**Chapter 2**

Zuse Plakalkul

* Designed in 1945 by Konrad Zuse
* Never implemented
* Only published in 1972
* Advanced data structures
  + Floating point, arrays, nested records
* Iterative for-like statement
* Selection statement, no else

Pseudocodes

* In the late 1940s / early 1950s programming was done using machine code
  + Coding was tedious
  + Poor readability
  + Poor modifiability
  + Machine deficiency

IMB 704 and Fortan

* FORmula TRANslating system
* Fortran 0: never implemented
* Fortran I (1957)
  + designed for IMB 704, had index registers
  + Led to the idea of compiled languages
  + IMB 704 had very little memory and was slow and unreliable
  + Applications were simple
  + Machine efficiency was the most important concern
  + Compiled programs had to be fast
  + No need for dynamic storage
  + Good array handling
  + No string handling or input/output
  + Names could have up to 6 characters
  + Programs larger than 400 lines didn’t compile correctly most times
* Fortran II
  + Independent compilation meant fixing compilation for larger programs
* Fortran IV
  + Explicit type declarations
  + Logical selections statements
* Fortran 77
  + Character string handling
  + Logical loop control statement
  + IF-THEN-ELSE statements
* Fortran 90
  + Modules
  + Dynamic arrays
  + Pointers
  + Recursion
  + Case statements
* LISP
  + LISt Processing language
  + Processing data in lists rather than arrays
  + Recursive operations
  + Automatic dynamic storage handling
  + Only 2 data types
    - Atoms (values) and lists
  + List iteration controlled via recursion
* ALGOL 60
  + Machine dependant, no portability
  + No universal language for communicating algorithms
  + Good for describing algorithms
  + Must be translated into machine code
  + No readability
  + Non portable
* ALGOL 58
  + Names could be any length
  + Arrays could have infinite subscripts
  + Compound statements (begin…end)
  + If and else-if statements
  + No I/O, this would be machine independent
* COBOL
  + Names up to 12 characters
  + English names for operations, no +, =…\
  + Poor compiler performance
  + Increased costs
* BASIC
  + Easy to learn
  + Good writability
* PL/I
  + Exception handling
  + Switch selectable recursion
  + Pointer data type
  + Array cross sections
  + Unfortunately it was poorly designed
* APL
  + No readability
  + No writability
* SNOBOL
  + String manipulation language
  + Initially used to write text editors
* ALGOL 68
  + The design was based on orthogonality
  + User defined data structures
  + Reference types
  + Heap dynamic arrays
* Pascal
  + Small and simple
  + Used to teach structured programming
* C
  + Powerful set of operations and functions
  + Poor type checking and other unsafe features
  + Less reliable
  + Good generality
* Prolog
  + Highly ineffienct
  + High cost
* Ada
  + Intended to standardize programming languages in the US DoD
  + Very high cost to develop and lots of time
  + Support for data abstraction
  + Exception handling
  + Concurrency
  + No useable compilers
* Smalltalk
  + Data abstraction
  + Inheritance
  + Dynamic binding
  + User GUI’s
  + The OOP paradigm
* C++
  + A large and complex language
  + Exception handling, operator overloading, pointers, references, template functions and classes
  + Cheap compilers
  + Backward compatible with C
  + Only OO language suitable for large projects at the time
* Java
  + Simplified and more reliable
  + Only supports OOP
  + Support for applets, concurrency, & GUI design
  + Less powerful and flexible
  + Less generality
  + Many built in libraries
  + Portable