UNIVERSITÀ
degli STUDI
di CATANIA

Traduzione

Corso di programmazione I (A-E / O-Z) AA 2025/26

Corso di Laurea Triennale in Informatica

Fabrizio Messina
fabrizio.messina@unict.it

Dipartimento di Matematica e Informatica

Linguaggi di programmazione

Un algoritmo va codificato mediante un opportuno **linguaggio di programmazione**.

Un linguaggio di programmazione rappresenta **una specifica di un insieme di istruzioni che possono essere usate per codificare programmi** e quindi per produrre dati in output.

- 1945: linguaggi macchina.
- 1950: linguaggi assembly.
- Dopo il 1950, linguaggi di **alto livello**:
  - **Astrazione**
  - **Semplificazione**
  - **Similarità con il ragionamento umano**.

Esempio E4.1: Assembly vs linguaggio di alto livello

Un programma C che fa..nulla!

```c
/* test.c */
int main(){
  return 0;
}
```

Si provi a generare codice assembly del codice mostrato prima con il seg. comando su piattaforma Linux (produrrà il file test.s)

$ gcc -S test.c

Esempio E4.1: Assembly vs linguaggio di alto livello

Contenuto (parziale) del file test.s (generato al passo precedente)

```
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $1, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

Paradigmi di programmazione

Un paradigma di programmazione rappresenta la **filosofia** con cui si scrivono i programmi.

Un **linguaggio si basa generalmente su un paradigma di programmazione**. Esso caratterizza:

- la **metodologia** con cui si scrivono i programmi (ES: strutture di controllo oppure procedure)
- il concetto di **computazione** (ES: istruzioni o funzioni matematiche)

Paradigmi di programmazione

Programmazione funzionale: Il flusso di esecuzione è una serie di valutazioni di **funzioni matematiche** (assenza di "side effect"!)

Programmazione logica: Descrivere la **struttura logica** del problema anziché il modo per risolverlo (logica del primo ordine).

Paradigmi di programmazione

Programmazione imperativa: Sequenza di **istruzioni** da "impartire" al calcolatore (ES: assegnazioni).

Programmazione strutturata: Si basa sul teorema di Bohm-Jacopini, quindi sui **tre costrutti** sequenza, selezione, iterazione (...e istruzione GOTO, come sarà detto in seguito) .

Programmazione procedurale: Insieme di **blocchi di codice sorgente** ben delimitati identificati da nome ed eventuali argomenti (funzioni/procedure).

Paradigmi di programmazione

Programmazione modulare: paradigma basato sulla modularità, ovvero strutturare un programma in **moduli ben separati con interfaccia ben definita**.

Programmazione orientata agli oggetti. Include aspetti di programmazione imperativa, strutturata, procedurale e modulare.

Programmazione imperativa

Insieme di istruzioni di natura **imperativa**

Esempio: “Leggi A”; “Stampa B”; “assegna il valore 5 alla variabile X”

Programmazione procedurale, strutturata, modulare, orientata agli oggetti usano istruzioni imperative

Paradigmi di programmazione

Programmazione strutturata

Il Teorema di **Bohm-Jacopini** costituisce la **base della programmazione strutturata**.

Strutture di controllo per **condizione** (if-then-else).

Struttura di controllo per **iterazioni** (while).

Sequenza di istruzioni.

Anche istruzione **GOTO** (assente/deprecata nei moderni linguaggi procedurali, a partire dal C!!).

Programmazione procedurale

Insieme di **procedure** o funzioni.

Ogni funzione contiene i) istruzioni di tipo imperativo e ii) costrutti della programmazione strutturata.

I GOTO sono **deprecati**

Side-effects! Il passaggio di parametri alle funzioni potrebbe avvenire per indirizzo quindi la funzione potrebbe modificare aree di memoria esterne ad esso.

Paradigmi di programmazione

Programmazione modulare

Tecnica di **design** del software.

Programma suddiviso in **moduli**.

Ogni modulo è ben **separato e indipendente** dall'altro.

Ogni modulo è dotato di **un'interfaccia ben definita**.

La programmazione OOP (Object Oriented Programming) è : il *modulo è una sorta di "antenato" della classe.*

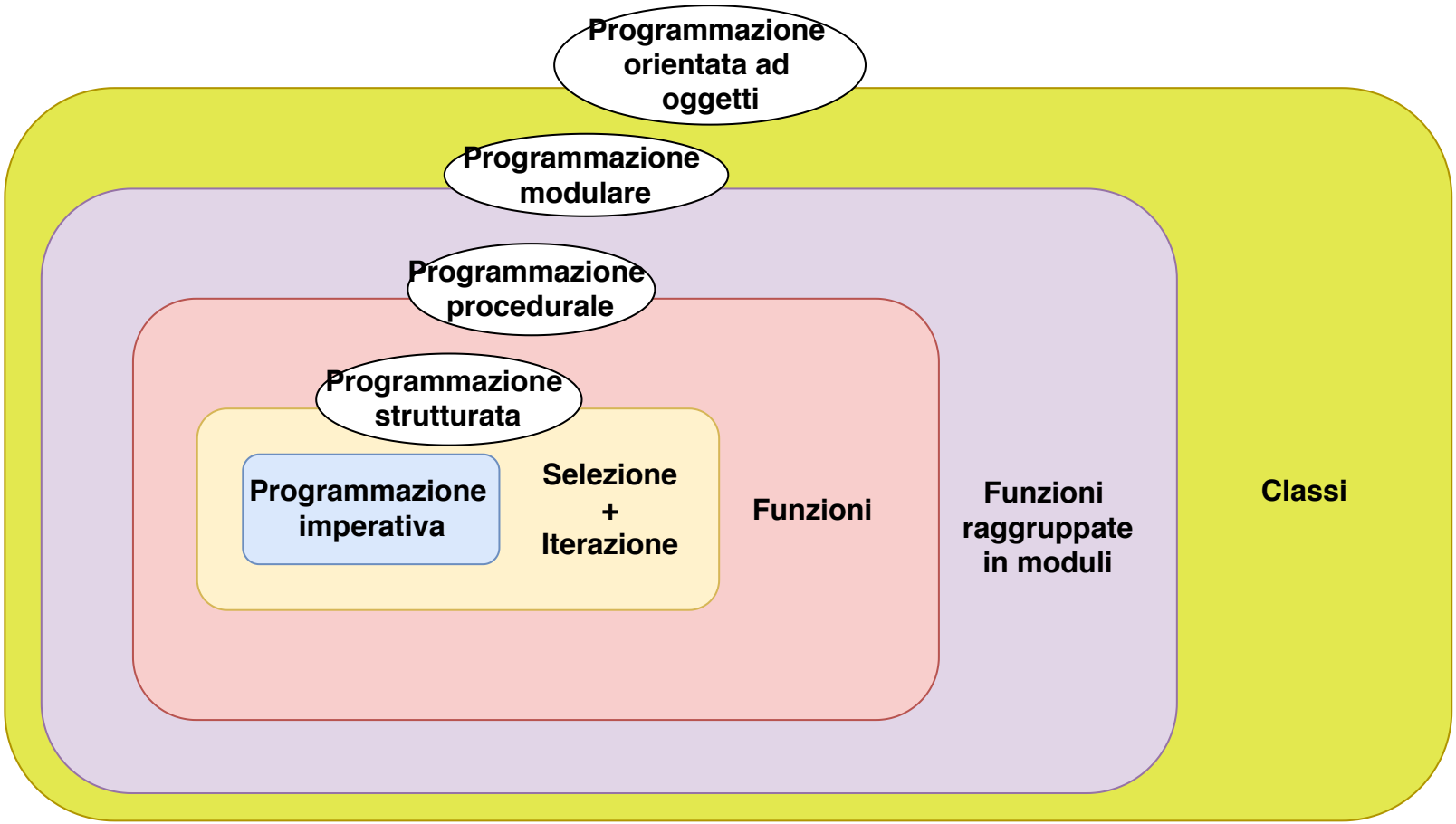
Programmazione OOP

OOP is Object-Oriented Programming.

Introdotto per migliorare **efficienza** del processo di **produzione** e **mantenimento** del software.

Discende dalla programmazione procedurale, favorisce la **modularità**.

Paradigmi di programmazione



Linguaggi di programmazione

I linguaggi di programmazione sono stati introdotti per **facilitare la scrittura dei programmi**.

Sono definiti da un insieme di regole formali, le regole grammaticali o **sintassi**.

La sintassi include la specifica di tutti i **simboli** da usare durante la scrittura di un programma.

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.

La correttezza sintattica **non implica** la correttezza semantica del programma.

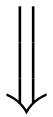
La seguente funzione C è sintatticamente corretta, ovvero il compilatore genera codice eseguibile.

```
1   int sum(int *a, short l){
2     int sm;
3     for(int i=0; i<l; i++)
4       sm+=a[i];
5     return sm;
6   }
```

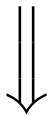
Tuttavia essa contiene un errore **logico**.

Traduzione

Programma composto di istruzioni in un linguaggio il più possibile simile a quello umano



TRADUZIONE



CODICE MACCHINA

Traduzione

Linguaggio di alto livello

Linguaggio dotato di un alto grado di espressività.

```
tot = var1 + var2
```

Linguaggio di basso livello

Linguaggio vicino al linguaggio macchina.

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

Traduzione

Grazie al concetto di traduzione:

- **evoluzione** dei linguaggi di programmazione verso sistemi simbolici più **espressivi**.
- molto più **agevole** scrivere programmi
- **indipendenza dalla piattaforma**: il programmatore scrive il programma senza preoccuparsi della piattaforma sottostante.

Traduttori

Un traduttore **genera** codice in **linguaggio macchina** a partire da codice scritto in un linguaggio di alto livello.

Tipi di traduttori

1. Interpreti
2. Compilatori

Traduzione

Un compilatore prende in input un codice sorgente e lo **traduce** in **codice oggetto**.

Programma P codificato in linguaggio di alto livello.

$\Downarrow$ TRADUZIONE

Programma Q codificato in linguaggio macchina equivalente a P.

Traduzione

C++, Pascal, C, Cobol, Fortran sono linguaggi compilati

Il compilatore deve "supportare" una certa architettura.

Traduzione

Un interprete prende in input un codice sorgente, come il compilatore, ma:

Operazioni di un interprete

- Viene **tradotta** la singola istruzione generando il corrispondente codice macchina.
- Si **esegue** il codice in linguaggio macchina e si passa alla istruzione successiva.

ES: Prolog, Lisp, VBasic. Java è un caso particolare...

Traduzione

Compilatori: pro e contro

(+) **Performance** (esecuzione): codice sorgente viene tradotto preventivamente, non durante esecuzione.
(+) **Efficienza**: compilatore non occupa risorse (memoria) durante esecuzione programma.
(+) **Non invasivo**. Compilatore **non installato su macchine di produzione**, ma su macchine per sviluppo.
(-) Portabilità su architetture differenti:

- Compilatore eseguito su architettura X produrrà codice macchina *compliant* a X.
- Affinchè il compilatore produca **codice per architettura** X **differente da architettura target**, bisogna operare "cross-compilazione" (necessita una tool-chain...!).

Interpreti: pro e contro

(-) **Performance** (run-time): esecuzione programma più lenta rispetto ad esecuzione nativa.
(-) **Efficienza**: interprete occupa risorse su macchina di produzione.
(-) **Invasivo**. Interprete va installato su macchine di produzione.
(+) **Portabilità**: il programmatore non deve preoccuparsi della architettura, interprete "nasconde" architettura della macchina.

Traduzione

Esempio: JAVA

Java fu introdotto nel 1995 dalla Sun Microsystem, Inc. (successivamente acquisita da Oracle)

Il linguaggio Java è basato sul paradigma OOP (Object Oriented Programming).

Java fu concepito per produrre programmi **facilmente trasportabili** su architetture differenti.

Inizialmente concepito anche per essere eseguito su browser (Applet).

Traduzione

Java: traduzione ed esecuzione

1. Il programmatore scrive il programma in linguaggio Java.

2. La compilazione produce un programma in **Java Bytecode**, simile al linguaggio macchina ma **indipendente** dalla architettura.

3. Il Bytecode viene eseguito mediante **interprete Java** sulle specifiche architetture.

JVM (Java Virtual Machine)

Il Bytecode viene interpretato dalla **JVM (Java Virtual Machine)**, un programma che viene eseguito sulla macchina di produzione.

La JVM legge il Bytecode ed **esegue le computationi corrispondenti** alla semantica delle istruzioni in Bytecode.

L'intrepretazione del Bytecode comporta un **overhead rispetto all'esecuzione di codice macchina**, ovvero risorse addizionali.

Traduzione

JIT (Just in Time Compiler)

Il compilatore Just-In-Time (JIT) fu concepito per **ridurre overhead** di JVM.

Un metodo eguito dalla JVM viene **compilato in codice macchina dal JIT**, in modo da poter essere eseguito in secondo momento senza interpretazione.

NB: Il compilatore **JIT fa uso di risorse** (memoria e cpu).

$\Longrightarrow$ **Ottimizzazione**: un metodo non viene compilato la prima volta in cui viene caricato e interpretato dalla JVM, ma quando il **metodo stesso è stato eseguito un certo numero di volte**.

Riferimenti e approfondimenti

[1] $\rightarrow$ Capitolo 1 (in particolare 1.4)

---

[1] Paul J. Deitel and Harvey M. Deitel.
C Fondamenti e tecniche di programmazione.
Pearson, 2022.