Introduzione al corso di Informatica 3

Angelo Gargantini

AA 2007/2008

Argomenti

- Concetti generali dei linguaggi di programmazione
  - Computabilità
  - Tipi e sicurezza dei tipi
  - Visibilità, decomposizione
  - Semantica
- Object Oriented programming
  - ereditarietà, polimorfismo
  - Java
  - C++

Docenti

- Io, angelo.gargantini@unibg.it
- + prof. Mario Verdicchio
  - Computabilità
  - C++

Testo per info3

- Concepts in Programming Languages (Cambridge Univ Press, 2002) John C. Mitchell
- consigliato l'acquisto
- (disponibile in pdf)
- leggetelo !!!
  - Programming Language Concepts, Ghezzi e Jazayeri, Wiley
  - Programming Languages, Sebesta, Addison Wesley
  - Advances in Programming languages, Finkel, Addison Wesley – si può scaricare

Progetto di Informatica 3

Obiettivo: imparare a progettare/**implementare** un programma largo in Java

- usare Java in modo approfondito
  - classi astratte, interfacce, metodi statici, ...
- usare le cose nuove di Java
  - generics, enumeration
- avanzate:
  - uso di librerie esterne
  - Junit
  - coverage con emma

.... coming soon

Altre info

- ricevimento
  - Martedì mattina
- Laboratori, pochissime volte
  - per fare esercizi
- Dal sito web cs.unibg.it/gargantini/
  - registro
  - Pdf delle lezioni
  - User: info32007
  - Password: 2007info3

Esame di Info3

- L'esame è composto da tre parti:
  - Esercizi pratici scritti (forse su PC)
  - Domande orali
  - Progettino
- Peso: scritto 50%, orale 30%, progetto 20%

Iniziamo ...
Linguaggi di programmazione

- Libro 1.3
- wikipedia
- C'è anche una lista
- C'è anche in time table ...

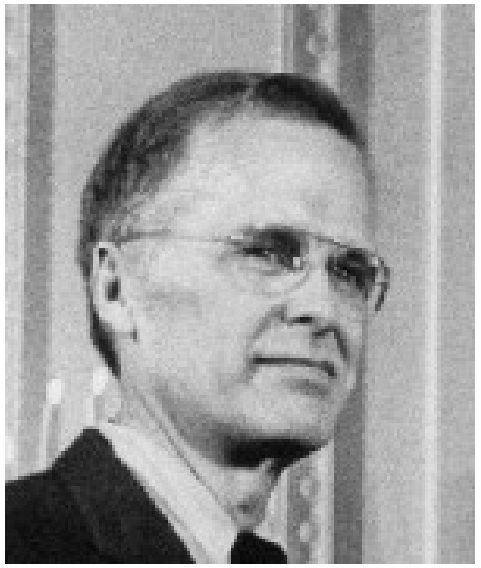
Programming paradigms

- Procedural/Imperative-style programming
  - FORTRAN, Algol, Pascal, C, ...
- Functional/Applicative-style programming
  - LISP, Scheme, ML, Haskell, ...
- Declarative/Logic programming
  - Prolog, ...
- Object-oriented programming
  - C++, C#, Java, ...
- Hybrids
  - concurrent, parallel, dataflow, intensional, domainspecific,
    ...
    scripting & extension languages

Key language milestones

- Assembly languages
  - invented by machine designers in the early 1950s
  - shift from binary machine code to mnemonics
  - first occurrence of reusable macros & subroutines
- FORTRAN - FORmula TRANslation
  - designed by John Backus at IBM in the mid-1950s
  - first high-level “algebraic” language with a compiler
- LISP - LISt Processor
  - designed by John McCarthy in 1958
  - first language to be based on the theory of recursive functions

FORTRAN



- John Backus, b. 1924
  - 1977 Turing Award
    On FORTRAN: “We did not know what we wanted and how to do it. It just sort of grew. The first struggle was over what the language would look like. Then how to parse expressions - it was a big problem and what we did looks astonishingly clumsy now.... “
- Defined BNF: “The syntax and semantics of the proposed international algebraic language of the Zurich ACM GRAMM conference.” ICIP Paris, June 1959.
  - influenced by Chomsky’s work on context-free grammars

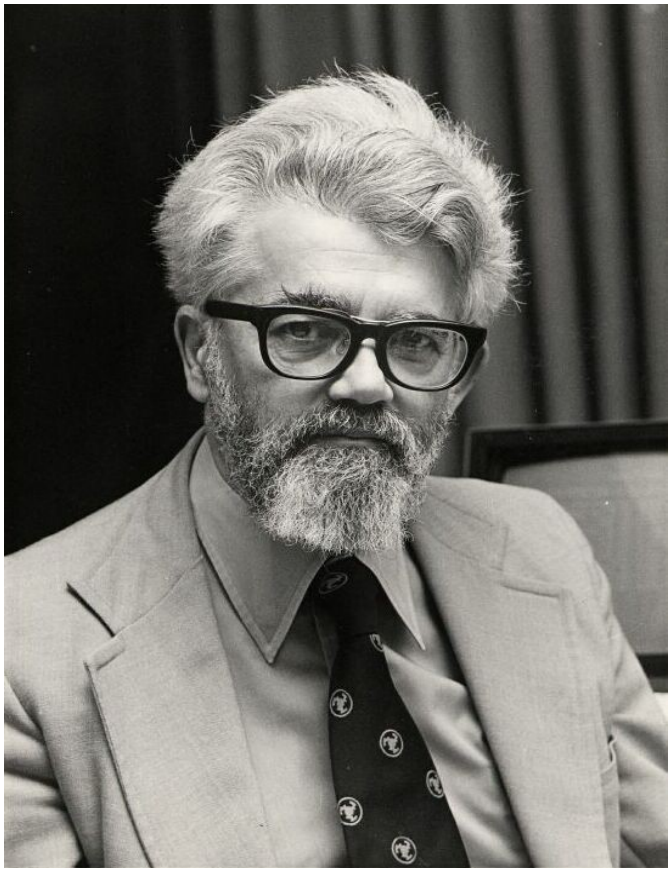
<letter> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<identifier> ::= <letter> | <identifier> <letter> | <identifier> <digit>

LISP

- John McCarthy, b. 1927
  - 1971 Turing Award
    “In the course of its development the LISP system went through several stages of simplification and eventually came to be based on a scheme for representing the partial recursive functions of a certain class of symbolic expressions.”
- Recursive Functions of Symbolic Expressions and their Computation by Machine, Part I, CACM, April 1960.



The roots of modern languages

- Algol 60 - International Algorithmic Language
  - designed by IFIP Working Group 2.1 in 1958-1960
  - earlier versions: IAL, Algol 58
  - John Backus, Peter Naur, John McCarthy, Alan Perlis & others
  - formally specified syntax using Backus-Naur Form (BNF)
  - significant influence on all of today's modern languages
  - introduced explicit variable type declarations, block structure (begin-end), nested lexical scopes & recursive procedures
- Pascal, Modula, Ada, C, C++, & Java are direct descendants of Algol
- Scheme adopted lexical scoping from Algol
- Simula 67 - first object-oriented language
  - designed by Ole-Johan Dahl and Kristen Nygaard
  - influenced all subsequent OO programming languages
  - objects & classes
  - inheritance (subtyping) & virtual methods (subtype polymorphism)

Other important languages

- Algol-like
  - Jovial, Euler, Pascal, Algol-68, Forsythe, Clu, Ada
- Functional
  - ISWIM, FP, SASL, Miranda, Haskell
  - LCF, ML, SML, Caml, OCaml
  - Scheme, Common LISP
- Object-Oriented
  - Smalltalk, Objective-C, C++, Eiffel, Modula-3, Self, C#, CLOS
- Logic programming
  - Prolog, Gödel, LDL, automated theorem provers (ACL2)
- Research-oriented
  - Dylan, ABCL/1, ACT, and literally hundreds more ...

Ada

- Primarily used by the US Dept of Defense
  - designed by a French language design team as
    part of an open competition
  - Named after Ada Byron (Lady Lovelace), 1815-1851
    - At a young age, Ada learned of Charles Babbage's ideas for a new calculating engine, the Analytical Engine. Babbage conjectured: what if a calculating engine could not only foresee but could act on that foresight. Ada was impressed by the universality of this idea. She suggested the idea of writing a plan for how this new calculating engine could be used to calculate Bernoulli numbers. This plan, is now regarded as the first "computer program.”
    - see the book: Ada, The Enchantress of Numbers, by Betty Alexandra Toole

Application specific languages

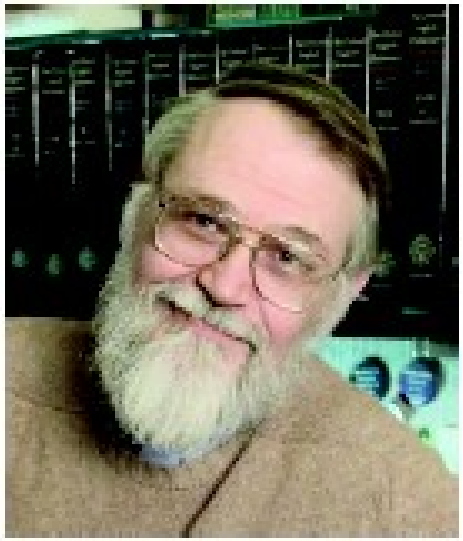
- Commercial data processing & database querying
  - Cobol, SQL, 4GLs, XQuery
- Systems programming
  - PL/I, PL/M, BCPL, BLISS, Modula, Modula-2, Oberon
- Specialized applications
  - BASIC, APL, Forth, Icon, Logo, SNOBOL4, GPSS, VisualBasic
- Concurrent, Parallel & Distributed
  - Concurrent Pascal, Concurrent C, C*, SR, Occam, Erlang, Obliq
- Command shells, scripting & “web” languages
  - sh, csh, tcsh, ksh, zsh, bash, ...
  - Perl, Php, Python, Rexx, Ruby, Tcl, AppleScript, VBScript, etc.
  - HTML/XML are markup languages not programming languages
  - but they often imbed executable scripts like Active Server Pages (ASPs) & Java Server Pages (JSPs)
- Programming tool “mini-languages”
  - awk, make, lex, yacc, autoconf, ...

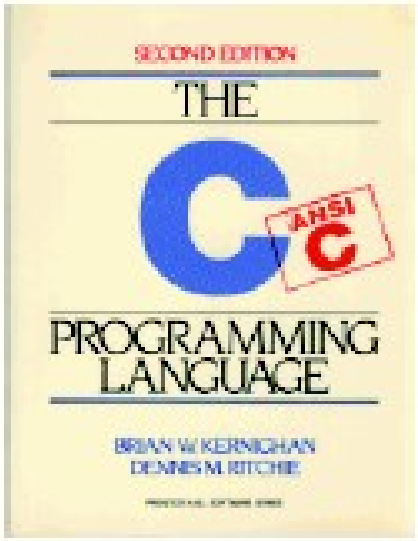
Cobol

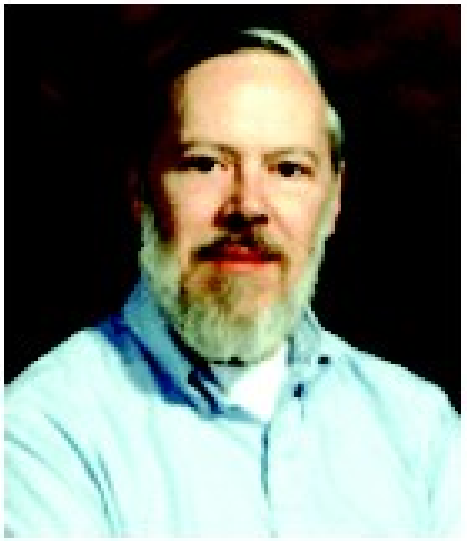
- Common Business Oriented Language
  - invented in the 1950's
  - primarily used for business data processing applications
  - billions spent to fix Y2K issues in old Cobol programs
- Admiral Grace Murray Hopper, 1906-1992
  - PhD Mathematics, Yale, 1934
  - joined the Navy in 1943 and worked at Harvard with Howard Aiken on the Mark I and Mark II computers
    - called the "mother of Cobol" for her contributions to the standardization of the language
    - credited with inventing an early compiler (1952)
    - She did this, she said, because she was lazy and hoped that "the programmer may return to being a mathematician."
    - Conference on Women in Computing is regularly held in her honor "ACM Grace Murray Hopper Award" see http://www.acm.org/awards for winners

From K&R C to ISO/ANSI C

- Brian Kernighan
  - also the ‘K’ in AWK
- Dennis Ritchie
  - 1983 Turing Award winner (with Ken Thompson)

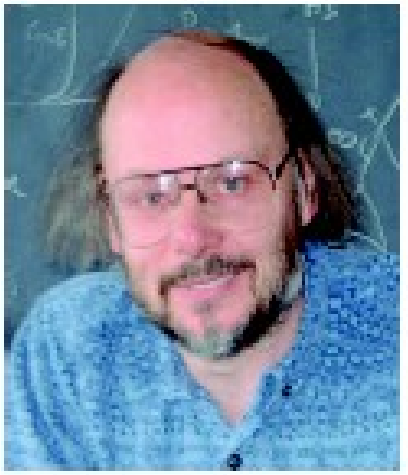


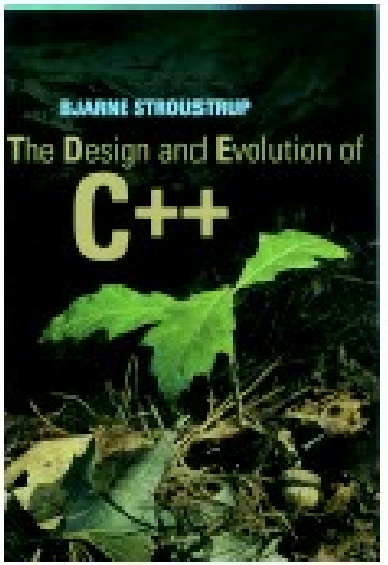




From K&R C to “C with Classes” to C ++

- Bjarne Stroustrup
  - Ph.D Univ. of Cambridge
  - used Simula in Ph.D. research and he knew about BCPL
  - then he went to Bell Labs & created “C with classes” in 1979
  - ‘++’ in C++ due to A. Koenig first “Cfront” translator from C++ to C around 1983
  - released to Universities in 1985-86





And then came Java...

- James Gosling (and “Duke”)
  - Gosling Emacs
  - Oak => Java
- Java is more influenced by C (syntax) and Modula-3 (object model) than by C++
  - Unlike C++
    - no operator overloading
    - no templates (but in Java 1.5)
    - no multiple inheritance
  - Like Modula-3
    - explicit interfaces
    - single class inheritance
    - exception handling
    - built-in threading model
    - references & automatic garbage collection (no pointers!)

From BASIC to C# to .NET

- The “un-Java” for Windows
  - an aside: the politics of language adoption
  - use of a programming language to win the mindshare of the software developer community to gain or maintain commercial market share
  - “open” language design/evolution process vs proprietary ownership of a language
  - this is not a new thing: IBM tried to do this with PL/I in the 60s, but free implementations appeared: e.g., PL/C - Cornell PL/I
  - C# has an interesting run-time environment
    - .NET CLR - common language runtime for Visual Basic, C++, C#, and future Microsoft languages

Why so many languages?

- In cosa differiscono tra di loro i linguaggi?
- Cosa hanno in comune?

Alcuni concetti in comune

- Variabile
- Istruzione
  - un comando, una funzione, oppure una regola descrittiva
- Espressione
- Strutture di controllo
  - Per governare il flusso dell'esecuzione del programma,
- Sottoprogramma
  - un blocco di codice che può essere richiamato da qualsiasi altro punto del programma.
- Strutture dati
  - meccanismi che permettono di organizzare e gestire dati complessi.

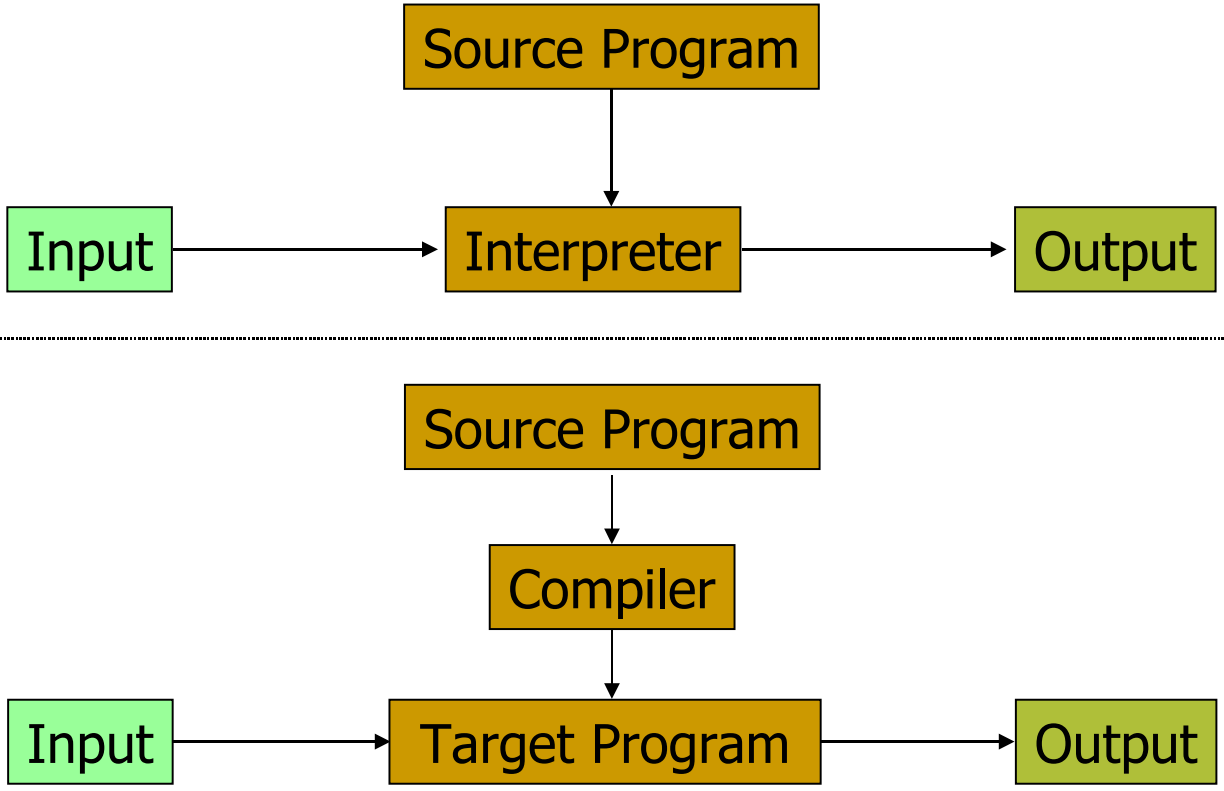
In comune – da approfondire

- Da approfondire invece abbiamo:
- Sintassi
  - Compilatore, ... --> Corso di linguaggi
  - Se volete, vedete il libro 4.1
  - Dovete sapere cosa fa un compilatore, interprete, ecc.
- Semantica di esecuzione
  - Semantica assiomatica - M7
- Tipi --> M1

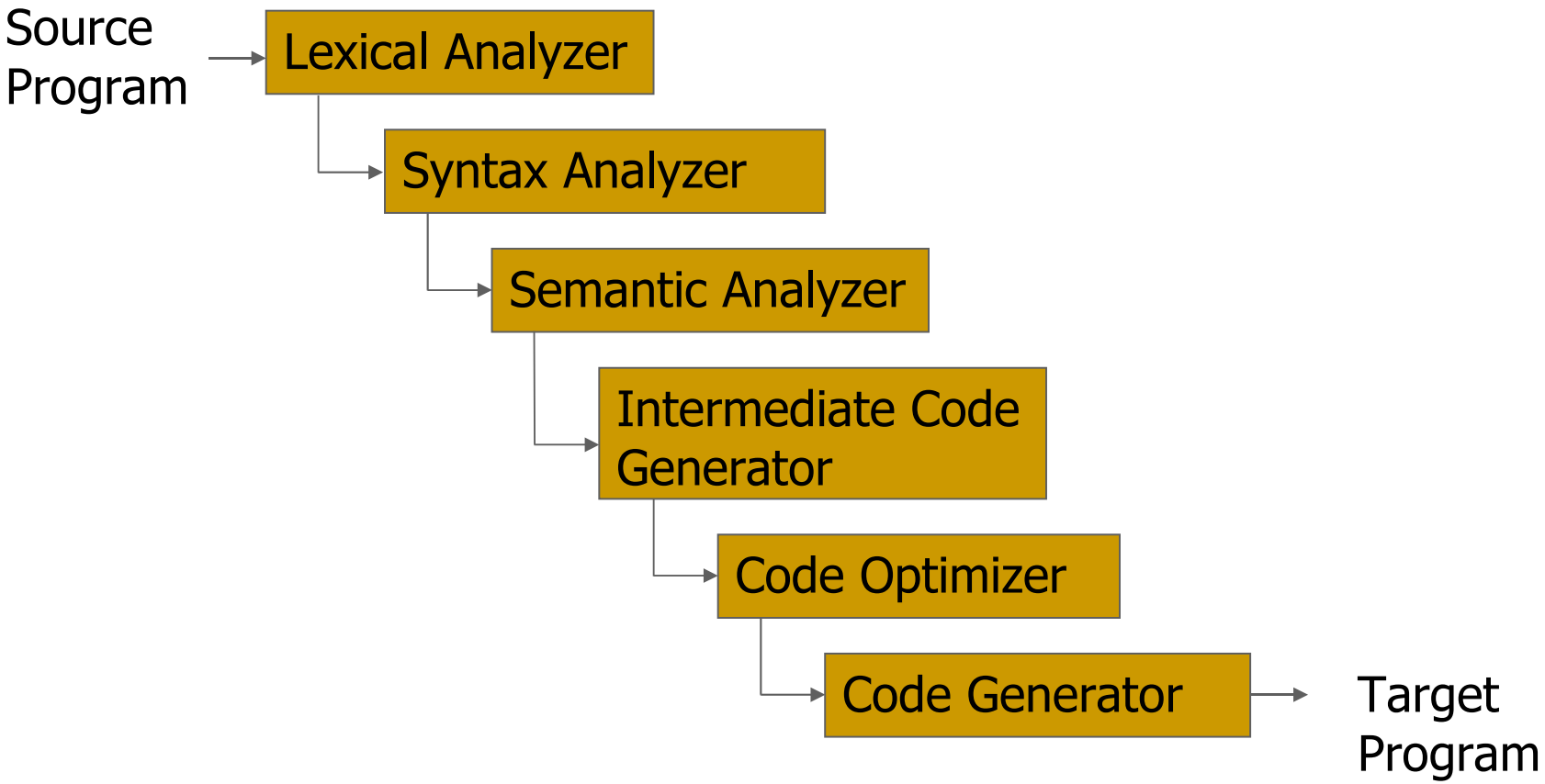
Syntax and Semantics of Programs

- Syntax
  - The symbols used to write a program
- Semantics
  - The actions that occur when a program is executed
- Programming language implementation
  - Syntax $\rightarrow$ Semantics
  - Transform program syntax into machine instructions that can be executed to cause the correct sequence of actions to occur

Interpreter vs Compiler



Typical Compiler



See summary in course text, compiler books

Syntax and correctness

- Not all the programs that are syntactically correct are correct
  - Correttezza di sintassi
    - Analisi statica
  - Con l'analisi statica posso trovare tutti gli errori nel mio programma?

Caso del C

```
[cairngorm:scrap] cat > crash.cc
void main() {
    ((int *) 5)[0] = 6;
}

[cairngorm:scrap] g++ crash.cc -o crash

[cairngorm:scrap]./crash
Segmentation fault
```

Java

```
[cairngorm:scrap] cat > Crash.java
class Crash {
    static public void main(String[] args) {
        ((int[]) 5)[0] = 6;
    }
}

[cairngorm:scrap] javac Crash.java
Crash.java:3: Invalid cast from int to int[].
        ((int[]) 5)[0] = 6;
          ^
```

Per cosa differiscono?

- Analisi
  - Analisi statica / analisi dinamica
    - Statica: analisi sul codice senza eseguirlo
    - Dinamica: durante l'esecuzione
- Espressività vs efficienza
- Le categorie sono:
  - Procedurali
    - Pascal, C, ...
  - Object oriented
    - Java, C#, ...
  - Funzionali
    - Lisp, ...
  - Logici
    - Prolog, ...

Linguaggi funzionali

- Sez 4.4. del libro
- In molti programmi la base sono le istruzioni imperative:
  - Es: x := 5
- Esistono anche dichiarazioni e definizioni ...
  - Es: f(int x) { return 5}
- Si pyò esprimere un programma con solo dichiarazioni di funzioni?
  - Funzione: in stile matamatico, dati certi input calcola un certo output
  - es. f(int x) = 5,   f(int x) = x *2 + g(x-3)
  - Sì --> LISP