Programmazione I

A.A. 2002-03

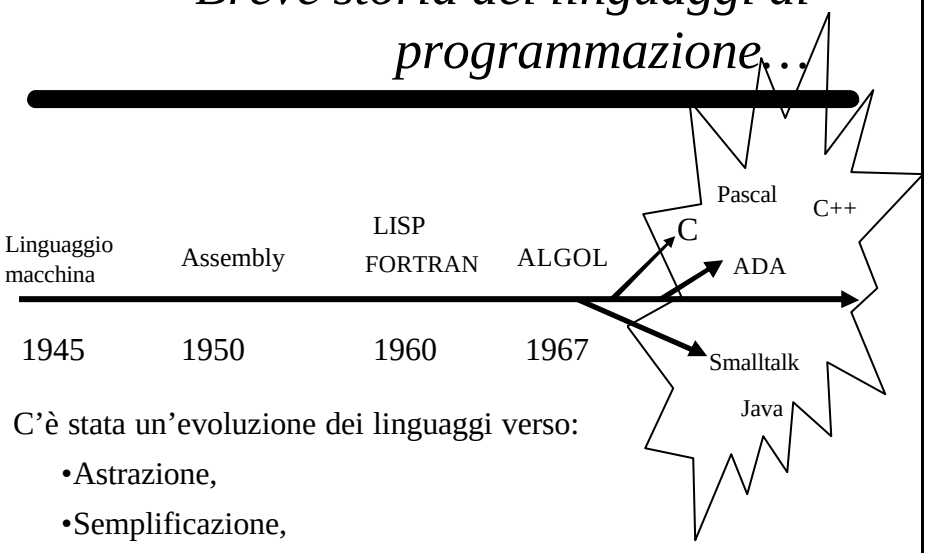
Elementi di Informatica

(Lezione VI)

Linguaggi di programmazione

Prof. Giovanni Gallo
Dr. Gianluca Cincotti
Dipartimento di Matematica e Informatica
Università di Catania
e-mail : {gallo, cincotti}@dmi.unict.it

Breve storia dei linguaggi di programmazione…



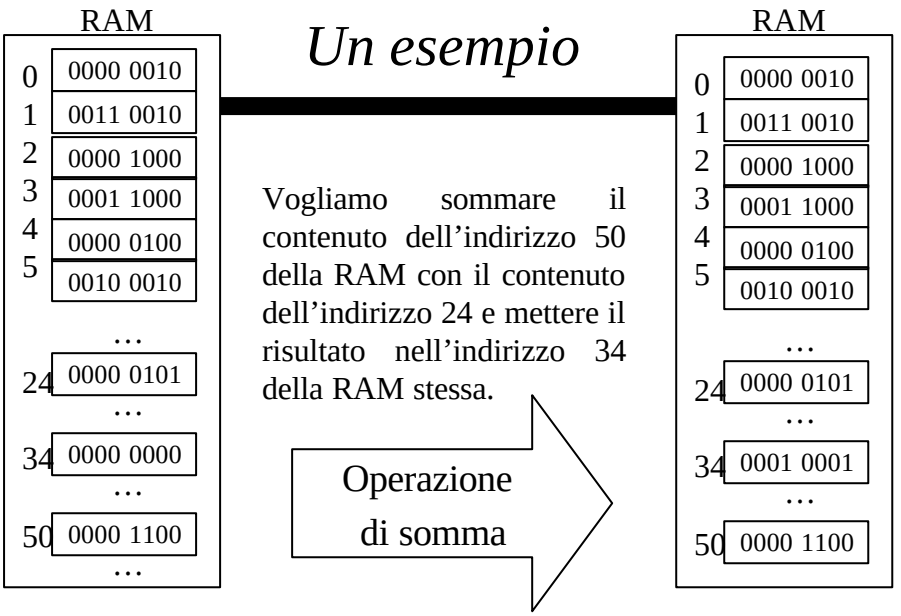
C'è stata un'evoluzione dei linguaggi verso:

- Astrazione,
- Semplificazione,
- Similarità con il ragionamento umano.

Linguaggi di programmazione

- I *linguaggi di programmazione* sono classificati in tre livelli:
  - linguaggi macchina,
  - linguaggi assembly,
  - linguaggi ad alto livello.

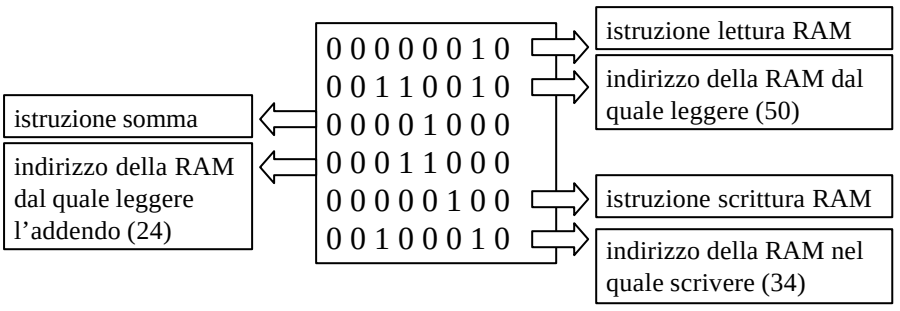
Un esempio



L'operazione di somma tra due numeri in memoria non è elementare!

- La sequenza di operazioni da fare è:
  - Copia il contenuto della word 50 dalla RAM al registro ACC (accumulatore);
  - Prendi il contenuto della word 24 ed incrementa ACC di tale valore;
  - Scrivi il contenuto del registro ACC nella parola 34 della RAM.

In linguaggio macchina …



Il programma viene inizialmente caricato in RAM; il PC viene inizializzato all'indirizzo della prima istruzione.

… ed in assembly …

- Il programmatore non deve più ricordare sequenze astruse di numeri binari, ma può usufruire di *assemblatori* che traducono automaticamente:
  - *codici operativi* per le istruzioni macchina,
  - *nomi simbolici* o *mnemonici* per registri e per locazioni di memoria.

- Esempio:

```
load  ACC, var1
add   ACC, var2
store tot, ACC
```


I problemi dei linguaggi macchina

- Sono specifici della macchina.
  - Ogni CPU ha il proprio linguaggio macchina.
  - Occorre conoscere l'architettura della macchina per scrivere programmi.
  - I programmi non sono portabili.
- I codici sono illeggibili all'uomo.
- I programmatori si specializzano nel cercare efficienza su una macchina specifica, anziché concentrarsi sul problema.

… e dell'assembly

- Sono comunque legati all'architettura della macchina.
- I linguaggi *assembly* non sono sufficienti a gestire l'enorme complessità dei programmi moderni.
  - TOP_DOWN o BOTTOM_UP ?
    - Il modo naturale di procedere è pensare prima alla struttura generale e poi curare i dettagli …
    - MA questo è impossibile con l'Assembly che è fatto SOLO da dettagli...

I linguaggi di programmazione

- I *linguaggi di programmazione* sono stati introdotti per facilitare la scrittura dei programmi.
  - Sono linguaggi *simbolici* e in continua evoluzione.
  - Sono definiti da un insieme di regole formali, le regole grammaticali o ***sintassi*.**

Sintassi e semantica

- Le *regole di sintassi* definiscono come si devono comporre i simboli e le parole per formare istruzioni corrette.
- La *semantica* di un'istruzione definisce il significato della stessa.
- Un programma sintatticamente corretto non è necessariamente semanticamente corretto.
  - I programmi fanno quello che prescriviamo che facciano e non quello che vorremmo che facessero.

L'idea della traduzione

IDEA: scrivo le mie istruzioni usando un linguaggio a me più comprensibile e poi le *traduco* in linguaggio macchina.

Alto e basso livello

- Nell'ambito dei *linguaggi di programmazione:*
  - Se ci si avvicina al linguaggio umano, si parla di linguaggi di *Alto livello.*
  - Se ci si avvicina al linguaggio macchina, si parla di linguaggi di *Basso livello.*

Diversi livelli di espressività

- In un linguaggio ad *alto livello*:

```
tot = var1 + var2;
```

- In un linguaggio *assembly:*

```
load  ACC, var1
add   ACC, var2
store tot, ACC
```

Diversi livelli di espressività (cont.)

- In un linguaggio ad *alto livello*:

```
se (a==b) allora c=0
          altrimenti c=a+b;
```

- In un linguaggio *assembly:*

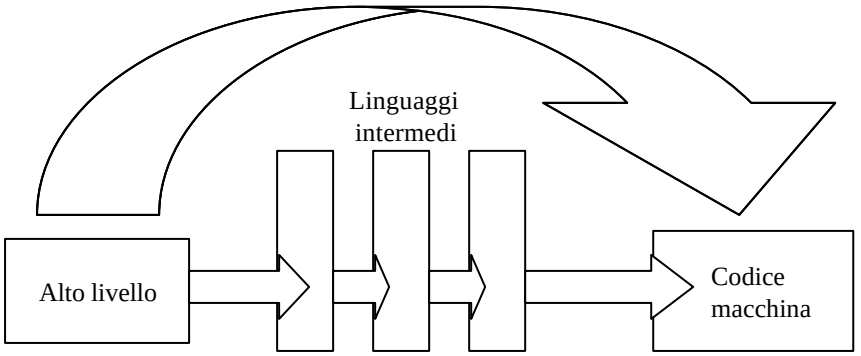
```
          load  R1, a
          load  R2, b
          sub   R1, R2
          jzero R1, fine
          load  R1, a
          add   R1, R2
fine:     store c,  R1
```


Traduzione dei linguaggi

- Il *concetto di traduzione* dei linguaggi ha permesso l'evoluzione verso sistemi simbolici più espressivi e più facilmente manipolabili dai programmatori
  - Il programmatore scrive un programma in un linguaggio ad alto livello senza preoccuparsi della macchina che esegue il programma.

Il meccanismo della traduzione

Tradurre è estremamente complesso. Si preferisce fare una *catena di traduzioni* tra linguaggi leggermente differenti andando sempre "verso" la macchina.



Dall'assembly al FORTRAN

- Il programmatore non deve necessariamente occuparsi della gestione della memoria.
  - può dicharare una variabile e ottenere dal computer l'assegnazione di una area di memoria alla stessa.
- Il programmatore può usare il linguaggio della matematica e non le istruzioni mnemoniche.

I problemi del FORTRAN

- Programmi difficili da leggere e da correggere a causa delle istruzioni “GOTO”.
- “Pezzi” di codice sono simili tra loro e potrebbero essere scritti solo una volta.
  - Nelle prime versioni non c’è alcuno strumento del linguaggio che aiuti a creare “moduli” o funzioni”.

Alcune soluzioni

- Modularizzazione:
  - creare sotto-programmi il più possibile indipendenti tra loro e isolare dentro tali “moduli” le operazioni più semplici.
    - Costruire da moduli semplici moduli via via più complessi…
- Astrazione:
  - slegare il programmatore dal modello della macchina e avvicinarlo al modello del problema da risolvere.
- Strutturazione:
  - i salti nell’esecuzione del programma debbono essere espliciti, visibili e chiari.
    - NON si deve usare il “GOTO”

Una nuova generazione di linguaggi

- Il linguaggio ALGOL non ha mai preso piede ma … è stato il modello per :

  C, Pascal, MODULA, FORTRAN77.

- Novità della programmazione strutturata:
  - funzioni che isolano i sotto-programmi;
  - controllo della gestione della memoria mediante variabili "tipizzate".

Ma i problemi non si sono esauriti…

- I linguaggi ALGOL-like non sono ancora soddisfacenti.
  - Le modalità di organizzazione dei dati (*strutture dati*) restano separate dai metodi necessari a manipolare i dati stessi.
  - Le funzioni/moduli non sono "a tenuta stagna" :
    - possono influenzarsi tra loro in maniera non sempre esplicita al programmatore.
  - Non è quasi mai facile *riutilizzare* una funzione in un altro programma.
- La *manutenzione* del software (correzione ed aggiornamento) diviene rapidamente più difficile e costosa dello sviluppo ex-novo dello stesso software.

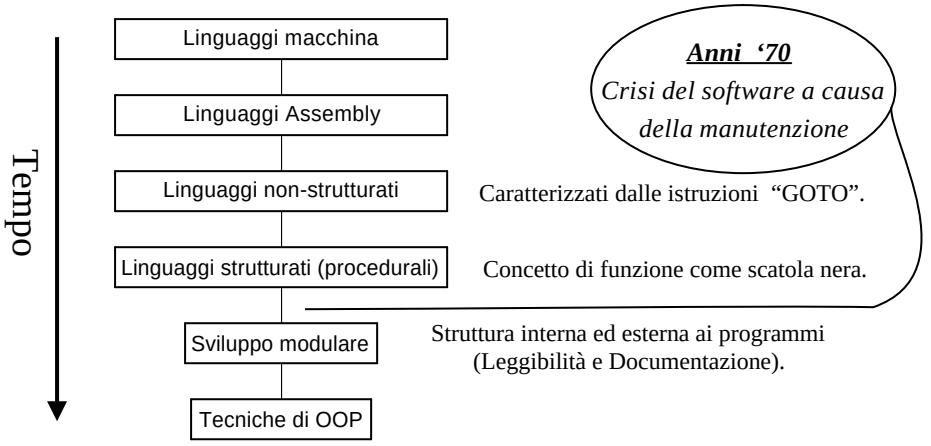
L'ultima tendenza …

- Linguaggi orientati agli oggetti:

  SMALLTALK, EIFFEL, C++, JAVA.

  - Ad ogni "entità" del problema da risolvere corrisponde un oggetto capace di memorizzare i dati in maniera organizzata e di manipolarli.
    - Un oggetto è una scatola nera con ingressi e uscite chiaramente ed esplicitamente definiti:
  - Il sogno della modularizzazione sembra così realizzarsi.

Evoluzione delle tecniche di programmazione



Perché ci son voluti 40 anni ?

- Perché ogni linguaggio nuovo si trova un po' più "lontano" dal linguaggio macchina e non è affatto semplice costruire programmi che TRADUCANO *automaticamente* dal linguaggio ad alto livello verso il codice macchina direttamente eseguibile.
  - Tali programmi traduttori sono complessi da creare e richiedono grandi risorse computazionali.
    - Nascono quindi problemi di efficienza che solo computer veloci possono risolvere adeguatamente.

I paradigmi di programmazione

- Forniscono la filosofia con cui si scrivono i programmi e stabiliscono:
  - la metodologia con cui si scrivono i programmi,
  - il concetto di computazione.
- I linguaggi devono *consentire* ma soprattutto *spingere* all'adozione di un particolare paradigma.
  - Funzionale
  - Logica
  - Imperativa
  - Modulare
  - Orientata agli oggetti

Paradigma procedurale

- Enfasi sulla soluzione dei problemi mediante modifica progressiva dei dati
  - Esecuzione sequenziale di istruzioni
  - Stato della memoria
  - Cambiamento di stato tramite esecuzione di istruzioni
- Aderenti al modello della macchina di von Neumann
- Molto efficienti
- Ha mostrato limiti nello sviluppo e mantenimento di software complessi
- **Pascal**, **C**

Influenza del modello di macchina

- Concetto di istruzione
- Concetto di sequenzialità e iterazione
  - Il programma assolve il compito eseguendo le istruzioni in sequenza
- Concetto di variabile e di assegnamento
  - Le celle di memoria hanno un indirizzo e contengono i dati da manipolare
  - Le variabili hanno un nome e un valore
  - L'assegnamento di un valore a una variabile equivale al trasferimento di un dato in una cella

Influenza del modello di macchina (cont.)

È sorprendente che il computer di Von Neumann sia rimasto così a lungo il paradigma fondamentale dell'architettura dei calcolatori.

Ma dato il fatto, non è sorprendente che i linguaggi imperativi siano I principali oggetti di studio e sviluppo.

Perchè come Backus ha sottolineato i linguaggi imperativi hanno solide radici nell'architettura della macchina di Von Neumann e ne sono l'immagine.

Horowitz *Fundamentals of Programming Languages*, 1983

Paradigma funzionale

- Primo tentativo di non rifarsi al modello di macchina di von Neumann
  - Il programmatore può IGNORARE la struttura fisica della macchina e scrivere i propri programmi in maniera assolutamente naturale basata sulla logica e la matematica.
- La computazione avviene tramite funzioni che applicate ai dati riportano nuovi valori
  - Le funzioni possono essere applicate a funzioni in catena e possono essere ricorsive
- **Lisp**, **ML**

Paradigma modulare

- Introduce il concetto di modulo che nasconde i dati all'utente
  - I dati possono essere letti solo tramite un'opportuna interfaccia
- **Modula-2**, **Ada**

Paradigma a oggetti (OOP)

- Spinge ulteriormente il concetto di modulo che incapsula i dati con le classi
  - Le classi hanno anche una struttura gerarchica ed ereditano caratteristiche e funzionalità
- Introdotto per migliorare l'efficienza del processo di produzione e mantenimento del software

Concetti base della OOP

- Incapsulamento dei dati
  - Il processo di nascondere i dettagli di definizione di oggetti, solo le interfacce con l'esterno sono visibili
- Ereditarietà
  - Gli oggetti sono definiti in una gerarchia ed ereditano dall'immediato parente caratteristiche comuni, che possono essere specializzate
- Astrazione
  - Il meccanismo con cui si specifica le caratteristiche peculiari di un oggetto che lo differenzia da altri
- Polimorfismo
  - Possibilità di eseguire funzioni con lo stesso nome che pure sono state specializzate per una particolare classe

Traduttori

- Servono a generare software.
  - Generano codice in *linguaggio macchina* a partire da codice scritto in un linguaggio di programmazione ad alto livello (ad es. C++, Java).
- Si distinguono in:
  - interpreti,
  - compilatori.

Compilatori

- Un compilatore è un programma che prende in input un codice *sorgente* e lo traduce fornendo in output un codice *oggetto.*
  - Per eseguire un programma *sorgente P*, scritto in un linguaggio di programmazione *L*:
    - *P* viene tradotto in un programma *Q* equivalente scritto in linguaggio macchina;
    - il programma *Q* viene eseguito.
- Esempi di linguaggi compilati:
  - C++, Pascal, Cobol, Fortran, …
- Il compilatore è legato all'architettura della macchina.

Interpreti

- Un interprete è un programma che prende in input un codice sorgente e, passo dopo passo, traduce ed esegue ogni singola istruzione.
  - La traduzione avviene dunque simultaneamente all'esecuzione.
  - Per ogni istruzione del programma sorgente *P*:
    - Viene tradotta la *singola* istruzione generando il corrispondente insieme di istruzioni in linguaggio macchina;
    - Si esegue il codice in linguaggio macchina e si passa all'istruzione sorgente successiva.
- Esempi di linguaggi interpretati:
  - VBasic, Lisp, Prolog, Java.

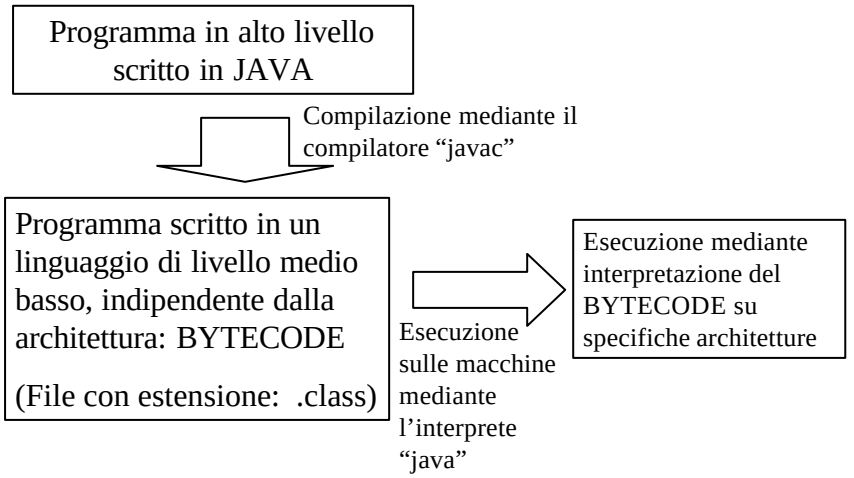
Compilatori vs. Interpreti

- Interpreti
  - Lenti nell'esecuzione.
  - Spesso si utilizza un linguaggio intermedio.
  - Progetti di dimensione limitata.
  - Facilità d'interazione col codice e velocità di sviluppo.
  - Facili da scrivere.
- Compilatori
  - Veloci nell'esecuzione.
  - Necessitano di poca memoria.
  - Permettono la compilazione separata.
  - Difficili da scrivere.

Java

- Java definito dalla Sun Microsystems, Inc.
- Introdotto nel 1995
- E' un linguaggio ***orientato agli oggetti***
- Derivato da Smalltalk e C++
- Definito per essere trasportabile su architetture differenti e per essere eseguito da browser

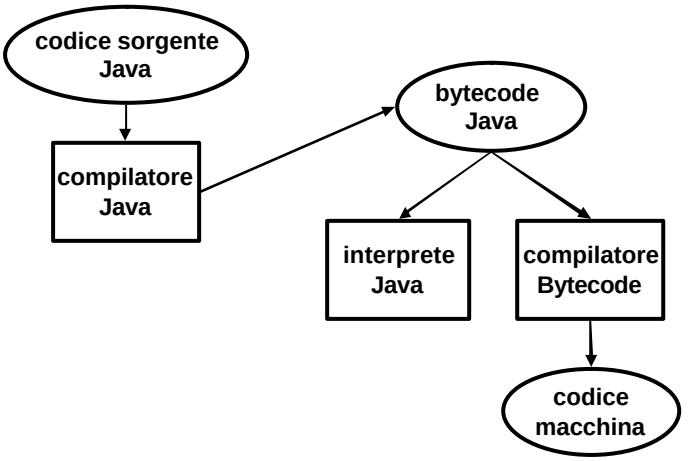
In medio stat virtus: JAVA



Traduzione ed esecuzione di Java

- Il compilatore Java traduce il programma *sorgente* in una rappresentazione speciale detta ***bytecode.***
- Il bytecode Java non è un linguaggio macchina di una CPU particolare, ma di una ***macchina virtuale Java.***
  - L'*interprete* traduce il bytecode nel linguaggio macchina e lo esegue.
- Un compilatore Java compiler non è legato ad una particolare macchina.
  - Java è *indipendente* dall'architettura della macchina.

Traduzione ed esecuzione di Java (cont.)



L'uovo di Colombo!

- Sembra unire i pregi della compilazione:
  - traduco una volta sola e la traduzione da JAVA a bytecode è relativamente semplice,

  con i pregi dell'interpretazione:
  - è facile scrivere un interprete java, Java Virtual Machine, per ogni differente architettura.
- Basta scrivere un solo codice per molte macchine:
  - ideale per il WEB.
- Problema : efficienza! (JIT)

Fine