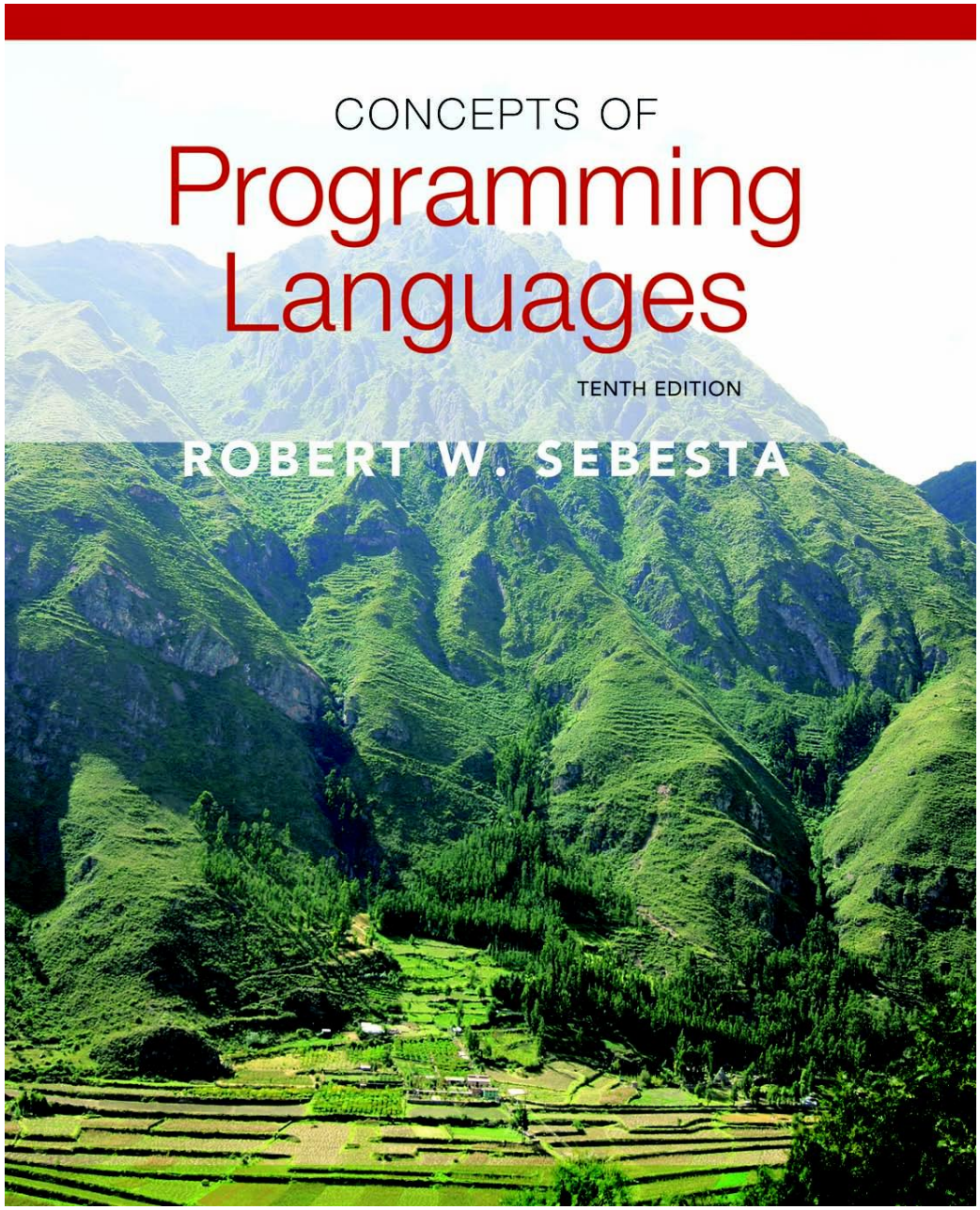
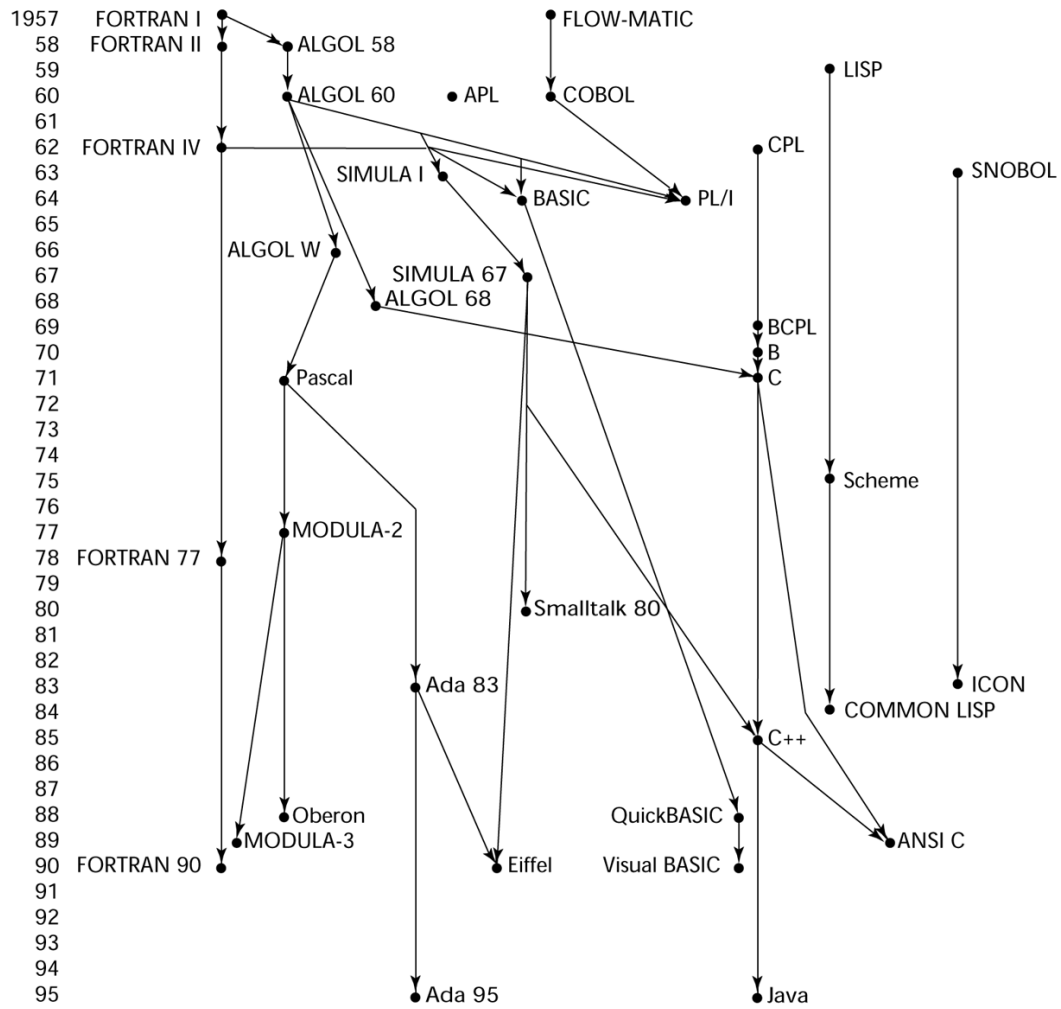
Chapter 2

Evolution of the Major Programming Languages



Genealogy of Common Languages



Zuse's Plankalkül

- Designed in 1945, but not published until 1972
- Never implemented
- Advanced data structures
  - floating point, arrays, records
- Invariants

Plankalkül Syntax

- An assignment statement to assign the expression A[4] + 1 to A[5]

	|	A + 1 => A	
V	|	4 5	(subscripts)
S	|	1.n 1.n	(data types)

Minimal Hardware Programming: Pseudocodes

- What was wrong with using machine code?
  - Poor readability
  - Poor modifiability
  - Expression coding was tedious
  - Machine deficiencies--no indexing or floating point

Pseudocodes: Short Code

- Short Code developed by Mauchly in 1949 for BINAC computers
  - Expressions were coded, left to right
  - Example of operations:

```
01 -    06 abs value
02 )    07 +
03 =    08 pause
04 /    09 (
```

50 times slower than machine code

Interpretation

Pseudocodes: Speedcoding

- Speedcoding developed by Backus in 1954 for IBM 701
  - Pseudo ops for arithmetic and math functions
  - Conditional and unconditional branching
  - Auto-increment registers for array access
  - Slow!
  - Only 700 words left for user program

Pseudocodes: Related Systems

- The UNIVAC Compiling System
  - Developed by a team led by Grace Hopper
  - Pseudocode expanded into machine code subprograms
- David J. Wheeler (Cambridge University)
  - developed a method of using blocks of re-locatable addresses to solve the problem of absolute addressing

IBM 704 and Fortran

- Fortran 0: 1954 – not implemented
- Fortran I:1957
  - Designed for the new IBM 704, which had index registers and floating point hardware
    - This led to the idea of compiled programming languages, because there was no place to hide the cost of interpretation before (no floating-point hardware)
  - Environment of development
    - Computers were small and unreliable
    - Applications were scientific
    - No programming methodology or tools
    - Machine efficiency was the most important concern

Design Process of Fortran

- Computers had
  - Small memories, slow, and unreliable
  - Primary use was scientific
- Speed of object code was primary concern
- Impact of environment
  - No need for dynamic storage
  - Need good array handling and counting loops
  - No string handling, decimal arithmetic, or powerful input/output (for business software)

Fortran I Overview

- First implemented version of Fortran
  - Names could have up to six characters
  - Post-test counting loop (DO)
  - Formatted I/O
  - User-defined subprograms
  - Three-way selection statement (arithmetic IF)
  - No data typing statements
    - Integer if starts with I, J, K, L, M, N
    - Floating point all others.

Fortran I Overview (continued)

- First implemented version of FORTRAN
  - Half of the code in IBM 704 was written in Fortran
  - Compiler released in April 1957
  - Programs larger than 400 lines rarely compiled correctly, mainly due to poor reliability of 704
  - Code was very fast
  - Quickly became widely used

Fortran II

- Distributed in 1958
  - Independent compilation introduced
    - So you don't have to compile whole program if you change only one function
  - Fixed the bugs

Fortran IV

- Evolved during 1960-62
  - Explicit type declarations
  - Logical selection statement
  - Subprogram names could be parameters

Fortran 77

- Became the new standard in 1978
  - Character string handling
  - Logical loop control statement
  - **`IF-THEN-ELSE`** statement

Fortran 90

- Most significant changes from Fortran 77
  - Modules
  - Dynamic arrays
  - Pointers
  - Recursion
  - `CASE` statement
  - Parameter type checking

Latest versions of Fortran

- Fortran 95 – relatively minor additions, plus some deletions
- Fortran 2003 – support for OOP, procedure pointers, interoperability with C
- Fortran 2008 – blocks for local scopes, co-arrays, `Do Concurrent`

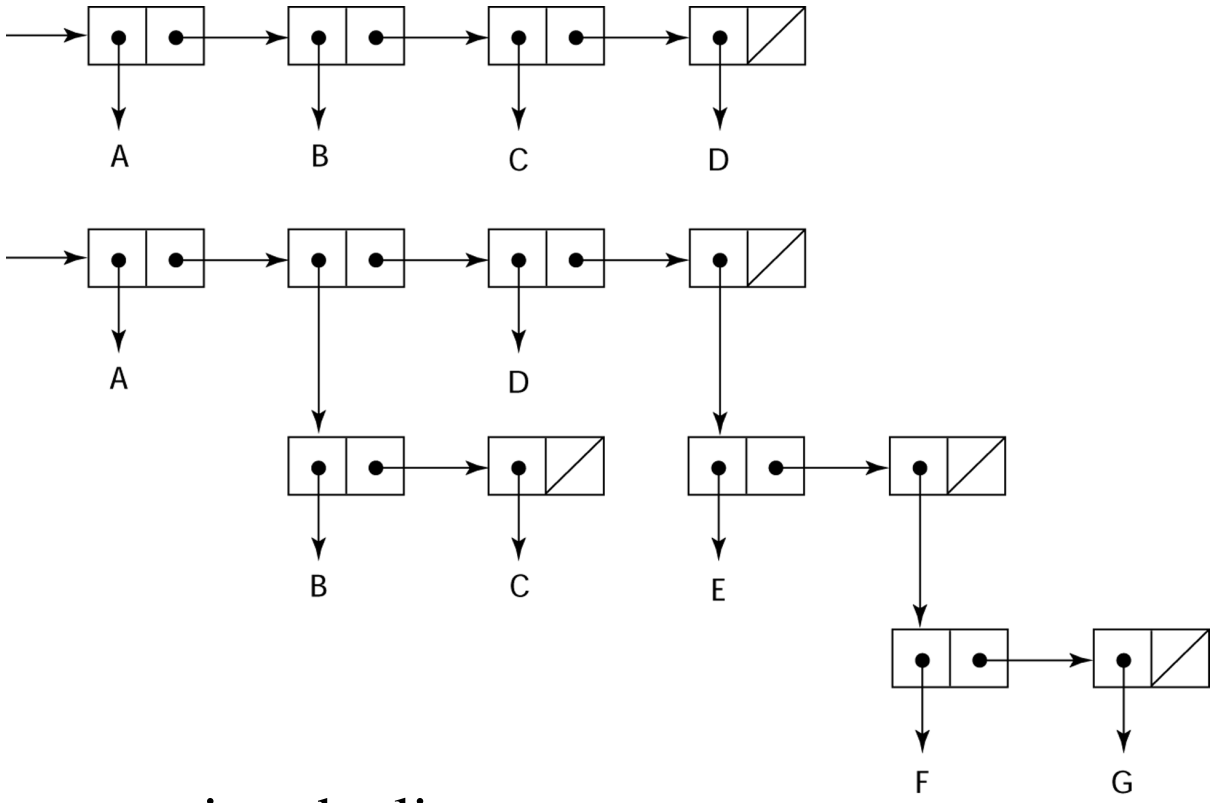
Fortran Evaluation

- Highly optimizing compilers (all versions before 90)
  - Types and storage of all variables are fixed before run time
- Dramatically changed forever the way computers are used

Functional Programming: LISP

- LISt Processing language
  - Designed at MIT by McCarthy
- AI research needed a language to
  - Process data in lists (rather than arrays)
  - Symbolic computation (rather than numeric)
- Only two data types: atoms and lists
- Syntax is based on *lambda calculus*

Representation of Two LISP Lists



Representing the lists `(A B C D)` and `(A (B C) D (E (F G)))`

LISP Evaluation

- Pioneered functional programming
  - No need for variables or assignment
  - Control via recursion and conditional expressions
- Still the dominant language for AI
- COMMON LISP and Scheme are contemporary dialects of LISP
- ML, Haskell, and F# are also functional programming languages, but use very different syntax

Scheme

- Developed at MIT in mid 1970s
- Small
- Extensive use of static scoping
- Functions as first-class entities
- Simple syntax (and small size) make it ideal for educational applications

COMMON LISP

- An effort to combine features of several dialects of LISP into a single language
- Large, complex, used in industry for some large applications

The First Step Toward Sophistication: ALGOL 60

- Environment of development
  - FORTRAN had (barely) arrived for IBM 70x
  - Many other languages were being developed, all for specific machines
  - No portable language; all were machine-dependent
  - No universal language for communicating algorithms
- ALGOL 60 was the result of efforts to design a universal language

Early Design Process

- ACM and GAMM met for four days for design (May 27 to June 1, 1958)

- Goals of the language
  1. Close to mathematical notation
  2. Good for describing algorithms
  3. Must be translatable to machine code

ALGOL 58

- Concept of type was formalized
- Names could be any length
- Arrays could have any number of subscripts
- Parameters were separated by mode (in & out)
- Subscripts were placed in brackets
- Compound statements (**`begin ... end`**)
- Semicolon as a statement separator
- Assignment operator was :=
- **`if`** had an **`else-if`** clause
- No I/O – "would make it machine dependent"

ALGOL 58 Implementation

- Not meant to be implemented, but variations of it were (MAD, JOVIAL)
- Although IBM was initially enthusiastic, all support was dropped by mid 1959

ALGOL 60 Overview

- Modified ALGOL 58 at 6-day meeting in Paris
- New features
  - Block structure (local scope)
  - Two parameter passing methods
  - Subprogram recursion
  - Stack-dynamic arrays

  - Still no I/O and no string handling

ALGOL 60 Evaluation

- Successes
  - It was the standard way to publish algorithms for over 20 years
  - **All subsequent imperative languages are based on it**
  - First machine-independent language
  - First language whose syntax was formally defined (BNF)

ALGOL 60 Evaluation (continued)

- Failure
  - Never widely used, especially in U.S.
  - Reasons
    - Lack of I/O and the character set made programs non-portable
    - Too flexible--hard to implement
    - Entrenchment of Fortran
    - Formal syntax description
    - Lack of support from IBM

COmmon Business-Oriented Language: COBOL

- Environment of development
  - UNIVAC was beginning to use FLOW-MATIC
  - USAF was beginning to use AIMACO
  - IBM was developing COMTRAN

COBOL Historical Background

- Based on FLOW-MATIC
- FLOW-MATIC features
  - Names up to 12 characters, with embedded hyphens
  - English names for arithmetic operators (no arithmetic expressions)
  - Data and code were completely separate
  - The first word in every statement was a verb

Shortest COBOL Code

```
$ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID. ShortestProgram.
PROCEDURE DIVISION.
DisplayPrompt.
  DISPLAY "I did it".
  STOP RUN.
```

COBOL Design Process

- First Design Meeting (Pentagon) – May 1959
- Design goals
  - Must look like simple English
  - Must be easy to use, even if that means it will be less powerful
  - Must broaden the base of computer users
  - Must not be biased by current compiler problems
- Design committee members were all from computer manufacturers and DoD branches
- Design Problems: arithmetic expressions? subscripts? Fights among manufacturers

COBOL Evaluation

- Contributions
  - First macro facility in a high-level language
  - Hierarchical data structures (records)
  - Nested selection statements
  - Long names (up to 30 characters), with hyphens
  - Separate data division

COBOL: DoD Influence

- First language required by DoD
  - would have failed without DoD
- Still the most widely used business applications language