Chapter 2

1. Plankalkül - 1945

- Never implemented
- Advanced data structures
  - floating point, arrays, records
- Invariants
- Notation:

A(7) := 5 * B(6)

	5 * B => A	
V	6	7 (subscripts)
S	1.n	1.n (data types)

2. Pseudocodes - 1949

What was wrong with using machine code?

a. Poor readability
b. Poor modifiability
c. Expression coding was tedious
d. Machine deficiencies--no indexing or fl. pt.

- *Short code; 1949; BINAC; Mauchly*
  - Expressions were coded, left to right
  - Some operations:
    - 1n => (n+2)nd power
    - 2n => (n+2)nd root
    - 07 => addition

Chapter 2

2. Pseudocodes (continued)

- *Speedcoding; 1954; IBM 701, Backus*
  - Pseudo ops for arithmetic and math functions
  - Conditional and unconditional branching
  - Autoincrement registers for array access
  - Slow!
  - Only 700 words left for user program

3. Laning and Zierler System - 1953

- Implemented on the MIT Whirlwind computer
- First "algebraic" compiler system
- Subscripted variables, function calls, expression translation
- Never ported to any other machine

4. FORTRAN I - 1957

(FORTRAN 0 - 1954 - not implemented)

- Designed for the new IBM 704, which had index registers and floating point hardware
- Environment of development:
  1. Computers were small and unreliable
  2. Applications were scientific
  3. No programming methodology or tools
  4. Machine efficiency was most important

Chapter 2

4. FORTRAN I (continued)

- *Impact of environment on design*
  1. No need for dynamic storage
  2. Need good array handling and counting loops
  3. No string handling, decimal arithmetic, or powerful input/output (commercial stuff)

- First implemented version of FORTRAN
  - Names could have up to six characters
  - Posttest counting loop (`DO`)
  - Formatted i/o
  - User-defined subprograms
  - Three-way selection statement (arithmetic `IF`)
  - No data typing statements
  - No separate compilation
  - Compiler released in April 1957, after 18 worker/ years of effort
  - Programs larger than 400 lines rarely compiled correctly, mainly due to poor reliability of the 704
  - Code was very fast
  - Quickly became widely used

5. FORTRAN II - 1958

- Independent compilation
- Fix the bugs

Chapter 2

6. FORTRAN IV - 1960-62

- Explicit type declarations
- Logical selection statement
- Subprogram names could be parameters
- ANSI standard in 1966

7. FORTRAN 77 - 1978

- Character string handling
- Logical loop control statement
- `IF-THEN-ELSE` statement

8. FORTRAN 90 - 1990

- Modules
- Dynamic arrays
- Pointers
- Recursion
- `CASE` statement
- Parameter type checking

FORTRAN Evaluation

- Dramatically changed forever the way computers are used

Chapter 2

9. LISP - 1959

- LISt Processing language
  (Designed at MIT by McCarthy)
  - *AI research needed a language that:*
    1. Process data in lists (rather than arrays)
    2. Symbolic computation (rather than numeric)
  - Only two data types: atoms and lists
  - Syntax is based on lambda calculus
  - *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, Miranda, and Haskell are related languages

10. ALGOL 58 - 1958

- *Environment of development:*
  1. FORTRAN had (barely) arrived for IBM 70x
  2. Many other languages were being developed, all for specific machines
  3. No portable language; all were machine-dependent
  4. No universal language for communicating algorithms

Chapter 2

11. ALGOL 58 (continued)

- ACM and GAMM met for four days for design
  - *Goals of the language:*
    1. Close to mathematical notation
    2. Good for describing algorithms
    3. Must be translatable to machine code

- *Language Features*:
  - Concept of type was formalized
  - Names could have 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

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

Chapter 2

12. ALGOL 60 - 1960

- 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

- *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)

- *Failure*:
  - Never widely used, especially in U.S.

    Reasons:

    1. No i/o and the character set made programs nonportable
    3. Too flexible--hard to implement
    4. Intrenchment of FORTRAN
    5. Formal syntax description
    6. Lack of support of IBM

Chapter 2

13. COBOL - 1960

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

- *Based on FLOW-MATIC*
  - FLOW-MATIC features:
    - Names up to 12 characters, with embedded hyphens
    - English names for arithmetic operators
    - Data and code were completely separate
    - Verbs were first word in every statement

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

Chapter 2

13. COBOL (continued)

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

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

14. BASIC - 1964

- Designed by Kemeny & Kurtz at Dartmouth
- Design Goals:
  - Easy to learn and use for non-science students
  - Must be ”pleasant and friendly"
  - Fast turnaround for homework
  - Free and private access
  - User time is more important than computer time
- Current popular dialects: QuickBASIC and Visual BASIC

Chapter 2

15. PL/I - 1965

- Designed by IBM and SHARE
- *Computing situation in 1964 (IBM's point of view)*
  1. Scientific computing
     - IBM 1620 and 7090 computers
     - FORTRAN
     - SHARE user group

  2. Business computing
     - IBM 1401, 7080 computers
     - COBOL
     - GUIDE user group

- By 1963, however,
  - Scientific users began to need more elaborate i/o, like COBOL had; Business users began to need fl. pt. and arrays (MIS)

  - It looked like many shops would begin to need two kinds of computers, languages, and support staff--too costly

- *The obvious solution:*
  1. Build a new computer to do both kinds of applications
  2. Design a new language to do both kinds of applications

Chapter 2

15. PL/I (continued)

- *PL/I contributions:*
  1. First unit-level concurrency
  2. First exception handling
  3. Switch-selectable recursion
  4. First pointer data type
  5. First array cross sections

- *Comments:*
  - Many new features were poorly designed
  - Too large and too complex
  - Was (and still is) actually used for both scientific and business applications

16. Early Dynamic Languages

- Characterized by dynamic typing and dynamic storage allocation

- APL (A Programming Language) 1962
  - Designed as a hardware description language (at IBM by Ken Iverson)
  - Highly expressive (many operators, for both scalars and arrays of various dimensions)
  - Programs are very difficult to read

- SNOBOL(1964)
  - Designed as a string manipulation language (at Bell Labs by Farber, Griswold, and Polensky)
  - Powerful operators for string pattern matching

17. SIMULA 67 - 1967

- Designed primarily for system simulation (in Norway by Nygaard and Dahl)
- Based on ALGOL 60 and SIMULA I
- *Primary Contribution:*
  - Coroutines - a kind of subprogram
    - Implemented in a structure called a class
      - Classes are the basis for data abstraction
        - Classes are structures that include both local data and functionality

18. ALGOL 68 - 1968

- From the continued development of ALGOL 60, but it is not a superset of that language
- Design is based on the concept of orthogonality
- *Contributions:*
  1. User-defined data structures
  2. Reference types
  3. Dynamic arrays (called `flex` arrays)

Chapter 2

18. ALGOL 68 (continued)

- *Comments:*
  - Had even less usage than ALGOL 60
  - Had strong influence on subsequent languages, especially Pascal, C, and Ada

19. Pascal - 1971

- Designed by Wirth, who quit the ALGOL 68 committee (didn't like the direction of that work)
- Designed for teaching structured programming
- Small, simple, nothing really new
- Still the most widely used language for teaching programming in colleges (but use is shrinking)

20. C - 1972

- Designed for systems programming (at Bell Labs by Dennis Richie)
- Evolved primarily from B, but also ALGOL 68
- Powerful set of operators, but poor type checking
- Initially spread through UNIX

21. Other descendants of ALGOL

- Modula-2 (mid-1970s by Niklaus Wirth at ETH)
  - Pascal plus modules and some low-level features designed for systems programming

Chapter 2

21. Other descendants of ALGOL (cont.)

- Modula-3 (late 1980s at Digital & Olivetti)
  - Modula-2 plus classes, exception handling, garbage collection, and concurrency

- Oberon (late 1980s by Wirth at ETH)
  - Adds support for OOP to Modula-2
  - Many Modula-2 features were deleted (e.g., `for` statement, enumeration types, `with` statement, noninteger array indices)

- Delphi (Borland)
  - Pascal plus features to support OOP
  - More elegant and safer than C++

22. Prolog - 1972

- Developed at the University of Aix-Marseille, by Comerauer and Roussel, with some help from Kowalski at the University of Edinburgh
- Based on formal logic
- Non-procedural
- Can be summarized as being an intelligent database system that uses an inferencing process to infer the truth of given queries

Chapter 2

23. Ada - 1983 (began in mid-1970s)

- Huge design effort, involving hundreds of people, much money, and about eight years

- *Contributions:*
  1. Packages - support for data abstraction
  2. Exception handling - elaborate
  3. Generic program units
  4. Concurrency - through the tasking model

- *Comments:*
  - Competitive design
  - Included all that was then known about software engineering and language design
  - First compilers were very difficult; the first really usable compiler came nearly five years after the language design was completed

- Ada 95 (began in 1988)
  - Support for OOP through type derivation
  - Better control mechanisms for shared data (new concurrency features)
  - More flexible libraries

Chapter 2

24. Smalltalk - 1972-1980

- Developed at Xerox PARC, initially by Alan Kay, later by Adele Goldberg
- First full implementation of an object-oriented language (data abstraction, inheritance, and dynamic type binding)
- Pioneered the graphical user interface everyone now uses

25. C++ - 1985

- Developed at Bell Labs by Stroustrup
- Evolved from C and SIMULA 67
- Facilities for object-oriented programming, taken partially from SIMULA 67, were added to C
- Also has exception handling
- A large and complex language, in part because it supports both procedural and OO programming
- Rapidly grew in popularity, along with OOP
- ANSI standard approved in November, 1997

- Eiffel - a related language that supports OOP
  - (Designed by Bertrand Meyer - 1992)
  - Not directly derived from any other language
  - Smaller and simpler than C++, but still has most of the power

Chapter 2

26. Java (1995)

- Developed at Sun in the early 1990s
- Based on C++
  - Significantly simplified
  - Supports *only* OOP
  - Has references, but not pointers
  - Includes support for applets and a form of concurrency