevono in ingresso due interi positivi
    - Continuando a decrementare Y inevitabilmente il valore arriva a zero
  - Se si ricevesse in ingresso un valore  $Y \leq 0$  entrerebbe in loop
    - D'altra parte il problema non apparterebbe più alla classe per cui l'algoritmo è stato progettato (prodotto di *interi positivi*)

## Esempio 4: M.C.D. di due naturali positivi

1. Leggi  $N$  ed  $M$
2.  $MIN$  = il minimo tra  $N$  ed  $M$
3. Parti con  $X=1$  ed assegna  $X$  a  $MCDtemp$
4. Fintantoché  $X < MIN$ 
  1.  $X = X + 1$
  2. se  $X$  divide sia  $N$  sia  $M$ , assegna  $X$  a  $MCDtemp$
5. Mostra come risultato  $MCDtemp$

Possiamo fare meglio?

# Trovare algoritmi migliori

- Partire con  $X = \text{MIN}$  e decrementare fino a trovare un divisore (il primo!)
  - alla peggio, ci si arresta senz'altro ad 1
  - il primo divisore trovato è il massimo
- Algoritmo di Euclide
  - se  $N$  è uguale a  $M$ , allora il risultato è  $N$
  - altrimenti il risultato sarà il massimo comune divisore tra il più piccolo dei due e la differenza tra il più grande e il più piccolo [*ne ripareremo*]

# Linguaggi per esprimere gli algoritmi

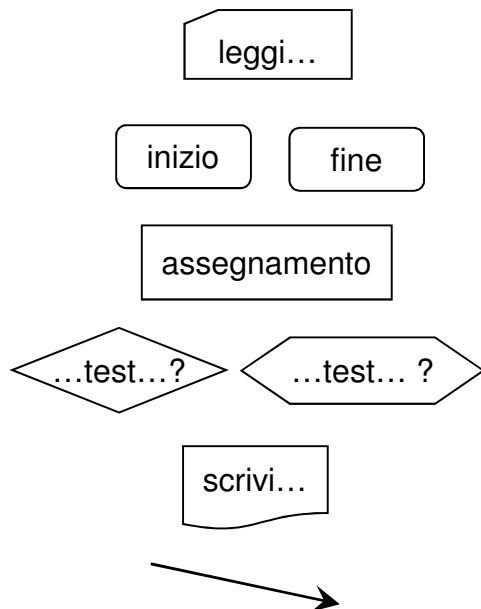
- Semi-formali
  - specifiche iniziali, ancora intelligibili solo all'essere umano
- Formali
  - programmi da eseguire, intelligibili anche alla macchina

**linguaggi di programmazione**

# Linguaggi semi-formali

(per la specifica iniziale)

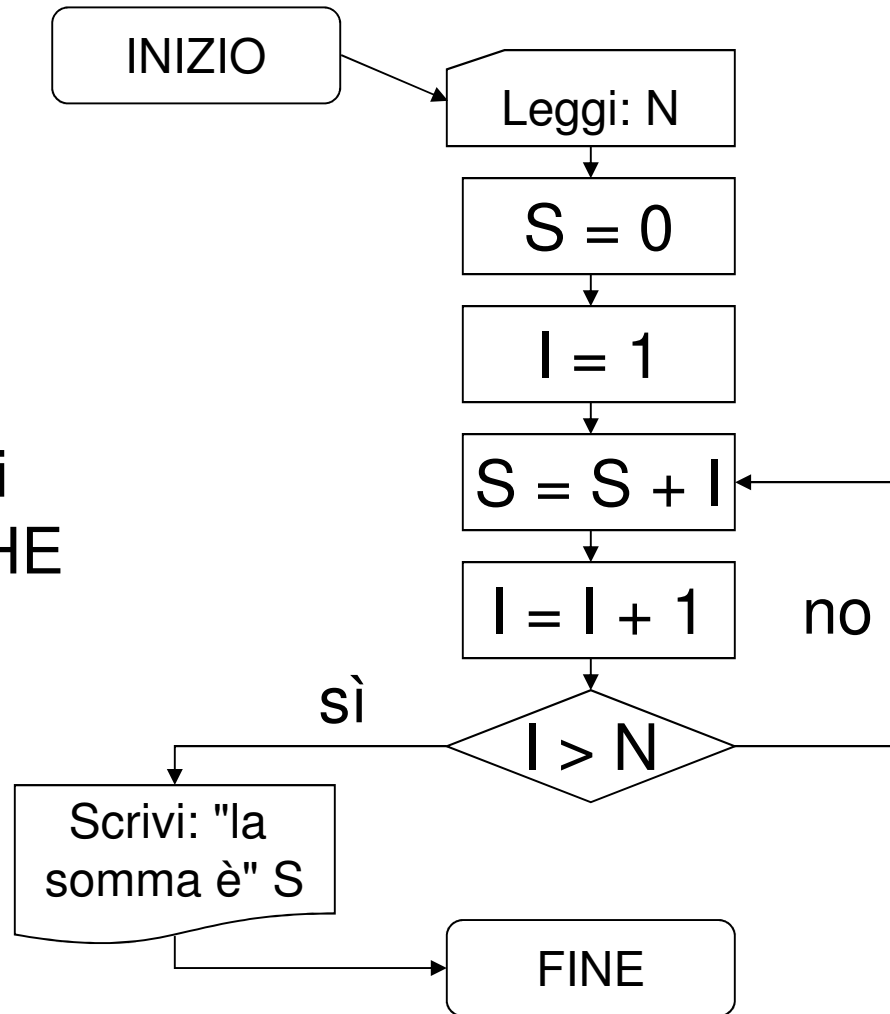
- **Pseudo-codice:** *se*  $A > 0$  *allora*  $A = A + 1$  *altrimenti*  $A = 0$
- **Diagrammi di flusso** (flow chart / schemi a blocchi)



- Blocco di input dati
- Blocchi di inizio/fine dell'esecuzione
- Blocco esecutivo
- Blocco condizionale
- Blocco di output dati
- Flusso di controllo delle operazioni

# Somma dei primi N numeri naturali

I simboli S, I e N sono definiti  
come **VARIABILI NUMERICHE**  
(di tipo intero)





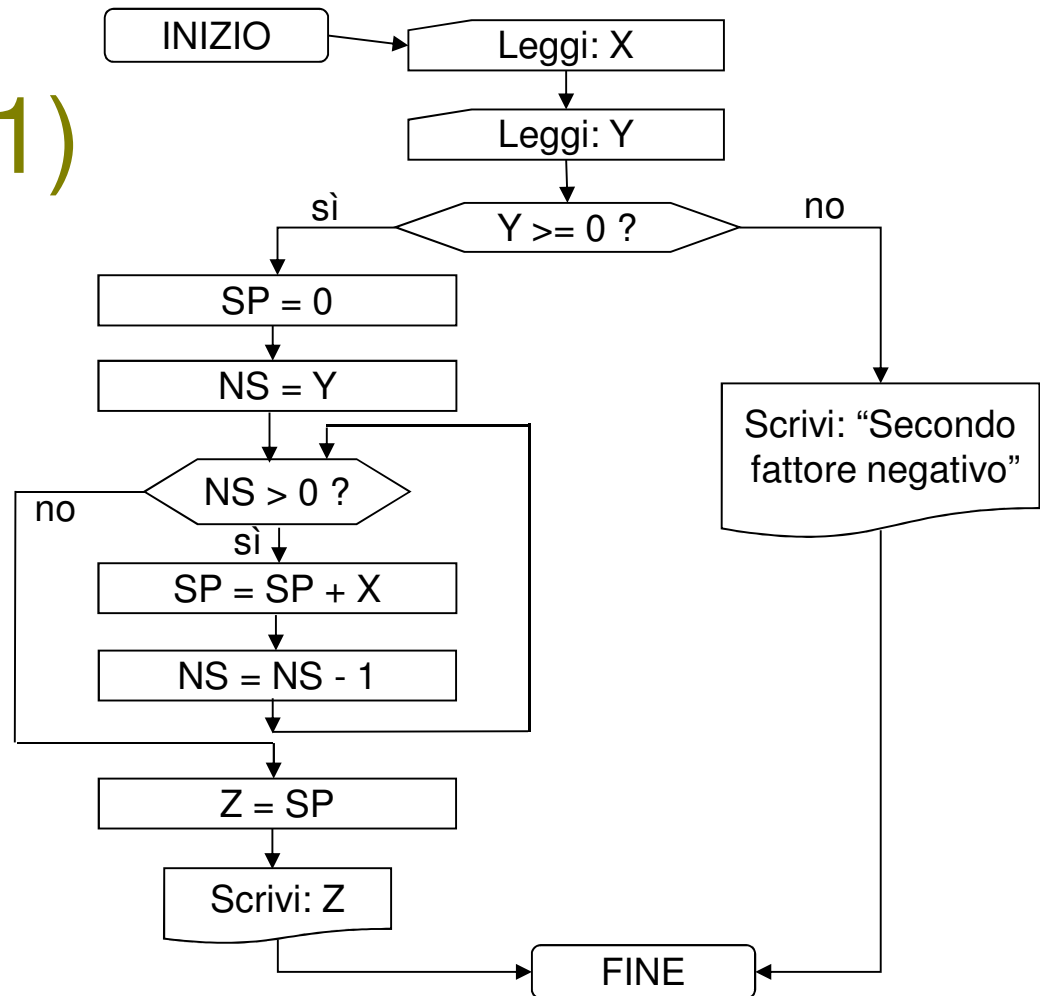
# Prodotto per somme ripetute (1)

**Ipotesi:** l'algoritmo non calcola il prodotto nei casi in cui  $Y < 0$

**Legenda:**

**NS:** numero somme

**SP:** somma parziale



# Prodotto per somme ripetute (2)

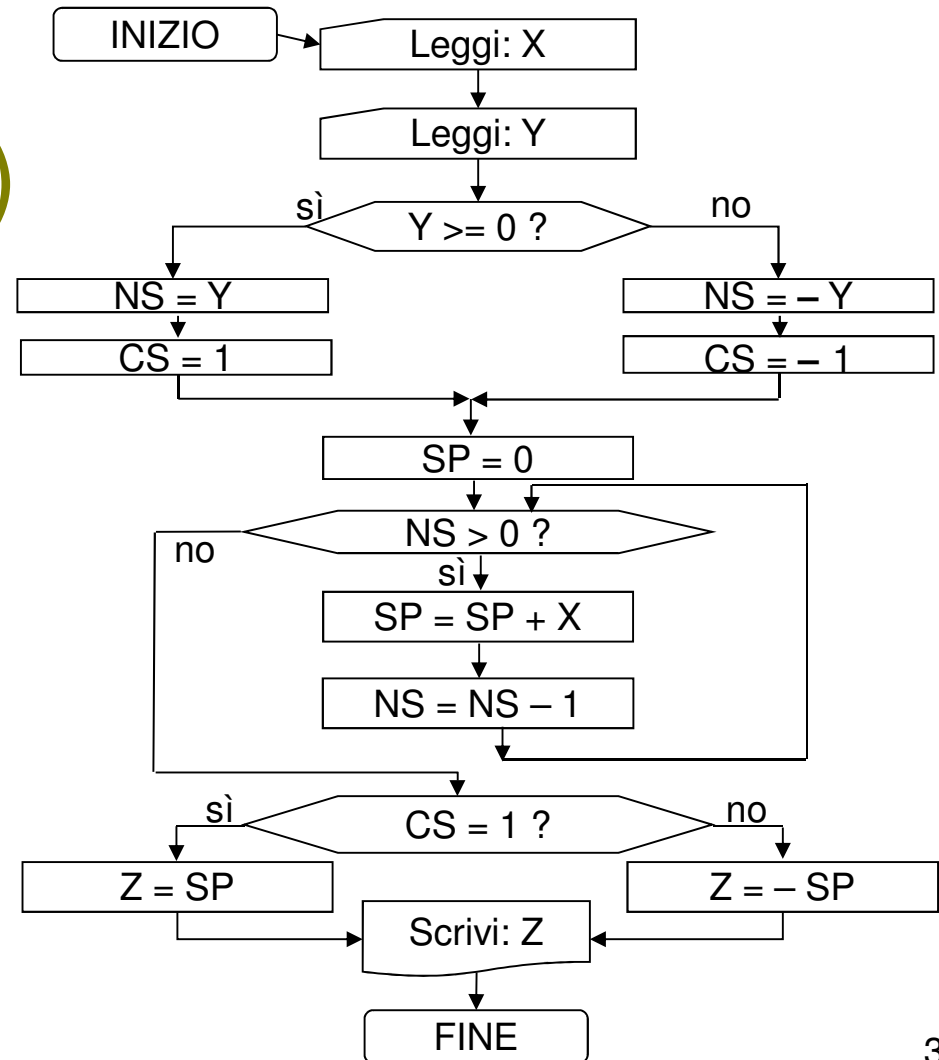
L'algoritmo calcola il prodotto in ogni caso, anche con fattori negativi

## Legenda:

**NS:** numero somme

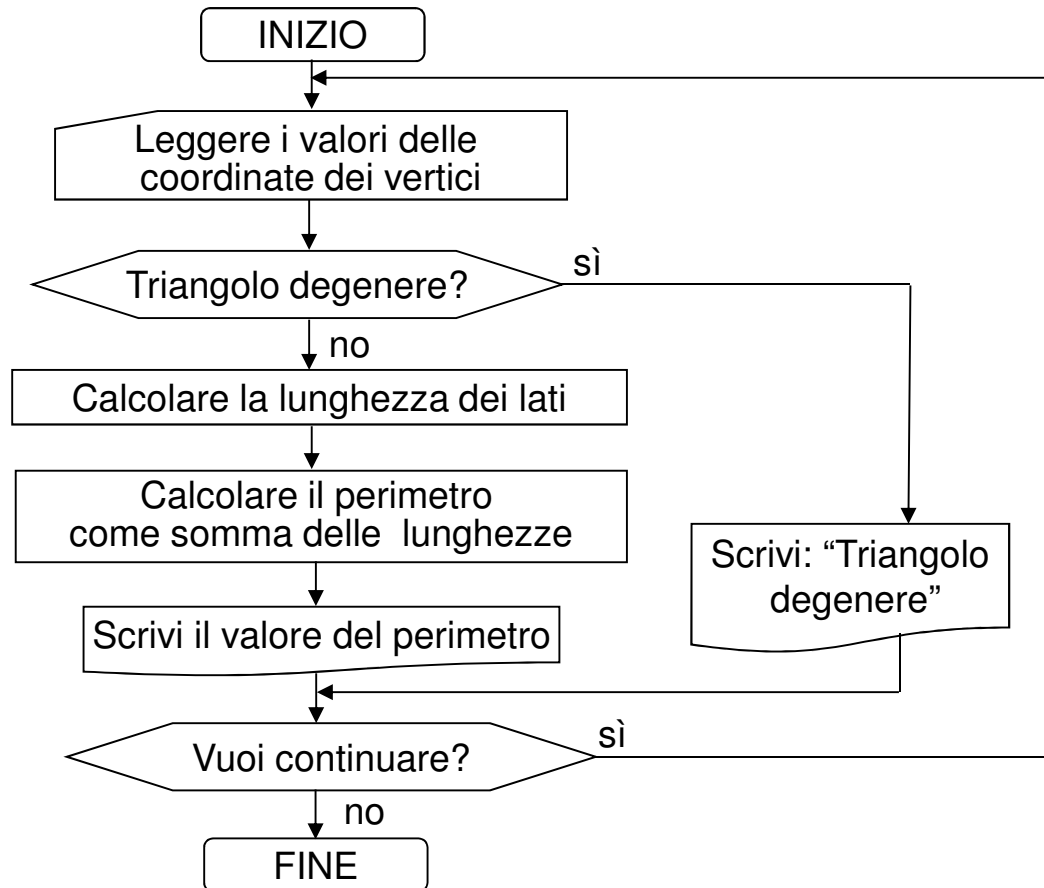
**SP:** somma parziale

**CS:** controllo segno

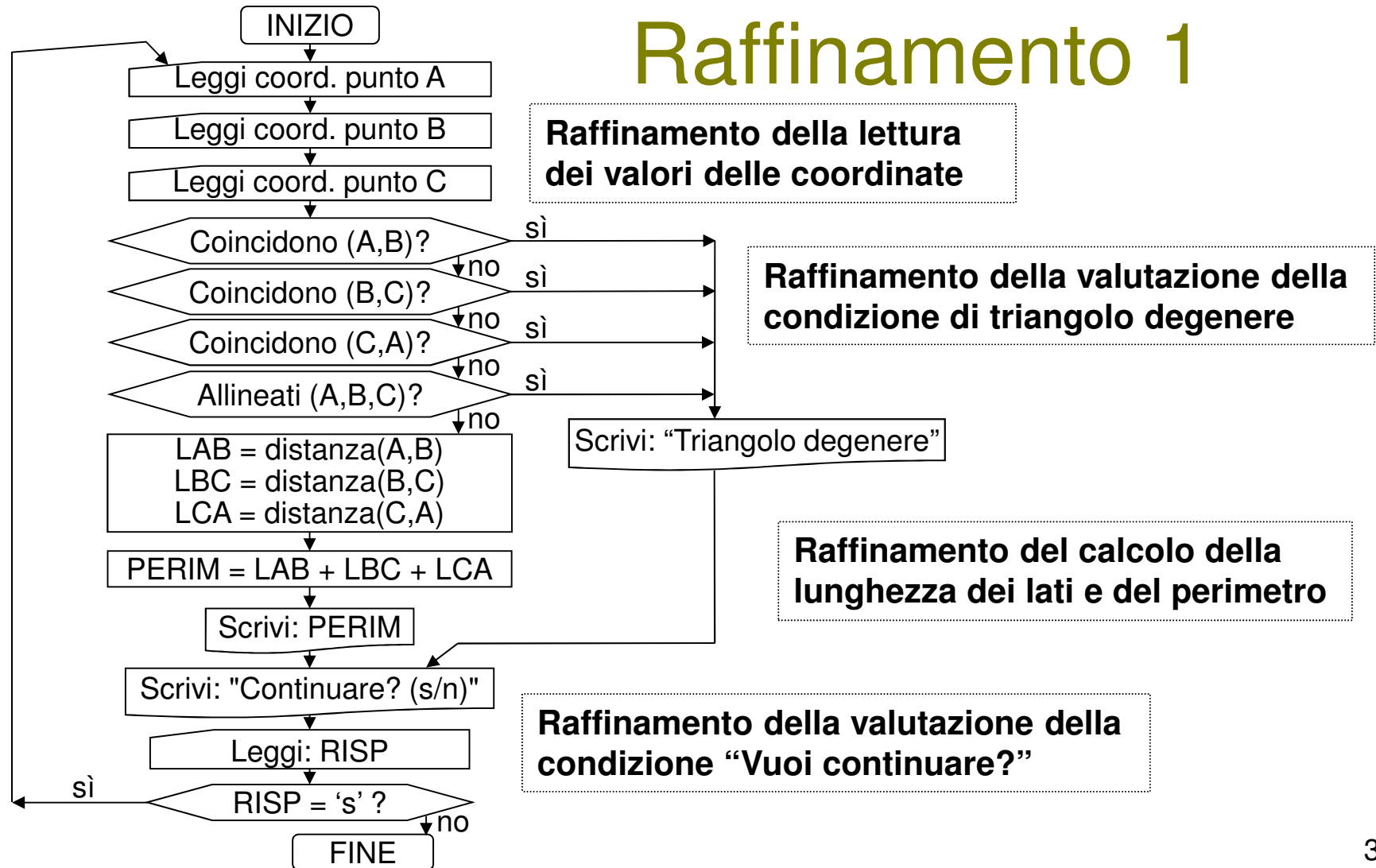


# Triangoli non degeneri e perimetro

**Problema:** date le coordinate di tre punti, riconoscere se sono i vertici di un triangolo non degeneri, e nel caso calcolarne il perimetro



# Raffinamento 1



# Concetto di sottoprogramma

- **Operazioni elementari:** direttamente eseguibili dall'esecutore
- **Direttive complesse:** devono essere raffinate ed espresse in termini di operazioni elementari
- **Raffinamento** di direttive complesse: realizzabile a parte rispetto all'algoritmo principale
- Le direttive complesse possono essere considerate come **sottoproblemi** da risolvere con un algoritmo dedicato
- **Sottoprogrammi:** codifiche di questi algoritmi "accessori"
- Direttive complesse: si possono considerare "**invocazioni**" dei sottoprogrammi all'interno dei programmi principali

# Vantaggi nell'impiego dei sottoprogrammi

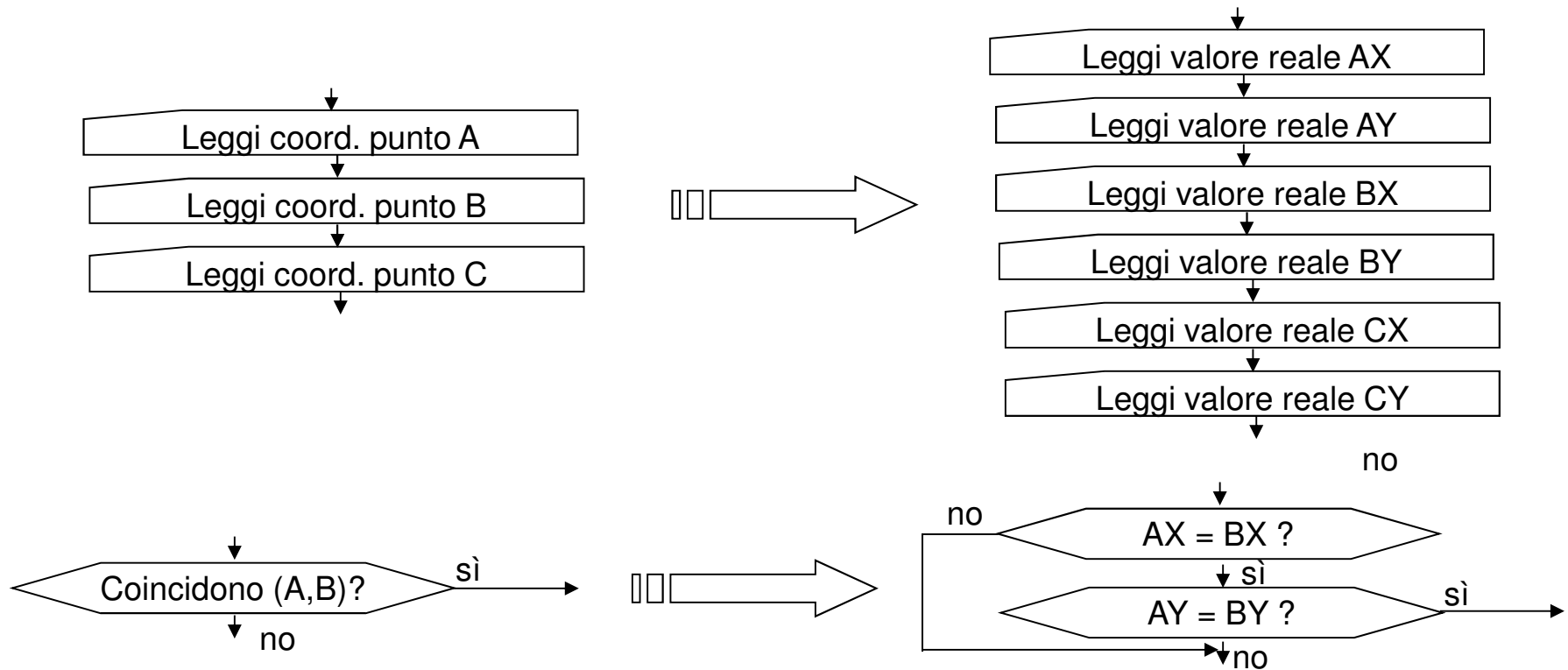
- Chiarezza del programma principale
  - molti dettagli sono descritti (e nascosti) nei sottoprogrammi
  - il programma principale descrive la struttura di controllo generale
- Si evitano ripetizioni
  - alcuni sottoproblemi devono essere affrontati più volte nella soluzione di un problema principale
  - il sottoprogramma può essere richiamato tutte le volte che sia necessario

# Vantaggi nell'impiego dei sottoprogrammi

- Disponibilità di "sottoprogrammi" prefabbricati
  - sottoproblemi ricorrenti già sviluppati da programmatori esperti, raccolti nelle cosiddette "librerie" di sottoprogrammi
  - si potranno riutilizzare anche in **altri** programmi
- La manutenzione è più semplice ed efficace
  - Si modifica una volta sola, in un punto, e l'effetto si propaga a tutti i programmi che fanno riferimento al sottoprogramma

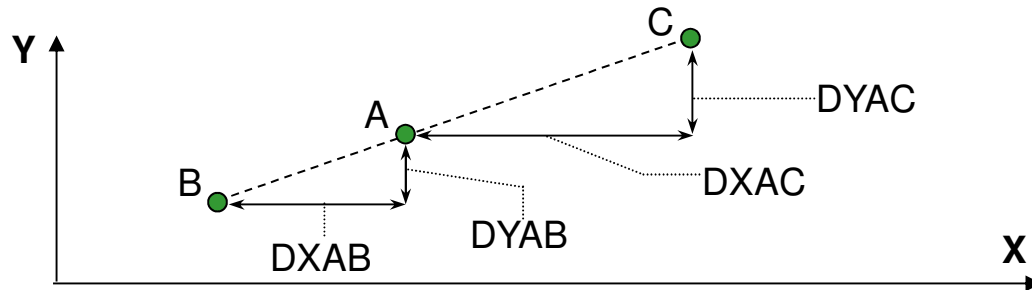
# Raffinamento 2

- Espansione delle direttive complesse

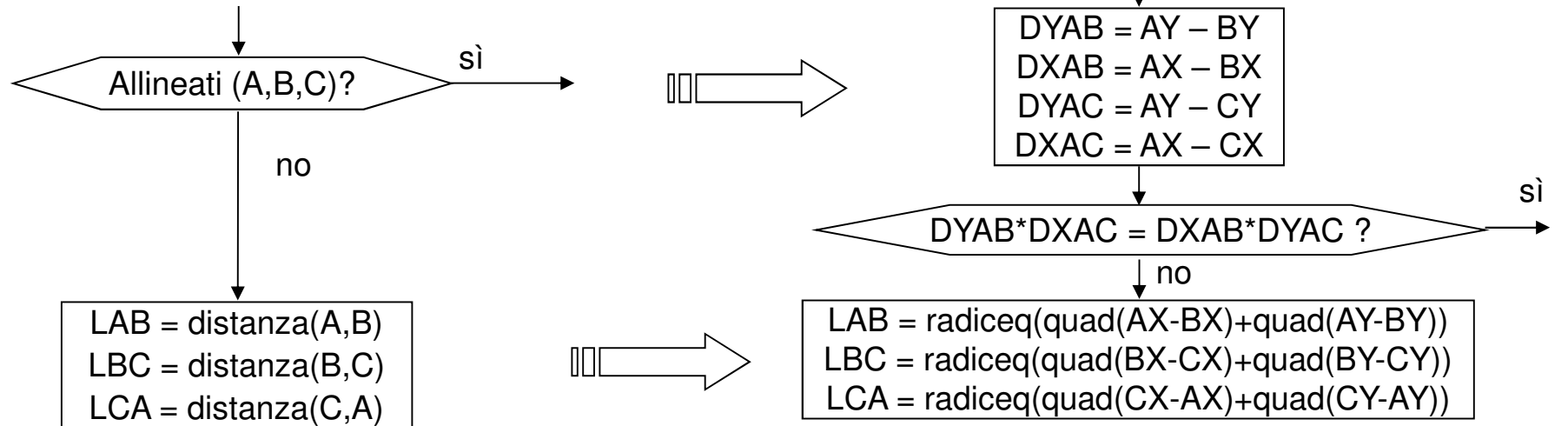




# Raffinamento 2 (continua)



Se A, B, C sono allineati, vale la proporzione  $DYAB : DXAB = DYAC : DXAC$



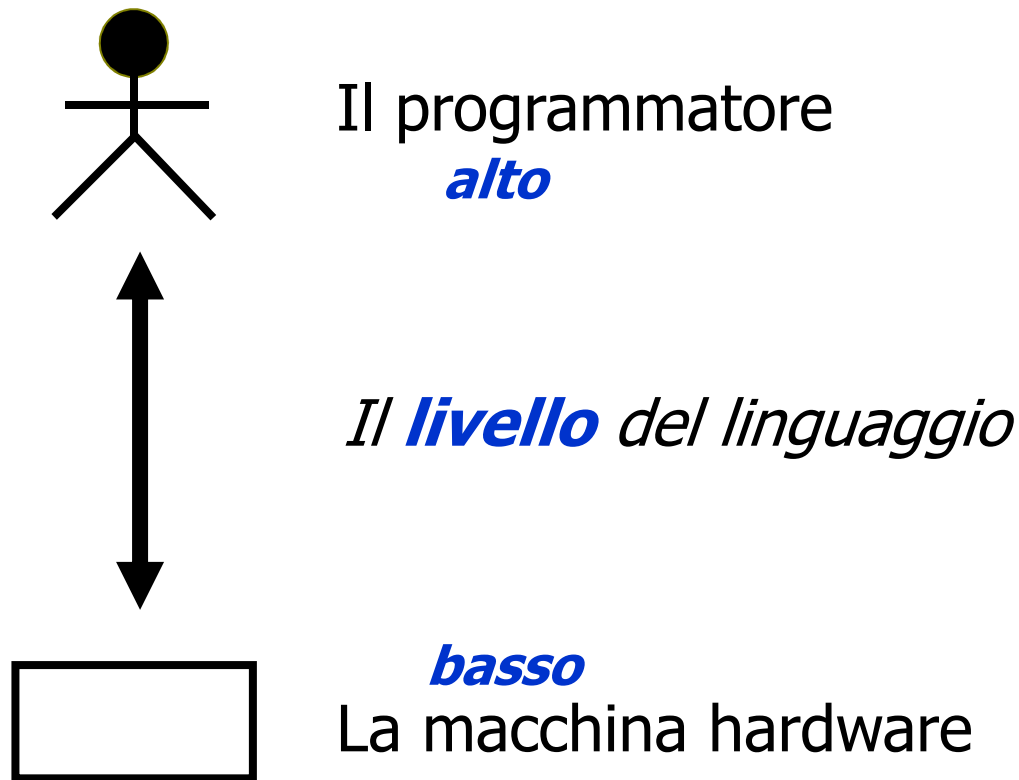
$quad(N)$  indica  $N*N$      $radiceq(N)$  indica  $\sqrt{N}$

# Linguaggi di programmazione

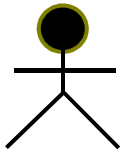
(formali, per la codifica)

- Consentono di scrivere gli algoritmi sotto forma di programmi eseguibili dal calcolatore
  - **Codificare** gli algoritmi
- Sono suddivisi in:
  - Linguaggi di alto livello
    - linguisticamente più vicini al linguaggio naturale
  - Linguaggi assembler
    - più vicini al codice macchina

# Il concetto di “livello” dei linguaggi di programmazione



# Esempi



- Linguaggio C
- Linguaggio assembler
- Linguaggio macchina



TOT=PAGA+STRAORD;

LOAD PAGA  
ADD STRAORD  
STORE TOT

0100001111  
1100111001  
0110001111

# La “Babele” dei linguaggi

– Problemi di comunicazione e compatibilità

+ Opportunità di specializzazione

- Inizialmente si usava direttamente il linguaggio della macchina
- Nella seconda metà degli anni '50, il linguaggio si alza di livello
  - Si usano programmi che traducono (programmi scritti in) i linguaggi di più alto livello nel linguaggio della macchina
  - Opportunità: traduzioni diverse dello **stesso programma** “alto” verso i linguaggi “bassi” di macchine diverse

# Componenti di un linguaggio

- ***Vocabolario***: parole chiave del linguaggio
  - riconosciute dal **parser (analizzatore lessicale)**
- ***Sintassi***: regole per comporre i simboli del vocabolario (le parole chiave)
  - Il controllo della sintassi avviene tramite l'**analizzatore sintattico**
- ***Semantica***: significato delle espressioni
  - Il controllo della semantica è il più difficile
  - Un errore semantico si può rilevare, in generale, solo a tempo di esecuzione

# Alcuni linguaggi

- I primi e tradizionali linguaggi
  - Fortran, Cobol
- Linguaggi più moderni
  - C, C++, ... Java, C# .... Python ...
- Linguaggi speciali
  - SQL, per interrogazione di database, ...
- Linguaggi che non “mimano” l’architettura della macchina
  - LISP, PROLOG

# Compilatori e Interpreti

- I ***compilatori*** sono programmi che traducono i programmi di alto livello in **codice macchina**
- Gli ***interpreti***, invece, ne interpretano direttamente le operazioni, **eseguendole**

Esempi di linguaggi *interpretati*

- LISP, PROLOG (usati nell'intelligenza artificiale)
- BASIC, PYTHON

Esempi di linguaggi *compilati*

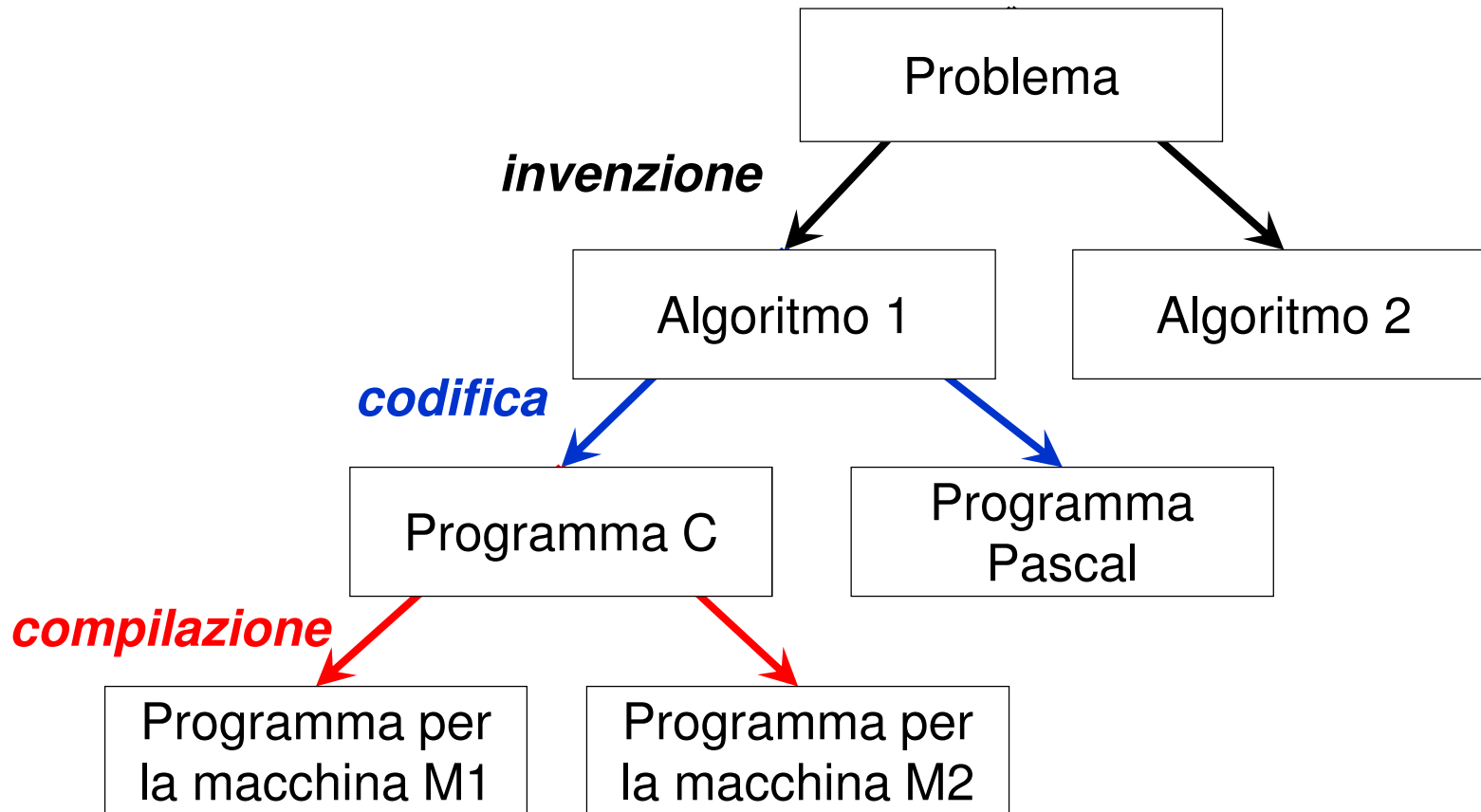
- COBOL, **C**, C++, PASCAL, FORTRAN



# Problemi, Algoritmi, Programmi

- Compito dell'informatico è inventare (*creare*) **algoritmi** ...
  - cioè escogitare e formalizzare le sequenze di passi che risolvono un problema
- ... e codificarli in **programmi**
  - cioè renderli comprensibili al calcolatore

# Problemi, Algoritmi, Programmi



# La catena di programmazione

(nel caso dei linguaggi *compilati*)

- Si parte dalla codifica di un algoritmo
  - in un linguaggio simbolico
    - di basso livello (Assembler)
    - o di alto livello (C, Fortran, ...)

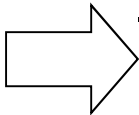
detta **programma sorgente**

- Si genera un programma scritto in codice macchina, chiamato **programma eseguibile**

# 1. Videoscrittura (editing)

- Il testo del programma sorgente, costituito da una sequenza di caratteri, viene **composto e modificato** usando uno specifico programma: l'***editor***
- Così otteniamo un **File Programma Sorgente** memorizzato in un file di testo di nome:
  - **XXX.asm** per programmi in assembler
  - **XXX.c** per programmi in C
  - **XXX.cpp** per programmi in C++
  - ...

## 2. Traduzione

- Linguaggio di alto livello  $\Rightarrow$  Linguaggio macchina (***compilatore***)
- Durante questa fase si riconoscono i simboli, le parole e i costrutti del linguaggio:
  -  – eventuali messaggi diagnostici segnalano errori lessicali e sintattici nel programma sorgente
    - Esempio: manca il ; alla fine di un'istruzione C
- Si genera la forma binaria del codice macchina corrispondente: il **File Programma Sorgente** è tradotto in un **File Programma Oggetto**, cioè in un file **binario** di nome **XXX.obj**

### 3. Collegamento (linking)

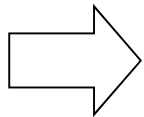
- Il programma *collegatore* (**linker**) deve collegare fra loro il file oggetto e i sottoprogrammi richiesti (es. le funzioni di C)
- I sottoprogrammi sono estratti dalle *librerie* oppure sono individuati tra quelli definiti dal programmatore (nel qual caso si trovano anch'essi nel file oggetto)
- Si rendono globalmente coerenti i riferimenti agli indirizzi dei vari elementi collegati
- Si genera un **File Programma Eseguibile**, un file binario che contiene il codice macchina del programma eseguibile completo, di nome **XXX.exe**
- Messaggi diagnostici possono rilevare errori nel citare i nomi delle funzioni da collegare (altro tipo di errore sintattico)
- Il programma sarà effettivamente eseguibile solo dopo che il contenuto del file **sarà stato caricato nella memoria di lavoro (centrale)** del calcolatore (a cura del Sistema Operativo)

## 4. Caricamento (loading)

- Il caricatore (***loader***) individua una porzione libera della memoria di lavoro e vi copia il contenuto del file **XXX.exe**
  - Eventuali messaggi rivolti all'utente possono segnalare che non c'è abbastanza spazio in memoria
    - errore "di sistema", dovuto a insufficienza di risorse di calcolo

## 5. Esecuzione

- Per eseguire il programma occorre fornire in ingresso i dati richiesti e in uscita riceveremo i risultati (su video o file o stampante)
- Durante l'esecuzione possono verificarsi degli errori (detti “errori di run-time”), quali:
  - calcoli con risultati scorretti (per esempio un overflow)
  - calcoli impossibili (divisioni per zero, logaritmo di un numero negativo, radice quadrata di un numero negativo,....)
  - errori nella concezione dell'algoritmo (l'algoritmo non risolve il problema dato)



Quelli citati qui sopra sono tre esempi di *errori semantici*



# Nel caso del C le fasi sono sei

## 1. Videoscrittura

- svolta dal programmatore tramite un *editor*

## 2. *Pre-compilazione (pre-processing)*

- svolta da un programma detto *preprocessore*

## 3. Traduzione (compilazione)

- svolta dal *compilatore (compiler)*

## 4. Collegamento (linking)

- svolto dal *collegatore (linker)*

## 5. Caricamento (loading)

- svolto dal *caricatore (loader)*

## 6. Esecuzione

- a cura del Sistema Operativo