1: Linguaggi di Programmazione

- paradigmi linguistici, costrutti
- semantica operazionale
- implementazione, strutture a tempo di esecuzione

- testi di consultazione
  - T.W. Pratt & M.V. Zelkowitz, Programming languages. Design and Implementation. Prentice-Hall, 1996
  - M. Gabbrielli & S. Martini, Linguaggi di programmazione – Principi e paradigmi, McGraw-Hill 2006.

2. Metodologie di Programmazione Object-Oriented

- tecniche per la programmazione orientata ad oggetti (in piccolo)
  - specifica, implementazione, dimostrazioni
- esemplificate utilizzando il linguaggio Java

- testo di riferimento (prima parte):
  - Barbara Liskov, Program Development in Java, Abstraction, Specification and Object-Oriented Design, Addison-Wesley 2001

Struttura 1 (linguaggi)

- macchine astratte, interpreti, compilatori, implementazioni miste
- cenni di semantica operazionale
  - il linguaggio di specifica-implementazione (Ocaml)
- tipi di dato, tipi di dato astratti, tipi
- espressioni e comandi
- ambiente, dichiarazioni, blocchi
- sottoprogrammi, regole di scoping, passaggio di parametri

Struttura 2 (linguaggi)

- classi e oggetti
- gestione dell'ambiente: implementazione
- gestione della memoria: implementazione
- ambiente globale, moduli, compilazione separata
- struttura della macchina intermedia: esempi

Spirito 1 (linguaggi)

- definizioni “formali” e realizzazioni sempre implementate
  - utilizzando un linguaggio (Ocaml) adatto sia alla specifica che all’implementazione
- trattando un linguaggio “didattico” orientato ad oggetti, che ha come frammenti un linguaggio funzionale puro ed un linguaggio imperativo puro “classici”
  - il linguaggio viene introdotto in modo incrementale
    - sia rispetto ai costrutti
    - che rispetto ai paradigmi
  - il linguaggio “didattico” è a sua volta molto simile a Ocaml
- dalla semantica operazionale al codice compilato attraverso metodi sistematici

Spirito 2 (linguaggi)

- accanto al filone principale, ci saranno delle "digressioni" (indicate esplicitamente) per descrivere
  - costrutti e meccanismi diversi da quelli del linguaggio didattico
  - implementazioni alternative
- non sempre le digressioni saranno descritte in modo "eseguibile"
- le digressioni riguarderanno spesso i linguaggi "veri"
  - sia quelli 'vivi"
  - che alcuni reperti archeologici importanti
- dei paradigmi linguistici importanti
  - ignoreremo completamente i linguaggi (e i costrutti) per la programmazione concorrente
  - tratteremo in modo molto superficiale la programmazione logica
    - perché non "omogenea" agli altri paradigmi

Spirito 3 (linguaggi)

- come vedremo, esistono un insieme di concetti semantici e di strutture di implementazione in termini dei quali si descrivono in modo naturale linguaggi diversi e loro implementazioni
  - ed esiste una chiara relazione tra concetti semantici e strutture di implementazione
- tali concetti e strutture mettono in evidenza ciò che è comune ai vari linguaggi e possono essere trattati a prescindere dal particolare linguaggio
- per questa ragione, non tratteremo mai alcun linguaggio "vero"
  - in particolare, ignoreremo completamente le differenze legate alle specifiche sintassi

Spirito 4 (linguaggi)

- il livello di descrizione non sarà mai quello di un “manuale d’uso”, che quasi sempre non contiene
  - nè una descrizione formale della semantica indipendente dall’implementazione
    - necessaria per poter ragionare sul significato dei programmi che scriviamo
  - nè una descrizione delle strutture a tempo di esecuzione della particolare implementazione
    - necessaria per ragionare sulla “performance” dei nostri programmi
- queste due cose (e la relazione fra di esse) sono l’oggetto di interesse di questa parte del corso

Spirito 5 (linguaggi)

- conoscere questi principi di semantica e di tecniche di implementazione consente di
  - migliorare la conoscenza del linguaggio che usate comunemente
    - perché quel meccanismo non è fornito o è particolarmente costoso?
  - migliorare il vostro “vocabolario di costrutti”
    - un costrutto che ci sarebbe utile, ma non viene fornito dal linguaggio, può spesso essere simulato
  - imparare agevolmente un nuovo linguaggio
  - scegliere il linguaggio più adatto alle vostre esigenze
    - o almeno avere argomenti tecnici per discutere con il capo che vi vuol fare programmare in COBOL!
  - progettare un nuovo linguaggio o estenderne uno esistente
    - capita più spesso di quanto possiate immaginare!

Struttura 1 (Java)

- programmazione come decomposizione guidata da astrazioni
  - meccanismi di astrazione: parametrizzazione, specifica
  - tipi di astrazione: procedure, tipi di dato astratti, iterazione astratta, gerarchie di tipi
- introduzione informale di Java
  - classi, oggetti, metodi, gerarchie
  - il modello di esecuzione

Struttura 2 (Java)

- astrazioni procedurali
- astrazioni sui dati
- iterazione astratta
- gerarchie di tipi
- polimorfismo

Spirito 1 (Java)

- metodologie di programmazione orientata ad oggetti
  - esemplificate utilizzando Java
  - non tutto Java e non solo Java
- un insieme di tecniche basate su vari tipi di astrazione
  - alcune supportate da Java in modo più o meno diretto
  - la più importante non è supportata da Java
    - l'astrazione attraverso specificazione
    - invece di (o in aggiunta a) codice Java
      - specifiche informali

Spirito 2 (Java)

- specifiche, implementazioni, dimostrazioni di "correttezza"
  - relazioni formali fra 2 specifiche, fra 1 specifica ed una implementazione, etc.
- le dimostrazioni sono tanto importanti quanto le implementazioni
- ogni meccanismo di astrazione ha associata una particolare sequenza di operazioni di specifica, implementazione e dimostrazione
  - che ci porterà ad utilizzare sottoinsiemi di costrutti Java "coerenti"
- non è compito di questo corso introdurre il linguaggio nella sua interezza
  - nè tanto meno le sue librerie (che vi imparate da soli, quando vi servono)

Linguaggi e astrazione

- i linguaggi di programmazione ad alto livello moderni sono il più potente strumento di astrazione messo a disposizione dei programmatori
  - che possono, con un solo costrutto del linguaggio, "rappresentare" un numero (anche infinito) di interminabili sequenze di istruzioni macchina corrispondenti
- i linguaggi si sono evoluti trasformando in costrutti linguistici (e realizzando una volta per tutte nell'implementazione del linguaggio)
  - tecniche e metodologie sviluppate nell'ambito della programmazione, degli algoritmi, dell'ingegneria del software e dei sistemi operativi
    - in certi casi perfino in settori di applicazioni (basi di dati, intelligenza artificiale, simulazione, etc.)
- di fondamentale importanza è stata l'introduzione nei linguaggi di vari meccanismi di astrazione, che permettono di
  - estendere il linguaggio (con nuove operazioni, nuovi tipi di dato, etc.) semplicemente scrivendo dei programmi nel linguaggio stesso

Un po' di storia dei linguaggi

- i linguaggi di programmazione nascono con la macchina di Von Neumann (macchina a programma memorizzato)
  - i programmi sono un particolare tipo di dato rappresentato nella memoria della macchina
  - la macchina possiede un interprete capace di fare eseguire il programma memorizzato, e quindi di implementare un qualunque algoritmo descrivibile nel "linguaggio macchina"
  - un qualunque linguaggio macchina dotato di semplici operazioni primitive per effettuare la scelta e per iterare (o simili) è Turing-equivalente, cioè può descrivere tutti gli algoritmi
- i linguaggi hanno tutti lo stesso potere espressivo, ma la caratteristica distintiva importante è il "quanto costa esprimere"
  - direttamente legato al "livello di astrazione" fornito dal linguaggio

I linguaggi macchina ad alto livello

- dai linguaggi macchina ai linguaggi Assembler
  - nomi simbolici per operazioni e dati
- (anni 50) FORTRAN e COBOL (sempreverdi)
  - notazioni ad alto livello orientate rispettivamente al calcolo scientifico (numerico) ed alla gestione dati (anche su memoria secondaria)
  - astrazione procedurale (sottoprogrammi, ma con caratteristiche molto simili ai costrutti forniti dai linguaggi macchina)
  - nuove operazioni e strutture dati (per esempio, gli arrays in FORTRAN, e i records in COBOL)
  - nulla di significativamente diverso dai linguaggi macchina

I favolosi anni ’60: LISP e ALGOL’60

- risultati teorici a monte
  - formalizzazione degli aspetti sintattici
  - primi risultati semantici basati sul λ-calcolo
- caratteristiche comuni
  - introduzione dell’ambiente
  - vera astrazione procedurale con ricorsione
  - argomenti procedurali e per nome
- ALGOL’60
  - primo linguaggio imperativo veramente ad alto livello
  - scoping statico
  - gestione dinamica della memoria a stack
- LISP (sempreverde, ancora oggi il linguaggio dell’A.I.)
  - primo linguaggio funzionale, direttamente ispirato al λ-calcolo
  - scoping dinamico
  - strutture dati dinamiche, gestione dinamica della memoria a heap con garbage collector

Estendere la macchina fisica o implementare una logica

- ALGOL'60, prototipo dei linguaggi imperativi
  - parte dalla struttura della macchina fisica
  - la estende con nuovi potenti meccanismi
- LISP, prototipo dei linguaggi logici e funzionali
  - parte da un calcolo logico (λ-calcolo)
  - ne definisce una implementazione sulla macchina fisica
- ne nascono concetti simili
  - non a caso basati sulla teoria
- gli approcci restano diversi e danno origine a due filoni
  - il filone imperativo
  - il filone logico

  che sono tuttora vitali

La fine degli anni ’60

- PL/I: il primo tentativo di linguaggio “totalitario” (targato IBM)
  - tentativo di sintesi fra LISP, ALGOL’60 e COBOL
    - fallito per mancanza di una visione semantica unitaria
- SIMULA’67: nasce la classe
  - estensione di ALGOL’60 orientato alla simulazione discreta
  - quasi sconosciuto, riscoperto 15 anni dopo

Evoluzione del filone imperativo

- risultati anni ’70
  - metodologie di programmazione, tipi di dati astratti, modularità, classi e oggetti
  - programmazione di sistema in linguaggi ad alto livello: eccezioni e concorrenza
- PASCAL
  - estensione di ALGOL’60 con la definizione di tipi (non astratti), l’uso esplicito di puntatori e la gestione dinamica della memoria a heap (senza garbage collector)
  - semplice implementazione mista (vedi dopo) facilmente portabile

Il dopo PASCAL

- C
  - PASCAL + moduli + tipi astratti + eccezioni + semplice interfaccia per interagire con il sistema operativo
- ADA: il secondo tentativo di linguaggio "totalitario" (targato Dipartimento della Difesa U.S.A.)
  - come sopra + concorrenza + costrutti per la programmazione in tempo reale
  - progetto ambizioso, anche dal punto di vista semantico, con una grande enfasi sulla semantica statica (proprietà verificabili dal compilatore)
- C++
  - C + classi e oggetti (allocati sulla heap, ancora senza garbage collector)

L’evoluzione del filone logico: programmazione logica

- PROLOG
  - implementazione di un frammento del calcolo dei predicati del primo ordine
  - strutture dati molto flessibili (termini) con calcolo effettuato dall’algoritmo di unificazione
  - computazioni non-deterministiche
  - gestione della memoria a heap con garbage collector
- CLP (Constraint Logic Programming)
  - PROLOG + calcolo su domini diversi (anche numerici) con opportuni algoritmi di soluzione di vincoli

L'evoluzione del filone logico: programmazione funzionale

- ML
  - implementazione del λ-calcolo tipato
  - definizione di nuovi tipi ricorsivi
  - i valori dei nuovi tipi sono termini, che possono essere visitati con un meccanismo di pattern matching (versione semplificata dell'unificazione)
  - scoping statico (a differenza di LISP)
  - semantica statica molto potente (inferenza e controllo dei tipi)
    - un programma "corretto" per la semantica statica quasi sempre va bene
  - gestione della memoria a heap con garbage collector
- HASKELL
  - ML con regola di valutazione "lazy"

JAVA

- molte caratteristche dal filone imperativo
  - essenzialmente tutte quelle del linguaggio più avanzato del filone, cioè C++
- alcune caratteristiche dei linguaggi del filone logico
  - gestione della memoria con garbage collector
- utilizza il meccanismo delle classi e dell'ereditarietà per ridurre il numero di meccanismi primitivi
  - quasi tutto viene realizzato con classi predefinite nelle librerie
- ha una implementazione mista (anch'essa tipica del filone logico)
  - che ne facilita la portabilità e lo rende particolarmente adatto ad essere integrato nelle applicazioni di rete

Lo strumento utilizzato nella prima parte del corso

- Ocaml (Objective CaML), una estensione, orientata ad oggetti (e con un frammento imperativo), di uno dei più importanti linguaggi funzionali (ML)
  - progettato ed implementato all'INRIA (Francia)
- l'implementazione (per tutte le piattaforme importanti) si può scaricare dal sito

  http://caml.inria.fr/
- il manuale on line al sito

  http://caml.inria.fr/ocaml/htmlman/index.html

Materiale didattico, esame, istruzioni per l'uso del corso

- il materiale didattico delle lezioni (in formato html e di presentazione powerpoint scaricabile) è disponibile sulla mia pagina web

  http://www.di.unipi.it/~levi/levi.html

  così come tutti i programmi Ocaml e Java che verranno discussi nelle esercitazioni
- esame = prova scritta + orale
  - ammissione all'orale con votazione >= 15/30 nello scritto
  - 2 prove intermedie che possono rimpiazzare la prova scritta
- consigli
  - seguire il corso (e soprattutto le esercitazioni), mantenendosi al passo con lo studio
  - partecipare (attivamente) alle esercitazioni
  - sostenere le prove intermedie