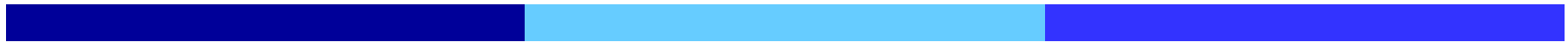


# Programmazione II

A.A. 2022-23

Prof. Maria Tortorella



## Introduzione al corso

# Requisiti per seguire il corso

---

- Conoscenza pregressa riguardante i fondamenti di informatica e la programmazione in linguaggio C:
  - Non è solo necessario aver seguito il corso di Programmazione I
  - È necessario aver studiato gli argomenti di Programmazione 1
  - È necessario avere in programma di sostenere l'esame di Programmazione 1 al più presto
- Quanti hanno già sostenuto l'esame di Programmazione 1?

# Note sul corso

---

Non è un corso di apprendimento di uno specifico linguaggio di programmazione

ma

... un corso di programmazione

... orientata agli oggetti, utilizzando JAVA

a tale scopo saranno ripresi i concetti studiati nel corso di Programmazione 1 e saranno presentate le differenze rispetto ad essi.

# Note sul corso

---

- È necessario studiare i concetti teorici che sono alla base della programmazione object oriented ...
- Studiare i lucidi NON è sufficiente !
  - Bisogna approfondire su libri, documentazione on line, ecc...
- ... è fondamentale che si faccia pratica !
  - sviluppare personalmente programmi utilizzando un computer ...
  - Evitare di scaricare da internet programmi già sviluppati
- Ricevimento studenti:  
RCOST II piano  
lunedì 15:00-17:00
- Interagisco via e-mail: [tortorella@unisannio.it](mailto:tortorella@unisannio.it)

# Organizzazione sul corso

---

- 6 ore settimanali di lezione frontale, normalmente distribuite in 2 lezioni
  - Giovedì (9:00-12:00),
  - Venerdì (9:00-12:00)
  
- Illustrare la sintassi del linguaggio richiederà una piccola parte del corso ma...
  
- ... saranno fondamentali le esercitazioni
  - In aula
  - In laboratorio – possibili incontri aggiuntivi

# Libri di testo

Cay S. Horstmann, "Concetti di Informatica e Fondamenti di Java"  
Apogeo



Per scambiarsi messaggi e condividere esercizi e altro:

<http://groups.google.it/group/Programmazione2-unisannio-22-23>  
email: [Programmazione2-unisannio-22-23@googlegroups.com](mailto:Programmazione2-unisannio-22-23@googlegroups.com)

# Lecture suggerite

M. T. Goodrich, R. Tamassa, "Strutture dati ed algoritmi in Java", Zanichelli cap. 6-13



Giovanni Pighizzini, Mauro Ferrari "Dai fondamenti agli oggetti - Corso di programmazione Java", Pearson / Addison-Wesley



## Ulteriori risorse:

Bruce Eckel, Thinking in Java

<http://www.mindviewinc.com/Books/TIJ4>

Tutorial Java

<http://docs.oracle.com/javase/tutorial/>

API di Java - <https://docs.oracle.com/javase/7/docs/api/>


# Programma Sintetico

- Introduzione alla programmazione object-oriented; concetti di classe, oggetto e messaggio; astrazione funzionale e astrazione sui dati;
- information hiding; ambienti di programmazione in Java; processo di compilazione di un programma Java.
- Implementazione di programmi utilizzando classi predefinite.
- Definizione di classe; implementazione di metodi; strutture di controllo; variabili d'istanza; tipi di dati fondamentali; variabili di tipi predefiniti; input/output di programmi.
- Collezioni di oggetti
- Ereditarietà e polimorfismo
- Testing e debugging di un programma
- Programmazione ad eventi ed interfaccia grafica
- Gestione dei file
- Algoritmi di ricerca ed ordinamento
- Tipi di dati astratti: liste, LIFO, FIFO, hash table, Alberi, Grafi, ...
- Esercitazioni in aula e in laboratorio



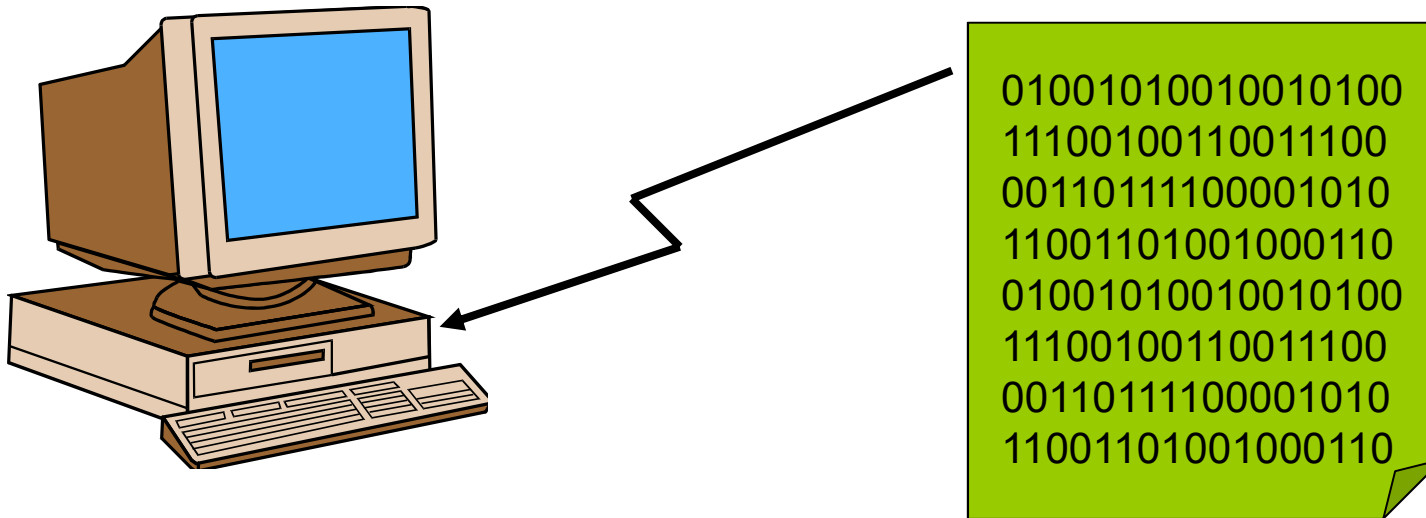
# Programmazione II

## Introduzione agli argomenti del corso

- 
- Linguaggi di programmazione
    - Linguaggi a basso ed a alto livello
    - Compilatori ed interpreti

# Programmi e calcolatori

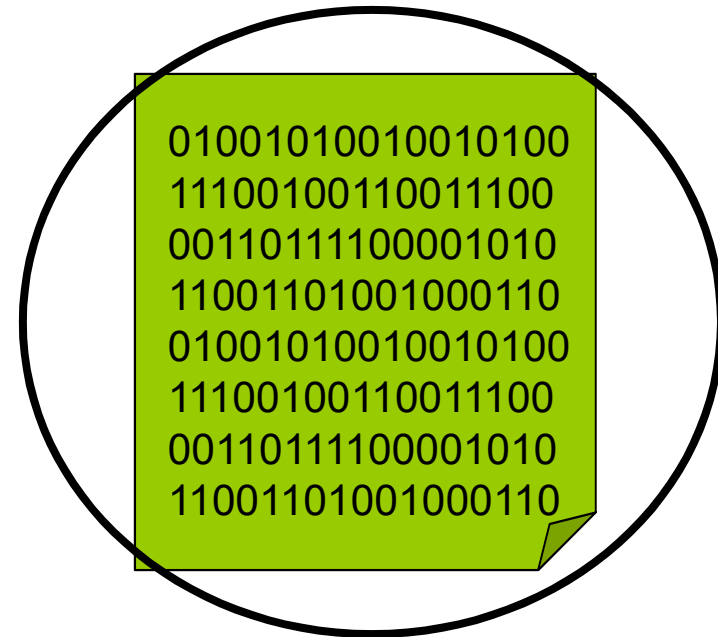
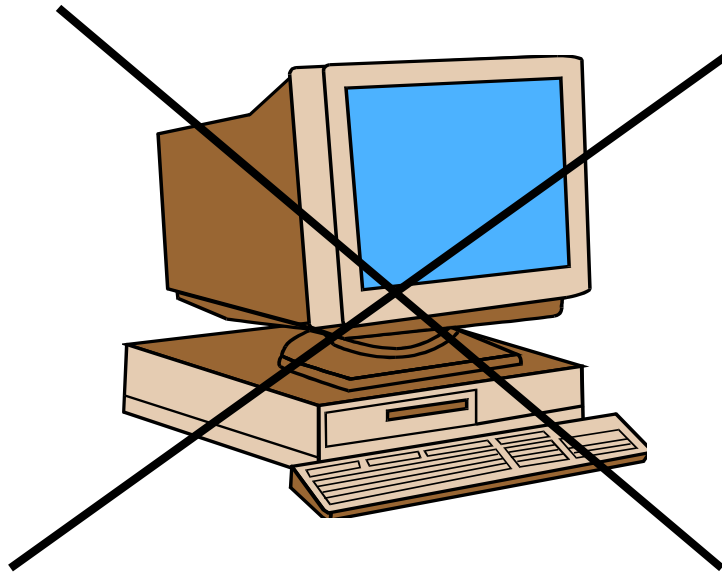
- Un programma è un testo che permette ad un calcolatore di svolgere un compito



- Il contenuto di un programma viene detto **codice**

# Focalizzazione del corso

---



- In particolare l' **approccio Orientato agli Oggetti** supporta questo concetto

# Programmi e linguaggi

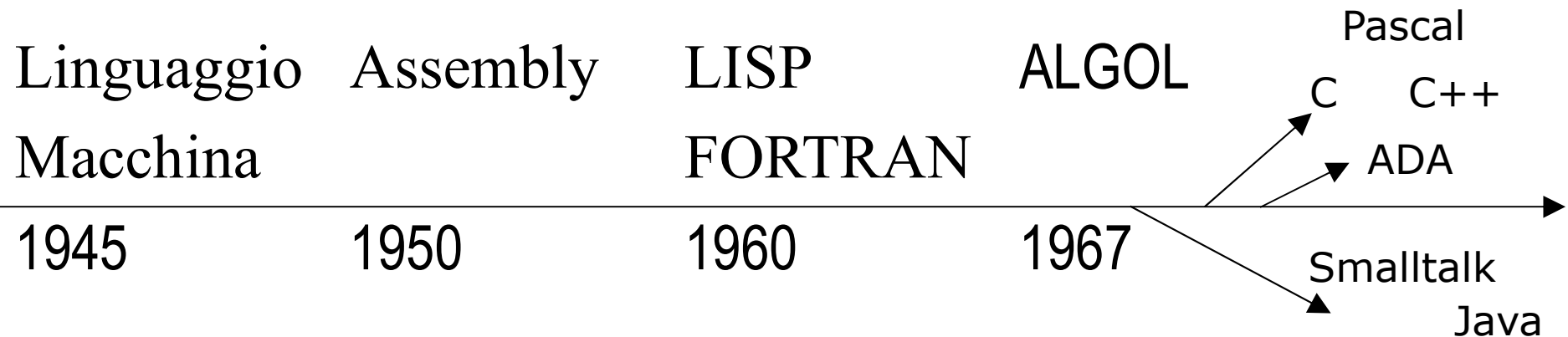
- ❑ Ogni testo è scritto in un qualche linguaggio
- ❑ I programmi sono scritti in un linguaggio specializzato detto

## **linguaggio di programmazione**

```
01001010010010100
11100100110011100
00110111100001010
11001101001000110
01001010010010100
11100100110011100
00110111100001010
11001101001000110
```

```
class Nothing {
    public static void main (String[] arg) {
    }
}
```

# 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.
  
- Cominciamo con il linguaggio macchina

# Un esempio ... in linguaggio macchina

---

Vogliamo sommare il contenuto dell'indirizzo 50 della RAM  
con il contenuto dell'indirizzo 24 e mettere il risultato  
nell'indirizzo 34 della RAM stessa.

```
00000010
00110010
00001000
00011000
00000100
00100010
```

# Un esempio

RAM	
0	0000 0010
1	0011 0010
2	0000 1000
3	0001 1000
4	0000 0100
5	0010 0010
...	
24	0000 0101
...	
34	0000 0000
...	
50	0000 1100
...	

Il programma viene inizialmente  
caricato in RAM;

il PC viene inizializzato  
all'indirizzo della prima  
istruzione.

Operazione  
di somma

RAM	
0	0000 0010
1	0011 0010
2	0000 1000
3	0001 1000
4	0000 0100
5	0010 0010
...	
24	0000 0101
...	
34	0001 0001
...	
50	0000 1100
...	

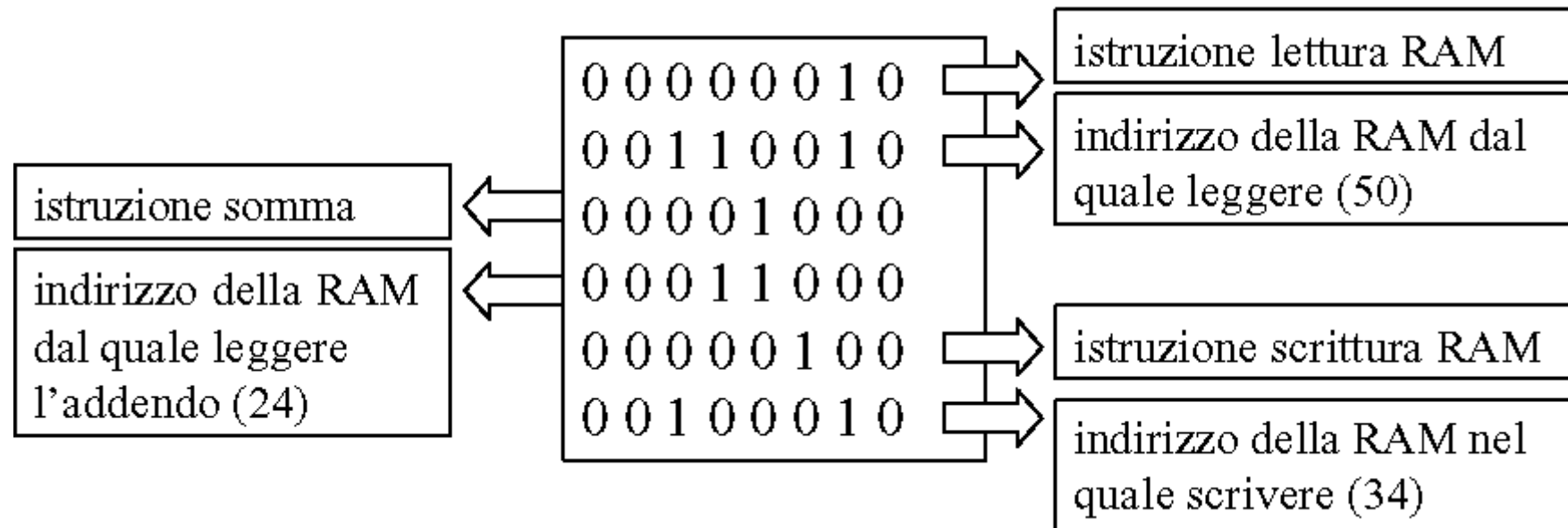


# L'operazione di somma tra due numeri in memoria non è elementare!

---

- La sequenza di operazioni da fare è:
  - Copia il contenuto della word 50 dalla RAM al registro ACC (accumulatore) della CPU;
  - 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.
- RICORDATE
  - I programmi che implementate devono comportarsi in modo *corretto* ... almeno per il contesto di interesse

# 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

---

- In un linguaggio ad *alto livello*:

```
if (a==b) then c=0
           else 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
```

# L'idea della traduzione

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



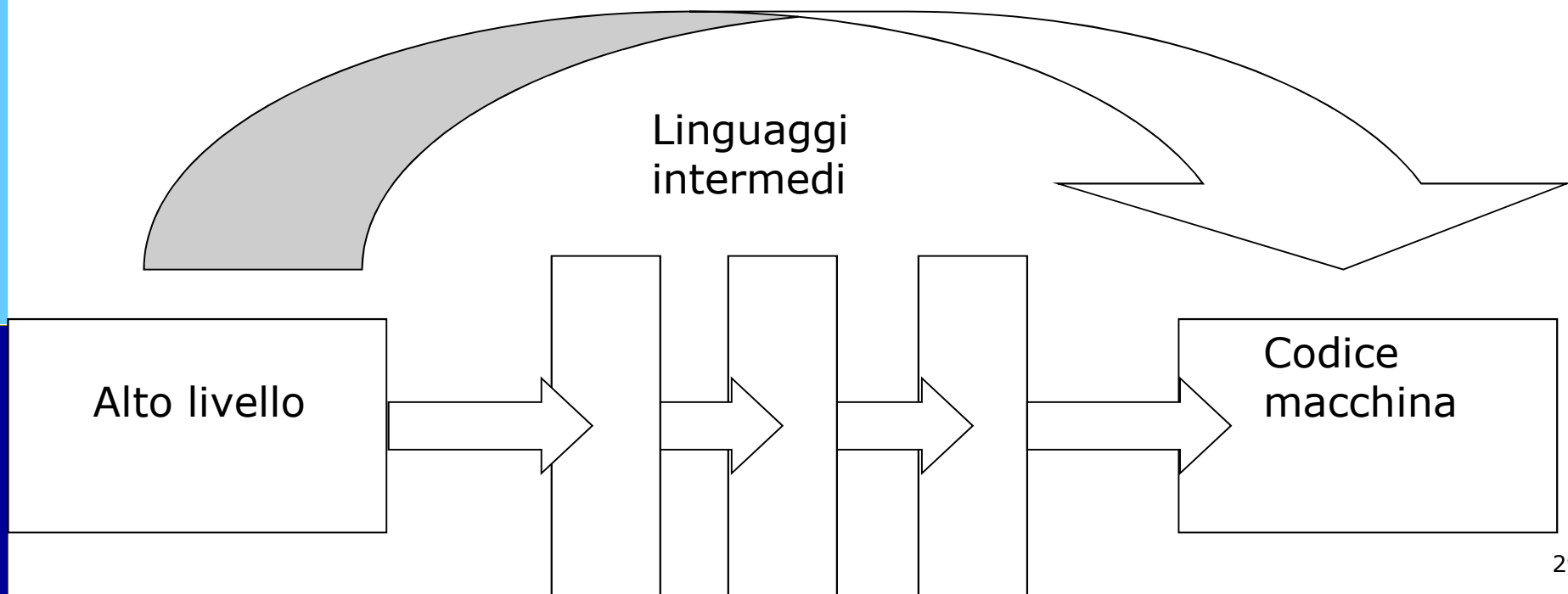
# Traduzione dei linguaggi

---

- 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.
  - Per renderlo eseguibile, interviene il processo di traduzione

# 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 dichiarare 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

I linguaggi si sono evoluti per facilitare il compito della programmazione.

In nuovi linguaggi includono tecniche, quali:

- **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”.
  - introduzione delle strutture di controllo

# ... si è giunti ...

---

## □ Linguaggi orientati agli oggetti:

SMALLTALK, EIFFEL, C++, JAVA.

- Ad ogni “entità” del problema da risolvere corrisponde un oggetto capace di memorizzare il suo stato, ovvero i dati che lo caratterizzano, 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 sono 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

# 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, ...**



# 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 specificano 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