History of Programming Languages

History

- Early History : The first programmers
- 1940s: Von Neumann and Zuse
- 1950s: The First Programming Language
- 1960s: Explosion in Programming languages
- 1970s: Simplicity, Abstraction, Study
- 1980s: Object-oriented, Logic programming
- 1990s: Internet, Java, C++, C#
- 2000s: Scripting, Web, …
- 2010s: Parallel computing, concurency

Early History: First Programmers

- Jacquard loom of early 1800s
  - Translated card patterns into cloth designs
- Charles Babbage's analytical engine (1830s and 40s)

  Programs were cards with data and operations. Steam powered!

- Ada Lovelace – first programmer

  "The engine can arrange and combine its numerical quantities exactly as if they were letters or any other general symbols; And in fact might bring out its results in algebraic notation, were provision made."

Konrad Zuse and Plankalkul

Konrad Zuse began work on Plankalkul (plan calculus), the first algorithmic programming language, with an aim of creating the theoretical preconditions for the formulation of problems of a general nature.

Seven years earlier, Zuse had developed and built the world's first binary digital computer, the Z1. He completed the first fully functional program-controlled electromechanical digital computer, the Z3, in 1941.

Only the Z4 – the most sophisticated of his creations -- survived World War II.

The 1940s: Von Neumann and Zuse

- Konrad Zuse (Plankalkul)
  - in Germany - in isolation because of the war
  - defined Plankalkul (program calculus) circa 1945 but never implemented it.
  - Wrote algorithms in the language, including a program to play chess.
  - His work finally published in 1972.
  - Included some advanced data type features such as
    - » Floating point, used two's complement and hidden bits
    - » Arrays
    - » records (that could be nested)

Plankalkul notation

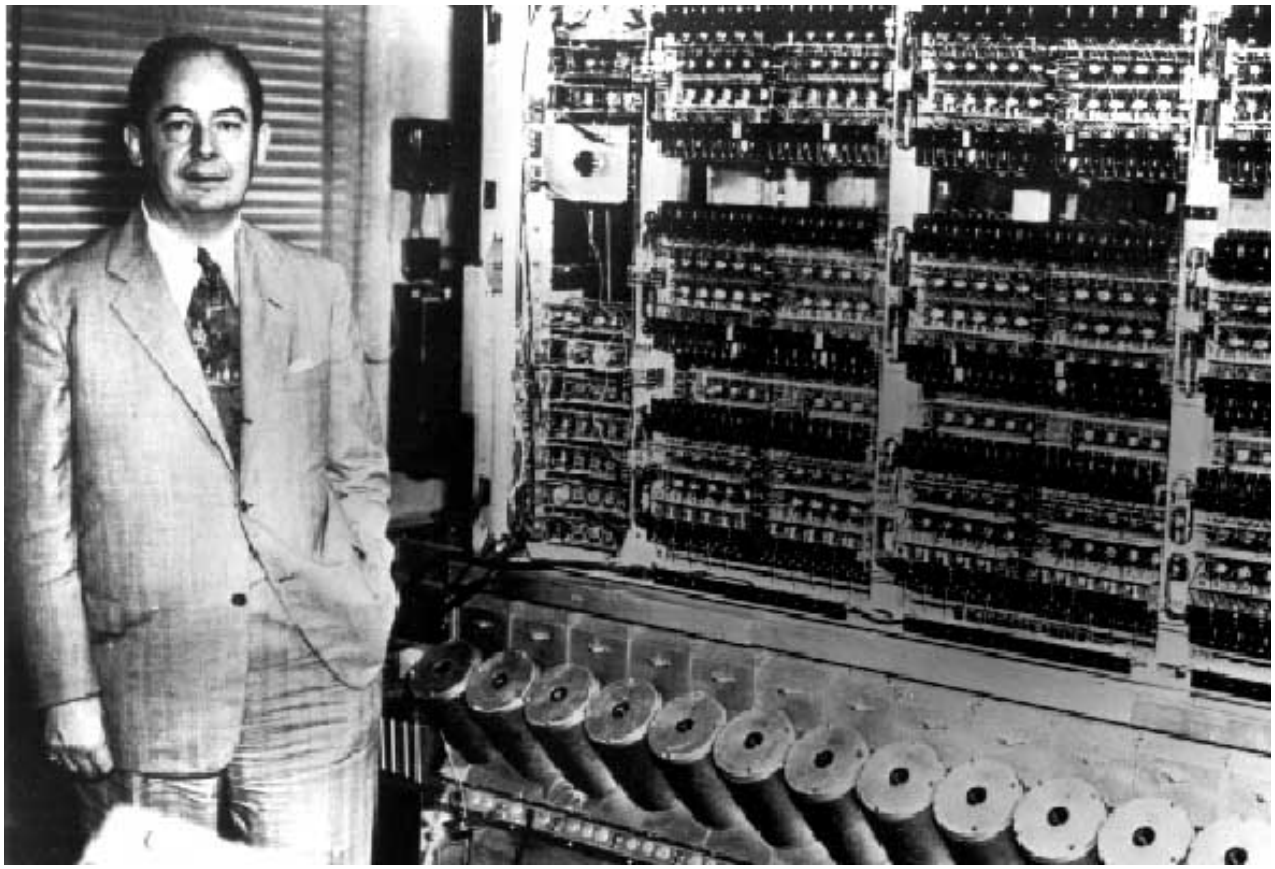
A(7) := 5 * B(6)

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

The 1940s: Von Neumann and Zuse

Von Neumann led a team that built computers with stored programs and a central processor

ENIAC was programmed with patch cords



Von Neuman with ENIAC

Machine Codes (40's)

- Initial computers were programmed in raw machine codes.
- These were entirely numeric.
- What was wrong with using machine code? Everything!
  - Poor readability
  - Poor modifiability
  - Expression coding was tedious
  - Inherit deficiencies of hardware, e.g., no indexing or floating point numbers

The 1950s: The First Programming Language

- **Pseudocodes:** interpreters for assembly language
- **Fortran:** the first higher level programming language
- **COBOL: t**he first business oriented language
- **Algol:** one of the most influential programming languages ever designed
- **LISP:** the first language outside the von Neumann model
- **APL:** A Programming Language

Pseudocodes (1949)

- Short Code or SHORTCODE - John Mauchly, 1949.
- Pseudocode interpreter for math problems, on Eckert and Mauchly's BINAC and later on UNIVAC I and II.
- Possibly the first attempt at a higher level language.
- Expressions were coded, left to right, e.g.:

  X0 = sqrt(abs(Y0))

  00 X0 03 20 06 Y0

- Some operations:

01 –	06 abs	1n (n+2)nd power
02)	07 +	2n (n+2)nd root
03 =	08 pause	4n if <= n
04 /	09 (	58 print & tab

More Pseudocodes

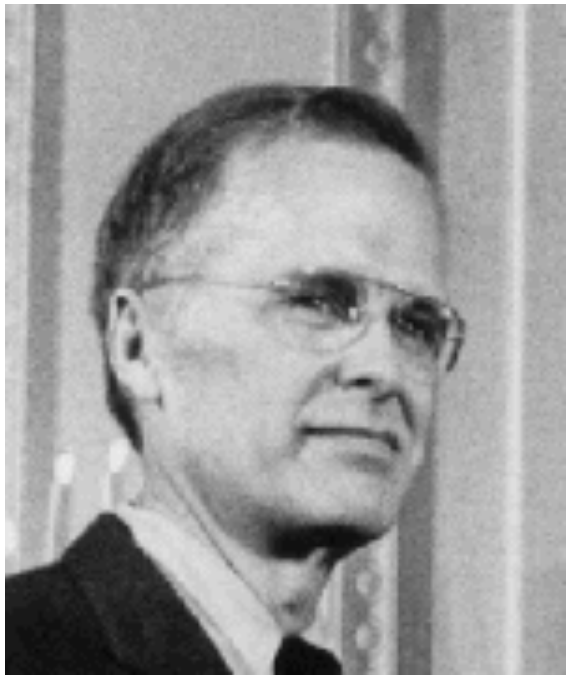
Speedcoding; 1953-4

- A pseudocode interpreter for math on IBM 701, IBM 650.
- Developed by John Backus
- Pseudo ops for arithmetic and math functions
- Conditional and unconditional branching
- Autoincrement registers for array access
- Slow but still dominated by slowness of s/w math
- Interpreter left only 700 words left for user program

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

Fortran (1954-57)



- FORmula TRANslator
- Developed at IBM under the guidance of John Backus primarily for scientific, computational programming
- Dramatically changed the way computers used
- Has continued to evolve, adding new features and concepts.
  - FORTRAN IV, FORTRAN 77, FORTRAN 2008 (!)
- Always among the most efficient compilers, producing fast code

Fortran 77 Examples

```
C Hello World in Fortran 77
C (lines must be 6 characters indented)
      PROGRAM HELLOW
      WRITE(UNIT=*, FMT=*) 'Hello World'
      END

      PROGRAM SQUARE
      DO 15, I = 1,10
      WRITE(*,*) I*I
 15   CONTINUE
      END
```

Fortran 0 and 1

FORTRAN 0 – 1954 (not implemented)

FORTRAN I - 1957

Designed for the new IBM 704, which had index registers and floating point hardware

Environment of development:

Computers were small and unreliable
Applications were scientific
No programming methodology or tools
Machine efficiency was most important

Impact of environment on design

- No need for dynamic storage
- Need good array handling and counting loops
- No string handling, decimal arithmetic, or powerful input/output (commercial stuff)

Fortran I Features

- Names could have up to six characters
- Post-test counting loop (DO I=1, 100)
- Formatted I/O
- User-defined subprograms
- Three-way selection statement (arithmetic IF with GOTO)
  IF (ICOUNT-1) 100 200 300
- Implicit data typing statements
  variables beginning with i, j, k, l, m or n were integers, all else floating point
- No separate compilation
- Programs larger than 400 lines rarely compiled correctly, mainly due to poor reliability of the 704
- Code was very fast
- Quickly became widely used

Fortran II, IV and 77

FORTRAN II - 1958

- Independent compilation
- Fix the bugs

FORTRAN IV - 1960-62

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

FORTRAN 77 - 1978

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

Fortran 90 (1990)

Added many features of more modern programming languages, including

- Pointers
- Recursion
- CASE statement
- Parameter type checking
- A collection of array operations, DOTPRODUCT, MATMUL, TRANSPOSE, etc
- dynamic allocations and deallocation of arrays
- a form of records (called derived types)

COBOL

- COmmon Business Oriented Language
- Principal mentor: (RADM Dr.) Grace Murray Hopper (1906-1992)
- *Based on FLOW-MATIC* which had such features as:
  - 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
- CODASYL committee (Conference on Data Systems Languages) developed a programming language by the name of COBOL

COBOL

First CODASYL Design Meeting - 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 were all from computer manufacturers and DoD branches

Design Problems: arithmetic expressions? subscripts? Fights among manufacturers

COBOL

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

Cobol Example

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HelloWorld.
AUTHOR. Fabritius.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.

DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.

PROCEDURE DIVISION.
DISPLAY "Hello World".
STOP RUN.
```

BASIC (1964)

- Beginner's All purpose Symbolic Instruction Code
- Designed by Kemeny & Kurtz at Dartmouth for the GE 225 with the goals:
  - Easy to learn and use for non-science students and as a path to Fortran and Algol
  - Must be “pleasant and friendly”
  - Fast turnaround for homework
  - Free and private access
  - User time is more important than computer time
- Well suited for implementation on first PCs (e.g., Gates and Allen’s 4K Basic interpreter for the MITS Altair personal computer (circa 1975)
- Current popular dialects: Visual BASIC

BASIC Examples

```
PRINT "Hello World"
```

```
FOR I=1 TO 10
   PRINT I*I;
NEXT I
```

LISP (1959)

- LISt Processing language (Designed at MIT by McCarthy)
- *AI research needed a language that:*
  - Process data in lists (rather than arrays)
  - Symbolic computation (rather than numeric)
- One universal, recursive data type: the s-expression
  - An s-expression is either an atom or a list of zero or more s-expressions
- Syntax is based on the lambda calculus
- *Pioneered functional programming*
  - No need for variables or assignment
  - Control via recursion and conditional expressions
- Status
  - Still the dominant language for AI
  - COMMON LISP and Scheme are contemporary dialects
  - ML, Miranda, and Haskell are related languages

LISP Examples

```
(print "Hello World")

(defun fact (n)
  (if (zerop n)
     1
     (* n (fact (1- n)))))

(format t “factorial of 6 is: ~A~%" (fact 6))

(defun print-squares (upto)
  (loop for i from 1 to upto
        do (format t "~A^2 = ~A~%" i (* i i))))
```

Algol

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

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

Algol 60 Examples

```
'begin'  --Hello World in Algol 60
   outstring(2, 'Hello World');
'end'
```

```
'begin' 'comment' Squares from 1 to 10
   'integer' I;
   'for' i := 1 'step' 1 'until' 10 'do'
   'begin'
      outinteger(2,i*i);
   'end' --for
'end' --program
```


Algol 58 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

Algol 60

Modified ALGOL 58 at 6-day meeting in Paris adding such new features as:

- 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)

Algol 60 (1960)

Failure: Never widely used, especially in U.S., mostly because

1. No I/O and the character set made programs nonportable
2. Too flexible--hard to implement
3. Entrenchment of FORTRAN
4. Formal syntax description
5. Lack of support of IBM

APL

- Designed by Ken Iverson at Harvard in late 1950's
- APL = A Programming Language
- A language for programming mathematical computations
  - especially those using matrices
- Functional style and many whole array operations
- Drawback is requirement of special keyboard

APL Examples

- APL required a special character set, usually provided by an IBM Selectric typewriter
- Here's an example that prints the squares of the first 10 integers: (ι 10) × (ι 10)
  - ι (iota) is an operator takes a number and returns a vector from 1 to that number
- The programming paradigm was focused on vector and matrix operations

The 1960s: An Explosion in Programming Languages

- The development of hundreds of programming languages
- PL/1 designed in 1963-4
  - supposed to be all purpose
  - combined features of FORTRAN, COBOL and Algol 60 and more!
  - translators were slow, huge and unreliable
  - some say it was ahead of its time.....
- Algol68
- SNOBOL
- Simula
- BASIC

PL/I

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

  Business computing
    - IBM 1401, 7080 computers
    - COBOL
    - GUIDE user group
- IBM’s goal: develop a single computer (IBM 360) and a single programming language (PL/I) that would be good for scientific and business applications.
- Eventually grew to include virtually every idea in current practical programming languages.

PL/I

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
- Subsets (e.g. PL/C) developed which were more manageable

Simula (1962-67)

- Designed and built by Ole-Johan Dahl and Kristen Nygaard at the Norwegian Computing Centre (NCC) in Oslo between 1962 and 1967
- Originally designed and implemented for discrete event simulation
- Based on ALGOL 60

Primary Contributions:

- Coroutines - a kind of subprogram
- Classes (data plus methods) and objects
- Inheritance
- Dynamic binding

=> Introduced the basic ideas that developed into object-oriented programming.

Algol 68

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:*
  - User-defined data structures
  - Reference types
  - Dynamic arrays (called flex arrays)
- *Comments:*
  - Had even less usage than ALGOL 60
  - Had strong influence on subsequent languages, especially Pascal, C, and Ada

The 1970s: Simplicity, Abstraction, Study

- Algol-W - Nicklaus Wirth and C.A.R. Hoare
  - reaction against 1960s
  - Simplicity, used as a teaching language in some places
- Pascal
  - small, simple, efficient structures
  - for teaching programming
- C - 1972 - Dennis Ritchie
  - aims for simplicity by reducing restrictions of the type system
  - allows access to underlying system
  - interface with O/S - UNIX

Pascal (1971)

- Designed by Wirth, who quit the ALGOL 68 committee because he didn't like the direction of that work
- Designed for teaching structured programming
- Small, simple
- Introduces some modest improvements, such as the case statement
- Was widely used for teaching programming in the 1980s
  - CMSC 201 used Pascal up to Spring 1994

C (1972-)

- Designed for systems programming at Bell Labs by Dennis Richie and colleagues.
- Evolved primarily from B, but also ALGOL 68
- Powerful set of operators, but poor type checking
- Initially spread through UNIX and the availability of high quality, free compilers.

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
- **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)

The 1980s: Consolidation and New Paradigms

- Ada
  - US Department of Defence
  - European team lead by Jean Ichbiah
- Functional programming
  - Scheme, ML, Haskell
- Logic programming
  - Prolog
- Object-oriented programming
  - Smalltalk, C++, Eiffel

Ada

- In study done in 73-74 it was determined that the US DoD was spending $3B annually on software, over half on embedded computer systems
- The *Higher Order Language Working Group* was formed and initial language requirements compiled and refined in 75-76 and existing languages evaluated
- In 1997, it was concluded that none were suitable, though Pascal, ALGOL 68 or PL/I would be a good starting point
- Language DoD-1 was developed thru a series of competitive contracts

Ada


Library of Congress

- Renamed Ada in May 1979
- Reference manual, Mil. Std. 1815 approved 10 December 1980. (Ada Bryon was born 10/12/1815)
- Ada was “mandated” for use in DoD work during late 80’s and early 90’s.
- Ada95, a joint ISO and ANSI standard, accepted in February 1995 and included many new features.
- The Ada Joint Program Office (AJPO) closed 1 October 1998 (Same day as ISO/IEC 14882:1998 (C++) published!)

Ada

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
- Very difficult to mandate programming technology

Logic Programming: Prolog

- 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

Prolog Example Program

```
parentOf(adam, able).
parentOf(eve, able).
parentOf(adam, cain).
parentOf(eve, cain).
male(adam).
female(eve).
motherOf(X,Y) :- parentOf(X,Y), female(X).
fatherOf(X,Y) :- parentOf(X,Y), female(X).
siblings(X,Y) :- parentOf(X,P1), parentOf(Y,P1), not(X=Y).
```

Functional Programming

- **Common Lisp:** consolidation of LISP dialects spured practical use, as did the development of Lisp Machines.
- **Scheme:** a simple and pure LISP like language used for teaching programming.
- **Logo:** Used for teaching young children how to program.
- **ML:** (MetaLanguage) a strongly-typed functional language first developed by Robin Milner in the 70's
- **Haskell:** polymorphicly typed, lazy, purely functional language.

Smalltalk (1972-80)

- Developed at Xerox PARC by Alan Kay and colleagues (esp. Adele Goldberg) inspired by Simula 67
- First compilation in 1972 was written on a bet to come up with "the most powerful language in the world" in "a single page of code".
- In 1980, Smalltalk 80, a uniformly object-oriented programming environment became available as the first commercial release of the Smalltalk language
- Pioneered the graphical user interface everyone now uses
- Saw some industrial use in late 80's and early 90's

C++ (1985)

- Developed at Bell Labs by Stroustrup
- Evolved from C and SIMULA 67
- Facilities for object-oriented programming, taken partially from SIMULA 67, 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

•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

1990's: the Internet and web

During the 90's, Object-oriented languages (mostly C++) became widely used in practical applications

The Internet and Web drove several phenomena:

- Adding concurrency and threads to existing languages
- Increased use of scripting languages such as Perl and Tcl/Tk
- Java as a new programming language

Java

- Developed at Sun in the early 1990s with original goal of a language for embedded computers
- Principals: Bill Joy, James Gosling, Mike Sheradin, Patrick Naughton
- Original name, Oak, changed for copyright reasons
- Based on C++ but significantly simplified
- Supports *only* OOP
- Has references, but not pointers
- Includes support for applets and a form of concurrency

C# (C Sharp)

- Microsoft and Sun were bitter rivals in the 90s
- C# is Microsoft's answer to Java
- C# is very similar to Java with (maybe) some minor improvements
- If you know Java, learning C# should be easy
- However: both languages have extensive libraries, and mastering them is a big part of mastering the language.

Scripting Languages

- Scripting languages like Perl, Ruby, Javascript and PHP have become important
- They shine at connecting diverse pre-existing components to accomplish new tasks
- Cf. shell languages in Unix
- Typical properties include:
  - privileging rapid development over execution efficiency
  - implemented with interpreters rather than compilers
  - strong at communication with program components in other languages

The future

- The 60's dream was a single all purpose language (e.g., PL/I, Algol)
- The 70s and 80s dream expressed by Winograd (1979)

  "Just as high-level languages allow the programmer to escape the intricacies of the machine, higher level programming systems can provide for manipulating complex systems. We need to shift away from algorithms and towards the description of the properties of the packages that we build. Programming systems will be declarative not imperative"
- Will that dream be realised?
- Programming is not yet obsolete