



INTRODUZIONE ALLA PROGRAMMAZIONE

HTML – JavaScript

Ing. Massimo Nannini

Mi presento

Formazione:

- Laurea in ingegneria Elettronica ind. Automazione
- Master in costruzione di macchine automatiche



Attività lavorativa:

- Consulente informatico e project manager
- Formatore
- Sviluppatore software
- Software architect
- Owner Gemax consulting

Conosciamoci



Di cosa parleremo ?

Modulo 1 – Introduzione alla programmazione

- Introduzione alla programmazione
- Algoritmi
- Programmazione strutturata
- Tipi di base
- Espressioni e funzioni matematiche
- Esecuzione condizionale
- Strutture iterative
- Controllo di flusso

Di cosa parleremo ?

Modulo 2 – Dai sorgenti al programma esecutivo

- Traduttori, compilatori ed interpreti
- Nodelli di esecuzione
- Pro & Cons

Di cosa parleremo ?

Modulo 2 – HTML – CSS – JavaScript

- Introduzione ai linguaggi per il WEB
- Concetti base di Frontend e Backend
- Il protocollo HTTPs
- Le direttive

Di cosa parleremo ?

Modulo 3 – Paradigmi

- Paradigmi di programmazione
- OOP
- Passaggio dei parametri

Cominciamo con una domanda ...

Che cosa è l'informatica ?

L'informatica

L'informatica è una scienza che è nata e vissuta anche quando il computer non era ancora stato inventato.

La parola deriva dal termine francese coniato nel 1962 da Philippe Dreyfus **informatique** che è l'unione delle parole *information* ed *automatique*.

Nascita dell'informatica

L'informatica ha radici molto antiche.

Da sempre l'uomo ha avuto la necessità di gestire le informazioni (*Homo sapiens*) ma alcuni meccanismi per automatizzare il trattamento dei dati e delle operazioni aritmetiche erano già noti ai babilonesi intorno al **X sec. a.C.**

L'informatica moderna

La nascita della concezione moderna dell'informatica ha radici post-rinascimentali nelle persone di Pascal, Leibniz, Babbage, Lovelace, Boole e Frege.

I primi lavori si avranno con Alan M. Turing e John Von Neumann.

L'informatica moderna

Tutti questi punti di vista puntano a definire l'informatica come **scienza del calcolabile**.

Non vi è comunque ancora unanimità nel mondo scientifico per definire l'informatica **una scienza**, alcuni sostengono che sia una disciplina evolutiva della matematica.

L'informatica moderna

Citazione:

“[...] Fino a poco tempo fa, i matematici teorici consideravano una problema risolto se esisteva un metodo conosciuto, o algoritmo, per risolverlo; il procedimento di esecuzione dell'algoritmo era di importanza secondaria. Tuttavia, c'è una grande differenza tra il sapere che è possibile fare qualcosa e farlo. [...]”

Enrico Bombieri

Medaglia Fields, 1974

Università di Pisa



Blaise Pascal

Matematico, fisico, filosofo e religioso francese (1623-1662) è considerato uno dei precursori dell'informatica.

Appena **diciottenne** progettò e costruì circa cinquanta esemplari di un calcolatore meccanico capace di eseguire addizioni e sottrazioni, detto **Pascalina**.

Morì a trentanove anni a causa delle sue da sempre malferme condizioni di salute.

Aneddoto su Pascal

Un giorno mentre frequentava l'università, Pascal arrivò in ritardo ad una lezione e vide sulla lavagna i testi di tre problemi scritti dal professore. Ritenendo che si trattasse di esercizi da risolvere li ricopiò senza sapere che si trattava di 3 famosi dilemmi matematici fino a quel momento senza soluzione.

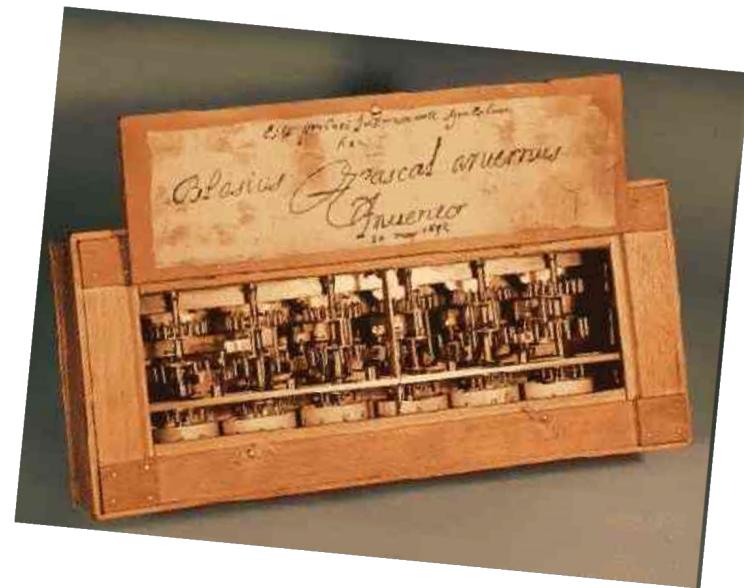
Il giorno dopo, Pascal andò dal professore dicendogli:

“Scusi professore, uno dei tre esercizi che ci ha assegnato ieri non mi è riuscito”

La Pascalina

Precursore della moderna calcolatrice, è stata inventata nel 1642.

Era costituita da una serie di **ruote dentate** indicanti le unità, le decine, le centinaia e così via. Ognuna era divisa in dieci settori, dallo 0 al 9, corrispondenti alle cifre del sistema decimale.



La Pascalina

Il primo esemplare fu costruito da Pascal per aiutare il padre, funzionario delle imposte per **gestire la propria contabilità**.

Fu considerata per anni la prima calcolatrice meccanica inventata, anche se questo merito andrebbe alla calcolatrice di **Wilhelm Schickard** (1623), eccellente in molteplici campi (arte, scienza, etc...) che visse in Germania.



George Boole

Matematico e logico britannico, è considerato il fondatore della logica matematica (1815-1864).

Autodidatta, studiò la matematica fin da giovane, morì all'età di 49 anni per l'aggravarsi di una forma febbrale causata da un banale raffreddore.

Algebra di Boole

In matematica ed informatica è una struttura algebrica che tratta i **bit** tramite operatori logici (e, o, non, ...).

Questa struttura oggi è molto utilizzata per progettare **circuiti elettronici** digitali.

Alcuni richiami di logica

p	q	p AND q
F	F	F
F	V	F
V	F	F
V	V	V

p	q	p OR q
F	F	F
F	V	V
V	F	V
V	V	V

Alcuni richiami di logica

p	NOT(p)
F	V
V	F

Esempio

*Se Pierino fa i compiti **E** riordina la stanza la mamma gli comprerà il gelato.*

In questo caso, avendo usato il connettivo **E** se Pierino adempie soltanto ad una delle due condizioni (o addirittura a nessuna) la mamma non gli comprerà il gelato.

Esempio

*Se Pierino fa i compiti **O** riordina la stanza la mamma gli comprerà il gelato.*

In questo caso, avendo usato il connettivo **O** la mamma comprerà il gelato a Pierino anche all'adempimento di una sola condizione (conoscendo Pierino non le svolgerà mai entrambe).

Una curiosità

La mamma dice a Pierino prima di uscire: “se fai i compiti ti comprerò il gelato”. Al ritorno, la mamma constata che Pierino non ha fatto i compiti ma gli compra comunque il gelato.

Com’è stata la mamma dal punto di vista logico?



Alan Mathison Turing

Matematico e logico britannico (1912-1954) è considerato uno dei padri fondatori della moderna informatica, introdusse una macchina ideale ed un test che portano il suo nome.

Personalità particolare, ebbe difficoltà a raggiungere il diploma perché poco interessato allo studio del latino e delle Sacre Scritture.

Alan Mathison Turing

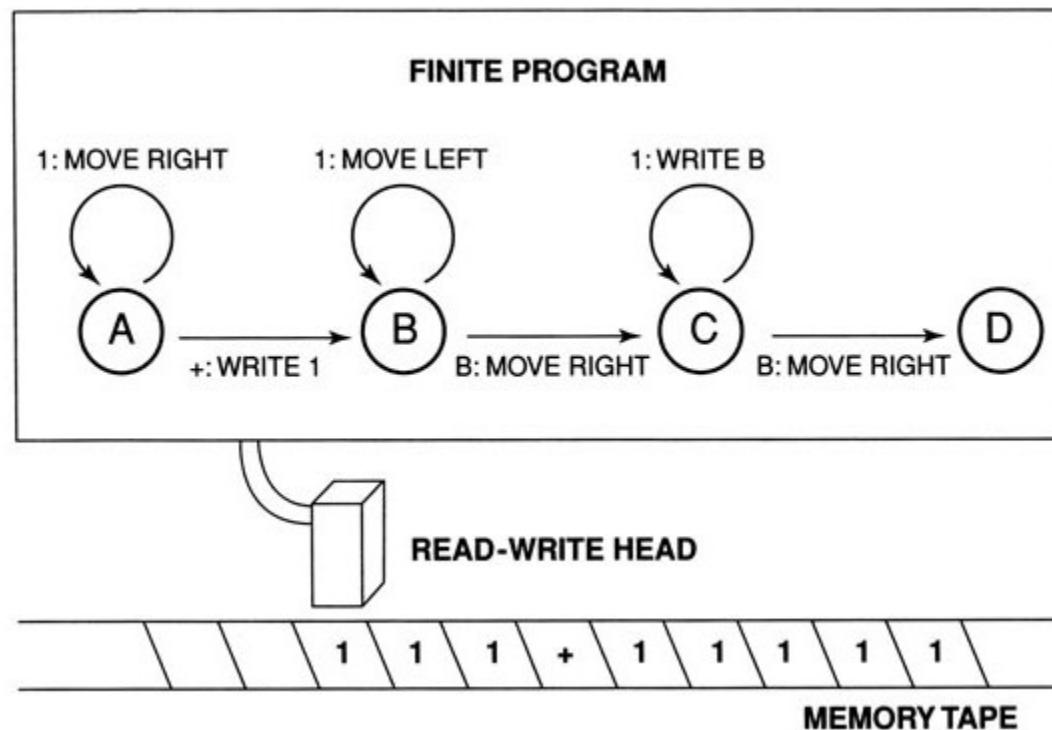
Più interessato a Relatività, calcoli astronomici e probabilità nel 1931 venne ammesso al King's College dell'Università di Cambridge dove studiò meccanica quantistica, logica e teoria della probabilità.

Laureato nel 1934 con il massimo dei voti, l'anno seguente ottenne un dottorato in ricerca (Ph.D.) e nel 1936 vinse il premio Smith.

Alan Mathison Turing

Lo stesso anno, pubblicò l'articolo nel quale descriveva, per la prima volta, quella che verrà poi definita come la “Macchina di Turing”, un sistema formale che in base a regole ben precise costituisce un modello di calcolo operante su stringhe.

Macchina di Turing



Macchina di Turing – come funziona

La macchina di Turing è simile ad una persona che esegue una procedura ben definita, cambiando il contenuto di un nastro di carta illimitato, suddiviso in quadrati che possono contenere simboli appartenenti ad un insieme finito.

La persona deve anche ricordare uno **stato** appartenente anche esse ad un insieme finito.

Quindi la macchina di Turing consiste in:

1. Un insieme finito di simboli detto *alfabeto*
2. Un nastro potenzialmente infinito suddiviso in celle discrete contenete un simbolo
3. Una *testina* mobile di lettura/scrittura
4. Una *memoria* capace di assumere un numero finito di stati diversi

Macchina di Turing – come funziona

La macchina di Turing può dunque compiere operazioni elementari di:

- scrivere o leggere un simbolo in una cella
- variare il suo stato di memoria
- spostare la testina a destra oppure a sinistra di una cella

L'attività della macchina consiste interamente in passi discreti, ciascuno dei quali è determinato dalla sua condizione iniziale.

Questa condizione iniziale è costituita a sua volta dallo stato della macchina e dal simbolo che in quell'istante occupa la cella del nastro esaminata.

Macchina di Turing - Istruzioni

Data una certa condizione iniziale la macchina riceve una **istruzione** per il passo successivo, in cui sono indicati:

- il simbolo che la macchina deve scrivere nella cella esaminata
- lo stato successivo della macchina
- la direzione dello spostamento della testina a destra (D) oppure a sinistra (S)

Una istruzione del tipo: (1, II, D) indica che la macchina deve scrivere il simboli 1 nella cella corrente, assumere lo stato II e spostarsi a destra.

Macchina di Turing Universale

- Bisogna creare una macchina di Turing per ogni specifico problema che si deve risolvere ?
- Esiste una macchina **programmabile** in grado di accettare la descrizione di una *qualsiasi* macchina di Turing e di *simularne* il comportamento ?

La risposta alle domande è la **Macchina di Turing Universale** esse è l'equivalente teorico dei calcolatori a programma memorizzato.

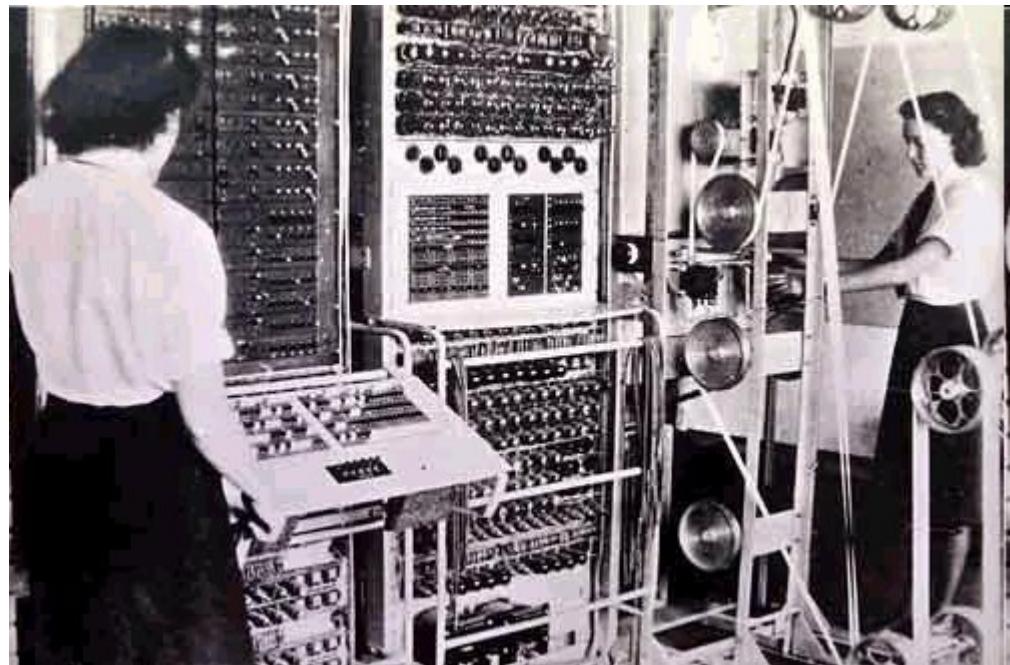


Alan Mathison Turing

Durante la seconda guerra mondiale, Turing mise le sue capacità matematiche al servizio del “Dipartimento delle comunicazioni” inglese per decifrare i codici usati nelle comunicazioni naziste cifrate tramite il sistema “**Enigma**”.

Nel 1942 realizzò una macchina chiamata **Colossus** in grado di decifrare in modo efficace ed efficiente i codici dei tedeschi.

Alan Mathison Turing



Alan Mathison Turing

L'anno 1950 pubblicò il suo studio sull'**intelligenza artificiale** creando così il "**Test di Turing**" atto a stabilire se una macchina che riproduca la mente umana possa essere considerata "pensante" oppure no.

Alan Mathison Turing

Il 31 marzo del 1952 venne arrestato per **omosessualità** e condotto in giudizio, dove fu messo davanti alla scelta fra il carcere e la “**cura ormonale**” che Turing scelse. Questa cura gli provocò molti scompensi fisici.

Morì nel 1954 a soli 42 anni ingerendo una **mela avvelenata** con il cianuro prendendo spunto dalla fiaba di **Biancaneve** da sempre molto apprezzata.



Alan Mathison Turing

“Machines take me by surprise with great frequency”
A.Turing

Le macchine mi prendono di sorpresa con grande frequenza.



John Von Neumann

Matematico ed informatico ungherese, personalità eminente nel mondo scientifico del XX secolo.

Ha contribuito in parecchi settori della scienza ed in particolare della matematica.

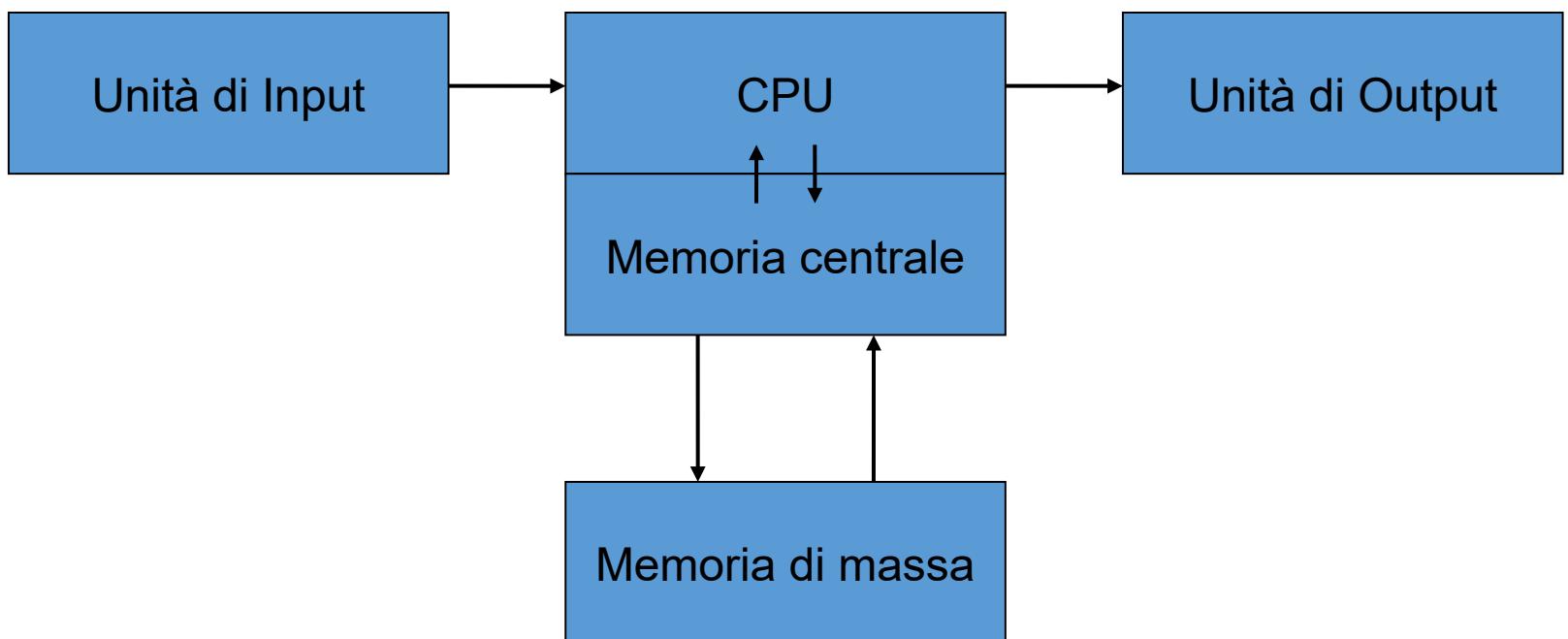
John Von Neumann

Basandosi sulle teorie di A.Turing nel 1945 creò l'EDVAC, la prima macchina digitale programmabile, nata per migliorare l'ENIAC, ammasso enorme di valvole e condensatori.

Nasce così

I'architettura di Von Neumann

Architettura di Von Neumann

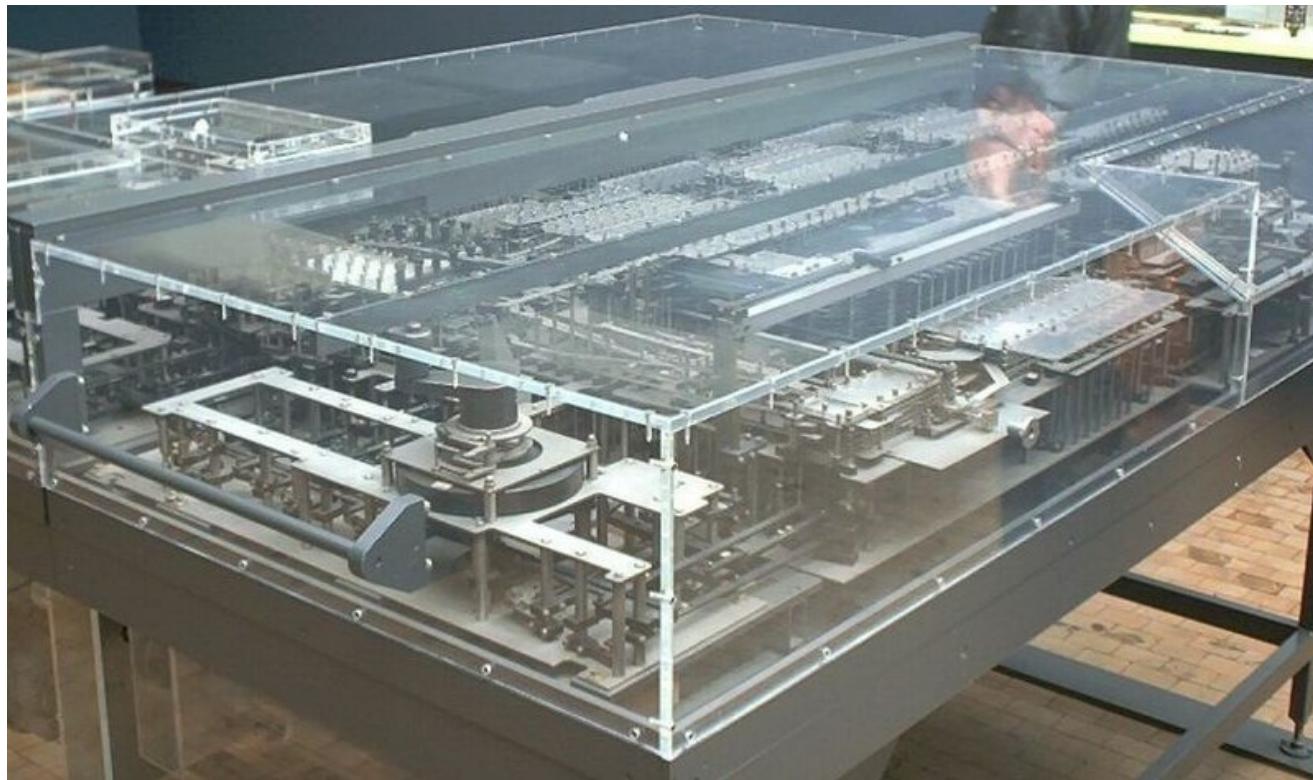


Aneddoto su Von Neumann

Pare che Stanley Kubrick nel suo film “**Il dottor stranamore**” si sia ispirato alla figura di Von Neumann per il personaggio dello scienziato pazzo il cui obiettivo era quello di decidere le sorti del Mondo.

In conclusione

Il primo computer programmabile nacque soltanto nel 1937.



Scienza o Tecnologia ?

L'informatica è una scienza, non una tecnologia.

La programmazione

.... stesura di un programma in una forma interpretabile dal calcolatore elettronico (o direttamente o dopo una rielaborazione opportuna), compiuta dal programmatore sulla base delle specifiche di programma fornite



Alcune definizioni

Risolvere un problema significa ricercare un procedimento risolutivo (**algoritmo**) che, eseguito, fornisce delle informazioni finali (**risultati**) dipendenti da alcune informazioni iniziali (**dati di ingresso**).

Queste informazioni iniziali e finali devono essere sempre codificabili mediante sequenze finite di simboli (ad esempio caratteri) dette **stringhe** (o mediante i numeri naturali) che indicheremo genericamente con il temine **dati**.

Dati vs. Informazioni

I **dati**, non costituiscono di per sé, **informazione**, ma la rappresentano quando riescono a portare qualche nuova conoscenza a chi li **riceve**.

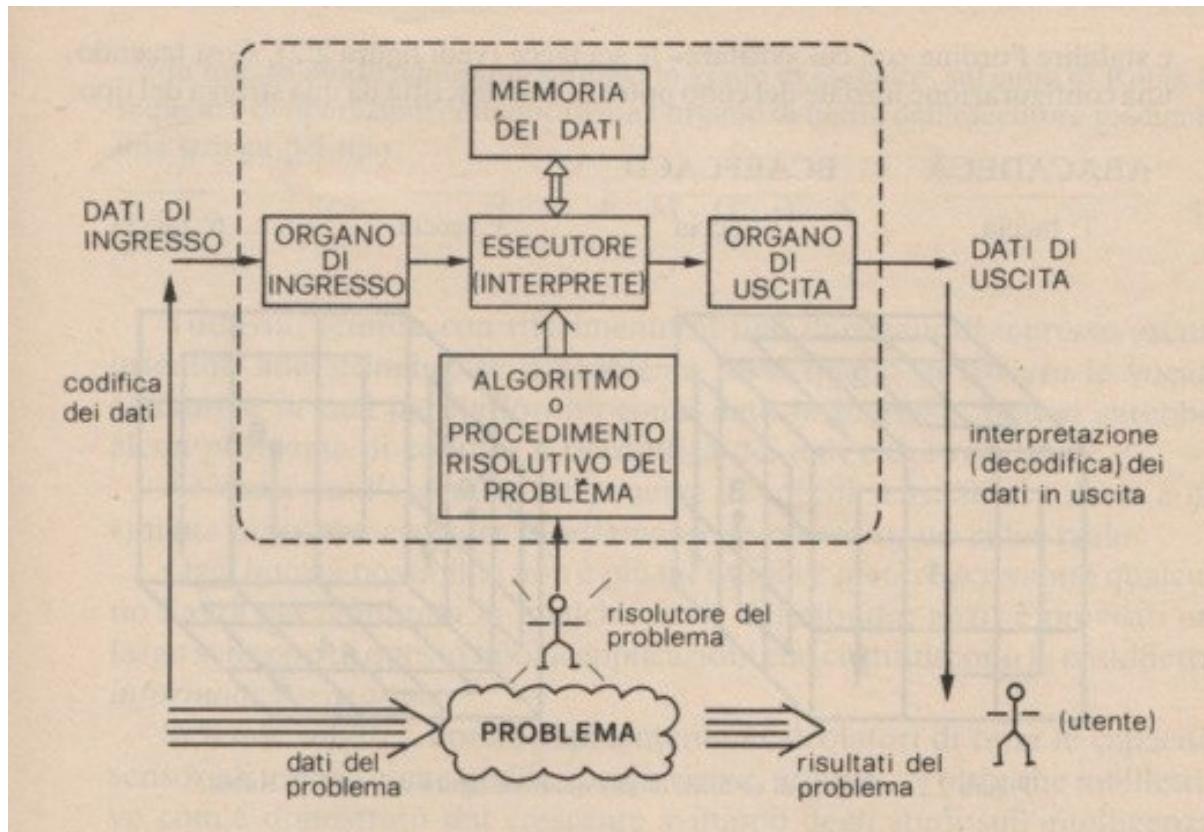
In questo caso chi riceve i dati è **l'esecutore** che attende i dati iniziali dell'**utente** per fornire i risultati richiesti.

Dati

I dati genericamente vengono divisi in:

- Dati di ingresso
(sono i dati iniziali che vengono forniti all'**esecutore**)
- Dati intermedi
(sono i dati che sono creati o trasformati dall'**esecutore** delle procedure risolutive o **algoritmo**)
- Dati di uscita
(sono dati che rappresentano i risultati)

Sistema di calcolo

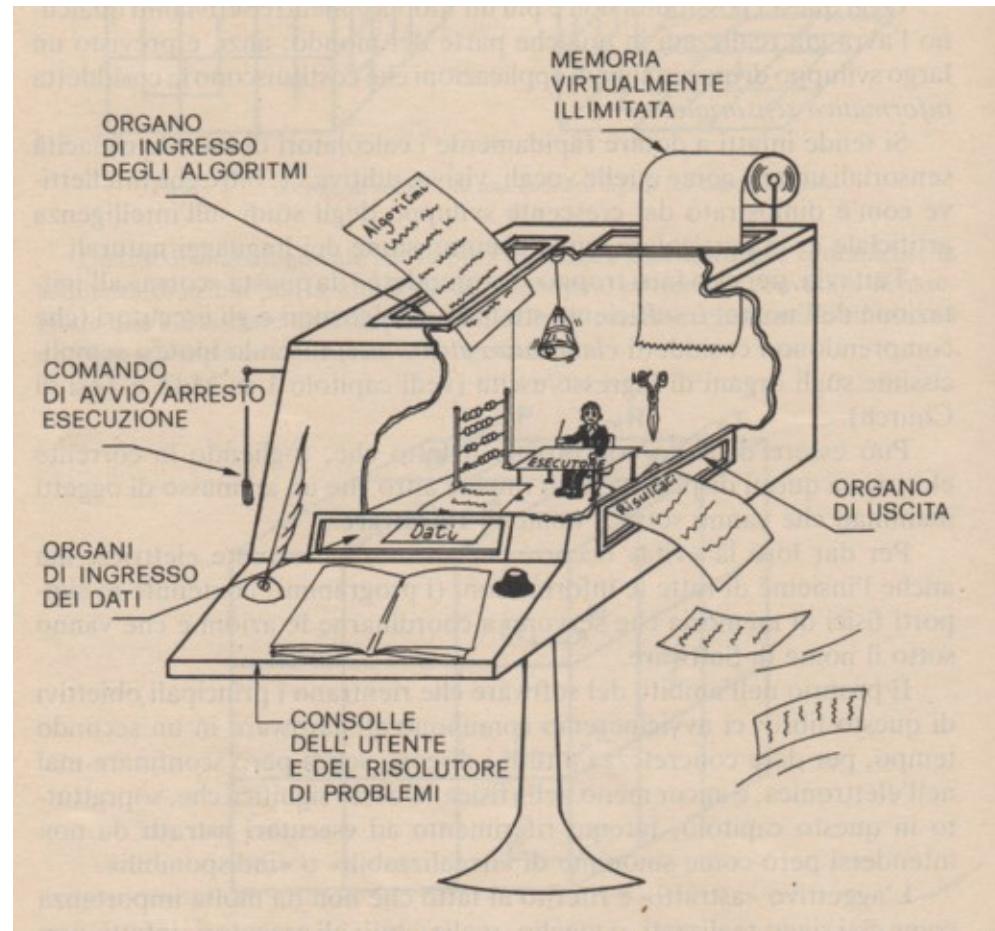


Esecutore astratto

Parliamo di **esecutore astratto** quando facciamo riferimento ad un sistema di calcolo in grado di svolgere delle operazioni/attività o meglio fornire un risultato dati dei valori di ingresso, senza fare alcun riferimento alla tecnologia con cui sono stati realizzati.

Gli esecutori non sono necessariamente dei dispositivi elettronici, meccanici ma entità in grado di eseguire un algoritmo. In ultima analisi anche noi se ci dotiamo di infinita pazienza e disponessimo di molto tempo potremmo diventare esecutori di algoritmi.

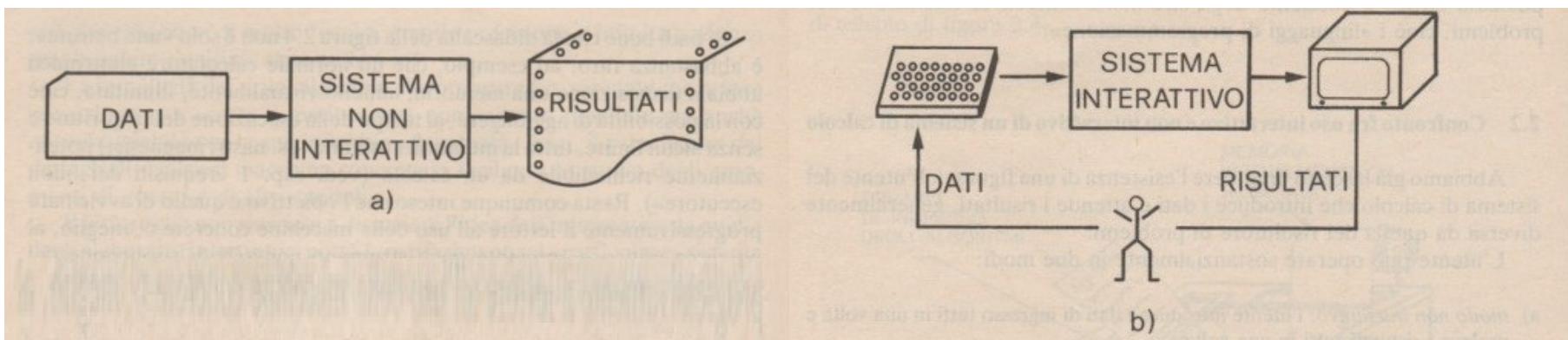
Uno dei più potenti e diffusi computer



Uso interattivo e non interattivo

L'utente può operare sostanzialmente in due modi:

- *modo non interattivo* : l'utente introduce solo i dati di ingresso tutti in una volta e preleva i dati di uscita tutti in una volta.
- *modo interattivo*: l'utente introduce una parte dei dati, attende un primo risultato, in base al quale decide quale nuovo ingresso fornire.



Algoritmo

- Dato un problema e un esecutore, l'algoritmo è:
 - una successione finita di passi elementari (operazioni e direttive)
 - eseguibili senza ambiguità dall'esecutore
 - che risolve il problema dato
- Quindi l'obiettivo è passare dal problema all'algoritmo
- Sarà compito dell'esecutore seguire le direttive dell'algoritmo

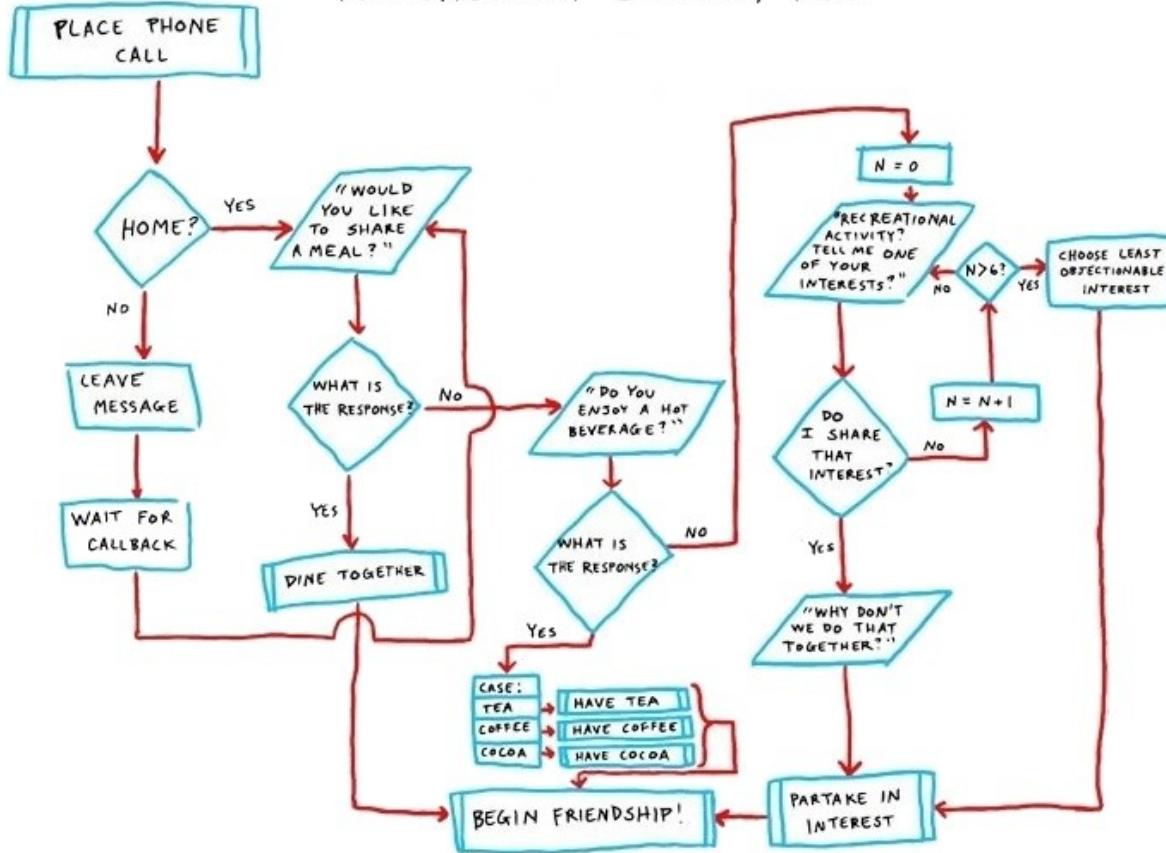
Algoritmo

- Un algoritmo non deve dipendere solo da valori predefiniti dei dati ma anche da dati inseriti dall'utente
- Quindi un algoritmo deve avere una formulazione generale
- In un algoritmo devono essere previsti particolari passi di acquisizione delle informazioni

Algoritmo dell'amicizia

THE FRIENDSHIP ALGORITHM

DR. SHELDON COOPER, Ph.D



Proprietà di un algoritmo

- **Correttezza:**
 - perviene alla soluzione del problema dato
- **Efficienza:**
 - perviene alla soluzione nel modo migliore possibile
 - la velocità è solo uno dei criteri. Si parla in generale di qualità
- **Finitezza:**
 - il numero di istruzioni che fanno parte di un algoritmo è finito
 - le operazioni definite in esso vengono eseguite un numero finito di volte

Proprietà di un algoritmo

- **Il determinismo:**
 - le istruzioni presenti in un algoritmo devono essere definite senza ambiguità
 - un algoritmo eseguito più volte e da diversi esecutori, a parità di premesse, deve giungere a medesimi risultati
- **Terminazione:**
 - l'esecutore deve terminare in tempo finito per ogni insieme di valori in ingresso
- **Realizzabilità pratica:**
 - l'esecutore deve essere in grado di eseguire l'algoritmo con le risorse a sua disposizione (informazioni + tecnologia)
- **Generalità**
 - un algoritmo, compatibilmente con i vincoli espressi dal problema, si occupa della risoluzione di famiglie di problemi

Elementi degli algoritmi

- Oggetti:
 - le entità su cui opera l'algoritmo
- Operazioni:
 - interventi da effettuare sugli oggetti
- Strutture di controllo:
 - sequenza, selezione, iterazione

Oggetti

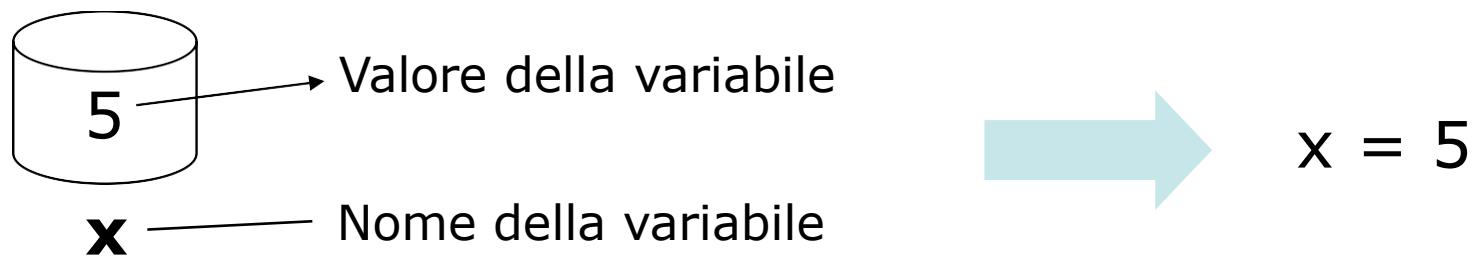
- Possono essere:
 - Costanti
 - Variabili
- Possono rappresentare:
 - Dati iniziali del problema
 - Informazioni ausiliarie
 - Risultati parziali
 - Risultati finali

Costanti

- Rappresentano dei valori che non variano durante l'esecuzione dell'algoritmo
- Possono essere sia numeriche che alfanumeriche
- Nel caso di costanti alfanumeriche il valore deve essere normalmente racchiuso tra doppi apici
- Esempio
 - 5 è una costante numerica (il numero 5)
 - “prova” è una costante alfanumerica
 - “5” è una costante alfanumerica (il carattere 5)

Variabili

- Sono dei contenitori di valori identificati da nomi simbolici
 - – x, y, somma, risultato, ... (il nome di una variabile deve sempre iniziare con un carattere non numerico)
- Il nome della variabile rappresenta una scatola
- Il valore della variabile il suo contenuto



Variabili

- Le variabili devono avere un tipo che rappresenta:
 - I valori ammissibili
 - Le operazioni che possono essere compiute sui suoi valori
- Esempio: una variabile di tipo numerico
 - può assumere solo valori numerici
 - può apparire come operando in una espressione matematica (e.g., x/y è corretta se x e y sono variabili di tipo numerico)
- Esempio: una variabile di tipo alfanumerico (stringa)
 - può assumere qualunque valore
 - non ha senso applicare l'operazione di divisione (e.g., se $a=“ciao”$ e $b=“mamma”$ cosa vuol dire a/b ?)

Attenzione!

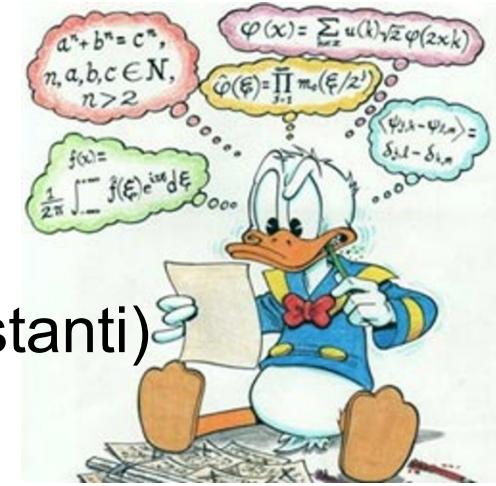
Tutte le variabili devono essere inizializzate

Assegnamento

- L'assegnamento è l'operazione che permette di modificare il valore di una variabile
- Si indica con il simbolo =
- $x = 6$ indica che ora la variabile x contiene il valore 6
- Attenzione! $x = 6$ è diverso da $x == 6$
 - Il primo modifica il valore della variabile
 - Il secondo predica sul valore corrente della variabile
- Regole fondamentali:
 - A sinistra dell'operatore di assegnamento deve esserci il nome di una e una sola variabile
 - A destra dell'operatore di assegnamento può esserci una costante, una variabile o anche un'espressione complessa
 - Il tipo di dato della variabile a sinistra deve essere compatibile con il tipo di dato ottenuto dall'espressione a destra

Operazioni

- Operatori aritmetici: $+$, $-$, $/$, $*$, ...
 - agiscono sugli operandi (variabili o costanti)
 - producono un valore numerico
- Operatori di confronto: $>$, $==$, $<$, ...
 - agiscono sugli operandi (variabili o costanti)
 - producono un valore logico (vero o falso)
- Funzioni: **cos(x)**, **log(x)**, ...
 - agiscono su valori detti parametri (variabili oppure costanti)
 - producono un valore



Operazioni

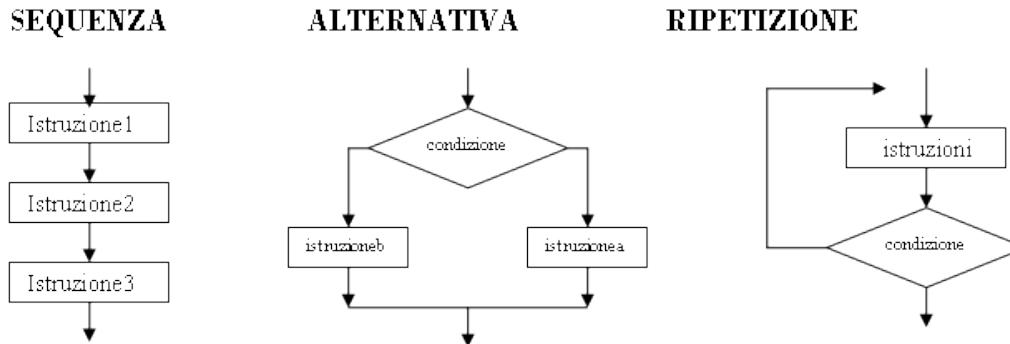
- Procedure: **Leggi(X)**, **Scrivi(N)**, ...
 - agiscono su valori detti parametri
 - effettuano operazioni
- Espressione: **5 + cos(Y)**, ...
 - composizione di operatori, funzioni, variabili e costanti
 - ad essa è associato un valore

Regole sulle operazioni

- Una operazione deve terminare entro un intervallo finito di tempo
(Es.: calcolare le cifre decimali di π , NO!)
- Una operazione deve produrre un effetto osservabile
(Es.: pensare al numero 5, NO!)
- Una operazione deve produrre lo stesso effetto ogni volta che viene eseguita a partire dalle stesse condizioni iniziali
(Es. $5+10=15$ deve valere sempre)

Strutture di controllo

- **Sequenza:** le operazioni sono eseguite una in seguito all'altra
- **Selezione:** blocchi di operazioni sono eseguite in alternativa
- **Iterazione:** blocchi di operazioni sono eseguite diverse volte



Rappresentazione di un algoritmo

- I linguaggi per descrivere gli algoritmi devono essere noti all'uomo che progetta e al calcolatore che deve eseguirli
- Fasi iniziali di progetto: struttura generale dell'algoritmo
 - descrizione rigorosa (non necessariamente formale) del flusso di controllo
 - descrizione semplificata delle direttive, per es. mediante l'uso di linguaggio naturale

Requisiti dei linguaggi

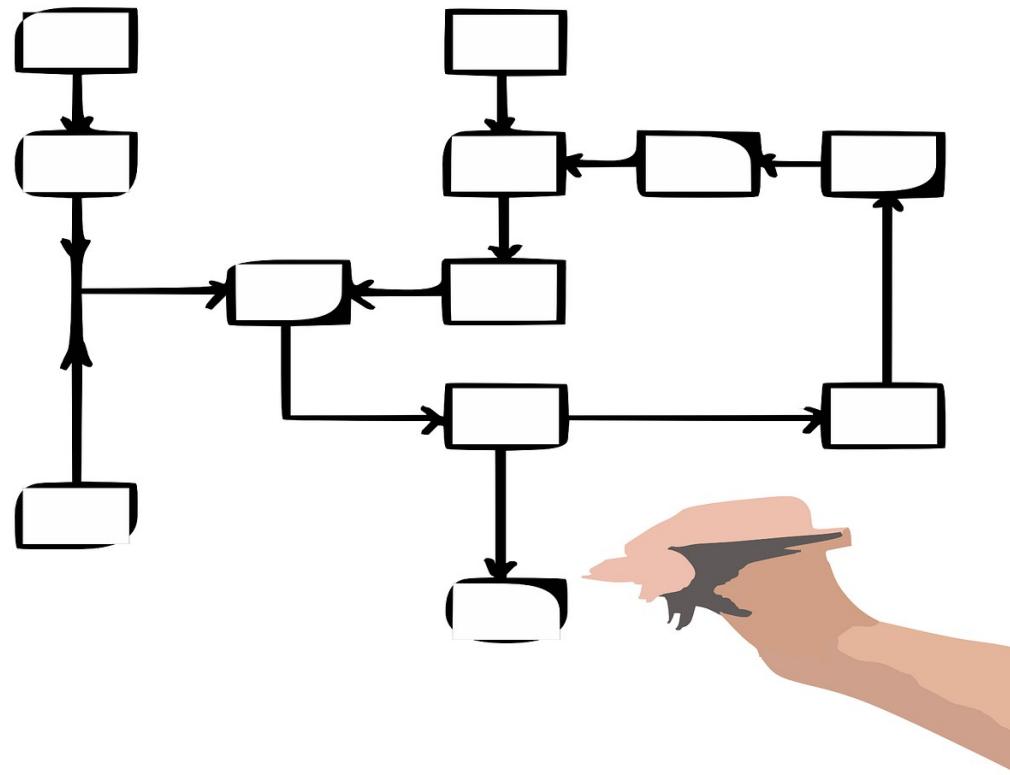
- **Lessico (vocabolario):** insieme di elementi per la descrizione di oggetti, operazioni e flusso di controllo
- **Sintassi:** insieme di regole di composizione degli elementi in frasi eseguibili e costrutti di controllo
- **Semantica:** insieme di regole per l'interpretazione degli elementi e delle istruzioni sintatticamente corrette



Rappresentazione del flusso di controllo

- schemi a blocchi (o diagrammi di flusso)
 - elementi grafici per indicare il flusso di controllo e i tipi di operazioni
 - elementi testuali per descrivere le operazioni e gli oggetti
- pseudo-codice
 - completamente testuale
 - costrutti di controllo descritti con la forma e le parole chiave dei linguaggi di programmazione
 - le operazioni possono essere descritte in modo informale

Schemi a blocchi



Schemi a blocchi: lessico

- Blocchi di inizio e fine:
- Blocco di esecuzione:
- Blocco di condizione:
- Blocco di ingresso dati:
- Blocco di uscita dati:

Inizio/Fine

Fai qualcosa



Leggi qualcosa

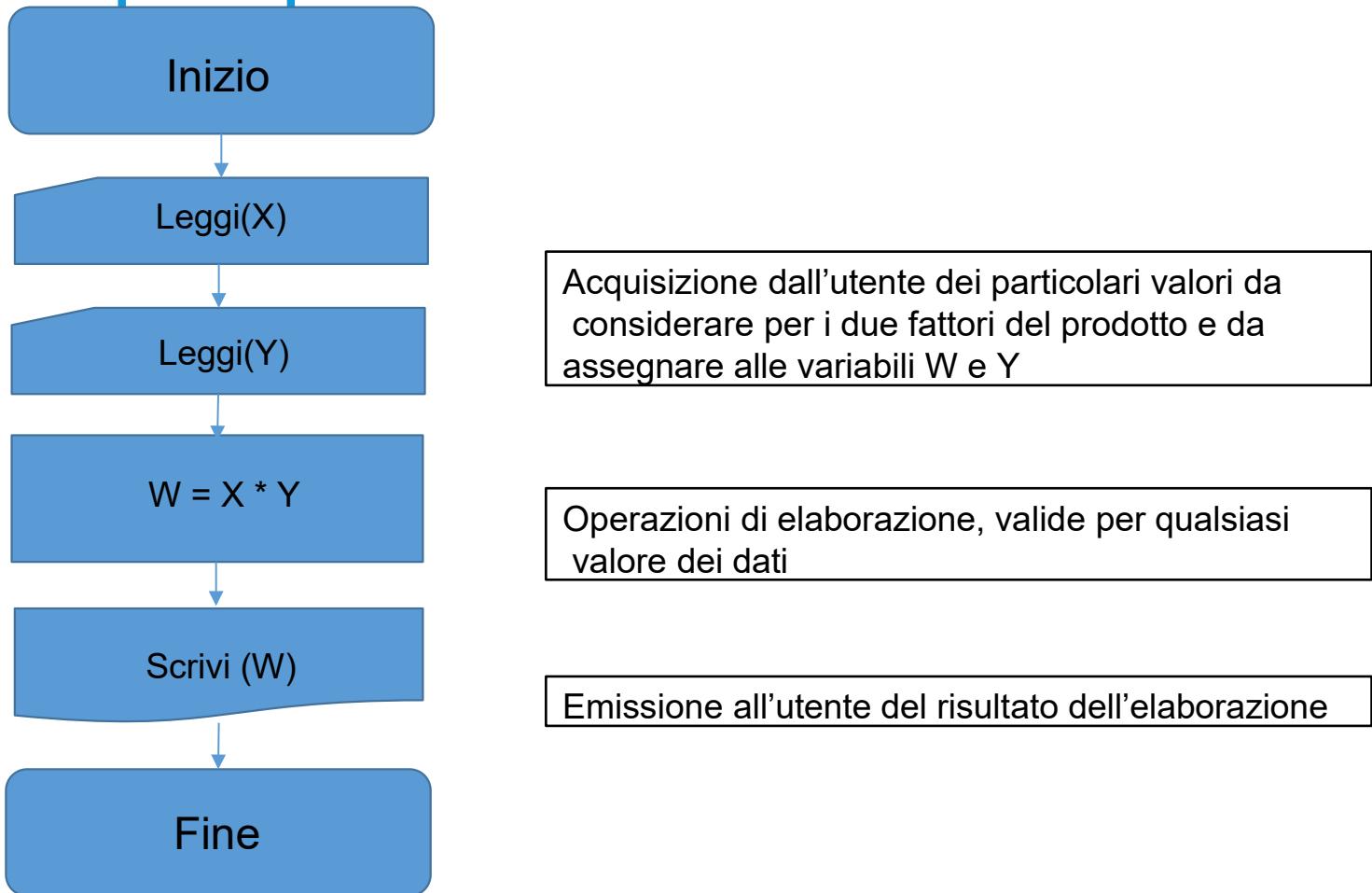
Scrivi qualcosa



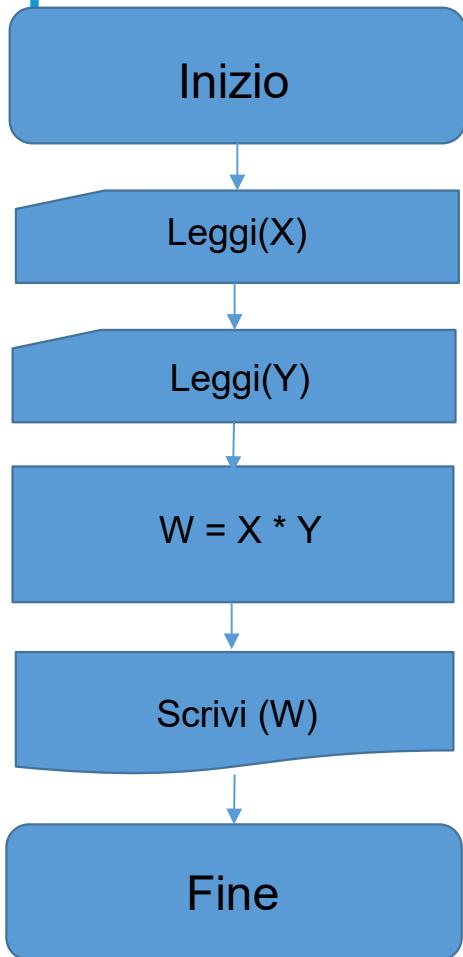
Sintassi: alcune regole di base

- Per il flusso di controllo
 - Un solo blocco di inizio
 - Almeno un blocco di terminazione
 - Dal blocco di inizio e da ogni blocco esecutivo deve uscire una sola freccia
 - Da ogni blocco condizionato devono uscire due frecce contrassegnate dalle indicazioni vero (si) e falso (no)
- Per il flusso di esecuzione
 - In tutti i casi, è unica la scelta del blocco successivo da eseguire

Esempio: prodotto di due numeri



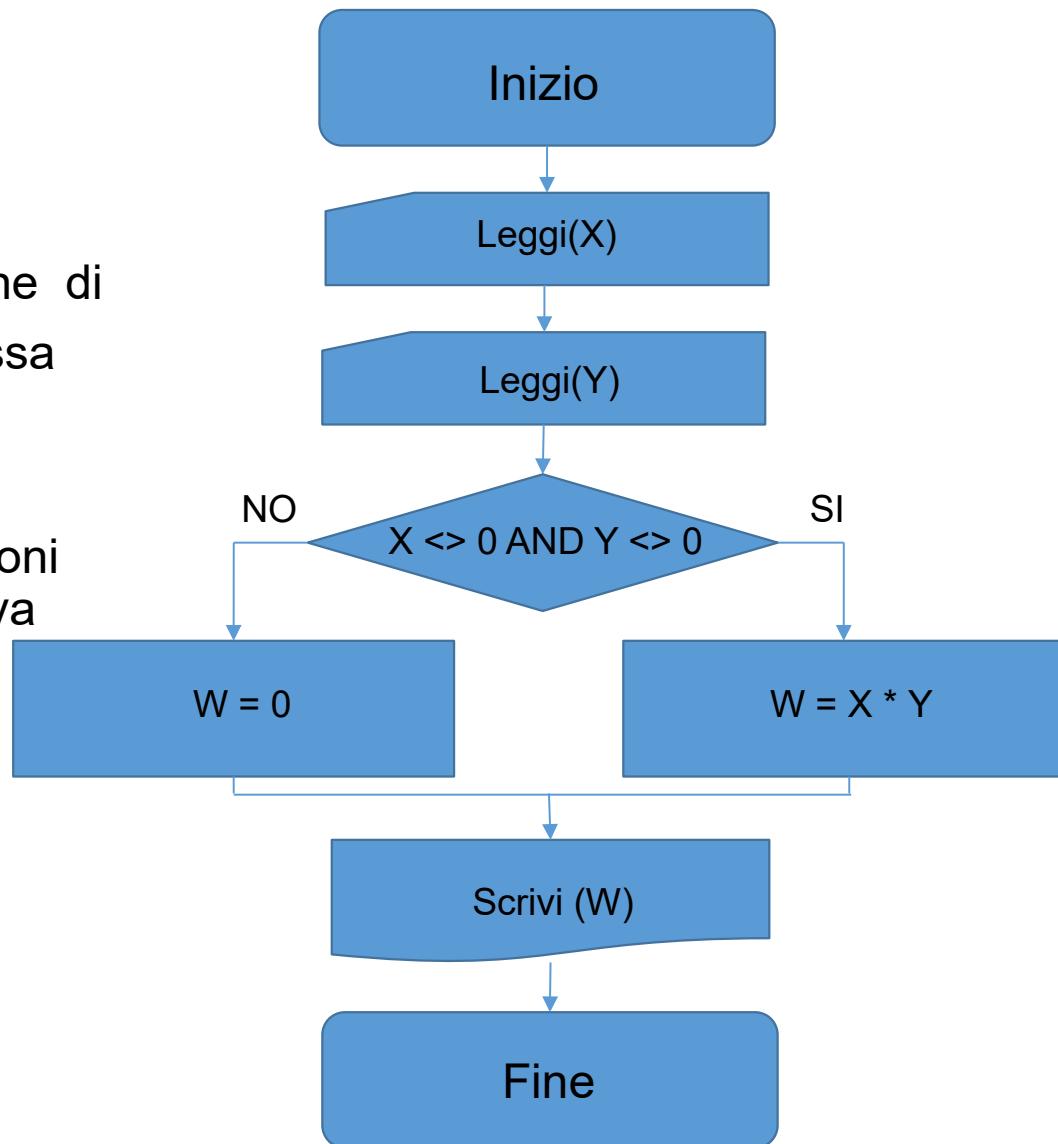
Sequenza



- Le frecce tra blocchi indicano una sequenza di operazioni
- Una operazione non inizia finché quella precedente non termina
- Non possono esserci due operazioni svolte contemporaneamente

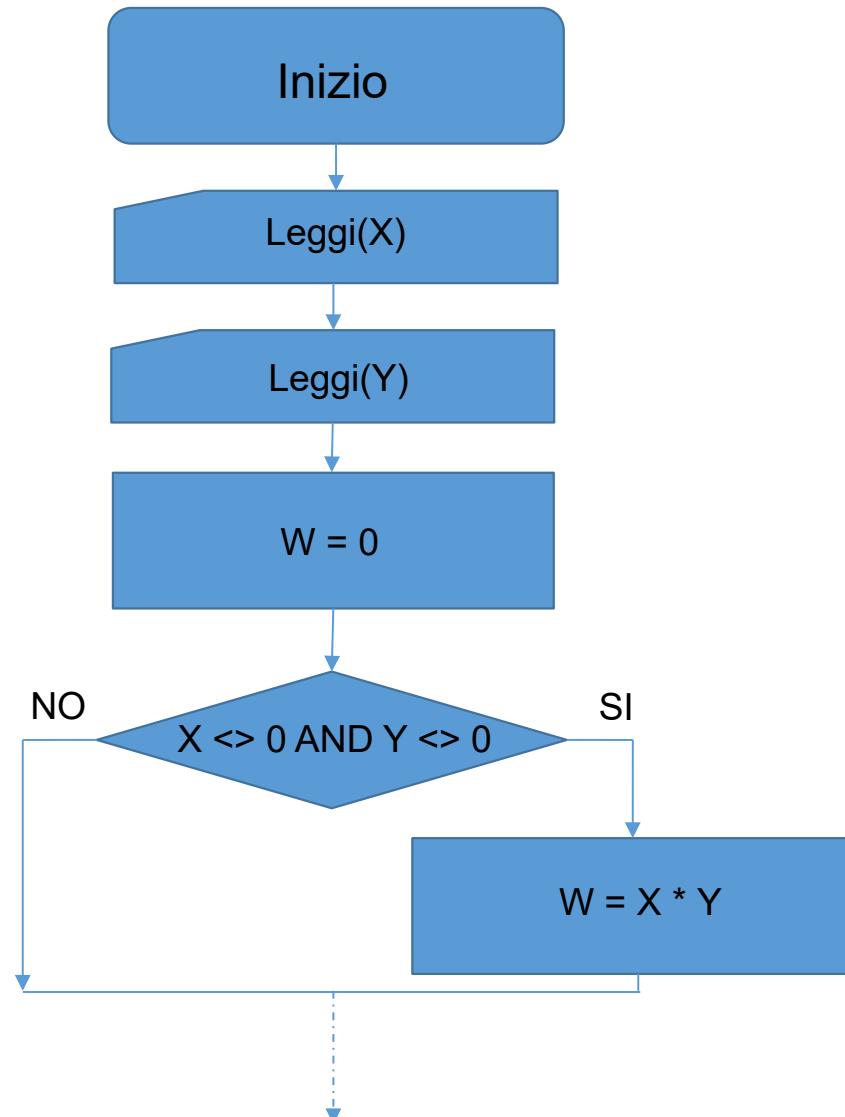
Selezione

- Si basa sulla valutazione di una condizione espressa all'interno del simbolo romboidale
- Due blocchi di operazioni sono svolti in alternativa

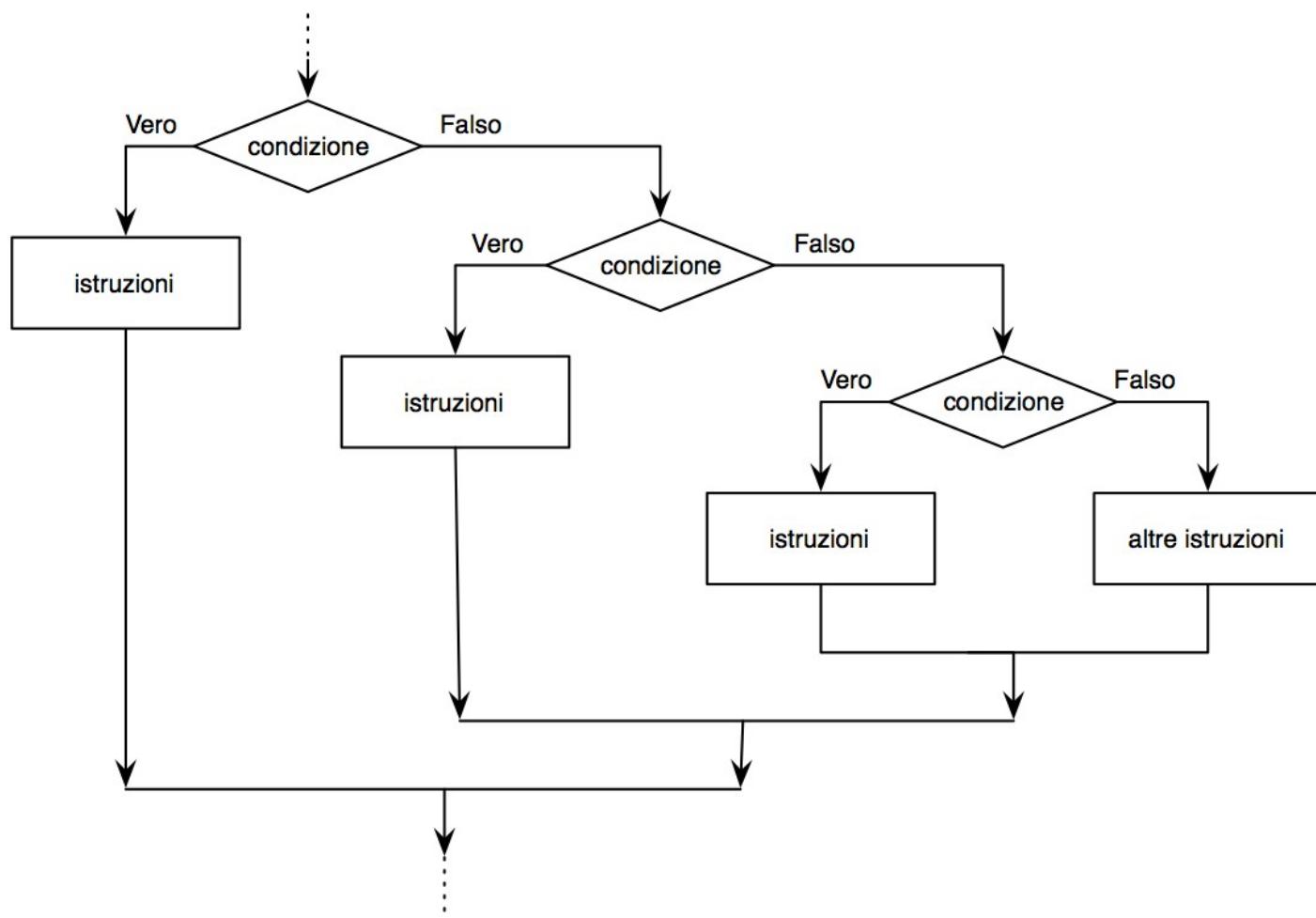


Selezione

- Il blocco sul ramo “si” deve essere sempre presente
- Il blocco sul ramo “no” è opzionale



Selezione nidificata



Iterazione

- Esprime la possibilità di ripetere un blocco di istruzioni un numero finito di volte
- Elementi:
 - condizione di permanenza: funzione della variabile di controllo
 - corpo del ciclo: contiene la modifica della variabile di controllo
- La ripetizione è controllata dalla valutazione della condizione di permanenza del ciclo
 - Se la condizione è vera viene ripetuto il ciclo
 - Se la condizione è falsa si procede con la prima operazione successiva al ciclo
- La condizione di permanenza può essere verificata
 - all'inizio dell'iterazione (pre-condizione)
 - alla fine dell'iterazione (post-condizione)

Post-condizione

Acquisizione dei dati in ingresso e attribuzione dei loro valori a X e Y

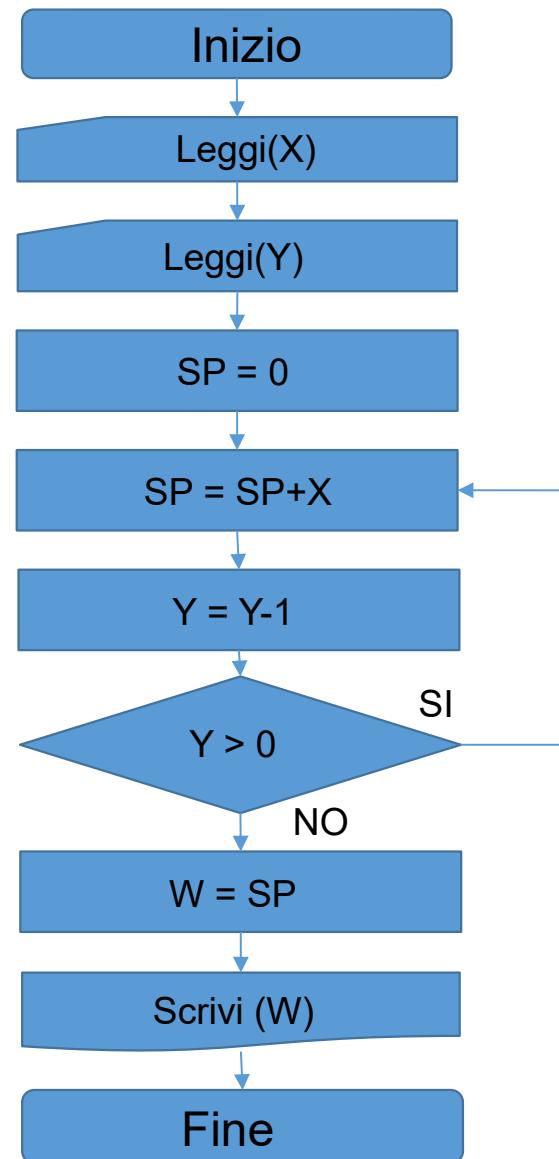
Inizializzazione delle variabili ausiliarie

Corpo del ciclo

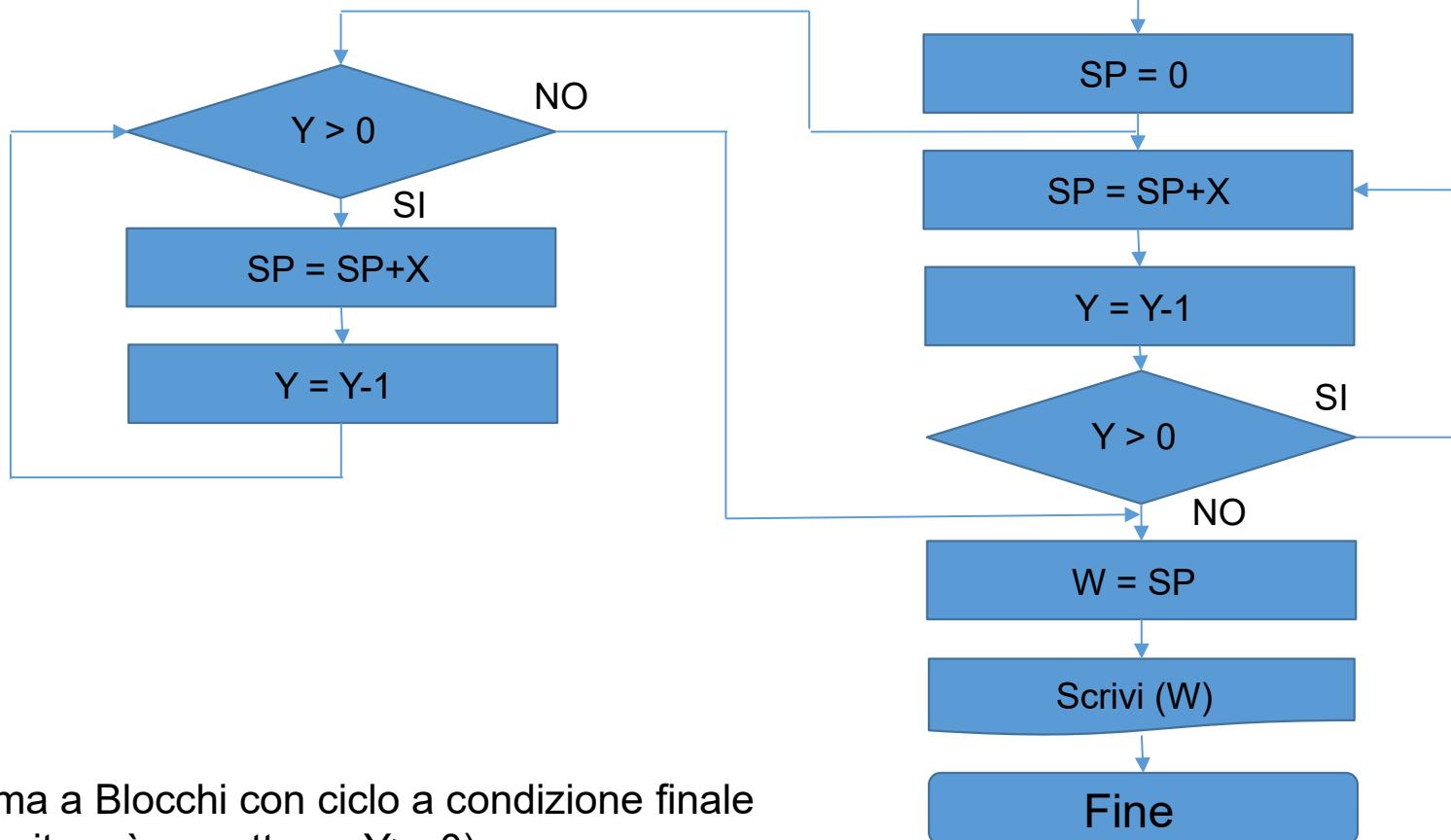
Valutazione della condizione di uscita dal ciclo

Emissione del risultato

Schema a Blocchi con ciclo a condizione finale
(l'algoritmo è corretto se $Y > 0$)



Pre-condizione



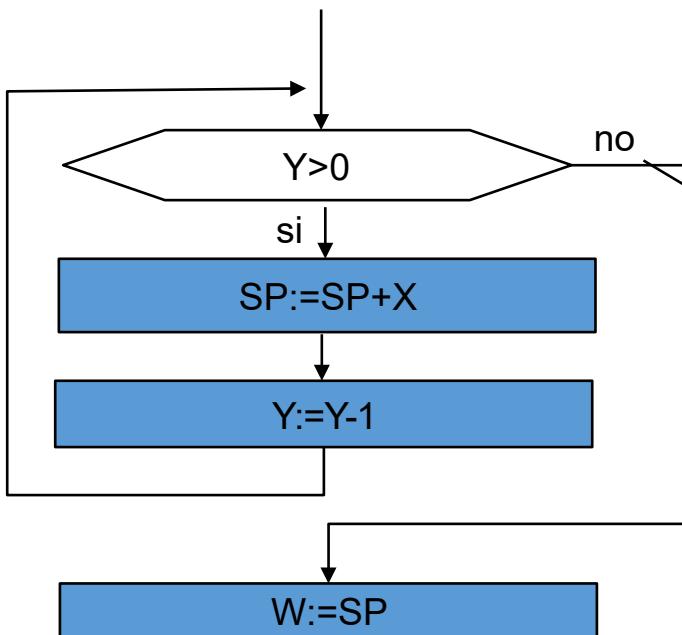
Schema a Blocchi con ciclo a condizione finale
(l'algoritmo è corretto se $Y \geq 0$)

Pre- e post-condizioni nei cicli

- Nell'esempio la verifica della condizione è stata spostata dalla fine all'inizio del ciclo
- Nell'esempio abbiamo migliorato l'algoritmo ma in realtà i due algoritmi NON sono funzionalmente equivalenti:
 - Il primo è corretto per $y > 0$
 - Il secondo è corretto per $y \geq 0$
- Se voglio due algoritmi equivalenti non basta quindi semplicemente spostare la condizione ma vanno verificati i casi limite

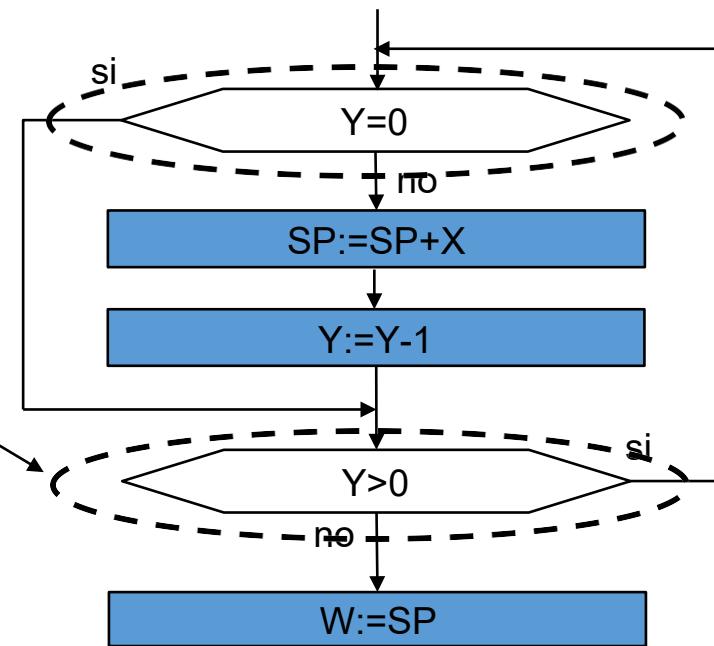
Pre- e post-condizioni nei cicli

Se volessi una versione con post-condizione funzionalmente equivalente (valida per $y \geq 0$) a quella con pre-condizione



Pre-condizione

Sposto la condizione alla fine del ciclo e inserisco una selezione all'inizio del ciclo per evitare la prima iterazione

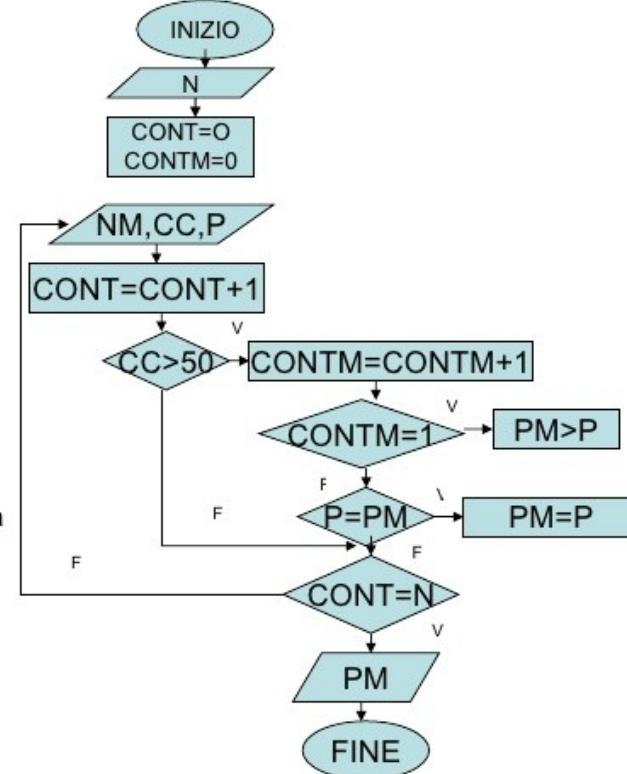


Post-condizione

Pseudo codice

PSEUDOCODICE & DIAGRAMMA A BLOCCHI

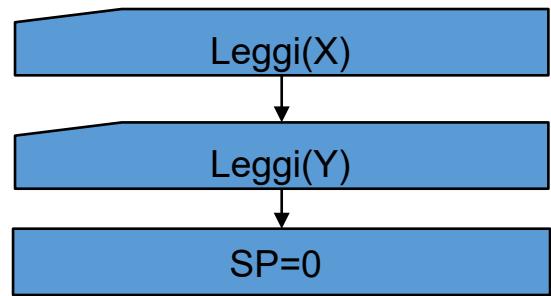
```
INIZIO
Leggi(N)
CONT=0
CONTM=0
RIPETI
    Leggi(NM;CC;P)
    CONT=CONT+1
    SE CC>50 allora
        CONTM=CONTM+1
    SE CONTM=1 allora
        PM=P
    ALTRIMENTI
        SE P>PM allora
            PM=P
        FINE SE
        FINE SE
    FINE SE
FINO A CHE CONT=N
SCRIVI (PM)
FINE
```



Pseudo codice

- Ha le stesse finalità dello schema a blocchi senza gli stessi simboli
- Viene adottato un linguaggio a metà tra il linguaggio naturale e quello di programmazione
- Utilizza gli stessi operatori ed espressioni visti per lo schema a blocchi
- Le strutture di controllo sono identificate da opportune parole chiave

Sequenza

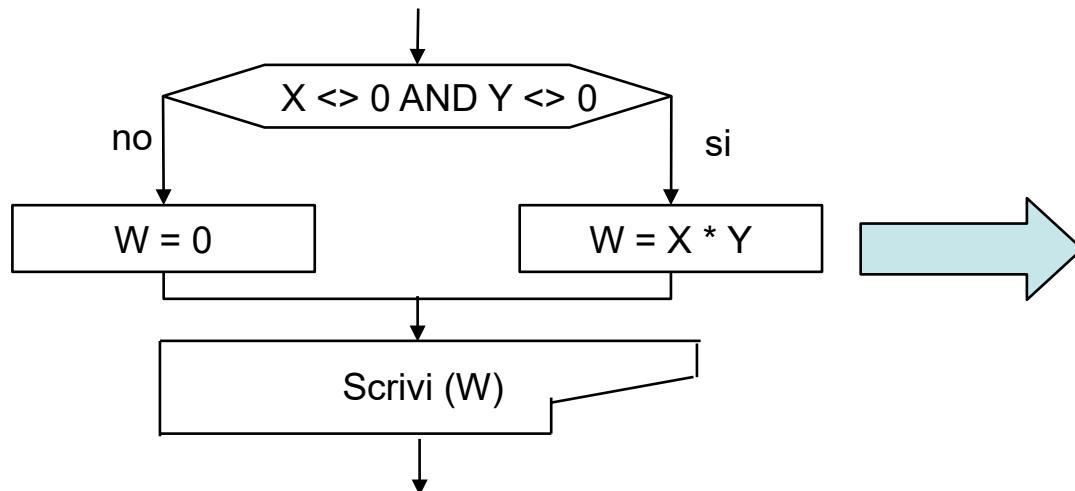


leggi (x)

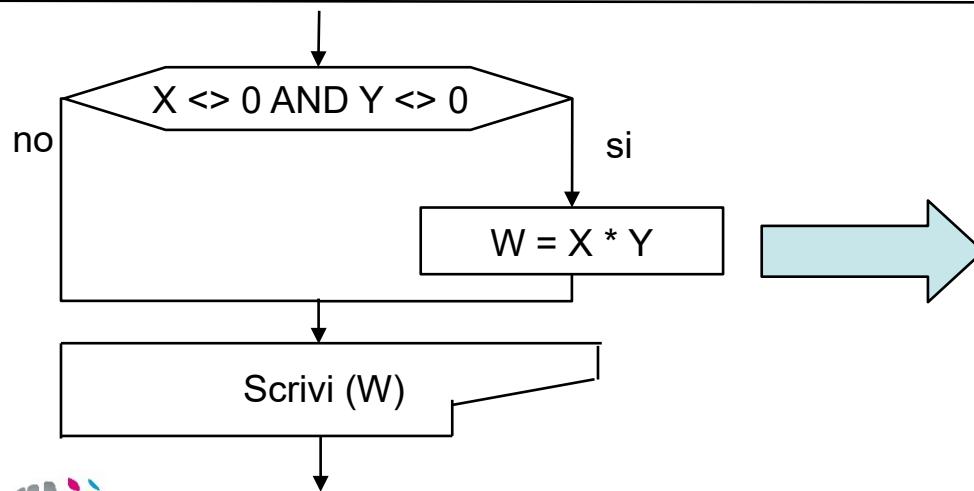
leggi (y)

sp = 0

Selezione if-then-else

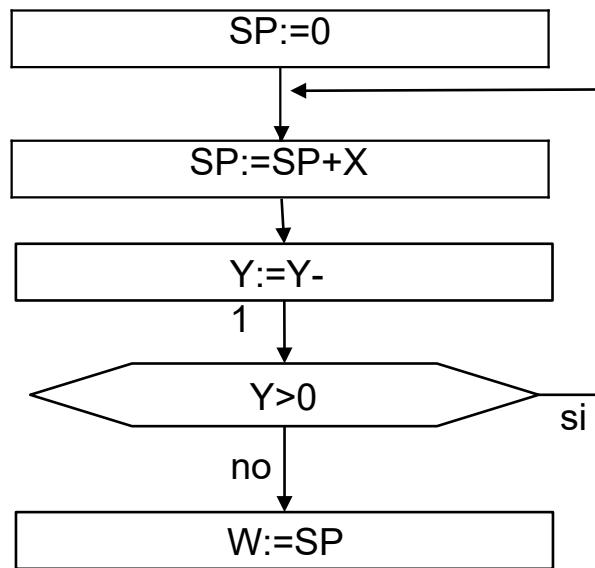


```
if (x<>0 AND y<>0) then  
    w := x * y  
else  
    w := 0  
end if  
scrivi (w)
```



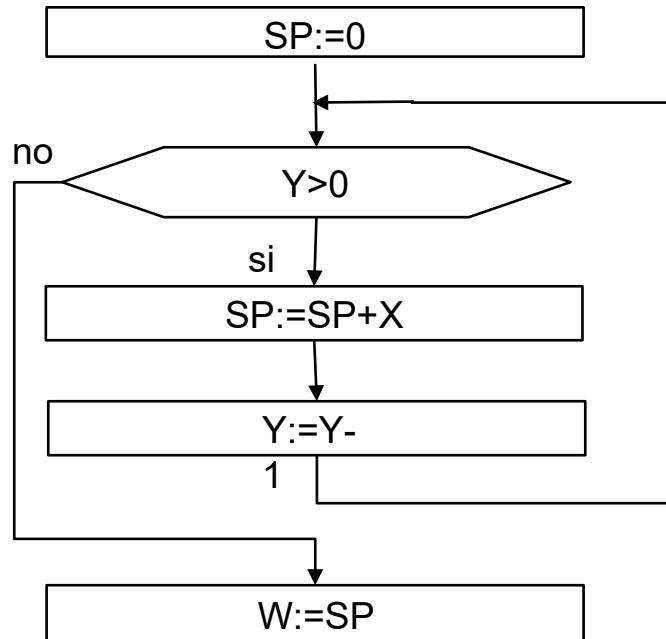
```
if (x<>0 AND y<>0) then  
    w := x * y  
end if  
scrivi (w)
```

Iterazione post-condizione do-while



```
sp:=0  
do  
    sp:=sp+x  
    y:=y-1  
while (y>0)  
w:=sp
```

Iterazione pre-condizione while-do



```
sp := 0  
while (y > 0)  
  
    sp := sp + x  
    y := y - 1  
do  
    w := sp
```

Flusso di esecuzione

- Il flusso di controllo:
 - definisce tutti i cammini possibili di esecuzione
 - si basa su ipotesi relative ai valori assumibili dalle variabili
- Il flusso di esecuzione:
 - definisce un particolare cammino di esecuzione
 - si basa su valori precisi date alle variabili ottenute anche dai dati in ingresso dall'utente
- Il tracing è il meccanismo che permette di controllare il funzionamento di un algoritmo analizzando il valore delle variabili durante l'esecuzione

Tracing

- In una griglia vanno messe tutte le variabili definite nell'algoritmo
- In ogni riga va indicato il valore della variabile per ogni operazione eseguita

Ipotesi di valore inserito dall'utente

Contenuto della variabile non definito

	X	Y	SP	W
leggi (x)	< 5 >	#	#	< # >
leggi (y)	5	2	#	#
sp:=0 (y>0)	5	2	0	#
whi \$p:=sp+x	5	2	5	#
y:=y-1	5	1	5	#
do	5	1	10	#
w:=sp	5	0	10	#
scriv i (w)	5	0	10	10

Strutturazione degli algoritmi

Come abbiamo visto dagli esempi precedenti un ciclo può essere realizzato da:

- Una sequenza
- Un costrutto di selezione binaria
- Una variabile contatore
- Una o più istruzioni di salto ad altre parti del programma

Programmazione strutturata

Tuttavia per realizzare i cicli la maggior parte dei linguaggi dispone appunto di costrutti specifici `for`, `while-do`, `do-while`, il cui uso costituisce uno dei fondamenti della **programmazione strutturata**.

```
100  GOTO 500
110  PRINT I;
120  GOTO 400
150  PRINT I * 12;
160  GOTO 450
200  PRINT " = ";
210  GOTO 150
300  PRINT " 12 ";
310  GOTO 200
400  PRINT " * ";
410  GOTO 300
450  I = I + 1
460  IF I > 12 THEN STOP
460  GOTO 110
500  PRINT "Tabella dei valori"
510  I = 1
520  GOTO 110
```

forma non strutturata	forma strutturata
1) leggi x, y 2) se $x < y$ scambiali 3) dividi x per y , chiama r il resto 4) poni $x = y$ 5) poni $y = r$ 6) se $y \neq 0$ vai a 3) 7) stampa x 8) fine	leggi x, y se $x < y$ scambiali ripeti dividi x per y , chiama r il resto poni $x = y$ poni $y = r$ fino a che $y = 0$ stampa x fine

Iterazione e ricorsività

Nell'informatica in concetto di **iterazione** si incontra in molte forme tra le quali i modelli dei dati, dove molti concetti, come le **liste**, sono definiti in modo ripetitivo.

Ad esempio una lista si può definire: « *una lista è vuota oppure è un elemento seguito da un altro elemento* »

Alternativa all'iterazione è la **ricorsività**, una tecnica per cui il concetto viene definito, direttamente o indirettamente, in termini di se stesso.

Ad esempio una lista si può definire: « *una lista vuota oppure un elemento seguito da una lista* »

Esempio di ricorsività - fattoriale

base: $1!$

induzione: $n! = n * (n-1)!$

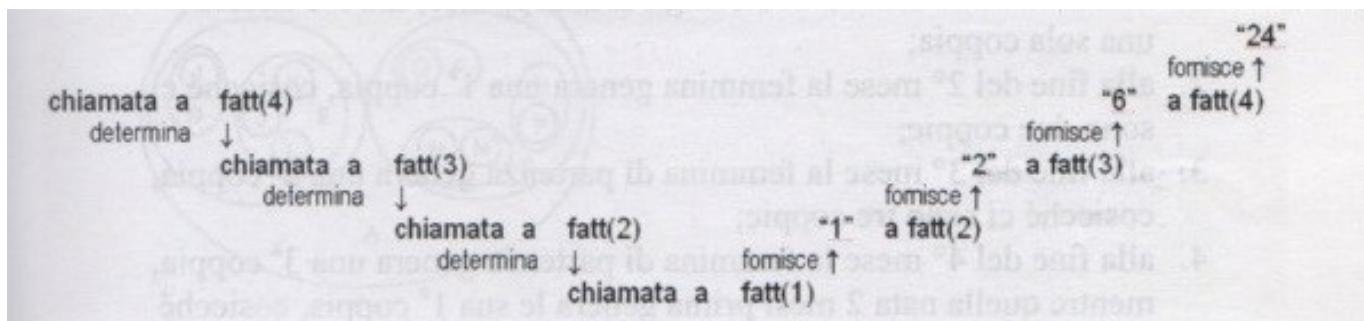
Da cui per esempio:

$$5! = 5 * (5-1)! = 5 * 4!$$

$$4! = 4 * (4-1)! = 4 * 3!$$

ecc...

```
....  
if (n==0)  
    return (1);  
return (n*fatt(n-1));
```



I linguaggi di programmazione



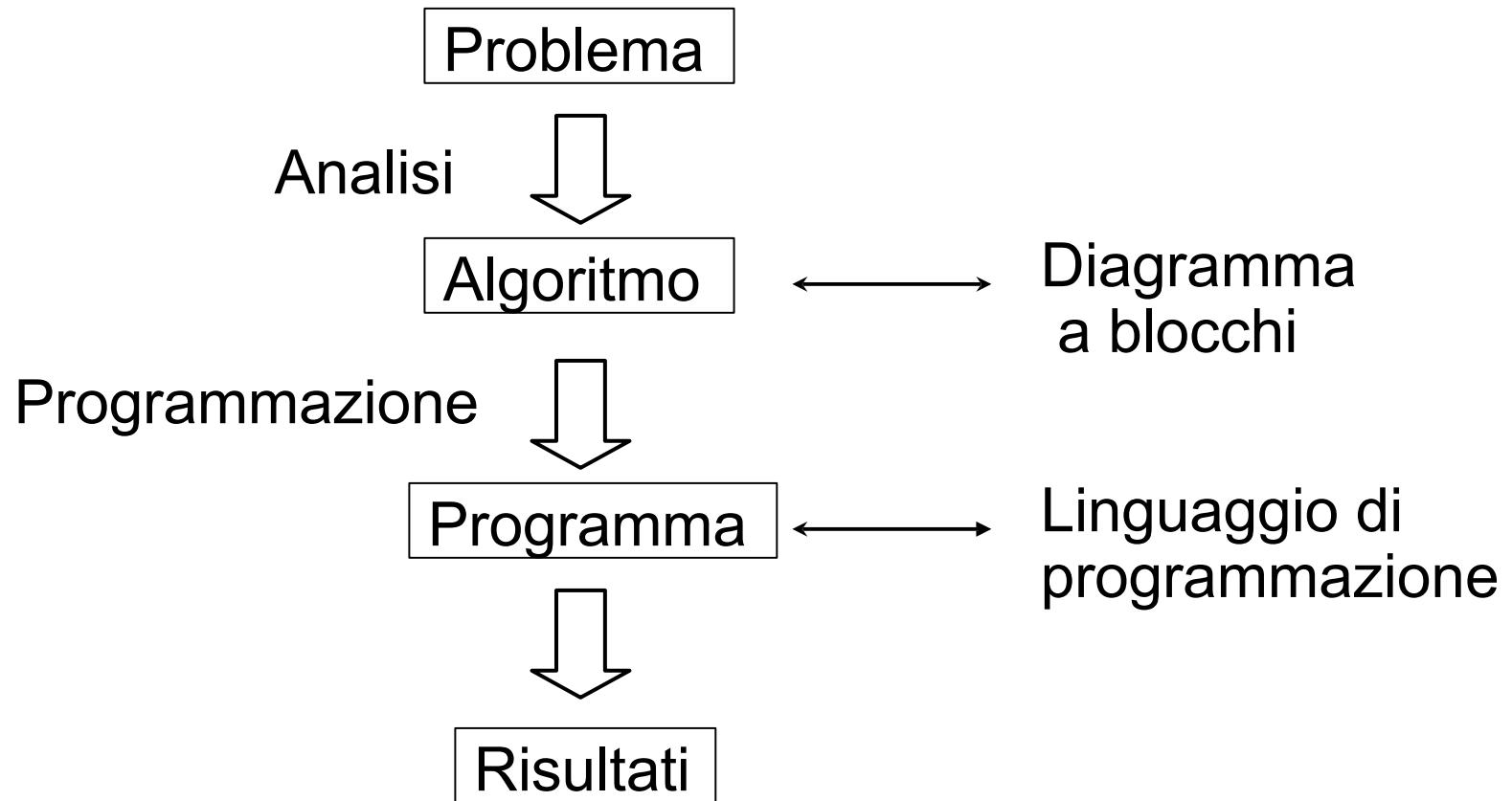
Linguaggio naturale e macchina

- La comunicazione uomo-macchina avviene attraverso formalismi che assumono la forma di un linguaggio.
- Caratteristiche del **Linguaggio Naturale**:
 - Vantaggi:
 - Ricchezza espressiva
 - Svantaggi:
 - Ambiguità
 - Ridondanza
- Caratteristiche del **Linguaggio Macchina (e Assembler)**:
 - Vantaggi:
 - Legato alla struttura fisica del Calcolatore (CPU)
 - Potente e veloce
 - Svantaggi:
 - Programmi lunghi e di difficile scrittura
 - Difficoltà di messa a punto dei programmi

Linguaggio di programmazione

- **Programma:**
 - formulazione di un algoritmo nei termini di un linguaggio di programmazione
- **Linguaggio di Programmazione:**
 - Linguaggio intermedio fra il linguaggio macchina e il linguaggio naturale
 - Descrive gli algoritmi con una ricchezza espressiva comparabile con quella dei linguaggi naturali, ma non è ambiguo
 - Descrive gli algoritmi in modo rigoroso

Uso dei linguaggi di programmazione

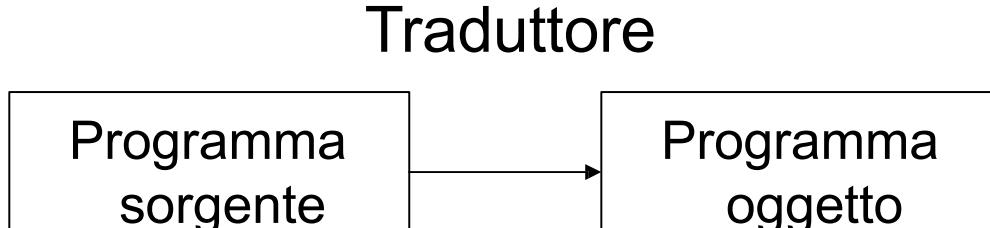


Traduttori

- Il linguaggio macchina è l'unico linguaggio compreso dalla CPU
- Qualsiasi altro linguaggio di programmazione ha bisogno di un traduttore (non è possibile progettare un traduttore per i linguaggi naturali, per via dell'ambiguità)
- I linguaggi di programmazione sono comprensibili sia dalla macchina (attraverso un traduttore) che dall'uomo

Traduttori

- **Programma sorgente:**
 - Programma espresso in linguaggio di programmazione
- **Programma oggetto:**
 - Programma espresso in linguaggio macchina
- **Traduttore:**
 - Programma che traduce un programma sorgente in un programma oggetto
 - E' funzione del linguaggio di programmazione e dell'architettura dell'elaboratore (CPU)



Programma sorgente e
Programma oggetto
sono equivalenti (stessi
risultati sugli stessi dati)

Tipi di traduttori: assemblatori, compilatori, interpreti

Linguaggi ad *alto
livello* (orientati all'uomo)

Linguaggi a *basso
livello* (linguaggi Assembler,
orientati alla macchina)

Compilatori
o Interpreti

Assemblatori

Linguaggio macchina

Linguaggi ad alto livello

- E' svincolato dall'architettura della macchina, nel senso che il programmatore non deve necessariamente conoscere le caratteristiche specifiche della CPU
- E' più comprensibile al programmatore in quanto più vicino al linguaggio naturale
- Per CPU diverse (ad es. Intel invece che IBM), è necessario predisporre compilatori diversi
- Uno stesso Programma sorgente può essere eseguito su macchine (CPU) diverse, creando il Programma oggetto con il compilatore opportuno
- Scelta di un Linguaggio in base ad applicazioni specifiche

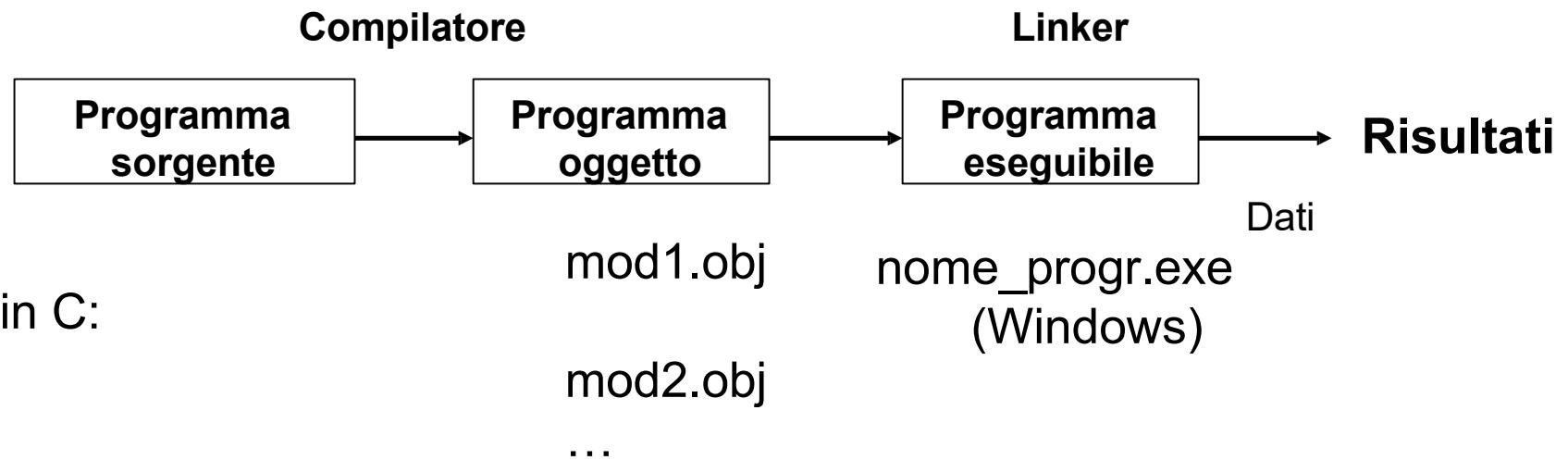
Traduttori specifiche

- A ciascuna istruzione del **Linguaggio Assembler** corrisponde una sola istruzione in Linguaggio macchina (corrispondenza biunivoca);
- In generale, ad un'istruzione in Linguaggio ad alto livello corrispondono una o più istruzioni in Linguaggio macchina.
- Mentre la traduzione di un Programma in Assembler è un'operazione abbastanza banale, la traduzione di un Programma scritto il Linguaggio di alto livello è un'operazione più ardua, e non univoca.
- I **Compilatori** (e **Interpreti**) sono programmi molto più sofisticati degli Assemblatori, e sono determinanti ai fini dell'efficienza di un Programma.

Schema di compilazione

Compilatore:

- E' un programma che riceve un intero programma sorgente (file) in Input e restituisce in Output il programma oggetto.



Es in C:

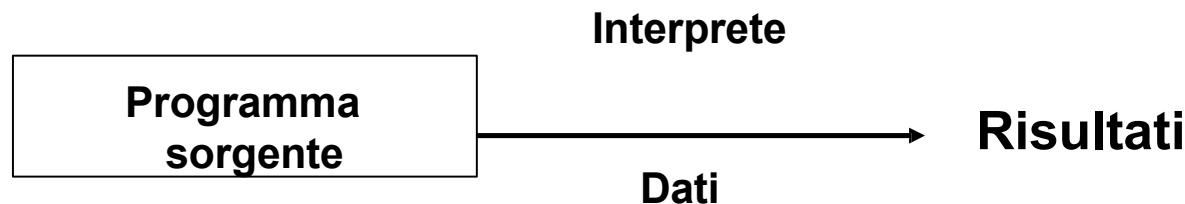
Schema di compilazione

Linker:

- Collega più moduli oggetto prodotti dal compilatore in un unico programma eseguibile
- Risolve i riferimenti esterni ad ogni modulo (es. librerie)

Schema di interpretazione

- **Interprete:**
 - Legge una singola frase in linguaggio sorgente, la trasforma in una sequenza di istruzioni macchina e le manda in esecuzione;
 - Traduzione e esecuzione sono contestuali.



Confronto tra compilatori ed interpreti

- Compilatori:
 - Sono in grado di *ottimizzare* il Codice;
 - Programmi più veloci da eseguire;
 - Esempi di Linguaggi: COBOL, Pascal, Fortran, C.
- Interpreti:
 - Programmi più lenti (traduzione ed esecuzione sono contestuali);
 - Messa a punto del programma (*debugging*) migliore (si conosce immediatamente la riga di programma dove si verifica un errore);
 - Esempi: VB script, Matlab, Prolog.

Modello ibrido

- Il Programma sorgente viene *compilato* in un linguaggio intermedio (es: bytecode) una volta per tutte.
- Il linguaggio intermedio viene *interpretato* da una Virtual Machine (VM) ossia un interprete di basso livello che emula una CPU.
- La portabilità su una CPU diversa è garantita dall'esistenza della VM per quella specifica CPU.
- Esempi: Java, C#, VB .NET, Python.

Paradigmi di programmazione

Esistono numerose famiglie di linguaggi di programmazione, riconducibili a diversi *paradigmi di programmazione*.

Ciascun paradigma fornisce al programmatore strumenti concettuali di diversa natura per descrivere gli algoritmi da far eseguire al calcolatore.

- Paradigma procedurale o imperativo
- Paradigma funzionale
- Paradigma logico
- Paradigma concorrente
- Paradigma ad oggetti

Paradigma procedurale o imperativo

La programmazione imperativa è un paradigma di programmazione secondo cui un programma viene inteso come un insieme di istruzioni (dette anche direttive, comandi), ciascuna delle quali può essere pensata come un “ordine” che viene impartito alla *macchina virtuale* definita dal linguaggio di programmazione utilizzato.

Da un punto di vista sintattico, i costrutti di un linguaggio imperativo sono spesso identificati da verbi all'imperativo, per esempio: print (stampa), read (leggi), do (fà).

Paradigma funzionale

A livello puramente astratto un programma (o, equivalentemente, un algoritmo) è una funzione che, dato un certo input, restituisce un certo output (dove per output si può anche intendere la situazione in cui il programma non termini la propria esecuzione o restituisca un errore).

La programmazione funzionale consiste esattamente nell'applicazione di questo punto di vista. Ogni operazione sui dati in input viene effettuata attraverso particolari funzioni elementari, appropriatamente definite dal programmatore, che opportunamente combinate, attraverso il concetto matematico di composizione di funzioni, danno vita al programma.

Paradigma logico

Il Prolog è stato il primo rappresentante di questa classe. Nati a partire da un progetto per un dimostratore automatico di teoremi, i linguaggi logici, o dichiarativi, rappresentano un modo completamente nuovo di concepire l'elaborazione dei dati: invece di una lista di comandi, un programma in un linguaggio dichiarativo è una lista di regole che descrivono le proprietà dei dati e i modi in cui questi possono trasformarsi.

Non esistono né cicli, né salti, né un ordine rigoroso di esecuzione, affinché sia possibile usarli in un programma dichiarativo, tutti i normali algoritmi devono essere riformulati in termini ricorsivi

Paradigma concorrente

Ormai tutti i calcolatori di fascia alta e media sono equipaggiati con più di una CPU. Come ovvia conseguenza, questo richiede la capacità di sfruttarle; per questo sono stati sviluppati dapprima il multithreading, cioè la capacità di lanciare più parti dello stesso programma contemporaneamente su CPU diverse, e in seguito alcuni linguaggi studiati in modo tale da poter individuare da soli, in fase di compilazione, le parti di codice da lanciare in parallelo.

Paradigma ad oggetti

La programmazione orientata agli oggetti (OOP, Object Oriented Programming) è un paradigma di programmazione, che prevede di raggruppare in un'unica entità (la classe) sia le strutture dati che le procedure che operano su di esse, creando per l'appunto un “oggetto” software dotato di proprietà (dati) e metodi (procedure) che operano sui dati dell'oggetto stesso.

La modularizzazione di un programma viene realizzata progettando e realizzando il codice sotto forma di classi che interagiscono tra di loro.

Linguaggi di scripting

I linguaggi di questo tipo nacquero come **linguaggi batch**: vale a dire liste di comandi di programmi interattivi che invece di venire digitati uno ad uno su una linea di comando, potevano essere salvati in un file, che diventava così una specie di comando composto che si poteva eseguire in modalità batch per automatizzare compiti lunghi e ripetitivi.

I primi linguaggi di scripting sono stati quelli delle shell Unix.

Linguaggi di scripting

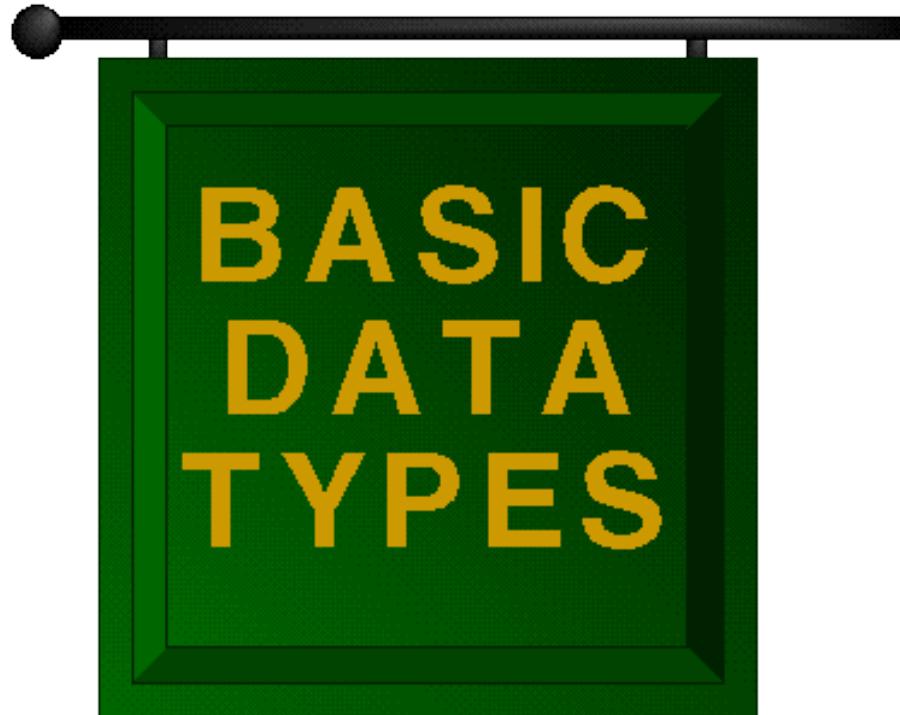
Il passo successivo fu quello di far accettare a questi programmi anche dei comandi di salto condizionato e delle istruzioni di ciclo, regolati da simboli associati ad un certo valore: in pratica implementare cioè l'uso di variabili.

Ormai molti programmi nati per tutt'altro scopo offrono agli utenti la possibilità di programmarli in modo autonomo tramite linguaggi di scripting più o meno proprietari.

Molti di questi linguaggi hanno finito per adottare una sintassi molto simile a quella del C: altri invece, come il Perl e il **Python**, sono stati sviluppati ex novo allo scopo.

SONO TUTTI LINGUAGGI INTERPERATI o IBRIDI

Tipi di dato



Tipi di dato e variabili e costanti

In matematica si usa classificare le variabili rispetto ad alcune caratteristiche importanti, distinguendo tra variabili reali, complesse e logiche, oppure variabili che rappresentano valori individuali, insiemi di valori o insiemi di insiemi.

Questa classificazione è ugualmente importante all'elaborazione del dato stesso e vale il seguente principio:

tutte le costanti, la variabili, le espressioni e le funzioni appartengono ad un certo tipo

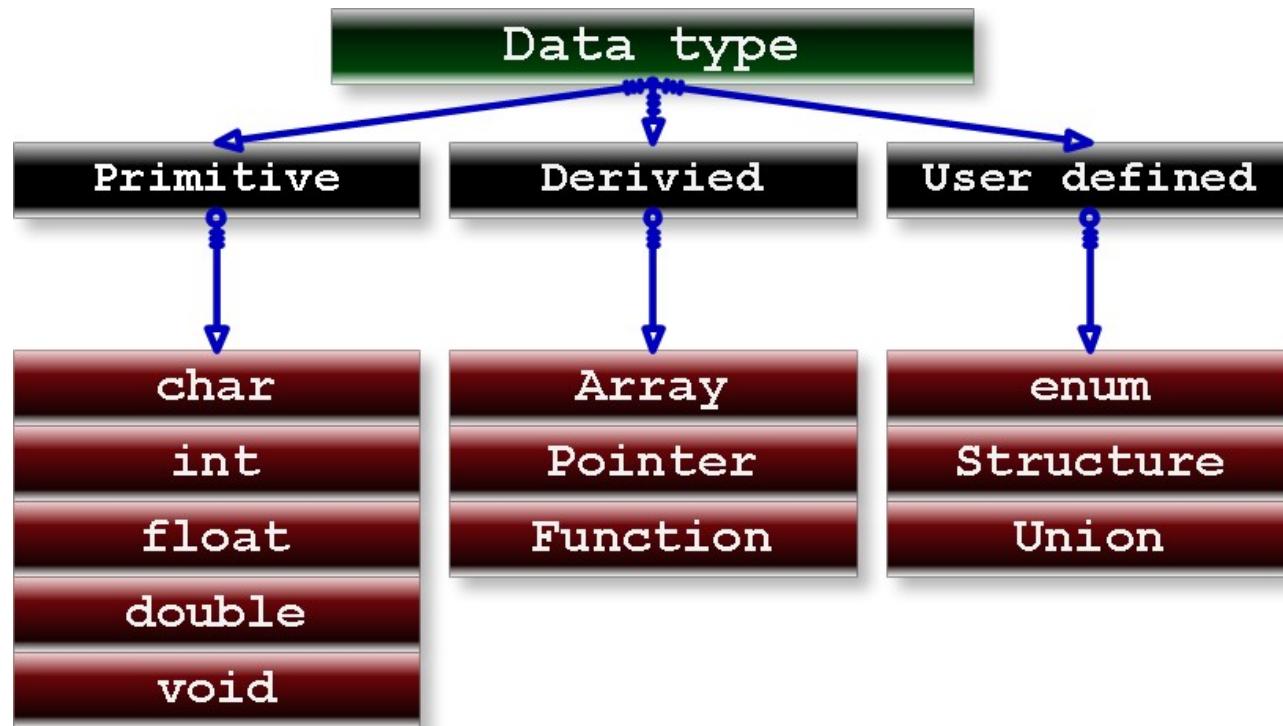
Dichiarazione

Nei testi di matematica il **tipo** di una variabile può essere dedotta normalmente dai caratteri tipografici, indipendentemente dal contesto. Ciò non è possibile nei programmi, perciò il tipo associato ad una variabile, ad una costante o ad una funzione, viene reso esplicito da una **dichiarazione**.

Questo appare ragionevole anche pensando al compilatore che deve essere in grado di **allocare** una quantità di memoria considerando l'ampiezza dei valori che la variabile potrà assumere.

Quando questa informazione è nota al compilatore esso riserva una quantità di memoria necessaria (**allocazione statica**), se invece questa informazione non è nota in fase di compilazione esso riserverà uno spazio di memoria dinamico (**allocazione dinamica**).

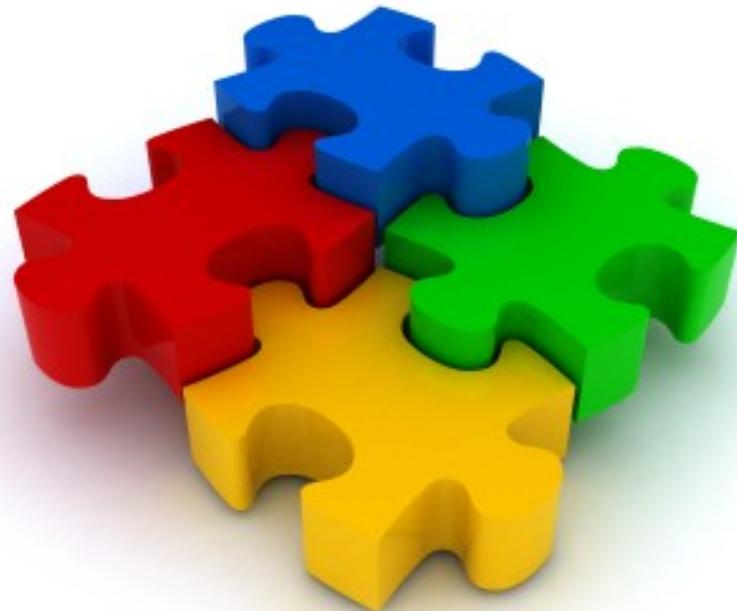
Tipi dato comuni



Tipi dato comuni

Data Type	Range	Bytes	Format
signed char	-128 to + 127	1	%c
unsigned char	0 to 255	1	%c
short signed int	-32768 to +32767	2	%d
short unsigned int	0 to 65535	2	%u
signed int	-32768 to +32767	2	%d
unsigned int	0 to 65535	2	%u
long signed int	-2147483648 to +2147483647	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4e38 to +3.4e38	4	%f
double	-1.7e308 to +1.7e308	8	%lf
long double	-1.7e4932 to +1.7e4932	10	%Lf
Note: The sizes and ranges of int, short and long are compiler dependent. Sizes in this figure are for 16-bit compiler.			

La modularità



La modularità - definizione

Un programma ben progettato è costruito usando un approccio simile a quello usato per costruire un edificio, anche nel caso di un programma una parte fondamentale del progetto è costituita dalla **struttura**.

Nella programmazione questo termine ha due significati collegati: la costruzione globale del programma e la forma usata per eseguire i singoli compiti all'interno del programma.

I programmi la cui struttura consiste in **segmenti correlati** tra loro e disposti in un ordine logico e facilmente comprensibile a formare una unità integrata e completa, sono detti **modulari**.

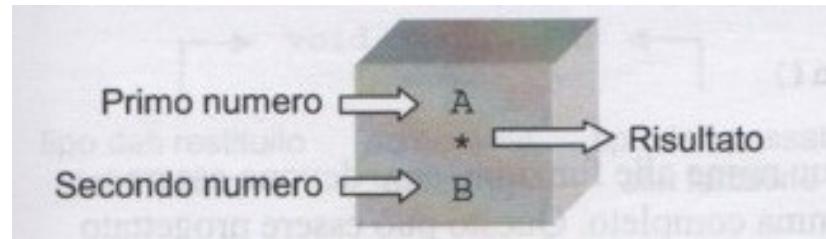
Moduli

I più piccoli componenti usati per costruire un programma modulare sono detti **moduli**.

Ogni modulo è progettato e sviluppato per svolgere un compito specifico, ed è in realtà un sotto-programma.

Il vantaggio della costruzione modulare è che il progetto complessivo può essere sviluppato prima di scrivere qualsiasi modulo.

Essendo il modulo un sotto-programma, riceve in ingresso dei dati e produce un risultato in uscita viene anche detto **funzione**.



Dichiarazione di funzione

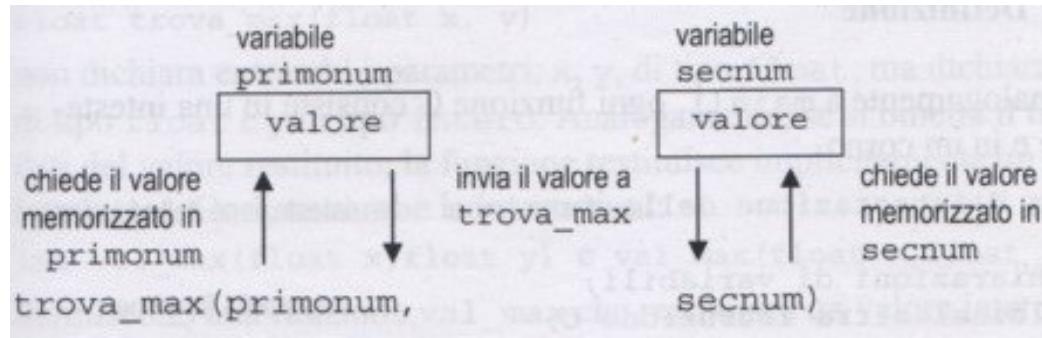
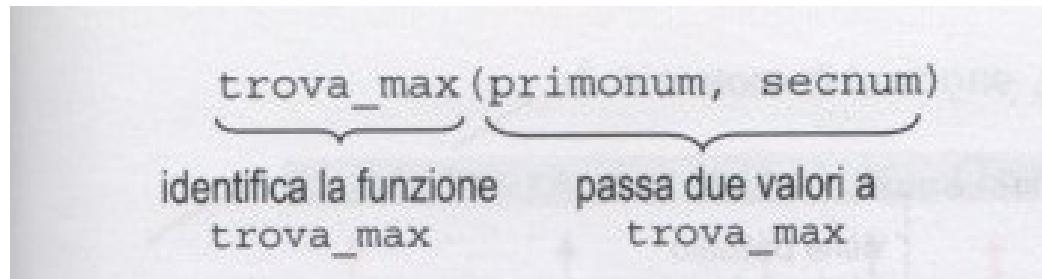
Prima di essere chiamata (utilizzata) una funzione deve essere dichiarata attraverso un'istruzione detta **prototipo di funzione**.

La forma generale del prototipo di funzione è:

Tipo-dati-restituito **nome-funzione** (lista tipi dati argomenti)



Chiamata di una funzione

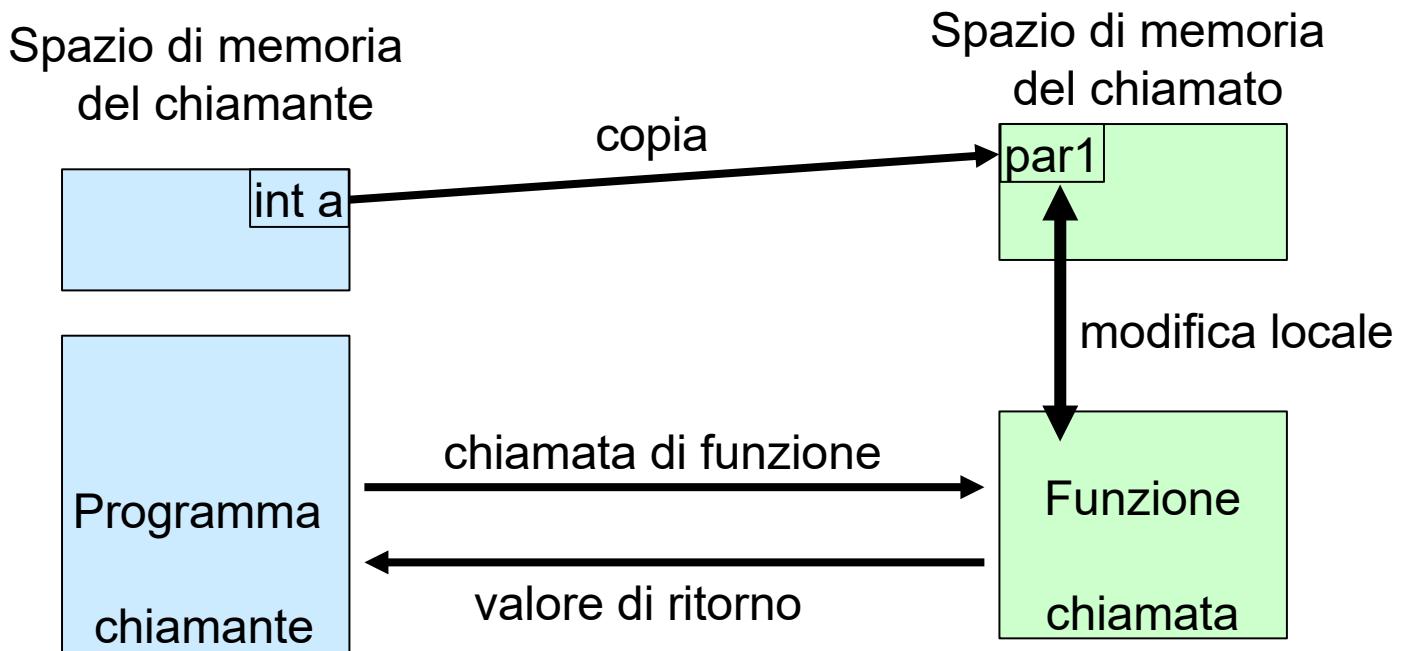


Passaggio parametri

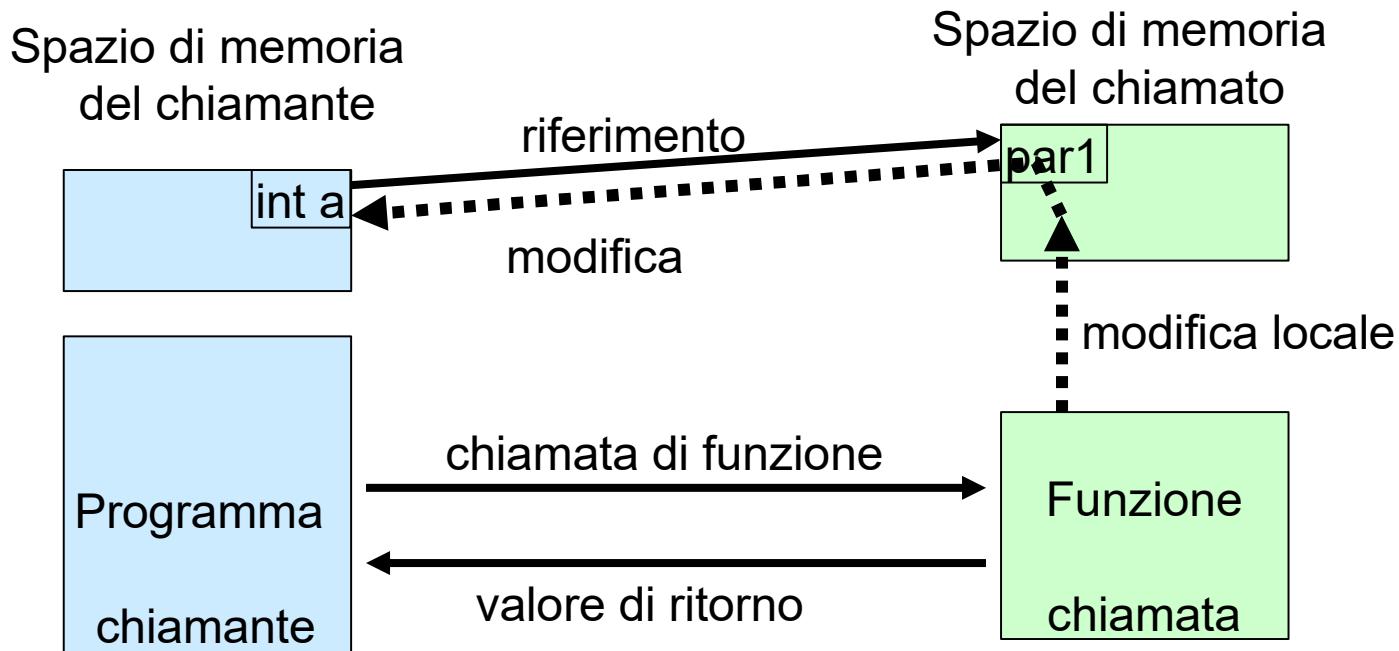
Il passaggio dei parametri ad una funzione può avvenire in due modalità distinte:

- **Passaggio per valore**
- **Passaggio per riferimento (indirizzo)**

Passaggio per valore



Passaggio per riferimento



Esempio di utilizzo

```
1 int a = 10, b = 5, c;
2
3 int product(int x, int y);
4
5 int main(void)
6 {
7     c = product(a,b);
8
9     printf("%i\n",c);
10
11    return 0;
12 }
13
14 int product(int x, int y)
15 {
16     return (x * y);
17 }
```

Function Prototype

Main Function

Function call

Function Definition

- int is the return type and int x and int y are the function arguments

- int is always the return type and there are no arguments, hence the (void). Curly braces {} mark the start and end of the main function

- product(a,b); a and b are global variables the function is passed. Here the values returned by the function are assigned to the variable c

- contains the function statement return(x * y); the function returns x times y to the main function where it was called. Curly braces {} mark the start and end of the function

Programmazione ad oggetti



Object Oriented Programming(OOP)

I linguaggi di programmazione procedurali sono stati quasi tutti sostituiti dalle rispettive versioni ad oggetti (tranne nei casi in cui la programmazione ad oggetti non è “consigliata”, ad esempio programmazione di basso livello, S.O., programmazione PLC ecc...)

Ma cos’è questo diverso tipo di programmazione? In cosa differisce dalla programmazione “classica”?

Object Oriented Programming(OOP)

La maggior parte dei programmi utili non manipola soltanto numeri e stringhe, ma gestisce dati più complessi e più aderenti alla realtà:

- Conti bancari, informazioni sugli impiegati, forme grafiche...

L'idea della programmazione ad oggetti è di rappresentare questi dati così come sono anche nel linguaggio di programmazione:

- gli oggetti o classi

La programmazione ad oggetti estende al massimo l'idea di dato strutturato presente nella programmazione classica:

- una classe (o oggetto) è un tipo di dato strutturato a cui sono associate variabili e funzioni

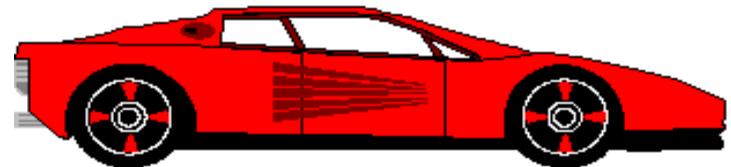
Object Oriented Programming(OOP)

- Il programmatore individua degli oggetti nella applicazione che deve progettare
 - Conto corrente, client e server, documento, ecc...
- Ogni oggetto ha delle proprietà e delle capacità (o funzioni)
- L'applicazione è poi costruita facendo interagire correttamente gli oggetti esistenti nell' “ambiente” che si va a progettare

Un esempio di oggetto

- Un automobile può essere caratterizzata in questo modo:

attributi	funzionalità
potenza	frena
marce	accelera
peso	cambio marcia
cilindrata	cambia direzione



- È possibile interagire con l'automobile, per determinarne il suo comportamento attraverso il suo interfaccia che permette di effettuare le operazioni consentite:
 - Pedale del freno
 - Pedale dell'acceleratore
 - Leva del cambio
 - volante

Ulteriore esempio: un videogioco



Classi ed oggetti: un esempio



Orco

Classi

Istanze (Oggetti)



Valiere



La programmazione orientata agli oggetti

- La programmazione orientata agli oggetti si basa su alcuni concetti fondamentali:
 - Classe
 - Incapsulamento
 - Oggetto
 - Ereditarietà
 - Polimorfismo

Classi e oggetti

- Nei linguaggi a oggetti, il costrutto **class** consente di definire nuovi tipi di dato e le relative operazioni
- Le operazioni possono essere definite sotto forma di operatori o di funzioni (dette metodi o funzioni membro), i nuovi tipi di dato possono essere gestiti quasi allo stesso modo dei tipi predefiniti del linguaggio:
 - si possono creare istanze, e
 - si possono eseguire operazioni su di esse
- Un oggetto è una variabile che appartiene ad un particolare tipo di dato definito dall'utente per mezzo del costrutto class.
- Nel gergo dei linguaggi OO una variabile definita di un certo tipo (classe) rappresenta **un'istanza** della classe
- Lo **stato** di un oggetto, invece, è rappresentato dai valori correnti delle variabili che costituiscono la struttura dati utilizzata per implementare il tipo di dato definito dalla classe

Caratteristiche di una classe

La classe è un modulo software con le seguenti caratteristiche:

- E' dotata di un'interfaccia (specificata) e di un corpo (implementazione)
- La struttura dati "concreta" di un oggetto della classe, e gli algoritmi che ne realizzano le operazioni, sono tenuti nascosti all'interno del modulo che implementa la classe
- Lo stato di un oggetto evolve unicamente in relazione alle operazioni ad esso applicate
- Le operazioni sono utilizzabili con modalità che prescindono completamente dagli aspetti implementativi; in tal modo è possibile modificare gli algoritmi utilizzati senza modificare l'interfaccia

Incapsulamento

- L'incapsulamento (in inglese: *information hiding*) è la suddivisione di un oggetto in due parti:
 - **Interfaccia:**
 - costituito dall'insieme di metodi (detti anche messaggi) che possono essere invocati dall'utente per accedere alle funzionalità dell'oggetto;
 - **Implementazione**
 - Contiene l'implementazione delle funzionalità dell'oggetto e delle strutture dati necessarie
 - È nascosta all'utente
- L'incapsulamento è realizzato per mezzo delle istruzioni:
 - **Private:** le funzioni e le variabili definite private NON sono accessibili all'esterno della classe.
 - **Public:** le funzioni e le variabili definite pubbliche sono accessibili all'esterno della classe.

Ereditarietà e poliformismo

- L'**ereditarietà** consente di definire nuove classi per specializzazione o estensione di classi preesistenti, in modo incrementale
- Il **polimorfismo** consente di invocare operazioni su un oggetto, pur non essendo nota a tempo di compilazione la classe cui fa riferimento l'oggetto stesso

Ereditarietà

- Il meccanismo dell'ereditarietà è di fondamentale importanza nella programmazione ad oggetti, in quanto induce una strutturazione gerarchica nel sistema software da costruire.
- L'ereditarietà consente infatti di realizzare relazioni tra classi di tipo generalizzazione-specializzazione, in cui una classe, detta base, realizza un comportamento generale comune ad un insieme di entità, mentre le classi derivate (sottoclassi) realizzano comportamenti specializzati rispetto a quelli della classe base

Esempio

- Tipo o classe base: Animale
- Tipi derivati (sottoclassi): Cane, Gatto, Cavallo, ...
- In una gerarchia gen-spec, le classi derivate sono specializzazioni (cioè casi particolari) della classe base

Ereditarietà: vantaggi

- Esiste però anche un altro motivo, di ordine pratico, per cui conviene usare l'ereditarietà, oltre quello di descrivere un sistema secondo un modello gerarchico; questo secondo motivo è legato esclusivamente al concetto di riuso del software.
- In alcuni casi si ha a disposizione una classe che non corrisponde esattamente alle proprie esigenze. Anziché scartare del tutto il codice esistente e riscriverlo, si può seguire con l'ereditarietà un approccio diverso, costruendo una nuova classe che eredita il comportamento di quella esistente, salvo che per i cambiamenti che si ritiene necessario apportare
- Tali cambiamenti possono riguardare sia l'aggiunta di nuove funzionalità che la modifica di quelle esistenti

Polimorfismo

- Per polimorfismo si intende la proprietà di una entità di assumere forme diverse nel tempo.
- Una entità è polimorfa se può fare riferimento, nel tempo, a classi diverse.
- Siano ad esempio a, b due oggetti appartenenti rispettivamente alle classi A, B, che prevedono entrambe una operazione m , con diverse implementazioni. Si consideri l'assegnazione:

$a := b$

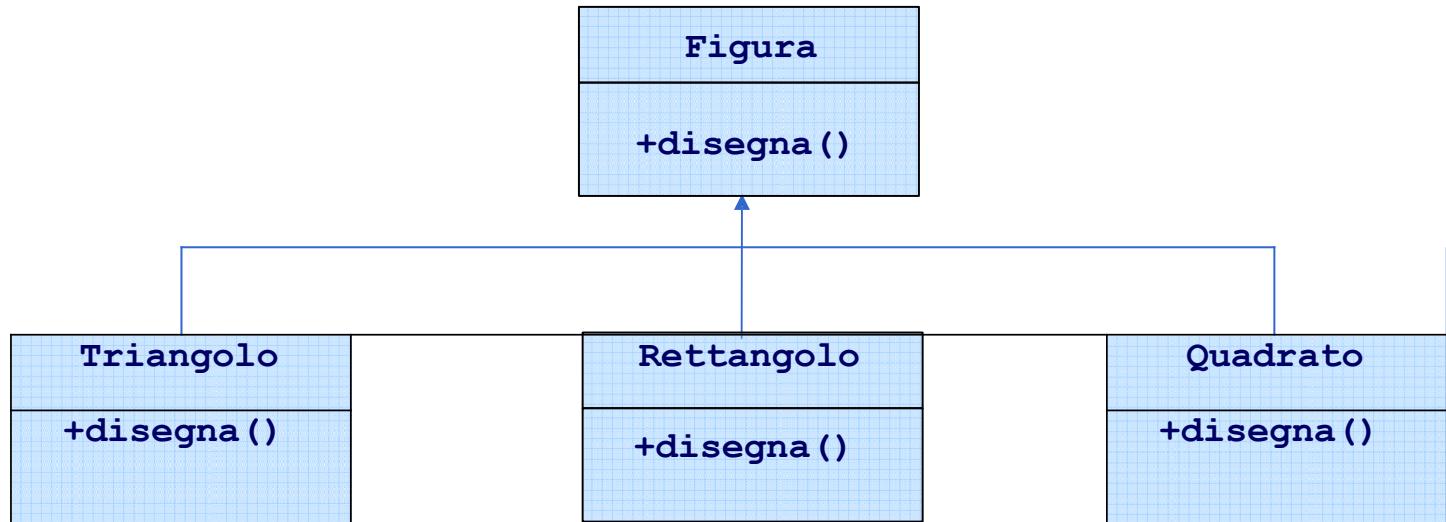
- L'esecuzione della operazione m sull'oggetto a dopo l'assegnazione, per la quale è spesso adoperata la sintassi:

$a.m()$

produce l'esecuzione della implementazione di m specificata per la classe B.

Polimorfismo: un esempio

- Si consideri una gerarchia di classi di figure geometriche:



- Sia ad es. A un vettore di N oggetti della classe Figura, composto di oggetti delle sottoclassi Triangolo, Rettangolo, Quadrato:
Figura A[N]

(ad es.: A[0] è un quadrato, A[1] un triangolo, A[2] un rettangolo, etc.)

Polimorfismo: un esempio

- Si consideri una funzione `Disegna_Figure()`, che contiene il seguente ciclo:
 - `for i = 1 to N do`
 - `A[i].disegna()`
- L'esecuzione del ciclo richiede che sia possibile determinare dinamicamente (cioè a tempo d'esecuzione) l'implementazione della operazione `disegna()` da eseguire, in funzione del tipo corrente dell'oggetto `A[i]`.
- L'istruzione `A[i].disegna()` non ha bisogno di essere modificato in conseguenza dell'aggiunta di una nuova sottoclassificazione di `Figura` (ad es.: `Cerchio`), anche se tale sottoclassificazione non era stata neppure prevista all'atto della stesura della funzione `Disegna_Figure()`. (Si confronti con il caso dell'uso di una istruzione `case` nella programmazione tradizionale).

Vantaggi della programmazione OO

- Rispetto alla programmazione tradizionale, la programmazione orientata agli oggetti (OOP) offre vantaggi in termini di:
 - ✓ **modularità**: le classi sono i moduli del sistema software;
 - ✓ **coesione dei moduli**: una classe è un componente software ben coeso in quanto rappresentazione di una unica entità;
 - ✓ **disaccoppiamento dei moduli**: gli oggetti hanno un alto grado di disaccoppiamento in quanto i metodi operano sulla struttura dati interna ad un oggetto; il sistema complessivo viene costruito componendo operazioni sugli oggetti;
 - ✓ **information hiding**: sia le strutture dati che gli algoritmi possono essere nascosti alla visibilità dall'esterno di un oggetto;
 - ✓ **riuso**: l'ereditarietà consente di riusare la definizione di una classe nel definire nuove (sotto)classi; inoltre è possibile costruire librerie di classi raggruppate per tipologia di applicazioni;
 - ✓ **estensibilità**: il polimorfismo agevola l'aggiunta di nuove funzionalità, minimizzando le modifiche necessarie al sistema esistente quando si vuole estenderlo.

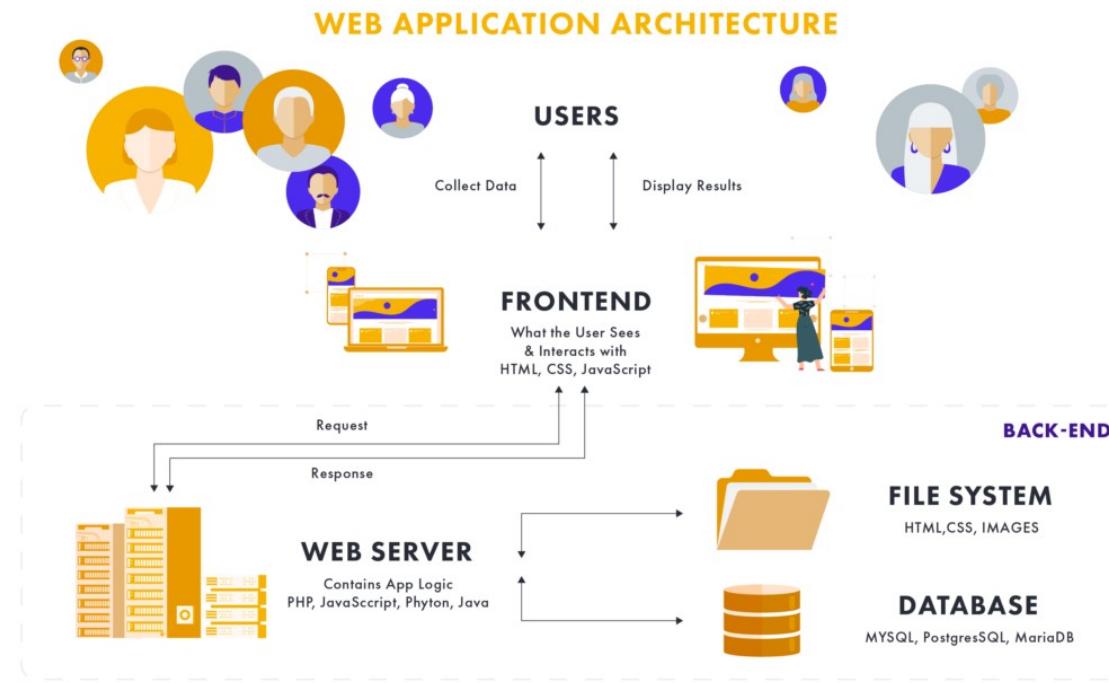
I più popolari linguaggi di programmazione



The 2022 Top Ten Programming Languages

Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C		99.7
3. Java		99.5
4. C++		97.1
5. C#		87.7
6. R		87.7
7. JavaScript		85.6
8. PHP		81.2
9. Go		75.1
10. Swift		73.7

Frontend vs Backend



Focus



Frontend

Focuses on layout, animations, content organization, navigation, graphics.

Programming languages:
JavaScript, HTML, CSS

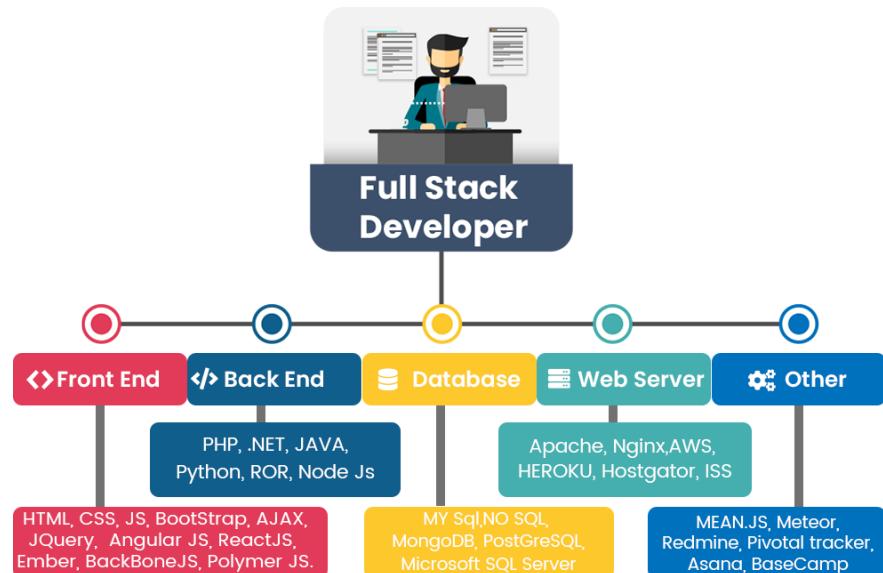
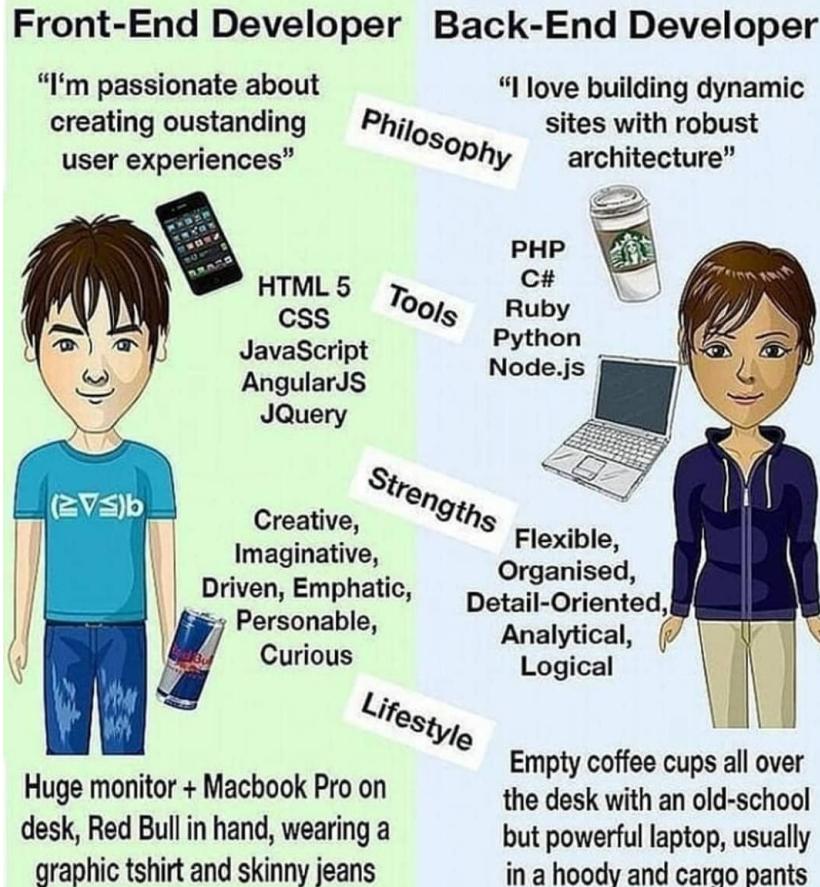


Backend

Focuses on building code, debugging, database management.

Programming languages:
Node.js, Python, Java

Full stack developer



HTML



Un pò di storia

Siete curiosi di sapere come tutto è nato?

Venerdì 4 giugno 2004, in una notte buia e tempestosa, Ian Hickson annuncia la creazione del gruppo di ricerca WHAT, acronimo di **Web Hypertext Application Technology** in un sintetico ma ricco messaggio.

L'obiettivo del gruppo è quello di elaborare specifiche per aiutare lo sviluppo di un web più orientato alle applicazioni che ai documenti; tra i sottoscrittori di questa iniziativa si annoverano aziende del calibro di Apple, Mozilla e Opera.

Questa piccolo scisma dal W3C è determinato dal disaccordo in merito alla rotta decisa dal consorzio durante un famoso convegno del 2004 dove, per un pugno di voti, prevalse la linea orientata ai documenti di XHTML2.

Un pò di storia

Nei due anni successivi i gruppi XHTML2 e WHAT proseguono i lavori sulle proprie specifiche in modo indipendente e parallelo, sollevando dubbi e confusione sia da parte dei produttori di browser che degli sviluppatori web.

Emblematico in tal senso è un articolo firmato da Edd Dumbill nel dicembre 2005 intitolato [The future of HTML](#). Il 27 ottobre 2006 in un post sul proprio blog intitolato [Reinventing HTML](#), Tim Berners-Lee annuncia la volontà di creare un nuovo gruppo di ricerca che strizzi l'occhio al WHAT e ammette alcuni sbagli commessi seguendo la filosofia XHTML2

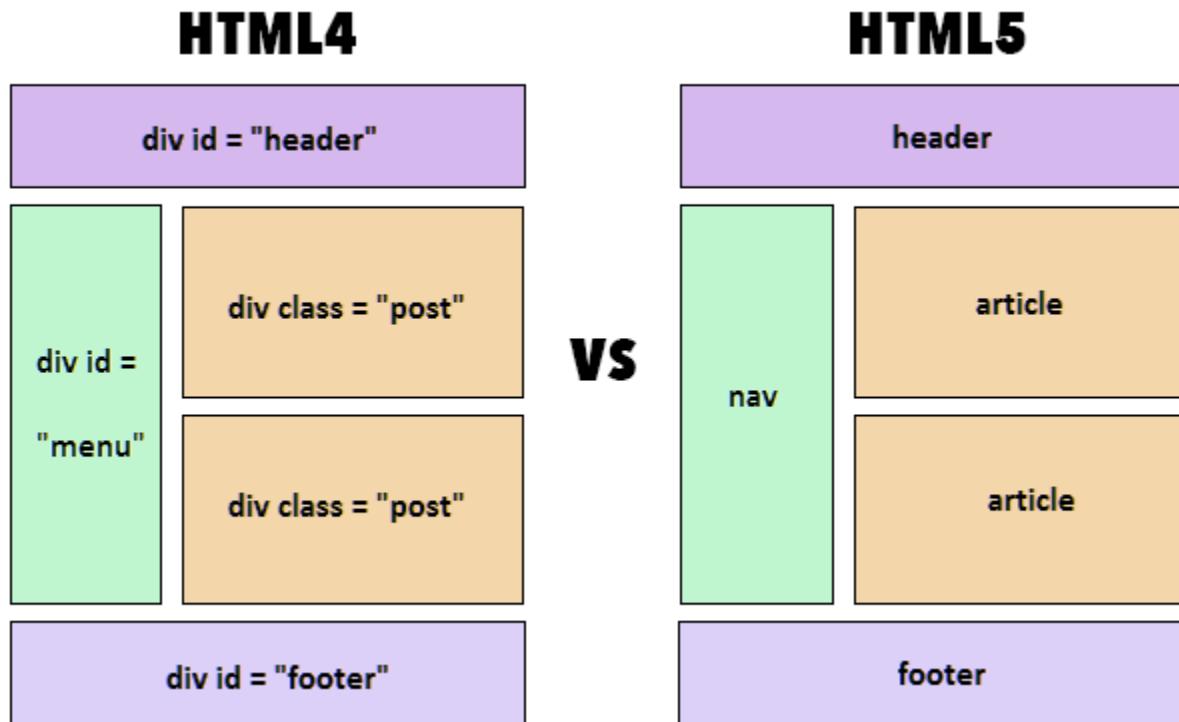
Che cosa è l'HTML5

Per prima cosa è importante ricordare che, anche in virtù della storia della sua nascita, all'interno dell'HTML5 convivono in armonia due anime: la prima, che raccoglie l'eredità semantica dell'XHTML2, e la seconda che invece deriva dallo sforzo di aiutare lo sviluppo di applicazioni Web.

Il risultato di questo mix di intenti è più articolato di quanto si immagini; in prima istanza si assiste ad una **evoluzione del modello di markup**, che non solo si amplia per accogliere nuovi elementi, ma modifica in modo sensibile anche le basi della propria sintassi e le regole per la disposizione dei contenuti sulla pagina.

A questo segue un **rinvigorimento delle API JavaScript** che vengono estese per supportare tutte le funzionalità di cui una applicazione moderna potrebbe aver bisogno.

Che cosa è l'HTML5



CSS

CSS



Introduzione

L'acronimo **CSS** sta per **Cascading Style Sheets** (fogli di stile a cascata) e designa un linguaggio di stile per i documenti web. I [CSS](#) istruiscono un browser o un altro programma utente su come il documento debba essere presentato all'utente, per esempio definendone i font, i colori, le immagini di sfondo, il layout, il posizionamento delle colonne o di altri elementi sulla pagina, etc.

La storia dei CSS procede su binari paralleli rispetto a quelli di [HTML](#), di cui vuole essere l'ideale complemento. Da sempre infatti, nelle intenzioni degli uomini del W3C, HTML dovrebbe essere visto semplicemente come un linguaggio **strutturale**, alieno da qualunque scopo attinente la **presentazione** di un documento.

Versioni successive

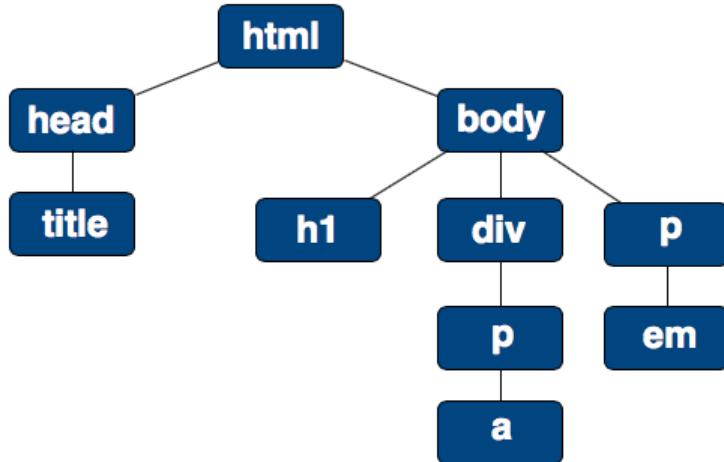
Per questo obiettivo, ovvero arricchire l'aspetto visuale e la presentazione di una pagina, lo strumento designato sono appunto i CSS. L'ideale perseguito da anni si può sintetizzare con una nota espressione: separare il contenuto dalla presentazione.

La prima specifica ufficiale di CSS ([CSS 1](#)) risale al dicembre del 1996. Nel maggio 1998 è stata la volta della seconda versione: [CSS 2](#).

Niente stravolgimenti, ma molte aggiunte rispetto alla prima. CSS 2 non è altro che CSS 1 più alcune nuove proprietà, valori di proprietà e definizioni per stili non canonici come quelli rivolti alla stampa o alla definizione di contenuti audio. Mentre prendeva corpo la specifica relativa ai [CSS3](#), veniva anche effettuata una revisione della seconda specifica, denominata CSS 2.1, che ha raggiunto lo stato di raccomandazione ufficiale nel giugno 2011.

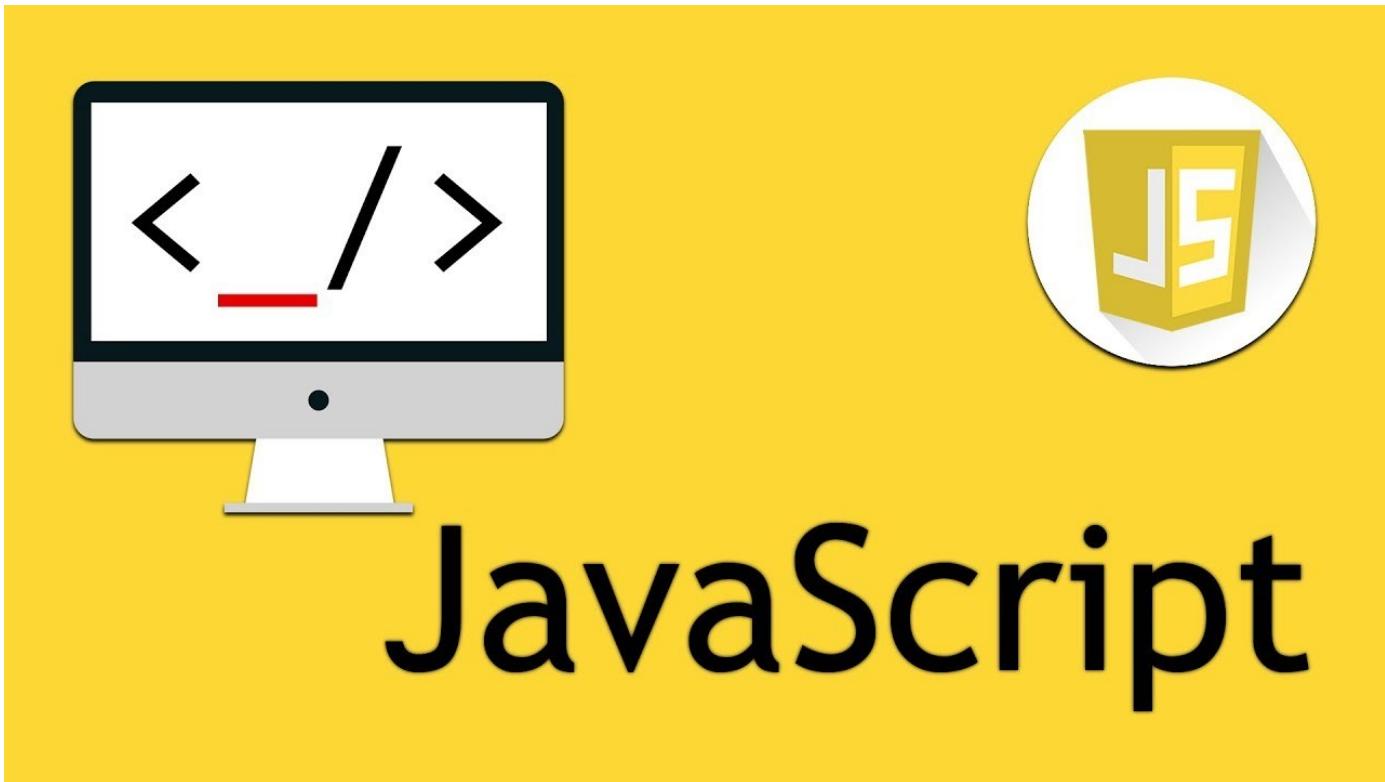
Oggi usiamo [CSS3](#).

Come funziona



```
body {  
background: white;  
color: black;  
}  
/* Stili per i titoli h1 */  
h1 {  
color: red;  
font: 36px Helvetica, Arial, sans-serif;  
}  
/* Colore del testo delle liste */  
li {color: green;}  
/* Colore dei titoli h1 per la stampa */  
@media print {  
h1 {color: black;}  
}
```

JavaScript



JavaScript

Un po' di storia

JavaScript è stato ideato nel 1995 da Netscape e rilasciato con la versione 2.0 del suo browser, *Netscape Navigator*, dapprima con il nome **LiveScript** e subito dopo con l'attuale nome, creando all'inizio non poca confusione con Java che proprio in quell'anno debuttava con grande attenzione da parte del mondo del software.

Da subito JavaScript aggiunse alle pagine HTML la possibilità di essere modificate in modo dinamico, in base all'interazione dell'utente con il browser (lato client). Questo grazie alle funzionalità di calcolo e di manipolazione dei documenti che era possibile effettuare anche senza coinvolgere il server. Questa caratteristica è stata sottolineata durante gli anni '90 con il nome che veniva dato all'accoppiata HTML-JavaScript: **Dynamic HTML (DHTML)**.

Un po' di storia

Nel 1997, sulla base del linguaggio di Netscape, nacque lo standard *ECMA-262* definito dall'organismo di standardizzazione industriale [ECMA International](#) che rappresentava le specifiche del linguaggio **ECMAScript**.

L'esigenza di rendere il Web sempre più interattivo e dinamico favorì la nascita di tecnologie concorrenti: Flash, ActiveX e gli Applet Java. Queste tecnologie fornivano la possibilità di realizzare funzionalità ed effetti grafici di maggior impatto rispetto a quanto possibile con JavaScript, ma richiedevano specifici runtime o, come i controlli ActiveX, giravano solo su uno specifico browser.

Un po' di storia

La concorrenza tra componenti esterni e JavaScript è durata diversi anni e vide Flash predominante sul fronte dell'interazione utente e dei formati per l'advertising, a discapito di JavaScript, che sembrava essere destinato ad un lento declino.

Fu l'avvento della tecnologia Ajax, la possibilità cioè di comunicare in maniera asincrona con il server tramite script, a riportare JavaScript sulla cresta dell'onda.

Il rinnovato interesse verso il linguaggio con le nuove potenzialità applicative ha fatto nascere il cosiddetto **Web 2.0** ed ha fatto fiorire numerose librerie con lo scopo di semplificare alcune delle attività più comuni e di bypassare le differenze che ancora c'erano tra i Browser, favorendo una programmazione unificata e più rapida.

Abbiamo per le mani un linguaggio maturo e utilizzabile in diversi contesti non più necessariamente legati al Web.

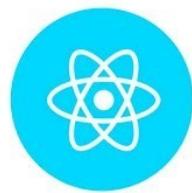
ECMAScript 6

Allo stato attuale, l'ultima versione ufficiale di ECMAScript è la 6, pubblicata a giugno del 2015. Questa versione delle specifiche, indicata generalmente con ES6 o come ECMAScript 2015, aggiunge delle interessanti novità al linguaggio di scripting e nel corso della guida verranno opportunamente evidenziate.

Tuttavia, il loro supporto da parte dei browser più recenti non è del tutto completo. Vedremo comunque come, nonostante tutto, le nuove funzionalità di JavaScript derivanti da ES6 possano essere utilizzate sin da ora.

Libraries and Frameworks

TOP 5 JAVASCRIPT LIBRARIES AND FRAMEWORKS



React.js

Library



Vue.js

Framework



jQuery

Library



Angular.js

Framework



Ember.js

Framework

Libraries and Frameworks

