Introduzione

Brief History of Programming Languages

“A programming language is a tool that has profound influence on our thinking habits”
-- -- Edsger Dijkstra

Angelo Gargantini

Università di Bergamo

What is a computation?

- Computation is an abstract and subtle mental notion
  - with a concrete realization by a “machine”
    - causing electrons/photons to change states/configurations
  - there are mathematical models of information & computation
- Effective transformation of inputs to outputs
  - for the class of Turing computable functions
  - “effective procedures” => algorithms
    - symbolic (syntactic) rewriting based on well-defined rules
    - (semantics) to compute (pragmatics) an answer (information)

The invention of the algorithm

See Knuth's Art of Computer Programming,
Vol. 1 - Fundamental Algorithms

- Euclid lived from ~325-265 BCE in Egypt (Alexandria)
- Euclid's Elements & Euclid's GCD algorithm
- Abu Ja'far Muhammad ibn Musa Al-Khwarizmi father of Ja'far , Mohammad, son of Moses,
native of Khwarizm lived from ~780-850 ACE in Persia (Baghdad)
wrote "Kitab al-jabr wa'l-muqabala"
"rules of equating & restoring"
- Etymology
Al-Khwarizm => algorism

Recent history (early 20th century)

Formal models of computation: 1900-1936

Church’s Thesis - all represent same class of computations

- Frege’s “concept script” (predicate logic)
- Church’s l-calculus
- Kleene’s recursive functions
- Turing’s abstract computing machine
- Curry’s combinatory logic
- Post’s production system

Turing machine

Principal notations for describing computations
TM: alphabet of tape symbols, r/w tape, set of states,
state transition functions,
imperative control mechanism

T = (Q, S, I, q0, F) = ({q0,q1,q2,q3, q4},{0,1,-},q0, {q4})
not programmer friendly!
ß l-calculus: l-terms, abstractions, and reduction rules
ß expression evaluation by parameter substitution & reduction
ß (lx.x+1) 2 -> 2+1 -> 3
ß lacks many practical features for programming a real computer
ß but Lisp, Scheme, Haskell, and ML make it a practical “calculus”

Logic, Turing Machines & Lambda Calculus

- Church’s thesis tells us that all these formalisms describe the same class of mathematical objects
  - i.e., the class of computable functions
  - choose the formalism best suited to the problem
    - Turing machines => imperative programming
    - focus on explicit state transitions and assignment
    - lambda-calculus => functional programming
    - pure expression evaluation and no assignment
    - predicate logic => logic programming
    - Horn clause resolution

What is a programming language?

- A formal notation for specifying an infinite number of computations
  - always requires an unambiguous syntax for the language
  - specified by a finite context-free grammar
  - should have a well-defined compositional semantics for each syntactic construct in the language
    - axiomatic vs denotational vs operational vs ad hoc
  - often requires a practical implementation: pragmatics
    - general purpose language versus a domain-specific language
- implementation on a real machine versus a virtual machine
  - efficiency vs portability
  - translation vs compilation vs interpretation
    - C++ was originally translated to C by Stroustrup’s Cfront translator
    - GNU g++ was the first native-code C++ compiler (by Michael Tiemann)
    - Java originally used a byte-code interpreter, but then just-in-time (JIT) compilers appeared, and now native code compilers are commonly used for greater run-time efficiency
    - Lisp, Scheme, and most other functional languages are interpreted by a virtual machine, but code is often pre-compiled to an internal executable form for efficient execution by the virtual machine

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
  - influenced by Church’s l-calculus notation
  - major influence on all subsequent functional languages as well as on Smalltalk

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

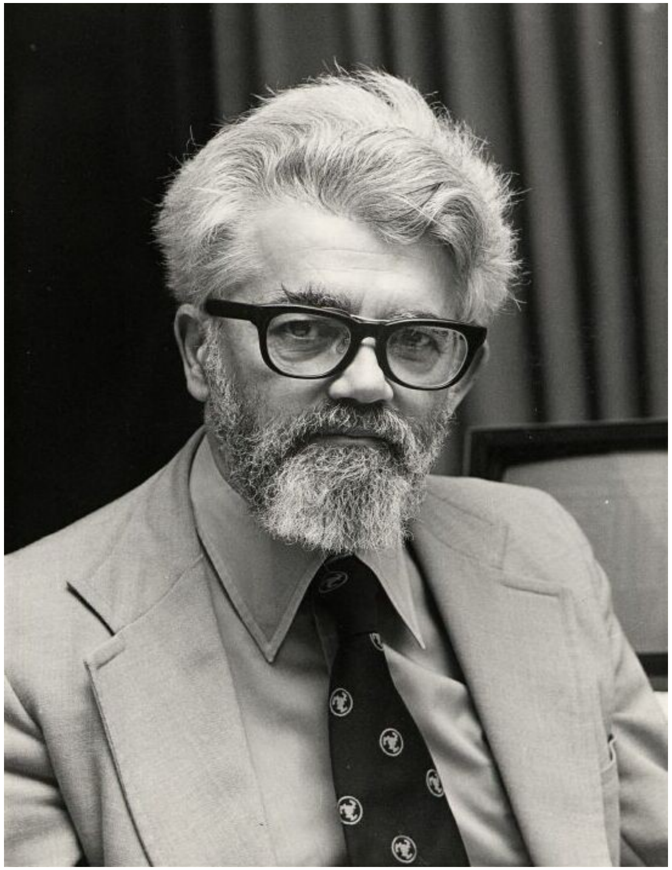
```
<letter> ::= a | b | c | d | e | f | g | h | i | j |
k | l | m | n | o | p | q | r | s |
t | u | v | w | x | y | z | A | B |
C | D | E | F | G | H | I | J | K |
L | M | N | O | P | Q | R | S |
T | U | V | W | X | Y | Z

<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)

Simula

Ole-Johan Dahl, 1931-2002 Kristen Nygaard, 1926-2002
joint 2001 Turing Award joint 2002 von Neumann Medal

SIMULA I (1962-65) and Simula 67 (1967) are the two first object-oriented languages.

- Simula 67 introduced most of the key concepts of object-oriented programming: both objects and classes, subclasses (usually referred to as inheritance) and virtual procedures, combined with safe referencing and mechanisms for bringing into a program collections of program structures described under a common class heading (prefixed blocks).

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
    - studied under the famous algebraist Oystein Ore
    - 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

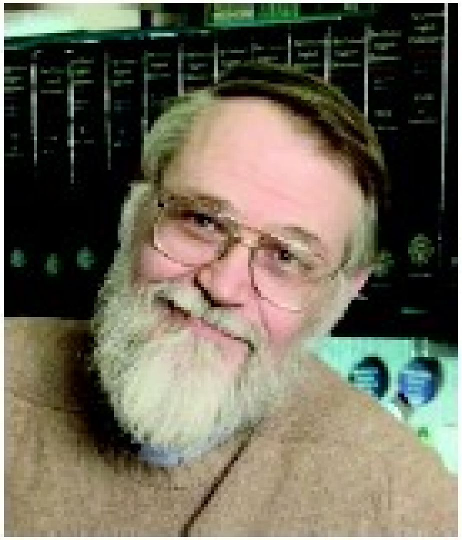
From K&R C to ISO/ANSI C

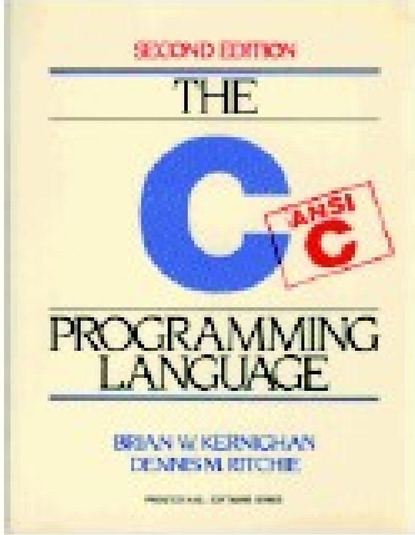
Brian Kernighan

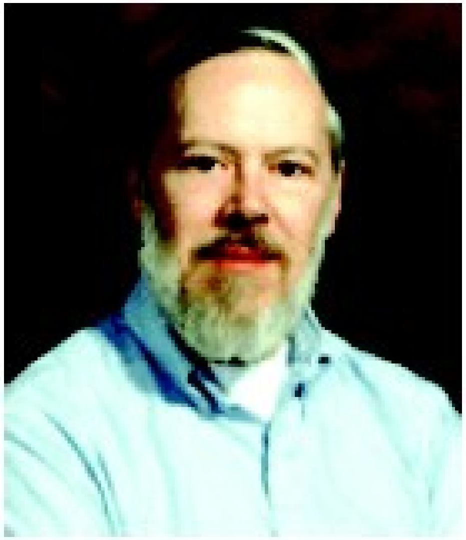
also the ‘K’ in AWK

Dennis Ritchie

1983 Turing Award winner (with Ken Thompson)



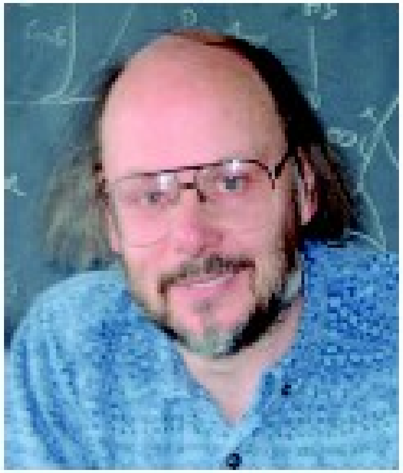


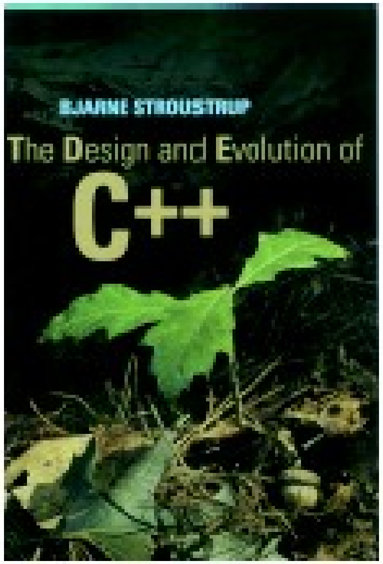


From K&R C to "C with Classes" to C++

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?

Language evolution versus revolution

- Are “new” languages really new?
  - first we must ask: “in what way is a language new”?
  - significantly improves upon 1st generation languages
  - e.g., better than Algol, Lisp and Simula in some key ways
- programmer productivity, program correctness, efficiency, reusability, extensibility, understandability, etc.
- evolutionary vs revolutionary progress since 1960
- Object-oriented? Functional? Logical?
  - Java is not really new - it “borrows” practically every feature from existing languages
    see http://java.sun.com/people/jag/green/index.html

The Von Neumann Bottleneck

- John von Neumann, 1903-1957
  - invented the concept of the stored program computer
  - based on the mathematical idea of Turing for a universal computing machine
- John Backus coined the term “von Neumann bottleneck” in his Turing lecture where he proposed a purely functional approach to programming
  - IEEE John von Neumann Medal is awarded annually for outstanding achievements in computer-related science and technology.

Language design

- What are good design criteria for a language?
- What do the experts say?
  - On the Design of Programming Languages, Niklaus Wirth
  - Hints on Programming Language Design, C.A.R. Hoare
  - Why Pascal is Not My Favorite Language, Brian Kernighan
- Lisp - Notes on its Past and Future, John McCarthy
  - Growing a Language, Guy Steele

What do all languages have in common?

- Lexical structure & analysis
  - tokens: keywords, operators, symbols, variables
  - ignore white space (i.e., _,\r,\n,\t) & comments
  - regular expressions & finite automata
  - lexical scanner generators, e.g., lex/flex
  - Syntactic structure & analysis
  - context-free grammars & parsing of syntactic phrases
  - LL(k) and LR(k) grammars
  - top-down vs bottom-up parser generators (e.g. ANTLR vs Yacc)
- Pragmatic implementation issues
  - lexical scopes, scope rules, block structure, local variables
  - procedures, functions, parameter passing, iteration, recursion
  - built-in types, type checking, strings, arrays, structures, etc.
- Semantics: what do programs mean and are they

Case Study: design and development of C

- Algol 68 & PL/I to BCPL -> B -> New B -> C
  - The Development of the C Language, Dennis Ritchie
    - A History of Algol 68, Charles Lindsey
    - PL/I as a Tool for Systems Programming, Fernando Corbató
    - The BCPL Reference Manual, Martin Richards
    - User’s Reference to B, Ken Thompson
    - C Reference Manual, Dennis Ritchie

Desiderata for Programming Languages

Expressiveness - Turing-completeness

But also a stronger kind of expressiveness - how easy it is to program simple concepts

Efficiency

Recursion in functional languages is expressive but sometimes inefficient

Simplicity - as few basic concepts as possible

Sometimes a trade-off with convenience (three kinds of loops in Pascal)

Uniformity and consistency of concepts

Why does **for** in Pascal require a single statement while **repeat** allows any number of statements?

Abstraction - language should allow to factor out recurring patterns

Clarity to humans

the distinction **=** *vs.* **==** in C a bit confusing

Information hiding and **modularity**

Safety - possibility to detect errors at compile time

Awk, REXX and SNOBOL type conversions are error prone

Informazioni sui corsi (Informatica 3 e progetto di informatica 3)

Obiettivo di Informatica 3

imparare alcuni nuovi pradigmi di programmazione e alcuni problemi piu' teorici per sapere programmare meglio e risolvere i problemi tramite programmi in modo piu‘ efficace e rigoroso.

- **Computabilita**': cenni macchina di Turing e problema dell'HALT
- **Sintassi** dei linguaggi
- Semantica **assiomatica** e correttezza di programmi
- **Type** systems: imparare un po' sui tipi, overloading, polimorfismo,
- **Object oriented**: familiarizzare con i concetti base dell'OO
  - C++:
  - OO Design Pattern
  - Java

Seguiremo l'ordine inverso

Testo

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

In laboratorio con me e Mario Verdicchio

Altre info

- ricevimento
    - lunedì dopo la lezione di info 2 e prima di info3
- laboratori
    - per fare esercizi
- sito web
    - Dal sito di unibg.it
- Esame: da decidere per Info3
- per progetto: il progetto + orale