Chapter 2

History of Programming Languages

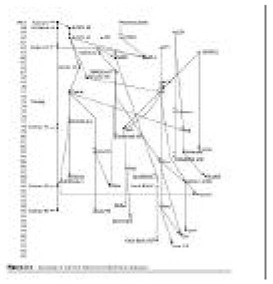
Topics

- Early History: low level languages
- The 1950s: first programming languages
- The 1960s: an explosion in programming languages
- The 1970s: back to simplicity
- Functional and logic programming
- Object-oriented programming

History of Programming Languages

- The history of programming languages is tied to the evolution of computers.
- Several hundred programming languages and dialects since low-level programming languages (1940s).
  - Most have limited life span and utility.
  - A few have enjoyed widespread success in one or more application domain.
  - Many have played an important role in influencing the design of future languages

Most Influential Programming Languages



Early History: Low Level Languages

- 1940s and early 1950s:
  - Computers: slow, unreliable, difficult to program
  - Machine language
    - Programming was tedious and error-prone.
    - Programs were difficult to read (numeric codes for instructions) and difficult to modify.
  - Assembly language
    - Use symbols and mnemonics to express the underlying machine code.
    - Highly machine dependent
    - Syntax: unlike natural language
  - Goal: implement what was possible on the available hardware.

The 1950s: The First PLs

FORTRAN

- First high-level programming language.
- Developed between 1954-1957 by a team at IBM lead by John Backus.
- "**FOR**mula **TRAN**slating"
- Designed primarily for scientific and computational programming (to express mathematical formulas).
- Goal: to eliminate bookkeeping details and repetitive planning.

The 1950s: The First PLs
FORTRAN

- Most features were new and become standard in later languages.
  - Array, loops controlled by an index variable, if-selection statement, do-loop statement, input-output formatting.
- Some characteristics:
  - Variable names could be up to 6 characters (Fortran 0 at most two).
  - Spaces were ignored because they could be confusing to the human typist (a coder wrote code on paper and a typist punched the codes into cards)

The 1950s: The First PLs
FORTRAN

- There were no data-typing statements (variables whose name began with I, J, K, L, M, and N were implicitly integer type, the other were implicitly floating-point)
- Fortran initiated the search for languages that communicate with the computer at a high level, a level closer to the way human think.
- Fortran and its descendants (Fortran II, IV, 77, 90, and 95) are still dominant in scientific applications today.
  - Their compilers are still among the most efficient available because they produce very fast code.

The 1950s: The First PLs
ALGOL 60

- Idea: design a universal language for communicating programs among users and to computers.
- Developed by an international committee between 1958 and 1960.
- "**ALGO**rithmic **L**anguage"
  - Initially called IAL (the International Algebraic Language).
- Goal: to provide a general, expressive language for describing algorithms, both in research and in practical applications.

The 1950s: The First PLs
ALGOL 60

- ALGOL had a profound effect on programming languages design.
  - Most of the current imperative languages are derivatives of ALGOL (Pascal, Ada, C, MODULA-2, C, Java).
  - Research papers today still often use ALGOL or ALGOL-like syntax to describe algorithms.

The 1950s: The First PLs
ALGOL 60

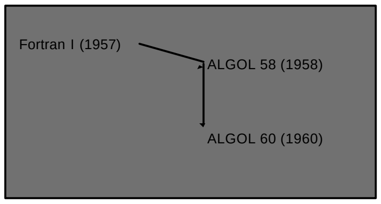
- Concepts introduced:
  - Free-format.
  - Structured statements.
  - Type declarations for variables .
  - The concept of block was introduced: begin-end blocks.
  - Procedures were allowed to be recursive.
  - Two different means of passing parameters to subprograms: pass by value and pass by name.
  - Stack-dynamic array were allowed.

The 1950s: The First PLs
ALGOL 60

- Some firsts:
  - The first time a language was designed by an international group.
  - The first language designed to be machine independent.
  - The first language whose syntax was formally described using BNF notation.
- ALGOL evolved:
  - ALGOL 60 $\rightarrow$ ALGOL W $\rightarrow$ ALGOL 68.
  - ALGOL68 had a long list of features: parallel computation, semaphores, implementation-dependant constants, large collection of types (complex numbers, bit patterns, long and short numbers, strings, and flexible arrays), and case-statement.

The 1950s: The First PLs
ALGOL 60

- Genealogy:



The 1950s: The First PLs
LISP

- First functional programming language.
- Intended to provide a tool for writing programs for symbol manipulation and list processing in the field of artificial intelligence.
- Developed by John McCarthy and his colleagues at MIT in the late 1950s-early 1960s
- "**LIS**t **P**rocessor"
- Primarily data structure: list of symbols.
  - Words in a sentence, a list of attributes, a payroll record, a symbolic differential equation.
- Based of function application.

The 1950s: The First PLs
LISP

- List: "nil" element and a pair element (a pair of pointers, one to an element of the list and one to the rest of the list).
- Data and programs are represented as lists.
- Pioneered general notions of computation: garbage collection.
- Lisp and its variants (MACLISP, UTLISP, COMMONLISP, SCHEME, etc) are still in use today in many artificial intelligence applications.

The 1950s: The First PLs
COBOL

- Developed by the U.S. Department of Defense in 1960.
- Goal: to obtain a common programming language for data processing applications.
- "**CO**mmon **B**usiness **O**riented **L**anguage "
- COBOL had the opposite effect than ALGOL 60
  - Quickly adopted by banks and corporations for a large-scale record-keeping and other business applications.
  - It had little effect on the design of subsequent languages, except PL/I.

The 1950s: The First PLs
COBOL

- Major goal: to have a more English-like programming language suitable for business data processing.
  - Uses English as a basis for its syntax.
  - Programs are constructed out of clauses, sentences, and paragraphs
  - Programs tend to be more wordy than comparable programs in other languages.
- Problems:
  - The design was supposed to permit nonprogrammers to read and understand programs
    - It only complicated the syntax without providing readability.
  - Complex algorithms are extremely difficult to program.

The 1950s: The First PLs
APL

- Designed by Kenneth Iverson between 1959-1960s at Harvard University.
- Designed to describe computer architecture
  - Influenced by the field of linear algebra
  - It was not intended for implementation
- Goal: to facilitate the rapid programming of matrix algebraic and other mathematical computations.
- "**A P**rogramming **L**anguage "

The 1950s: The First PLs

APL

- APL is also functional in style and has a large powerful set of operators that allow most iterations to be performed completely automatic.
  - Programs are extremely difficult to read and maintain.
- It is still used today and it has not changed a lot over all this years.
- A descendent of APL is the language J.

The 1950s: The First PLs

- Programming languages developed rapidly in a short period of time 1954-1960.
- Three major imperative programming languages (FORTRAN, COBOL, ALGOL 60)
  - All three, in modified form, still in use today.
- Functional programming began (LISP) which is in use today.

The 1960s: Explosion of PLs

- After success of initial programming languages, everyone wanted to design their own language.
- Hundreds of special-purpose programming languages
  - Designer's particular interests or concerns.
  - Most have vanished.
  - Only a few had significant effect on development of programming languages.

The 1960s: Explosion of PLs

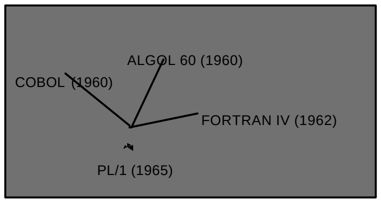
PL/I

- Designed by IBM between 1963 and 1964.
- Intended to use a new family of computers (the 360 family).
- Goal: to combine all features of FORTRAN, COBOL, and ALGOL 60 and add concurrency and exception handling as well.
- Failure:
  - Translators: difficult to write, slow, huge and unreliable
  - Language: difficult to learn and error prone to use, due to the large number of unpredictable interactions among language features.
- This language attempted to do too much, provide to many features, and satisfy too many users.

The 1960s: Explosion of PLs

PL/I

- Genealogy:



The 1960s: Explosion of PLs

SNOBOL

- Developed in the early 1960s by R. Griswold, D.J. Farber, and F Polensky at Bell Labs.
- Designed primarily to process string data (text processing).
- "**S**tri**N**g **O**riented **symBOlic L**anguage"
  - At first, SEXI (String Expression Interpreter) was proposed and rejected.
- Distributed free, which contributed to its dissemination and influence.
- Spin-offs: SPITBOL(speedy implementation), FASBOL, SLOBOL and SNOBAT.
  - The most familiar version is SNOBOLA.

The 1960s: Explosion of PLs
SIMULA 67

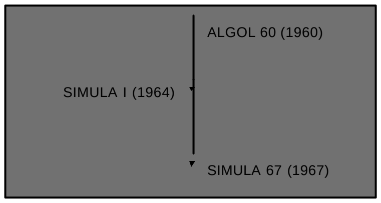
- Created by Kristen Nygaard and Ole-Johan Dahl in Oslo during the period 1965-1967.
- Based in SIMULA I, designed in the early 1960s and includes ALGOL 60 as a subset.
- Originally designed for simulations, such as queues at a supermarket, response times of emergency services, or chain reactions of nuclear reactors.
- Significant influence on programming languages development (not widely used)

The 1960s: Explosion of PLs
SIMULA 67

- Moved toward a general-purpose language.
- Contributed to the understanding of abstraction and computation
  - Introduction of the class concept fundamental to most object-oriented languages.
  - Basic idea of a class: data structure and routines that manipulate the data structure are packed together
  - It can be called the first object-oriented language.
- Successor (more than 20 years later): Beta

The 1960s: Explosion of PLs
SIMULA 67

- Genealogy:



The 1960s: Explosion of PLs
BASIC

- Designed by John Kemeny and Thomas E. Kurtz in the early 1960s
- Goal: facilitate the learning of programming and time-sharing principles.
  - Interactive language that was easy to learn, quick to compile, and easy to debug.
- "**B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode "
- Enjoyed widespread use but has gotten little respect.
- Final product: an interactive language that was easy to learn, quick to compile, and easy to debug.

The 1960s: Explosion of PLs
BASIC

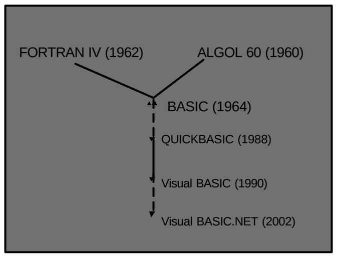
- Much of the design came from FORTRAN, with some minor influence from the syntax of ALGOL 60.
- Some characteristics:
  - Variables were formed of a single letter or a single letter followed by a single digit.
  - Variables are not declared and there is no distinction between integer and real numbers (like FORTRAN).
  - A program is organized by line numbers.
  - Original BASIC: only 14 different statement types and a single data type (floating point). The type was referred as "number".

The 1960s: Explosion of PLs
BASIC

- Enjoyed great popularity over the years, especially as a teaching language.
  - Simplicity appeals to many people who want to use the computer, but do not wish to learn programming.
- Multiple implementations of the language with vastly different characteristics.

The 1960s: Explosion of PLs
BASIC

- Genealogy:



The 1970s: Back to simplicity
PASCAL

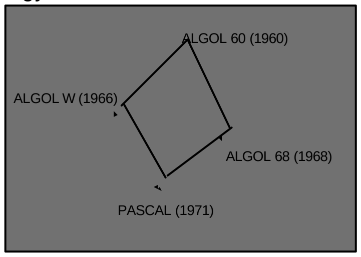
- Designed by Niklaus Wirth between 1971-1973
- Distilled the ideas of ALGOL into a small, simple, efficient, structured language that was intended for use in teaching programming
  - Gained acceptance not only for instruction but for many practical uses as well.
- Omitted very important practical features: adequate string handling, expandable input-output capabilities, separate compilation.

The 1970s: Back to simplicity
PASCAL

- Linked to the influence of Pascal was the structured approach to program methodology: step-wise refinement.
- Popularity based on remarkable combination of simplicity and expressivity.
- By the mid-1990s, the popularity of Pascal was on the decline, both in industry and in universities.

The 1970s: Back to simplicity
PASCAL

- Genealogy:

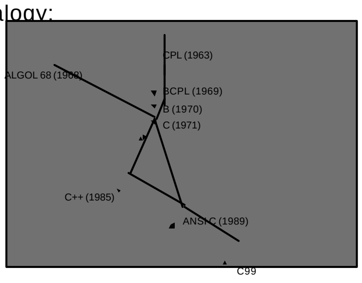


The 1970s: Back to simplicity
C

- Designed and implemented by Dennis M. Ritchie at Bell Laboratories in New Jersey in 1972.
- Influenced by the programming language B created for the first UNIX system. B was in fact based on BCPL.
- Characteristics:
  - Adequate control statements and data structuring facilities.
  - Rich set of operators (high degree of expressiveness).
  - Lack of complete type checking.
- Popularity: compiler is part of the UNIX operating system.

The 1970s: Back to simplicity
C

- Genealogy:



The 1970s: Back to simplicity
ADA

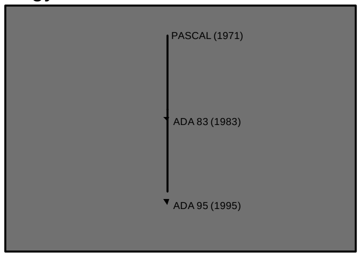
- In the mid 1970s the United States Department of Defense supported an effort to reduce the mounting software cost caused by the multitude of languages used by the defense establishment.
- In 1976, 23 already-existing programming languages were appraised in light of these goals.
  - No existing language was found entirely satisfactory
  - A cycle of language specifications and design reviews was conducted to develop a new language.
- The language was named ADA in honor of the first computer programmer.

The 1970s: Back to simplicity
ADA

- Features:
  - High-level constructs for concurrent execution and real-time programming.
  - Construct for aggregating data structures and subprocedures called a "package".
  - A mechanism for exception handling.
- Problem: too large and too complex.

The 1970s: Back to simplicity
ADA

- Genealogy:



The 1970s: Back to simplicity
MODULA-2

- Designed by Niklaus Wirth called Modula-2.
- Based on an earlier language, Modula that was designed for constructing operating systems.
- Attempted to correct deficiencies of the design of Pascal as well as to add abstraction and partial concurrency facilities.
- The designer attempted to keep the language as small as simple as possible
  - Left out a number of features, including exception-handling mechanisms.

The 1980s: Other Paradigms
SCHEME

- Developed from 1975 to 1978 by Gerald J. Sussman and Guy L. Steele Jr at MIT.
- Not until mid-1980s.
- A version of LISP that is more uniform that other versions
  - Designed to resemble more closely the lambda calculus.
- Another versions of LISP that appeared in the 1980s is Common LISP, which attempts to define a standard for the LISP family.

The 1980s: Other Paradigms
ML

- Developed by Robin Milner at Edinburgh University.
- "**M**eta**L**anguage"
- Syntax more closely related to Pascal.
- Mechanism for type checking, similar to Pascal, but much more flexible.
- A related language is Miranda, developed by David Turner at Manchester University in 1986.

The 1980s: Other Paradigms

PROLOG

- Developed by a group at Marseille led by A. Colmerauer .
- Uses formal logic notation to communicate processes to computer.
- Reasons it is not widely used:
  - Highly inefficient (non imperative).
  - Effective for only small areas of application.

Object-oriented Paradigm

- SMALLTALK
- C++
- EIFFEL
- JAVA