I linguaggi di programmazione

Programmazione
Corso di laurea in Informatica

Problem Solving

- Lo scopo di un programma è quello di risolvere un problema
- I passi generali per risolvere un problema sono:
  - Capire il problema
  - Scomporre il problema in sottoproblemi gestibili
  - Tracciare una soluzione e trovare gli algoritmi
  - Considerare alternative e raffinare la soluzione
  - Implementare la soluzione
  - Verificare la soluzione e correggere malfunzionamenti se ci sono

Problem Solving

- Molti progetti falliscono perchè lo sviluppatore non ha una chiara comprensione del problema
  - Il problema va ben specificato e le ambiguità vanno risolte
- Come i problemi e le loro soluzioni si fanno più complicati, è necessario organizzare il software in unità maneggiabili
  - Adottare una metodologia è fondamentale per lo sviluppo del sw
  - Le soluzioni che vedremo saranno concepite in unità chiamate ***classi*** e ***oggetti***, adottando un ***approccio orientato agli oggetti*** ***(object-oriented)***

I linguaggi di programmazione

- I linguaggi di programmazione sono stati introdotti per facilitare ai programmatori il compito di scrittura dei programmi
- Sono linguaggi ***simbolici***, in continua evoluzione
- Sono definiti da un insieme di regole formali, le regole grammaticali o ***sintassi***
- Diversi ***paradigmi di programmazione*** per affrontare
  - il processo della programmazione
  - la traduzione dell'algoritmo nel linguaggio adottato

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 cosa questa significhi (lo scopo dell'istruzione)
- Un programma sintatticamente corretto non è necessariamente semanticamente corretto
- I programmi fanno quello che prescriviamo che facciano e non quello che vorremmo che facessero

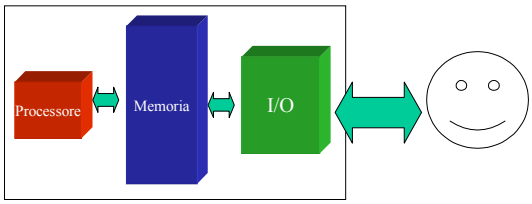
Sintassi

- Il sistema di regole formali che definisce il linguaggio
  - Le parole, i simboli e il modo di organizzarli
- Consente di stabilire se un'istruzione è ben formata
  - È corretto scrivere **se *x* + *n* > *m* allora STOP** ?
- Facilitano il compito al programmatore
- I programmi devono essere tradotti nel linguaggio nativo della macchina

Evoluzione dei linguaggi

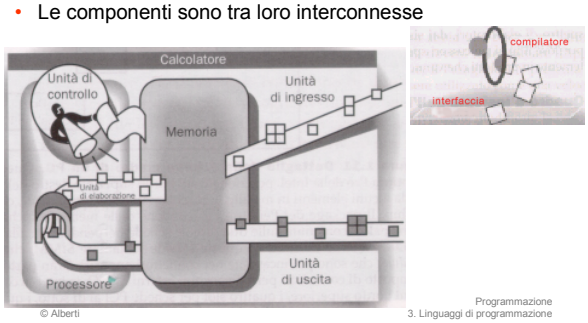
- La sfida degli anni '50 sulla traduzione dei linguaggi
- Evoluzione verso sistemi di codici complessi e potenti, orientati più all'uomo che alla macchina
- Linguaggi ad ***alto livello*** e a ***basso livello*** ovvero *i linguaggi macchina*

Schema dell'architettura



Schema delle interconnessioni

- Le componenti sono tra loro interconnesse



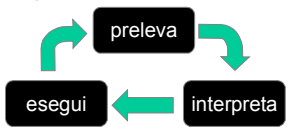
Il processore

- Il datapath o unità di elaborazione
  - L'insieme dei circuiti che operano e manipolano i dati
- Il controller
  - L'insieme dei circuiti che interpretano un programma e sovraintendono all'esecuzione delle istruzioni da parte delle altre componenti del calcolatore

Ciclo del processore

- Il processore esegue in continuazione il ciclo
  - *preleva-interpreta-esegui*

Preleva dalla memoria principale la prossima istruzione di un programma



Esegui l'istruzione **Decodifica l'istruzione**

Il linguaggio del processore

- Ogni modello di microprocessore ha un *proprio linguaggio macchina* diverso da quello di altri processori
- Ogni modello di microprocessore *riconosce* solo programmi scritti nel proprio linguaggio macchina
- Il linguaggio macchina contiene *tutte e sole le operazioni* che possono essere eseguite dal microprocessore

Linguaggio macchina

- Un processore con 2 registri `R1` e `R2`
- Si debbono sommare i contenuti delle locazioni di memoria `var1` e `var2` e archiviare in `tot`
- L'operazione di somma è possibile solo sui dati archiviati nei registri
  - Trasferisci il contentuto di `var1` nel registro `R1`
  - Trasferisci il contentuto di `var2` nel registro `R2`
  - Somma il contenuto dei due registri
  - Trasferisci il risultato nella locazione di memoria `tot`

Linguaggio macchina –2

- È necessario disporre di
  - Istruzioni per il trasferimento di dati dalla memoria ai registri e viceversa
  - Istruzioni aritmetiche/logiche
  - Eventualmente istruzioni di controllo
  - Esempio:

```
load R1, var1
load R2, var2
add R1, r2
store R1, tot
```


Diversi livelli di espressività

```
se a=b allora
     c:=0 altrimenti c:=a+b
          load R1, a
          load R2, b
          sub R1, R2
          jzero R1, fine
          load R1, a
          add R1, R2
fine:     store R1, c
```


Linguaggi di basso livello

- In un linguaggio macchina, ogni istruzione è una sequenza di cifre binarie
  - Totalmente illeggibile per l'uomo
  - Perfettamente non ambiguo per la macchina

```
0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1
0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0
```

traduzione delle istruzioni:

```
LOAD x
ADD y
STORE z
```


Le operazioni elementari

- Ogni istruzione del linguaggio macchina viene eseguita da un microprocessore svolgendo una serie di passi, le *operazioni elementari*
- Il numero di operazioni elementari necessario a portare a compimento un'istruzione in linguaggio macchina è dell'ordine di 7-10

Linguaggi macchina: problemi

- Sono specifici della macchina
- Occorre conoscere l'architettura della macchina per scrivere programmi
- I programmi non sono trasportabili
- I codici sono poco leggibili
- I programmatori si specializzano nel cercare efficienza su una macchina specifica, anziché concentrarsi sul problema

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 ***compilatore*** viene predisposto per una macchina specifica e poi traduce tutti i programmi scritti in uno specifico linguaggio ad alto livello

Programma compilatore

- Il compilatore traduce istruzioni scritte in un linguaggio ad alto livello, definite da regole sintattiche precise
- Necessità di strumenti formali per la descrizione delle regole (**tavole sintattiche**, **grammatiche**, **BNF**)
- Programma sorgente: il programma ad alto livello fornito dal programmatore al compilatore
- Programma oggetto: il risultato della traduzione del compilatore nel linguaggio obiettivo

Programma interprete

- Un interprete riceve un programma sorgente e oltre a tradurlo lo esegue istruzione per istruzione in un ciclo continuo
- I linguaggi compilati
  - Pascal, C
- I linguaggi interpretati
  - Lisp, Prolog

Descrivere le regole sintattiche

- Formalismi per scrivere le regole sintattiche dei linguaggi:
  - **Grammatiche**
  - **Forma di Backus-Naur** (BNF)
  - **Tavole sintattiche** o grafi sintattici

Grammatica

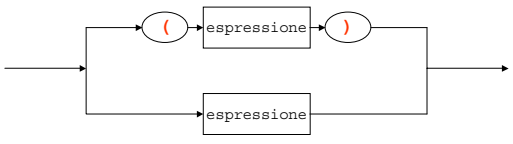
- Un simbolo iniziale, simboli terminali, simboli non terminali e regole di sostituzione

```
<frase>     ::= <soggetto> <predicato>
<soggetto>  ::= Paolo | Francesca
<predicato> ::= dorme | legge
```

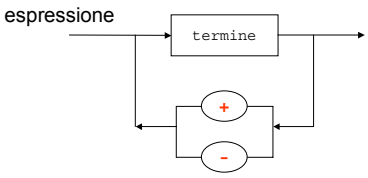
- Consente di ***produrre*** frasi del linguaggio e di ***decidere*** quali frasi vi appartengono
  - `Paolo legge`, `Francesca dorme`
  - Ma non `legge Paolo`, `Dario dorme`

I grafi sintattici

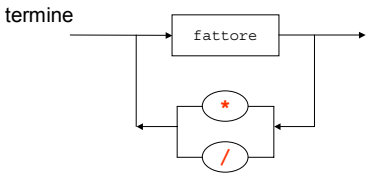
Per valutare un'espressione



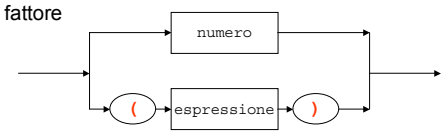
I grafi sintattici



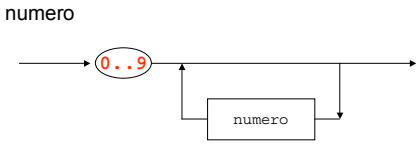
I grafi sintattici



I grafi sintattici

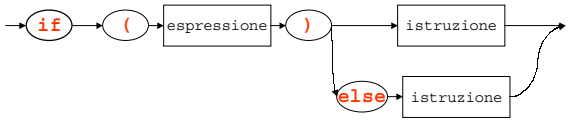


I grafi sintattici



I grafi sintattici

Istruzione condizionale



Evoluzione dei linguaggi – ‘50

- I primi linguaggi ad alto livello
  - **FORTRAN**, introduce il concetto di sottoprogrammi che operano su dati comuni
  - **ALGOL**, introduce il concetto di struttura dei programmi e di procedure *ricorsive*, ovvero che richiamano sè stesse
  - **COBOL**, introduce il concetto di FILE e di descrizione dei dati

Evoluzione dei linguaggi – '60

- Notazioni per la descrizione dei linguaggi
- Meno enfasi sull'efficienza, attenzione al modello di computazione
- I linguaggi orientati al problema
  - **LISP**, uniformità tra dati e programmi e un paradigma di programmazione basato sul concetto di funzione
  - **APL**, linguaggio matematico, ricco di notazioni e operatori per operare su strutture come vettori e matrici
  - **SNOBOL**, offre strumenti utili per la manipolazione di sequenze di caratteri

Evoluzione dei linguaggi – '70

- Metodologia di programmazione
  - **PASCAL**, ha lo scopo di insegnare la programmazione strutturata, una possbile risposta alla necessità di programmare con un metodo. Segue **Modula-2** che introduce il concetto di modulo.
  - **C**, linguaggio ad alto livello con visibilità e accesso alla macchina
  - **Prolog**, linguaggio basato sulla logica, non convenzionale diventato di nicchia

Evoluzione dei linguaggi – '80

- Programmazione in grande, esigenza di modularità e di astrazione
  - La programmazione a oggetti. Una classe definisce un insieme di oggetti e le procedure per manipolarli
  - **SmallTalk**, **Objective-C**, **C++**

I paradigmi di programmazione

- Forniscono la filosofia e la metodologia con cui si scrivono i programmi
- I linguaggi devono *consentire* ma soprattutto *spingere* all'adozione di un particolare paradigma
  - Procedurale
  - Funzionale
  - Modulare
  - Orientato agli oggetti

Paradigma procedurale

- Enfasi sulla soluzione algoritmica dei problemi mediante modifica ***progressiva*** dei dati in memoria
  - Esecuzione sequenziale di istruzioni
  - Cambiamento dello stato di memoria (le variabili) tramite assegnamento
  - Programmazione per effetto collaterale (*side-effect*)
- Aderenti al modello della macchina di von Neumann
- Molto efficienti
- Ha mostrato limiti nello sviluppo e mantenimento di sw complessi
- I linguaggi **imperativi**: **Pascal**, **C**

Influenza del modello di macchina

- **Concetto di istruzione**
  - l'unità di base del programma, memorizzata in successive celle di memoria
- **Concetto di sequenzialità e iterazione**
  - Il programma assolve il compito eseguendo le istruzioni in sequenza
  - Presente in diversi costrutti dei linguaggi e in tutto il processo di esecuzione
- **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

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

La macchina di Von Neumann

- Il calcolatore dello I.A.S. (Princeton, 1952): Von Neumann, Goldstein, Burks …
- Primo modello con programmazione e memorizzazione del programma
  - Organo aritmetico-logico (oggi CPU)
  - Memoria
  - Organo di controllo
  - Organo per la gestione input/output

Paradigma funzionale

- Primo tentativo di non rifarsi al modello di macchina di von Neumann
- La computazione avviene tramite funzioni che applicate ai dati riportani nuovi valori
  - Ogni funzione è un modulo a sé dipendente unicamente dal valore dei suoi argomenti
  - L'effetto globale è ottenuto concatenando opportunamente funzioni anche richiamando sé stesse (*ricorsione*)
  - Modello che si rifà alla teoria delle funzioni ricorsive
  - Scarso supporto ai costrutti di ripetizione tramite iterazione
- **Lisp**, **ML**

Componenti dei linguaggi funzionali

- Un insieme di funzioni primitive
- Una legge di **composizione** di funzioni
- Una legge di **applicazione** di una funzione ai suoi argomenti
- Un insieme di oggetti su cui operare

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**

Esempi

- Il problema di sommare i numeri dipsari in un insieme dato
  - In **Pascal** codice
  - In **APL** codice
  - In **Lisp** codice
  - In **Logo** codice
  - In **Forth** codice

Esecuzione in FORTH - 1

		0		2								
	0	4		9	1		2					
4	4	0	9	9	9		7	1		2		
9	9	9	0	0	0	9	7	7		34	0	
7	7	7	7	7	7	7	9	9	16	34	34	
34	34	34	34	34	34	34	34	34	34	16	16	
23	23	23	23	23	23	23	23	23	23	23	23	

Esecuzione in FORTH - 2

	2		
	23	1	
16	23	23	
23	16	16	39

Paradigma a oggetti

- Specifica il concetto di modulo che incapsula i dati con le classi
- Le classi hanno anche una struttura gerarchica e ereditano caratteristiche e funzionalità
- Obiettivo: migliorare l'efficienza del processo di produzione e mantenimento del software

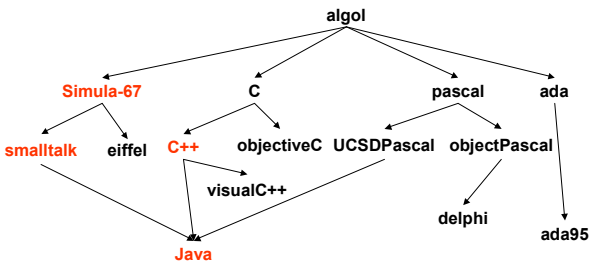
Concetti base della programmazione OO

- **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

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

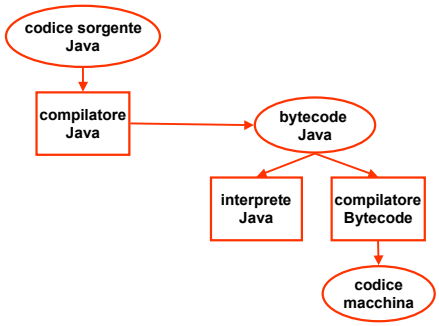
Pedigree di Java



Traduzione e 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
- Il compilatore Java non è legato ad una particolare macchina
  - Java è *indipendente* dall'architettura della macchina

Traduzione e esecuzione di Java



Ambiente di sviluppo

- Ci sono diversi ambienti di sviluppo per programmi Java:
  - Sun Java Software Development Kit (SDK)
  - Borland JBuilder
  - MetroWork CodeWarrior
  - Microsoft Visual J++
  - Symantec Café
- I dettagli operativi di questi ambienti sono diversi, ma il processo di compilazione ed esecuzione è sostanzialmente identico

Struttura del programma Java

- Ogni programma Java è una raccolta di una o più ***classi***
- Una classe contiene uno o più ***metodi***
- Un metodo contiene le *istruzioni*
- Ogni programma deve avere una e una sola classe contenente il metodo speciale `main`
- `Manzoni.java`

Struttura del programma Java - 2

```
//  commenti sulla classe
public class Mio_programma
{

}
```

Intestazione della classe

Corpo della classe

I commenti possono essere aggiunti ovunque

Struttura del programma Java – 3

```
//  commenti sulla classe
public class Mio_programma
{
   //  commenti sul metodo
   public static void main (String[] args)
   {
   }
}
```

Intestazione del metodo

Corpo del metodo

Commenti

- I commenti di un programma sono spesso chiamati *documentazione inline*
- Devono essere inclusi per documentare lo scopo e le funzionalità del programma
- Non ne influenzano il funzionamento
- Vengono trascurati dal compilatore
- Possono avere due forme:

```
// commenti fino alla fine della riga
/*  commento che
    può stare su più righe        */
```


Spazi vuoti

- Gli spazi, righe vuote e le tabulazioni sono chiamati *spazi bianchi*
- Gli *spazi bianchi* sono usati per separare le parole e I simboli di un programma
- Gli *spazi bianchi* vengono ignorati dal compilatore
- Un programma Java può essere formattato come si desidera con gli spazi bianchi
- La formattazione migliora la leggibilità di un programma e va usata in modo consistente
- Es `Manzoni 2.java` e `Manzoni 3.java`

Errori

- Errori di sintassi, che vengono intercettati dal compilatore (***errori di compilazione***)
  - Se c'è un errore durante la fase di compilazione, non viene creato un programma eseguibile
- Errori generati durante l'esecuzione (***errori d'esecuzione***)
  - Tentativi di divisione per zero, che causano la fine anomala del programma
- Errori che producono risultati diversi da quelli desiderati (***errori logici***)

Errori sintattici o di compilazione

- Nel programma `Manzoni.java` si sostituisca `System.out.println ("il cielo di …");` con le seguenti espresioni
- `System.aut.println ("il cielo di …");`
- `System.out.println ("il cielo di … );`
- `System.out.println ("il cilo di …");`

Errori sematici

- Nel programma `Divisione.java` si provi a passare in input `0` come divisore
- In esecuzione si genera il messaggio:

```
java.lang.ArithmeticException: /by zero
    at Divisione.main(Divisione.java:26)
Exception in thread "main" Process
Exit...
```


Errori logici

- Si vuole calcolare il MCD tra due numeri
- Si fornisce il programma `MCD errato.java`
  - Non si ottengono errori in compilazione,
  - né errori in esecuzione,
  - ma non si ottiene neanche il risultato voluto.