UNIVERSITÀ
degli STUDI
di CATANIA

Traduzione

Corso di programmazione I

Corso di Laurea Triennale in Informatica

Prof. Giovanni Maria Farinella
Web: http://www.dmi.unict.it/farinella
Email: gfarinella@dmi.unict.it

Dipartimento di Matematica e Informatica

Linguaggi di programmazione

I linguaggi di programmazione sono classificati in tre livelli:

- linguaggi macchina (1945)
- linguaggi assembly (1950)
- linguaggi di **alto livello**: dal 1950 in poi, evoluzione verso:
  - Astrazione
  - Semplificazione
  - Similarità con il ragionamento umano.

Paradigmi di programmazione

Un paradigma di programmazione rappresenta la **filosofia** con cui si scrivono i programmi. 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)
- Ogni linguaggio si basa generalmente su un paradigma di programmazione.

Paradigmi di programmazione

Programmazione funzionale: Il flusso di esecuzione è una serie di valutazioni di **funzioni matematiche**.

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

Poco diffusi...

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.

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 funzionale

Il flusso di esecuzione del programma e' costituito da una serie di **valutazioni di funzioni matematiche**.

Assenza di side-effect: la funzione non modifica lo "stato" del mondo esterno (un altro sistema, un modulo, etc).

(+) Più facile verifica della correttezza.

(+) Assenza di bug.

(-) Poco vicino al nostro modo di concepire i sistemi.

Programmazione logica

Adozione della **logica del primo ordine** per rappresentare l'informazione e la sua elaborazione.

Il programmatore si concentra sugli **aspetti logici del problema**.

Logica del primo ordine: sistema formale di enunciati (proposizioni) e deduzioni logiche.

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

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 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.

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**.

Parte della programmazione OOP (Object Oriented Programming).

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**.

Oggetto del corso di programmazione I.

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. $X = 2;$

La correttezza sintattica non è condizione sufficiente per la correttezza semantica del programma.

Esempio di non correttezza semantica

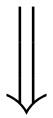
Programma che calcola la somma degli elementi in un array.

Dal compilatore nessun warning ne errori, genera correttamente il codice eseguibile sulla macchina X.

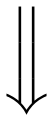
Scopriamo che il programma non somma l'ultimo elemento dell'array quindi non è semanticamente corretto.

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 grado di espressità che lo rende "vicino" al linguaggio umano.

```
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.

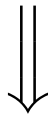
Tipi di traduttori

1. Interpreti
2. Compilatori

Traduzione

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

Programma P in linguaggio di alto livello



TRADUZIONE

Programma Q equivalente a P in linguaggio macchina

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.
(-) Protabilità 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.

Java concepito per produrre programmi che siano **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.

Traduzione

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 Bytecodes ed **esegue le computationi corrispondenti** alla semantica delle istruzioni in Bytecode.

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

Traduzione

JIT (Just in Time Compiler)

Il compilatore Just-In-Time (JIT) fu concepito **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.

Ma il compilatore **JIT fa uso di risorse** (memoria e cpu).

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**.