Chapter 2

1. Plankalkül – 1945(program calculus) pp. - 41

- Never implemented but based on Z4
- Advanced data structures(scalar type and for loop)
  - integer, floating point, arrays, records
- Mathematical expression, sorting and subscript
- Proposed on 1945, published on 1972(Zuse)

Notation:

```
A(7) := 5 * B(6)
     |  5  *  B  =>  A
V  |        6       7      (v subscripts)
S  |       1.n     1.n     (s data types, integer n bits)
```

2. Pseudocodes - 1949

*What was wrong with using **machine code**?*

a. Poor readability(numeric code op)
b. Poor modifiability(absolute address) for I/Delete
c. Expression coding was tedious
d. Machine deficiencies--no indexing or float point instructions provided

Short code 1949; BINAC computer; John Mauchly

- Expressions were coded, left to right pure interpreter
- Some operations: see pp. 42

  1n => (n+2)nd power ex. 00 x0 03 20 06 y0
  2n => (n+2)nd root x0=sqrt(abs(y0))
  07 => addition …

Chapter 2

2. Pseudocodes (continued)

- *Speedcoding; 1954; IBM 701, John Backus*
  - *include floating-point operation in Machine language*
  - Pseudo ops for arithmetic and math functions(Sq…)
  - Conditional and unconditional branching
  - Autoincrement registers for array access
  - Slow!(4.2 ms for an add instruction)
  - Only 700 words left for user program

3. Laning and Zierler System – 1953 pp.44

- 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
  5. Little syntax error checking

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(if constructs)
- 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 : public, private
- Dynamic arrays, Pointers
- Recursion
- CASE , Exit statement
- Parameter type checking

FORTRAN Evaluation

- Dramatically changed forever the way computers are used
  - Highly optimizing compiler
  - Simplicity and efficiency
  - No recursion allowed
  - Static allocated variables

Chapter 2

9. LISP – 1959 *pp. 49*

- LISt Processing language(1st Functional PL) pp. 52 (Designed at MIT by McCarthy)
- Application needs to process symbolic data:
  Linguist – Natural language processing
  Psychologist – Human information retrieval & store
  Mathematician – Theorem proving(by IBM in '50 mid)
- *AI research needed a language that:*
  1. Process data in lists (rather than arrays)
  2. Symbolic computation (rather than numeric)
- Characteristics of LISP:
  recursion, conditional expression,
  dynamic storage allocation, implicit deallocation
  - Only two data types: atoms and lists *pp. 51*
  - Syntax is based on lambda calculus
  - *Pioneered functional programming pp. 51- 52*
    - No need for variables or assignment(atom)
    - Control via recursion and conditional expressions replace the iterative process
  - Still the dominant language for AI
  - COMMON LISP and Scheme are contemporary dialects of LISP(static v.s dynamic scoping *pp. 199 - 206*)
    http://www.swiss.ai.mit.edu/projects/scheme/
  - 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

Chapter 2

3. No portable language; all were machine-dependent
4. No universal language for communicating algorithms
5. Need standard mathematical notations for describing computing process

11. ALGOL 58 (continued)

- ACM and GAMM met for four days for design
  - *Goals of the language: pp. 56*
    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(Fortran 6 words)
  - Any number of array dimension is allowed(3)
  - Parameters were separated by mode (in & out)
  - Subscripts were placed in brackets([ ])
  - Compound statements (`begin ... end`)
  - Semicolon as a statement separator
  - Assignment operator was `:= v.s. =>`
  - `if` had an `else-if` clause
- *Comments:*
  - Not meant to be implemented, but variations of it were (MAD, NELICA, JOVIAL)
  - JOVIAL is used by US air force for 25 years(1963)
  - Although IBM was initially enthusiastic, all support was dropped by mid-1959

Chapter 2

- Hard to persuade and train programmer to use it
- Hard to develop it!!

12. ALGOL 60 - 1960

- Modified ALGOL 58 at 6-day meeting in Paris
- *New Features*: pp. 58
  - Block structure (local scope)
  - Two parameter passing methods*(by value, name)*
  - Subprogram can be recursive*(LISP provided '59)*
  - Stack-dynamic arrays are allowed (ex. int a[n];)
  - Still no I/O formatting and no string handling
- *Successes*: pp. 59
  - 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 by BNF*(formal notation)*
  - affect the machine architecture (B5000 provide HW stack to support recursion)
- *Failure*:
  - Never widely used, especially in U.S.

    Reasons:
    1. No I/O statement made programs non-portable
    2. Too flexible--hard to implement(call-by-name)
    3. Too many people are FORTRAN users
    4. Formal syntax description(BNF is not widely accepted)
    5. 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 dash
    - English names for arithmetic operators(add,…)
    - 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: pp. 63*
  - First macro facility in a high-level language
  - Hierarchical data structures (records)
  - Nested selection statements
  - Long names (up to 30 characters), with dash
  - Data Division and File record description

- *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: pp. 66
  - 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(small) and 7090(large) computers
     - FORTRAN, Assembly language
     - SHARE user group

  2. Business computing
     - IBM 1401(small), 7080(large) computers
     - COBOL
     - GUIDE user group

- By 1963, however,
  - Scientific users began to need large data to be processed more efficient I/O facility, like COBOL had Business users began to need fl. pt. and arrays, like Fortran had (MIS)

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

- *The obvious solution:*
  1. Build a new computer to do both kinds of applications(IBM 360)
  2. Design a new language to do both kinds of applications*(replace Fortran, COBOL, LISP, Assembly)*

Chapter 2

15. PL/I (continued)

3-4 days meeting every week by IBM and SHARE

- *PL/I contributions: pp. 71*

  adopt ALGOL 60(recursion and block structure),
  Fortran IV(separate compilation),
  COBOL 60(data structure, I/O, report generator)

  1. First unit-level concurrency
  2. First provide 23 types of exception handling
  3. Recursion and non-recursion
  4. First pointer data type
  5. First array cross sections(a row of matrix can be a vector)

- *Comments:*
  - Many new features were poorly designed
  - Too large and too complex(pointer, except-handling…)
  - Was (and still is) actually used for both scientific and business applications(see pp. 72)

16.Early Dynamic Languages(APL&SNOBOL)

- Characterized by dynamic typing and dynamic storage allocation

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

Chapter 2

- SNOBOL(1964) pp. 73
  - Designed as a string manipulation language (at Bell Labs by Farber, Griswold, and Polensky)
  - Powerful operators for string pattern matching
  - Suitable for writing text editor

17. SIMULA 67 - 1967

- Designed primarily for system simulation in OR (in Norway by Nygaard and Dahl)
- Based on ALGOL 60 and SIMULA I('62 and '64)
- *Primary Contribution:*
  - Coroutines - a kind of subprogram(pp. 74 bottom)
  - Implemented in a structure called a class
    - Classes are the basis for data abstraction
    - Classes are structures that include both local data(attribute) and functionality(method)

      I.e. data structure and routine are packaged together

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(pp. 10)
- *Contributions:*
  1. User-defined data structures for data abstraction
  2. Dynamic arrays (called `flex` arrays) pp. 76

     Implicit heap-dynamic

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

Some Important descendants of the ALGOLs

19. Pascal – 1971 (pp. 79)

- Designed by Wirth, who quit the ALGOL 68 committee (didn't like the direction of that work)
- Designed for teaching structured programming
  basic data type, user-defined data type, pointer
  pass by value-result, case statement, recursion
- Small, simple, expressive, nothing really new
- Still the most widely used language for teaching programming in colleges (but use is shrinking in mid'90)

20. C – 1972 (pp. 81)

- Designed for systems programming (at Bell Labs by Dennis Richie)
- Evolved primarily from B, but also ALGOL 68
- a rich 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 (coroutine, abstract data type …) pp. 82

Chapter 2

21. Other descendants of ALGOL (cont.)

- Modula-3 (late 1980s at Digital & Olivetti)
  - Modula-2 plus classes, OOP, 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, non-integer array indices)
- Delphi (Borland)
  - Pascal plus features to support OOP
  - More elegant and safer than C++(range checking, pointer arithmetic, do not allow user-defined operator overloading, parameterized classes…)

22. Prolog – 1972 ***pp. 84***

- 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(predicate calculus)
- Non-procedural(facts and rules to deduce goal) pp. 85
- Can be summarized as being an intelligent database system that uses an inferencing process to infer the truth of given queries

Comment:
- logic programming has proven to be highly inefficiency
- suitable to an efficient method and small area of AP

Chapter 2

23. Ada - 1983 (began in mid-1970s) MIL-STD 1815
- Huge design effort, involving hundreds of people, much money, and about eight years because 450 PLs are used in DoD project (hard to maintain)

- *Contributions: pp. 88*
  1. Packages - support for data abstraction
  2. Extensive facility of exception handling – allow programmer to gain control when exception
  3. Generic program units
  4. Concurrency - through the tasking model(rendezvous)

- *Comments: pp. 89*
  - 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 '85

- Ada 95 (began in 1988) pp. 90
  - Support for OOP through type derivation
    . provide inheritance
    . polymorphism
  - Better control mechanisms for shared data (new concurrency features)
    . by sub-program or rendezvous
  - More flexible libraries
    . dynamic binding of subprogram call
  - provide GUI interface

Chapter 2

24. Smalltalk - 1972-1980

Designed based on SIMULA 67
- Pioneered the graphical user interface everyone now uses
- 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) pp. 92

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(template provide by C++)
- A large and complex language, in part because it supports both procedural and OO programming
- dynamic binding and multiple inheritance
- Rapidly grew in popularity, along with OOP
- ANSI standard approved in November, 1997
- Related language – Eiffel(simpler and smaller C++)

- 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 (pp. 100)
- Based on C++
- Significantly simplified
- Supports *only* OOP
- Has references, but no pointers
- Includes support for applets and a form of concurrency – thread or synchronize modifier
- Widely used in WWW's CGI programming
-**Disadvantage**
- **A complex language**
- **Lack of multiple inheritance**