




# **La programmazione**

Francesco Isgrò

# Programmazione


- La programmazione è quell'attività che porta a sviluppare un software
- Un procedimento può essere implementato in un software se:
  - Può essere descritto in modo non ambiguo
  - Raggiunge l'obiettivo in un tempo finito
  - E' deterministico
- Il procedimento viene chiamato algoritmo
  - Ricetta di cucina
  - Risolvere equazioni di 2° grado

- 
- Il programmatore descrive al calcolatore come risolvere un problema
  - Il microprocessore del calcolatore mette a disposizione molte operazioni di base: somma, sottrazione, AND, spostamenti di valori, letture e scritture da e verso periferici, ecc.
  - Abbiamo visto che un linguaggio di alto livello permette una maggiore astrazione



# Linguaggi di alto livello (HLL)

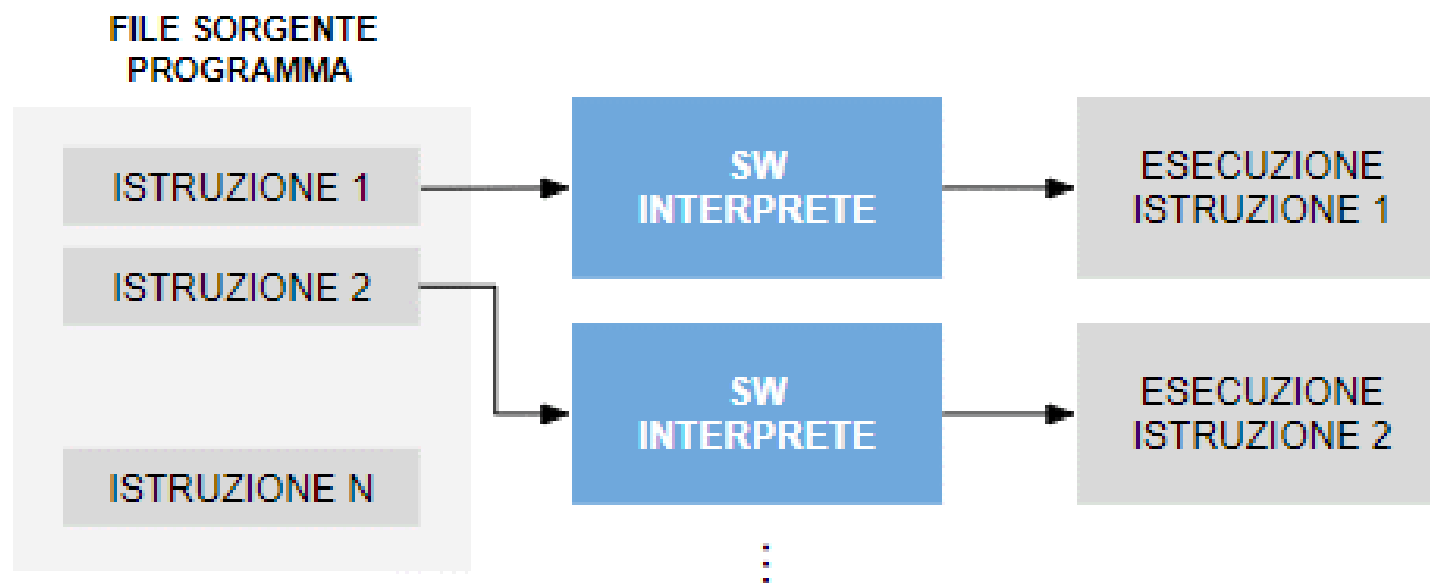
- alto livello di astrazione: le istruzioni hanno un significato più complesso e non serve al programmatore sapere come vengano realizzate dal processore (A + B somma due valori)
- tradotto in linguaggio macchina o in assembly da un traduttore; indipendente dal processore e dal sistema operativo
- è più simile al linguaggio umano e quindi più facile da ricordare (es. “print X” visualizza il valore di X)



Indipendentemente dalla soluzione adottata, il programmatore scrive in un file le istruzioni, queste costituiscono il codice sorgente o programma sorgente detto anche solo programma

# Interprete

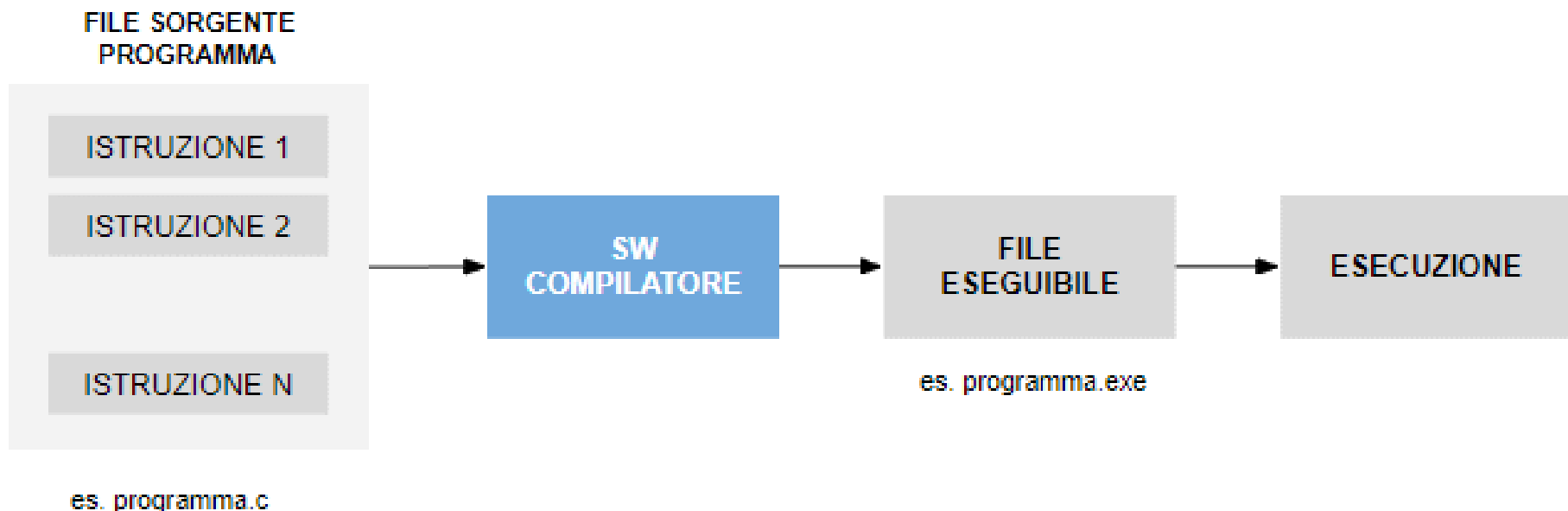
Le istruzioni del codice sorgente vengono ad una ad una tradotte in linguaggio macchina e subito eseguite dalla CPU



es. programma.py

# Compilatore

Tutto il codice sorgente viene tradotto in linguaggio macchina e memorizzato in un file detto programma (file/codice) eseguibile







# Differenze

- Velocità di esecuzione
  - Ogni volta che l'interprete esegue un programma, deve attuare la traduzione delle istruzioni in linguaggio macchina: lento
  - Il programma compilato è già tradotto e ha quindi una velocità di esecuzione superiore
  - Il compilatore è normalmente in grado di ottimizzare il codice per produrre o un programma più veloce o un programma più piccolo



- 
- Competenze necessarie
    - Eseguire un programma interpretato richiede che l'utente finale sia in possesso del programma interprete e impari a utilizzarlo (si può rendere trasparente)
    - Eseguire un programma compilato richiede solo una semplice interazione, ad esempio un doppio-click sulla sua icona
    - Solo il programmatore deve disporre del compilatore e solo lui deve avere la competenza necessaria per utilizzarlo

- 
- Copyright e gestione della complessità
    - L'interprete richiede di disporre del codice sorgente del programma che quindi risulta visibile e modificabile da chiunque
    - Il programma eseguibile non necessita del sorgente: protezione del copyright (reverse engineering possibile, ma complessa)



# Bytecode

- Alcuni linguaggi (es. Java, Python) compilano il sorgente producendo un file intermedio non eseguibile, ma in un assembly generico detto bytecode , di veloce traduzione in un codice macchina reale
- I calcolatori su cui deve essere eseguito il programma possono avere linguaggi macchina differenti, viene quindi installato un interprete (es. la Java Virtual Machine) che traduce il bytecode nel codice nativo di quel calcolatore

# Librerie

- In un HLL il programmatore non ha necessità di programmare le funzioni di base
- Queste operazioni sono state programmate e compilate dal produttore del traduttore e sono a disposizione del programmatore sotto forma di funzioni (es.  $\tan(x)$  calcola la tangente)
- I codici in linguaggio macchina vengono raggruppati in file detti librerie (collezioni di funzioni)



# Creazione eseguibile

- Il processo (build) consiste di 2 fasi
  - Compilazione:
    - il sorgente viene compilato in codice macchina, le “funzioni di libreria” sono ancora mancanti
      - viene generato un file intermedio detto file oggetto



## – Linking

- il file oggetto e le librerie (che sono già in codice macchina) vengono unite (collegate – “link”) così da aggiungere al file oggetto le parti mancanti e costituire un unico file eseguibile
- La fase di link crea un eseguibile collegando insieme uno o più file oggetto e una o più librerie

File sorgente .cpp  
contenente il main

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
```

File sorgente .cpp  
contenente funzioni

```
double potenza (double base, int esponente)
{
    double ris;
```

eccetera eccetera...

COMPILATORE

COMPILATORE

```
0000111000
0001110011
1100101000
0010010001
```

file binario oggetto .o  
non eseguibile

```
0000111000
0001110011
1100101000
0010010001
```

file binario oggetto .o  
non eseguibile

```
0000111000
0001110011
1100101000
0010010001
```

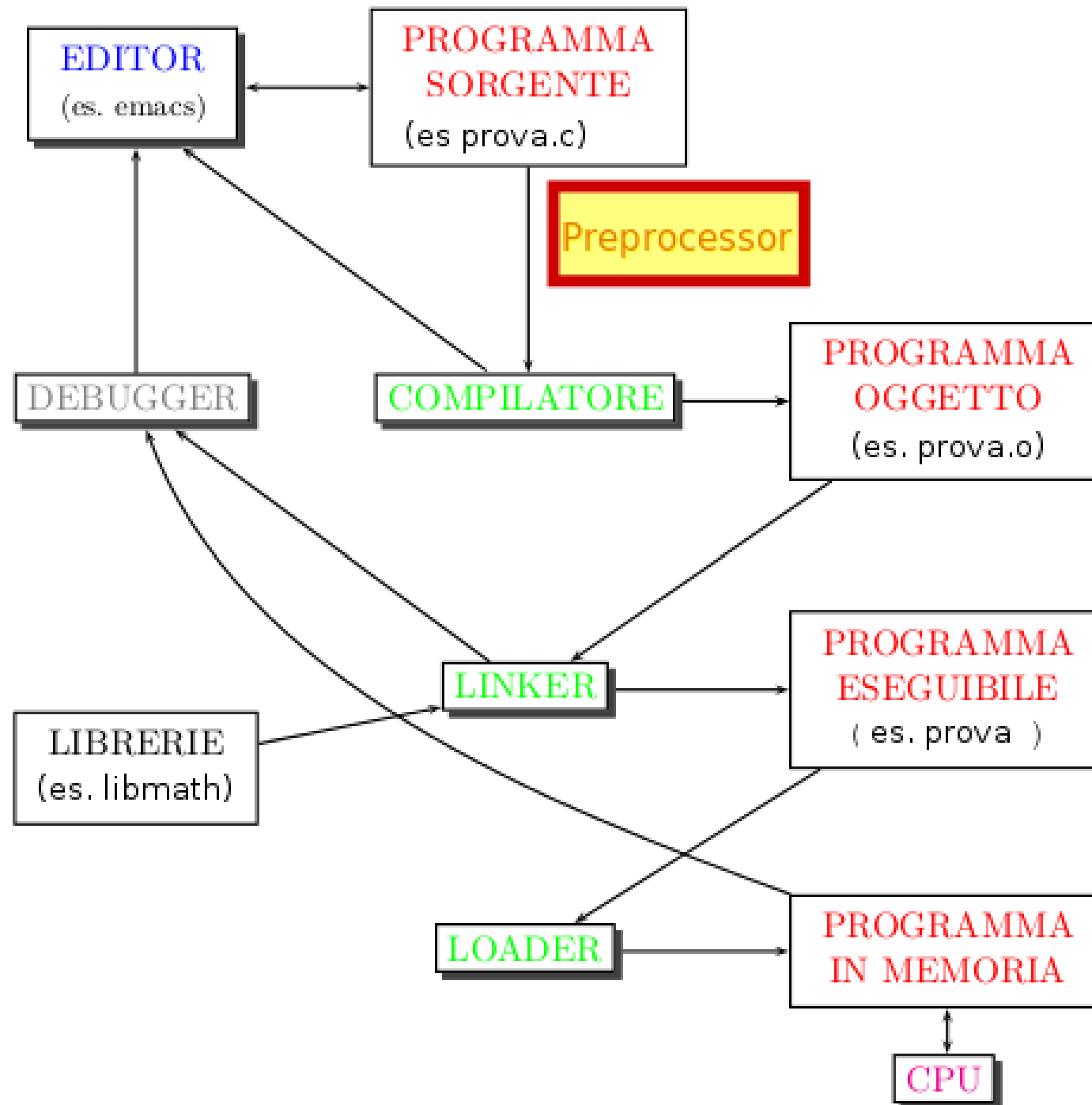
file binario oggetto .o  
(precompilato)  
o library di funzioni

LINKER



```
0000111000
0001110011
1100101000
0010010001
```

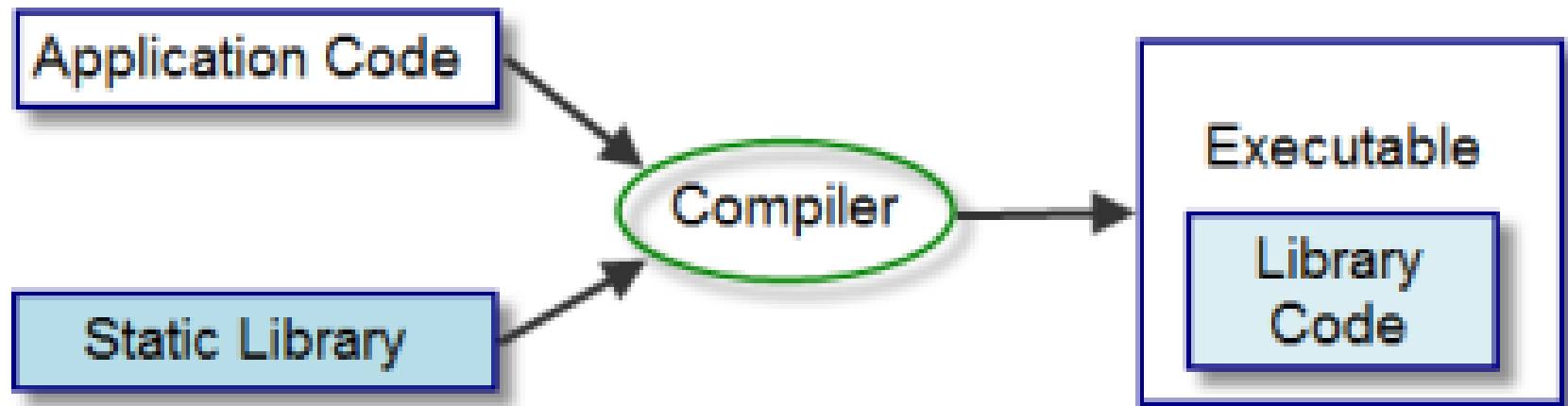
file binario .exe  
eseguibile



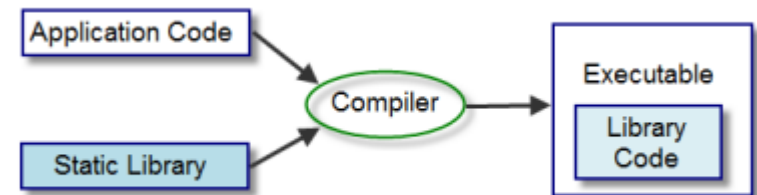
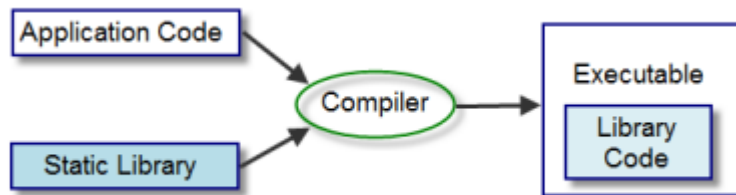
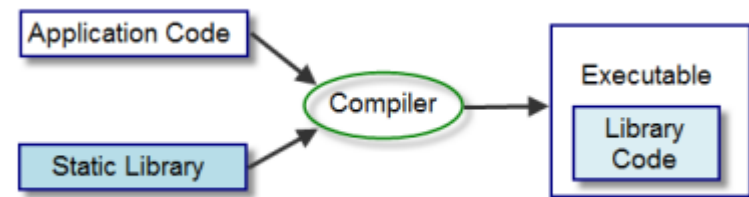
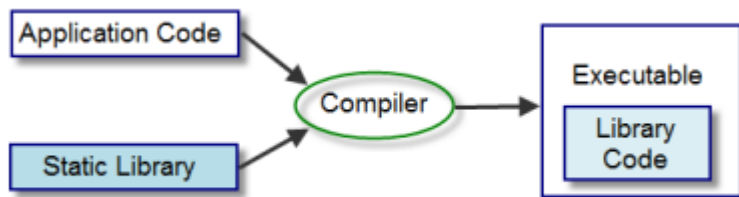


# Librerie statiche

- Nel linking, il codice delle funzioni di una libreria statica viene effettivamente inserito nel file eseguibile

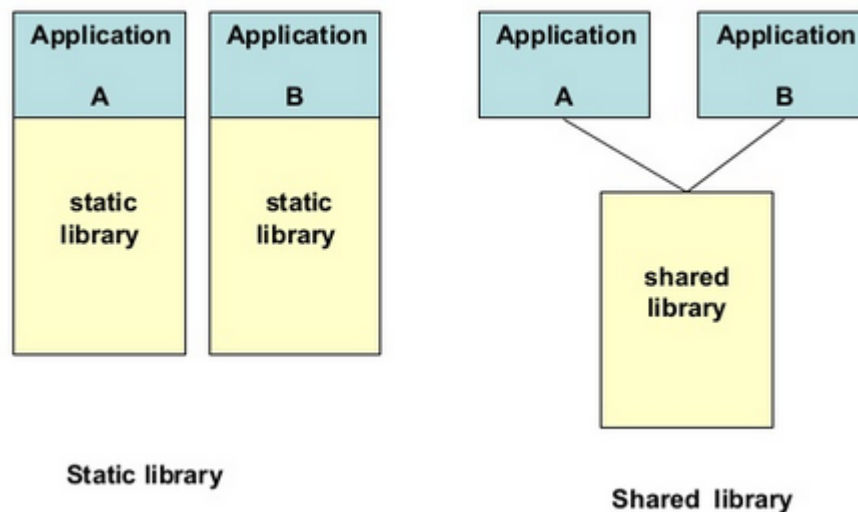



- Le stesse funzioni possono essere usate in più programmi




# Librerie dinamiche

- Nella creazione dell'eseguibile, nel file eseguibile NON viene inserito il codice delle funzioni di libreria, ma un riferimento al nome del file libreria e al nome della funzione
- Dynamic Link Libraries (DLL) - Windows
- Shared Libraries, Shared Objects - Linux/OSX




- 
- Quando viene eseguito il primo programma che usa quella libreria dinamica, questa viene caricata in memoria dal Sistema Operativo
  - Quando viene eseguito un altro programma che necessita di quella libreria, se questa già in memoria viene semplicemente condivisa
  - Il programma impiega meno tempo a partire se la libreria è già in memoria


- 
- Se una delle funzioni della libreria deve essere aggiornata (es. nuova versione), è sufficiente sostituire la libreria, mentre il programma non viene modificato
  - Problemi: se la libreria manca o è danneggiata il programma non funziona




# Algoritmo


- Un algoritmo è la descrizione operativa di come risolvere un problema
- La descrizione operativa è un elenco finito di istruzioni elementari del linguaggio da eseguire in successione
- Le singole istruzioni devono richiedere un tempo finito, non essere ambigue e avere un effetto osservabile
- Un algoritmo deve produrre un risultato, sempre lo stesso a partire dalle stesse condizioni iniziali


- 
- Le risorse necessarie alla realizzazione dell'algoritmo devono essere “ragionevoli” per la tecnologia attuale, considerando in particolare:
    - il tempo complessivo di esecuzione
    - la memoria massima necessaria
    - La necessità di particolari dispositivi periferici
  - Una richiesta eccessiva di risorse può pregiudicare la possibilità stessa di realizzare o utilizzare un algoritmo


- 
- Termine usato in Informatica, ma estendibile a qualsiasi ambito:
    - il metodo per risolvere le equazioni di 2 o grado
    - come preparare una ricetta
    - come montare un armadio IKEA
    - come avviare un motore di aereo
    - come installare un app sul cellulare




- 
- Le istruzioni saranno diverse
    - calcolare  $b^2 - 4ac$
    - sbattere 4 albumi
    - unire i pannelli 12 e 13 con una vite C
    - inserire i magneti
    - toccare il bottone installa
  - tutte hanno le caratteristiche indicate

- 
- Istruzioni non adeguate
    - Calcola tutte le cifre di  $\pi$
    - Aggiungere alcuni degli ingredienti
  - Un algoritmo che produce risultati diversi con gli stessi dati iniziali
    - non è corretto perché non è deterministico
    - es. un'equazione di 2 o grado deve sempre dare gli stessi risultati a partire dagli stessi valori di  $a$ ,  $b$  e  $c$

- 
- Un algoritmo che richiede un tempo irragionevole per la tecnologia attuale, considerati costi e benefici, non risolve il problema
  - Esempio: decifrare un file zip cifrato
    - Algoritmo: basta provare tutte le combinazioni di caratteri fino a trovare la password giusta
  - Ipotesi: consideriamo solo i 96 caratteri del codice ASCII standard e che la password sia di 8 caratteri

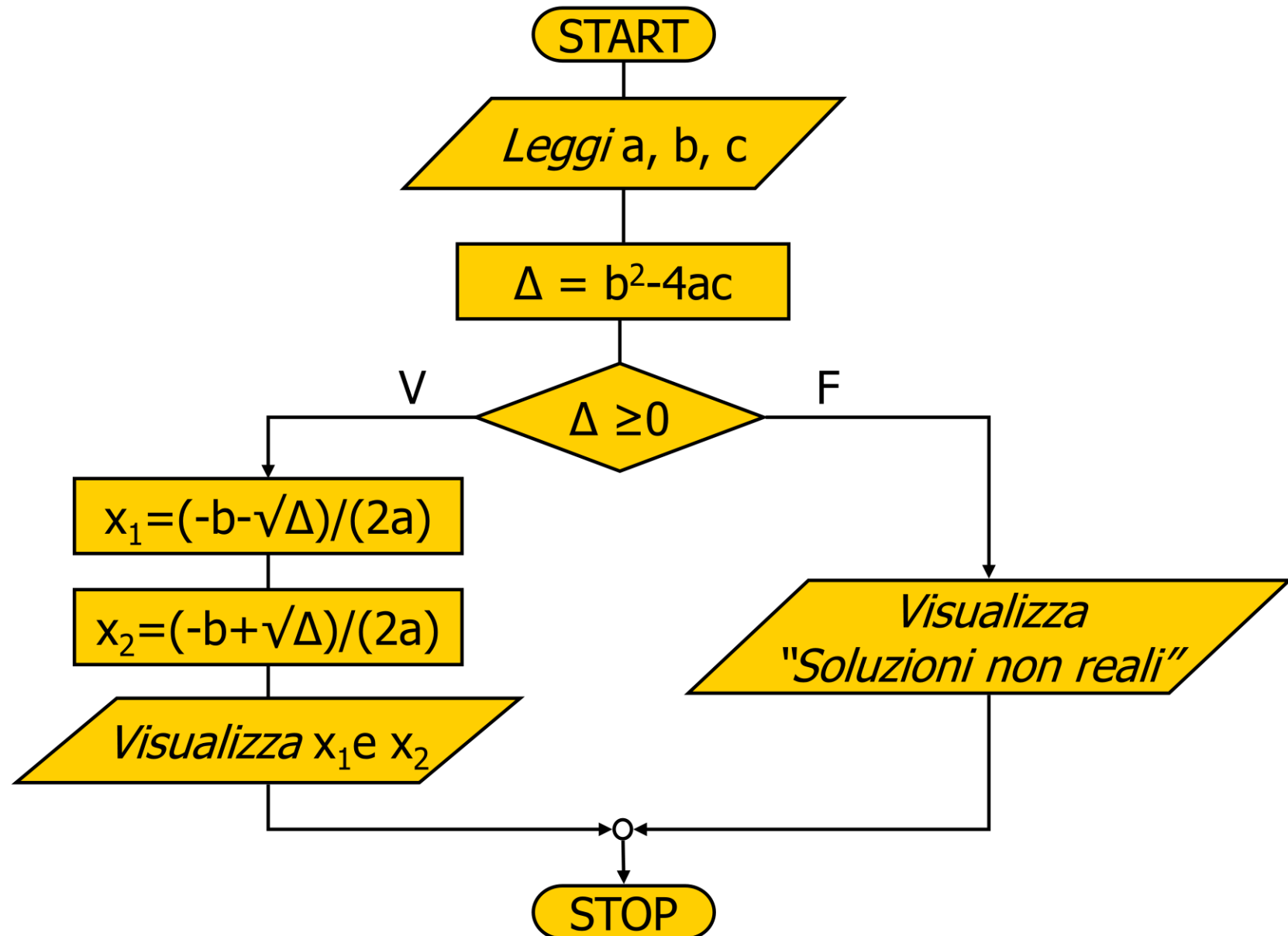
- 
- Ci sono  $96^8$  (7.2 milioni di miliardi) di possibili password
  - Con un computer che prova 34 milioni di password al secondo senza interruzioni occorrono quasi 5 anni per provarle tutte

- 
- Vari formalismi per descrivere un algoritmo
    - Pseudo-codice: descrizione testuale con un qualche livello di formalismo
    - Flow-chart: formalismo grafico
    - HLL: linguaggio di alto livello

# Pseudo-codifica

- *Descrivere l'algoritmo per il calcolo delle soluzioni reali di un'equazione di secondo grado*
- *Descritto con pseudo-codifica:*
  - Richiedere i valori dei coefficienti  $a$ ,  $b$ ,  $c$*
  - Calcolare il discriminante  $\Delta$  con la formula  $b^2 - 4ac$*
  - Se  $\Delta \geq 0$  calcolare le soluzioni e visualizzarle*
  - Altrimenti visualizzare "Soluzioni non reali"*

# Flow-chart





# HLL

- Vediamo l'algoritmo in C