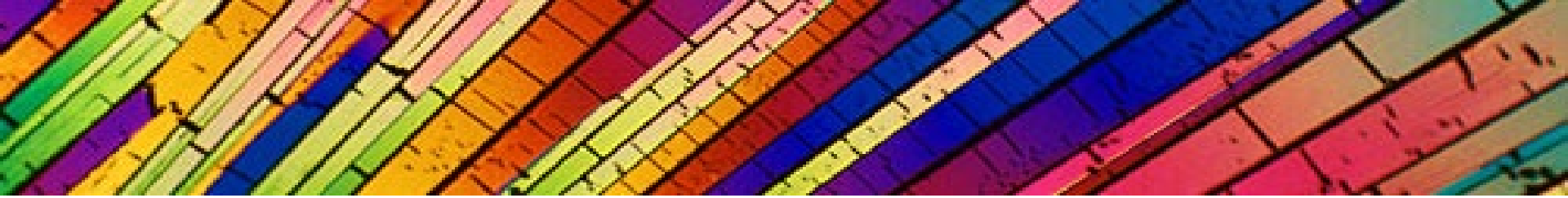


La programmazione

Ver. 3

Programmazione

- La programmazione è quell'attività che porta a sviluppare un software
- Se esiste un procedimento che:
  - può essere descritto in modo non ambiguo
  - conduce sempre all'obiettivo desiderato in un tempo finito

  allora esistono le condizioni per affidare questo compito a un calcolatore
- Questo procedimento si chiama *algoritmo*
- Esempi di algoritmi non informatici:
  - una ricetta di cucina
  - il metodo per risolvere le equazioni di 2º grado

Programma

- 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.
- L'attività del programmatore consiste nel *definire una sequenza di operazioni di base* eseguendo le quali il calcolatore risolve un determinato problema

Linguaggi

- Perché il programmatore possa descrivere al calcolatore quali operazioni fare, serve un linguaggio noto ad entrambi
- Problema:
  - Il calcolatore comprende solo sequenze di zeri e uno (ad es. la sequenza 1001001 potrebbe significare, per un ipotetico calcolatore, "fai la somma"): *linguaggio macchina*
  - Il programmatore comprende le parole "fai la somma" (mentre 1001001 non significa nulla per lui): *linguaggio umano*

Linguaggi di programmazione

- Soluzione 1

  Il programmatore impara il *linguaggio macchina* (sequenze di 0 e 1), detto anche *codice macchina* o *codice nativo*, ma questo:

  - ha un *basso livello di astrazione* (si scende molto nei dettagli realizzativi e si perde la visione di insieme del problema da risolvere), le istruzioni sono molto semplici (somma, sposta valore, ecc.)
  - è difficile da ricordare (le istruzioni possono essere diverse centinaia, ciascuna composta da molti bit, ad esempio 1001001 fa una somma)
  - è diverso per ogni piattaforma hardware (ogni tipo di microprocessore ha il suo set di istruzioni)

Linguaggi di programmazione

- Soluzione 2

  Il programmatore impara un linguaggio corrispondente al linguaggio macchina detto *assembly* (in Italia molti lo chiamano *assembler*)
- Il linguaggio assembly è sostanzialmente una *riscrittura del linguaggio macchina in forma mnemonica*: ogni istruzione del linguaggio assembly corrisponde a un'istruzione in linguaggio macchina, ma ha il vantaggio di essere *mnemonico* (es. ADD è l'istruzione per calcolare una somma)

Linguaggi di programmazione

- Soluzione 2 *(Continuazione)*
- Il linguaggio assembly:
  - ha lo stesso basso livello di astraz. del l. macchina
  - è più facile da ricordare (es. per avere una somma invece di scrivere 1001001 si scrive ADD)
  - viene tradotto in linguaggio macchina da un programma relativamente semplice (*assembler*) che, in linea di massima, sostituisce le istruzioni assembly con le corrispondenti istruzioni macchina
  - è diverso per ogni tipo di microprocessore e può esisterne più di uno per lo stesso processore
  - ha semplici costrutti che vengono convertiti in più istruzioni macchina per facilitare la programmazione

Linguaggi di programmazione

- Soluzione 3
  Il programmatore impara un *linguaggio di programmazione ad alto livello* (HLL) che:
  - ha un alto livello di astrazione: le istruzioni hanno un significato più complesso (es. A+B calcola la somma di due valori) e non serve al programmatore sapere come vengano realizzate dal processore
  - viene tradotto in linguaggio macchina o in assembly da un traduttore complesso (ogni istruzione HLL corrisponde a sequenze di istruzioni assembly), quindi è indipendente dal processore e dal sistema operativo (non tutti i linguaggi lo sono realmente)
  - è più simile al linguaggio umano e quindi più facile da ricordare (es. "`print X`" visualizza il valore di X)

Linguaggi di programmazione

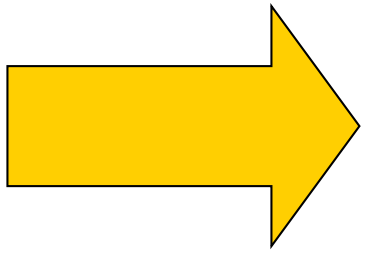
- Indipendentemente dalla soluzione adottata (che dipende dal linguaggio usato), il programmatore scrive in un file le istruzioni, queste costituiscono il *codice sorgente* o *programma sorgente* detto anche solo *programma*

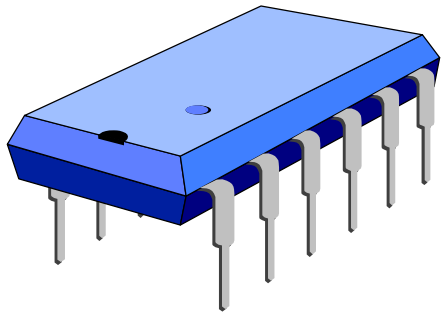
Traduttore di tipo interprete

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

Sorgente

```
somma

stampa

leggi

calcola

...
```



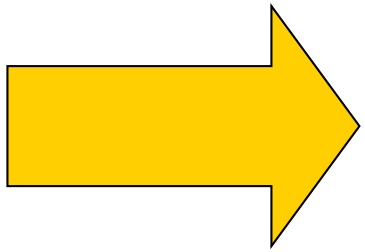


Traduttore di tipo compilatore

- Tutto il codice sorgente viene tradotto in linguaggio macchina e memorizzato in un file detto *programma* (o *file* o *codice*) *eseguibile*, o anche solo *programma* o *eseguibile*

Sorgente

```
somma

stampa

leggi

calcola

...
```



Eseguibile

```
1001001010101
0010100101010
1001001010010
1010010100101
0101010101011
0101001010100
1001001010010
```

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 molto superiore
- Il *compilatore* è normalmente in grado di *ottimizzare il codice* per produrre o un programma più veloce o un programma più piccolo

Differenze

Competenze necessarie per l'uso

- Eseguire un programma *interpretato* richiede che l'utente finale sia in possesso del *programma interprete* e impari a utilizzarlo
- 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

Differenze

Copyright e gestione della complessità

- L'*interprete* richiede di disporre del codice sorgente del programma che quindi risulta visibile e modificabile da chiunque (può essere reso difficilmente leggibile)
- Il programma eseguibile (ossia il risultato della *compilazione*) non necessita del sorgente: protezione del copyright (è molto complesso ricostruire un sorgente da un eseguibile)
- La procedura di *compilazione* permette di suddividere facilmente un programma complesso in più parti

Bytecode

- Alcuni linguaggi (es. Java, Python) compilano il sorgente producendo un file intermedio non in codice nativo (quindi non eseguibile), ma in un assembly generico detto *bytecode*, di veloce traduzione in un codice macchina reale
- I calcolatori su cui deve essere eseguito (fatto "girare" in gergo) 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 su cui viene eseguito

Programmi e processi

- Il termine *programma* identifica il codice sorgente e il codice eseguibile, si capisce dal contesto a quale accezione ci si riferisce
- Quando un programma viene eseguito (in gergo “fatto girare”, “lanciato”, “to run” in Inglese) è chiamato più propriamente *processo*, anche in questo caso si usa spesso il termine *programma*

Librerie

- In un HLL il programmatore non ha necessità di programmare le *funzioni di base* (quali leggere un numero dalla tastiera, calcolare la tangente, visualizzare una riga di testo, ecc.)
- 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 eseguibili* (quindi già tradotti in linguaggio macchina) che realizzano queste funzioni vengono raggruppati in file detti *librerie* (collezioni di funzioni)

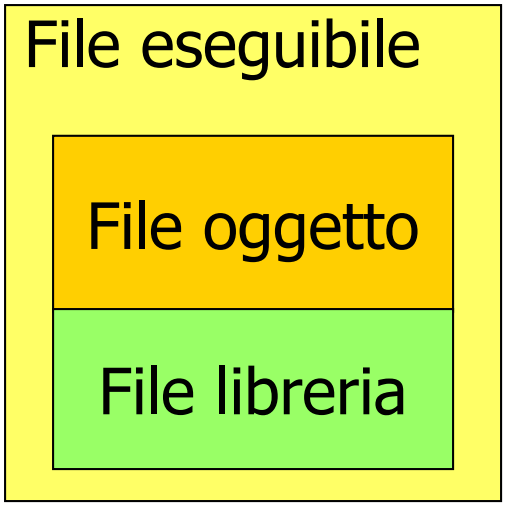
Creazione di un eseguibile

- Il processo di creazione di un eseguibile a partire dai sorgenti (tale processo è detto *build*) è composto da 2 fasi:
  - **compilazione**: il sorgente viene compilato in codice macchina, ma alcune parti (le "funzioni di base") 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 oggetti e *una o più* librerie

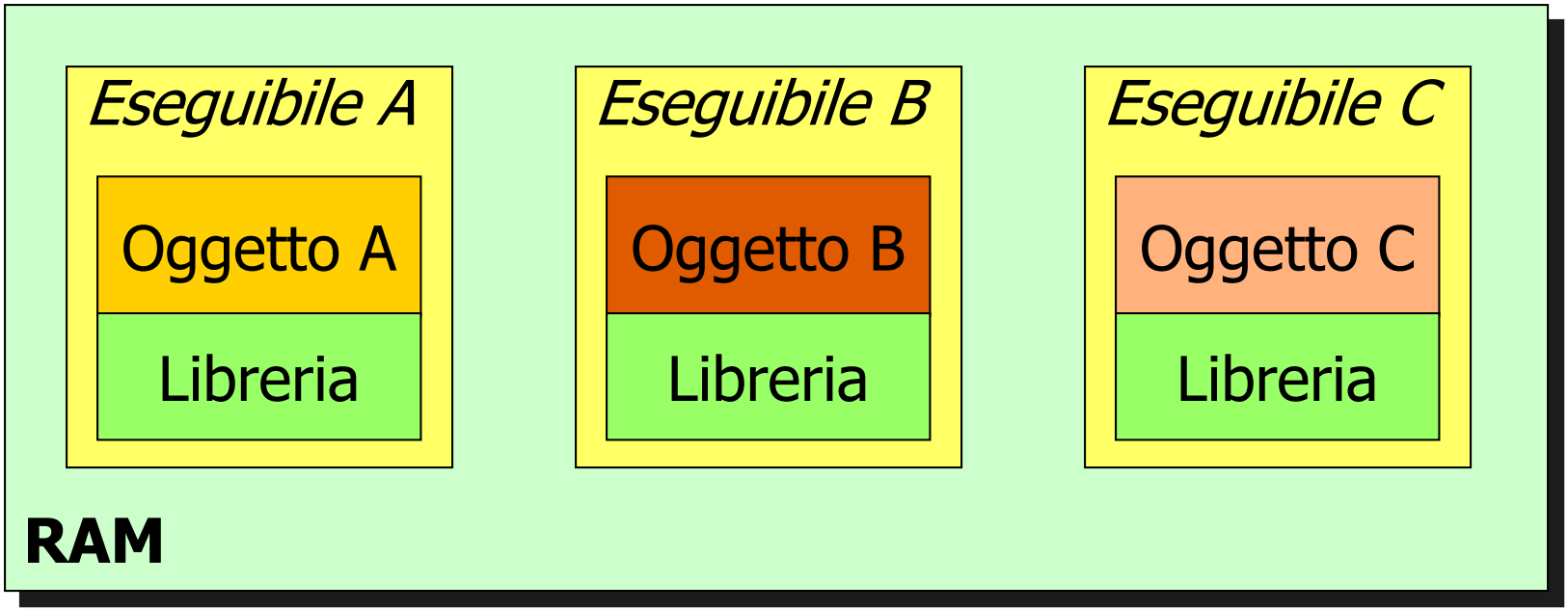
Librerie statiche

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



Librerie statiche

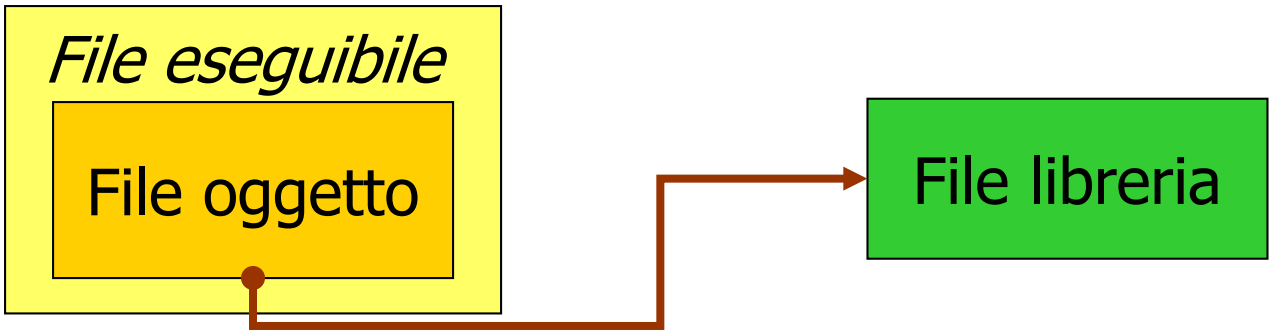
- Spesso la stessa libreria è usata da più programmi (ad es. la libreria delle operazioni di input/output)



- Basterebbe una sola copia di Libreria

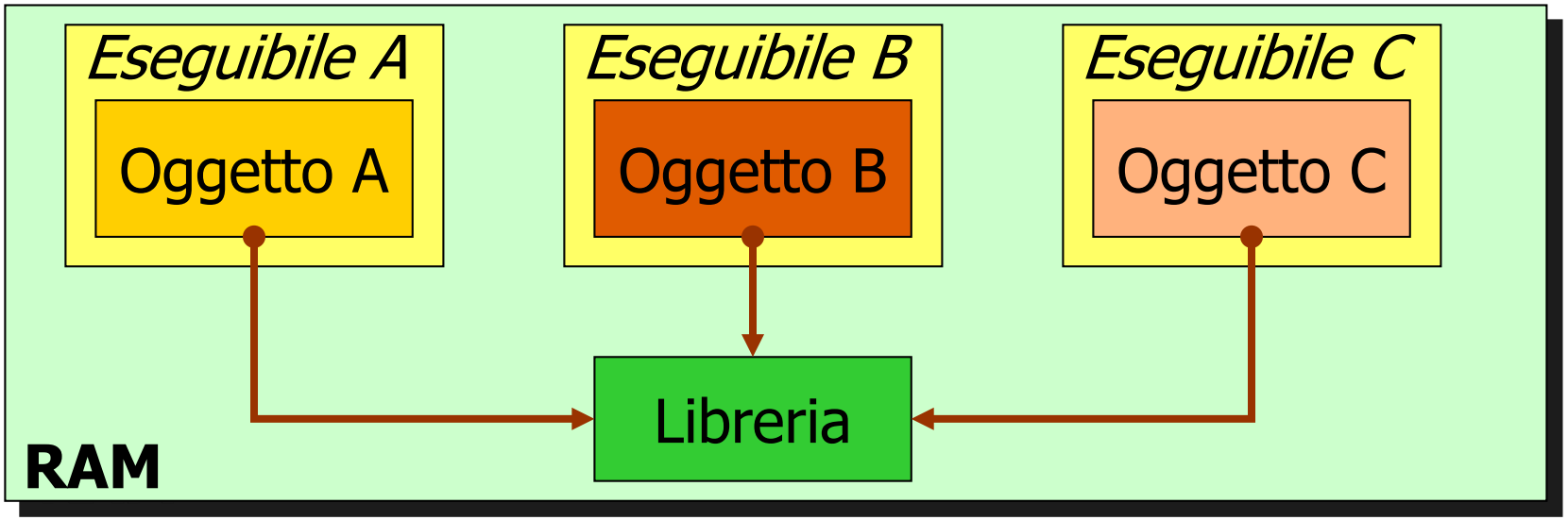
Librerie dinamiche

- Dynamic Link Libraries (DLL) - Windows
- Shared Libraries, Shared Objects - Linux/OSX
- 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



Librerie dinamiche

- 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, essendo questa già in memoria viene semplicemente condivisa



Librerie dinamiche

- 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 nel Sistema Operativo, mentre il programma non viene modificato
- Problemi: se la libreria manca o è danneggiata il programma non funziona

Interfacce utente

- L'interazione tra utente e programma può avvenire tramite interfacce:
  - a carattere (*console mode*):
    vengono visualizzate e immesse solo righe di testo;
    l'utente interagisce soltanto con video e tastiera
  - grafiche (GUI – *Graphical User Interface*):
    vengono visualizzati pannelli, bottoni, caselle di testo, immagini, ecc.;
    l'utente interagisce con video, tastiera e mouse
- Molti linguaggi dispongono di entrambe le interfacce

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 (significato univoco) e avere un effetto osservabile
- Un algoritmo deve produrre un risultato, sempre lo stesso a partire dalle stesse condizioni iniziali

Algoritmo

- 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

Algoritmo

- Termine usato primariamente in Informatica, ma estendibile a qualsiasi ambito:
  - il metodo per risolvere le equazioni di 2º grado
  - come cucinare 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*"), ma tutte hanno le caratteristiche indicate precedentemente

Algoritmo

- Istruzioni quali "calcola tutte le cifre del Pi greco" (richiede tempo e memoria infiniti), "aggiungere alcuni degli ingredienti" (ambigua), "pensa a un numero" (effetto non osservabile) non sono adeguate alla descrizione di un algoritmo
- Un algoritmo che produce risultati diversi con gli stessi dati iniziali non è corretto perché non è deterministico (es. un'equazione di 2° grado deve sempre dare gli stessi risultati a partire dagli stessi valori di a, b e c)

Algoritmo

- 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
  - Commento: 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

Descrizione di algoritmi

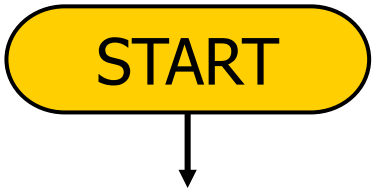
- Vari i formalismi per la descrizione degli algoritmi, ossia quali istruzioni elementari eseguire e in che ordine, fra questi:
  - la *pseudo-codifica* è una descrizione testuale (come negli esempi precedenti e nel primo esperimento che seguirà)
  - i *diagrammi di flusso* o *Flow-Chart* sono un formalismo grafico
- I flow-chart verranno utilizzati nelle slide per descrivere il funzionamento dei costrutti che costituiscono il linguaggio C

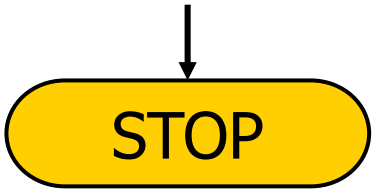
Flow-chart

- Un flow-chart:
  - se è poco dettagliato è adeguato per dare una *visione globale ma approssimativa della soluzione*, non è sufficiente per fornire una soluzione del problema direttamente implementabile; usa elementi descritti con un linguaggio umano
  - se è dettagliato descrive la soluzione con precisione tale da portare alla *soluzione implementabile del problema*, usa costrutti immediatamente riconducibili alle istruzioni di un linguaggio di programmazione (alto o basso livello)
- Un flow-chart può avere collegamenti ad altri eventualmente più dettagliati

Flow-chart – Start e Stop

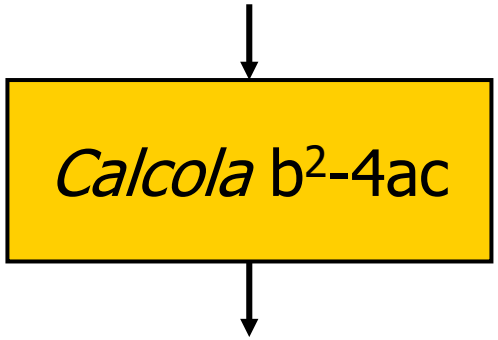
- Il formalismo grafico del flow-chart si avvale di pochi **simboli di base** che racchiudono la descrizione delle operazioni e di **linee o frecce** che ne indicano la sequenza di esecuzione
- Di ogni flow-chart deve esistere un *unico punto di partenza* e, se "strutturato" (concetto descritto in altre slide), un unico punto di fine
- I due simboli sono i seguenti (in Inglese o in Italiano):





Flow-chart - Operazione

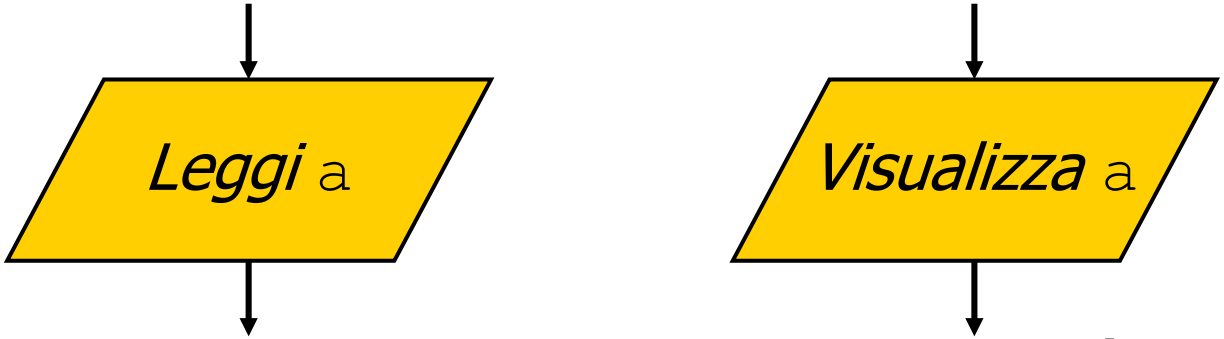
- Un'operazione che NON coinvolge l'utente viene racchiusa in un *rettangolo*



- Le operazioni possono essere descrittive o dettagliate a seconda del tipo di F.C.
- È possibile, se opportuno, inserire più operazioni nello stesso rettangolo

Flow-chart – Operazione di I/O

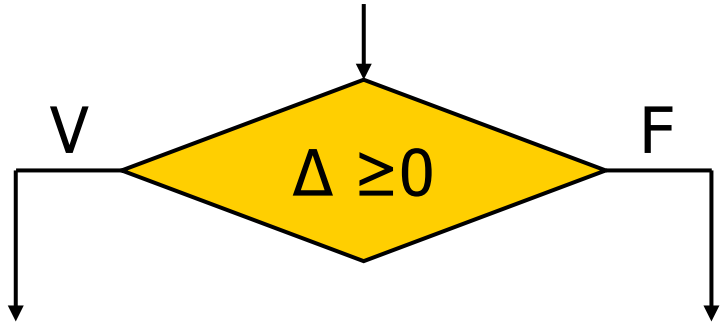
- Un'operazione che coinvolge l'utente (Input o Output, il contenuto esplicita quale dei due tipi di operazione) viene racchiusa in un *parallelogramma*



- È possibile, se opportuno, inserire più operazioni elementari nello stessa elemento
- In alcuni casi è implicito dal contesto se l'operazione è un Input o un Output

Flow-chart - Decisione

- Un'operazione che indica la valutazione di una condizione che può dare risultato Vero (V) o in Inglese True (T) o Falso (F)/False (F) viene racchiusa in un *rombo*



- Essendo *condizioni logiche*, non ci possono essere condizioni che danno altri risultati se non vero o falso

Esempio di algoritmo

- *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 Δ con la formula b^2-4ac

  Se Δ ≥ 0 calcolare le soluzioni e visualizzarle

  Altrimenti visualizzare "Soluzioni non reali"
- Descritto con flow-chart: alla pagina seguente, "Δ = b^2-4ac" significa "*metti in Δ il risultato del calcolo* b^2-4ac" o in modo del tutto equivalente "Δ *prende il valore di* b^2-4ac")

Esempio di algoritmo