
CS315
Joie Ann M. Mac

History

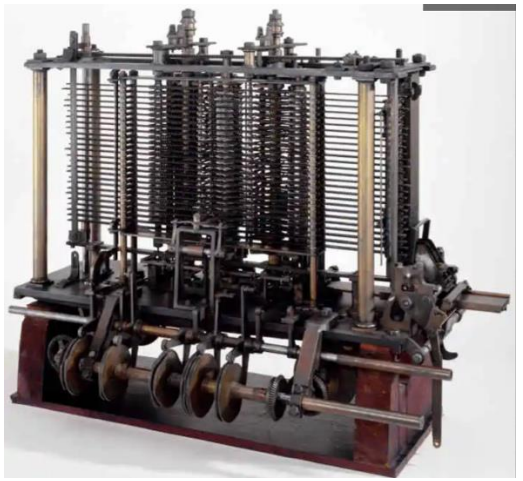
- Early History : The first programmers
- The 1940s: Von Neumann and Zuse
- The 1950s: The First Programming Language
- The 1960s: An Explosion in Programming languages
- The 1970s: Simplicity, Abstraction, Study
- The 1980s: Consolidation and New Directions
- The 1990s: Internet and the Web
- The 2000s: tbd

Early History: The First Programmer

- Jacquard loom of early 1800s
  - Translated card patterns into cloth designs

- Charles Babbage's analytical engine (1830s & 40s)

  Programs were cards with data and operations



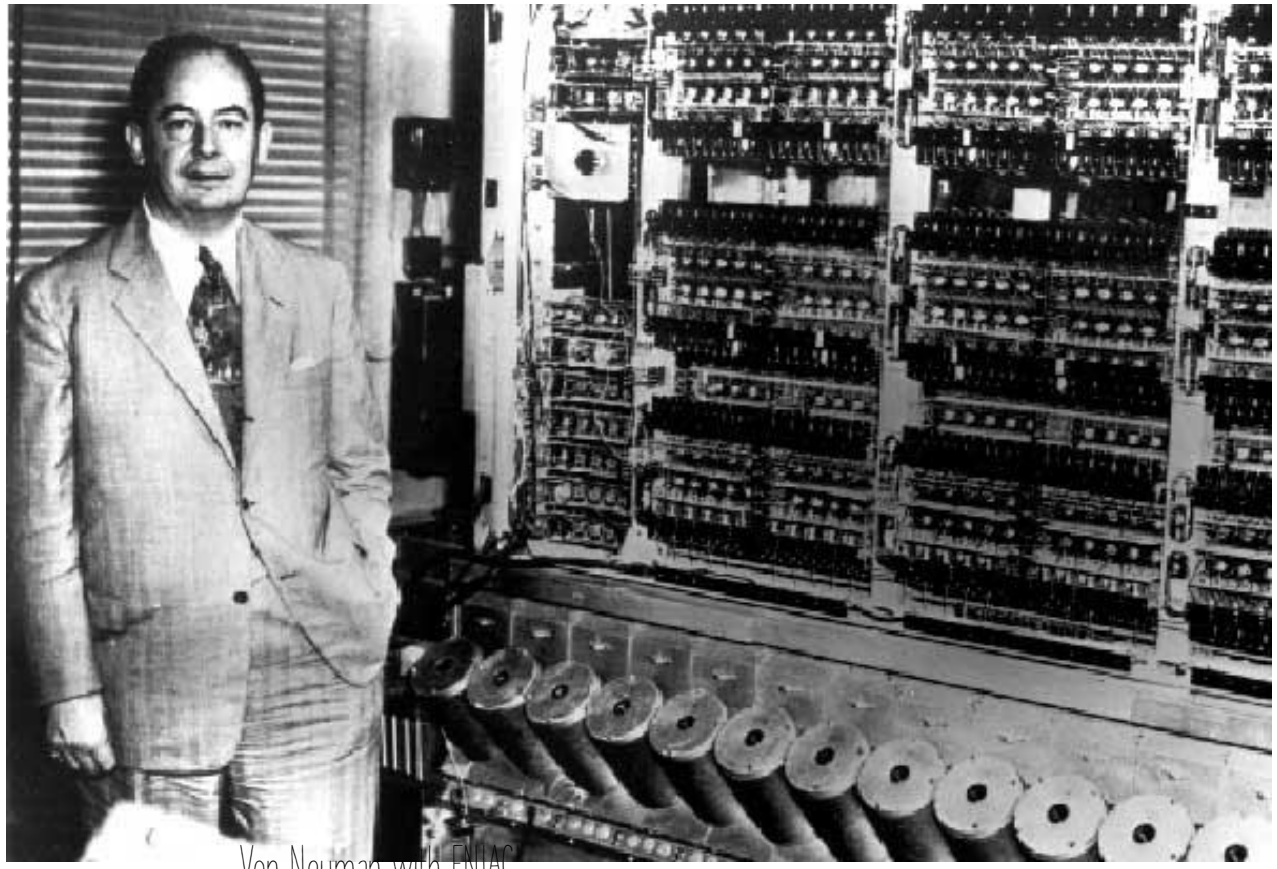
Early History: The First Programmer

- 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.”

The 1940s: Von Neumann and Zuse

- John Von Neumann led a team that built computers with stored programs and a central processor
- ENIAC, however, was also programmed wit patch cords.



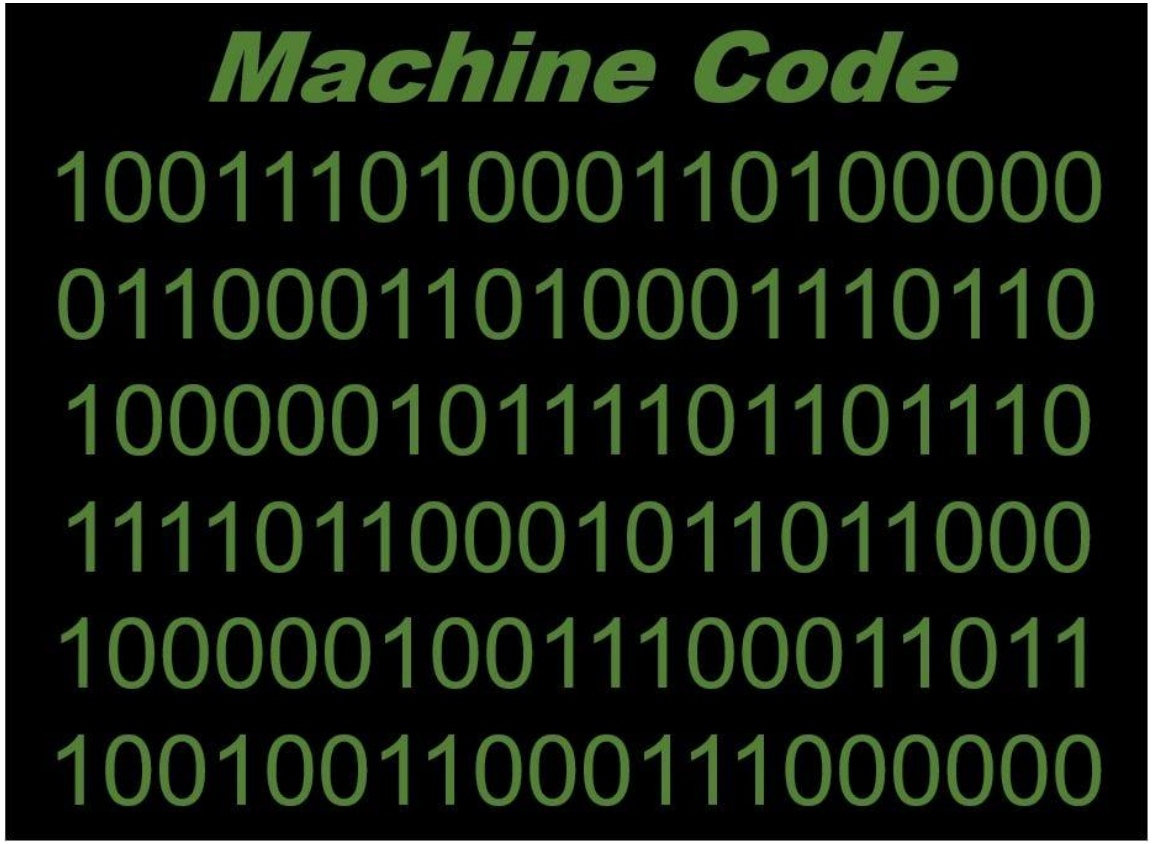
Von Neuman with ENIAC

Konrad Zuse and Plankalkul

- Plankalkül was an attempt by Konrad Zuse in the 1940's to devise a notational and conceptual system for writing what today is termed a program.
- Although this early approach to a programming language did not lead to practical use, the plan is described here because it contains features that are standard in today's programming languages

Machine Code (1940's)

- Initial computers were programmed in raw machine code.
- 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



Pseudocodes (1949)

- Short Code or SHORTCODE - John Mauchly, 1949.
- was one of the first higher-level languages developed for an electronic computer
- 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.

```
a = (b + c) / b * c
```

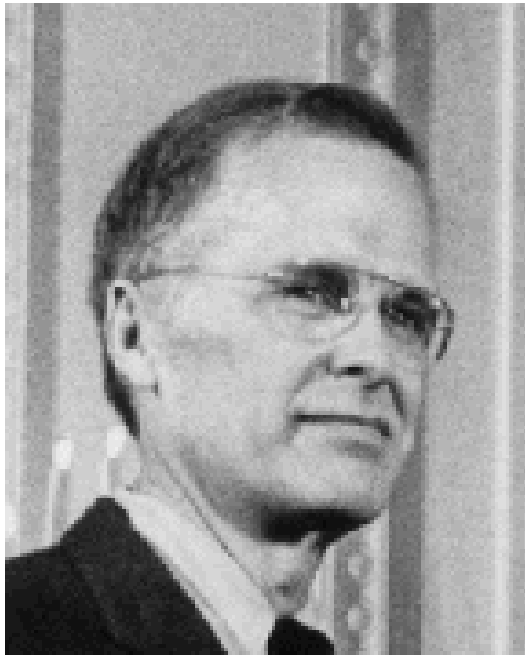
was converted to Short Code by a sequence of substitutions and a final regrouping:

```
X3 =  (  X1 +  Y1 )  /  X1 * Y1   substitute variables
X3 03 09 X1 07 Y1 02 04 X1   Y1   substitute operators and parentheses.
                                       Note that multiplication is
                                       represented by juxtaposition.
07Y10204X1Y1                      group into 12-byte words.
0000X30309X1
```

The 1950s: The First Programming Language

- **Pseudocodes**: interpreters for assembly language like
- **Fortran**: the first higher level programming language
- **COBOL**: the first business oriented language
- **BASIC**: a family of general-purpose, high-level programming languages designed for ease of use
- **Algol:** one of the most influential programming languages ever designed
- **LISP:** the first language to depart from the procedural paradigm
- **APL:** an array-language

Fortran (1954-57)



- FORmula TRANslator
- Developed at IBM under the guidance of John Backus primarily for scientific programming
- Dramatically changed forever the way computers used
- Has continued to evolve, adding new features & concepts.
  - FORTRAN II, FORTRAN IV, FORTRAN 66, FORTRAN 77, FORTRAN 90
- Always among the most efficient compilers, producing fast code
- Still popular, e.g. for supercomputers

COBOL (1959)

- COmmon Business Oriented Language
- Principal mentor: (Rear Admiral 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

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

Algol (1958-1960)

- A computer programming language designed by an international committee of the Association of Computing Machinery (ACM), led by Alan J. Perlis of Carnegie Mellon University, during 1958–60 for publishing algorithms, as well as for doing computations

- ALGOL contributed a notation for describing the structure of a programming language, Backus–Naur Form, which in some variation became the standard tool for stating the syntax (grammar) of programming languages

```
begin
        real x;
        integer i, j;
        for i := 2 until Z do begin
                x := Y[i];
                for j := i-1 step -1 until 1 do
                        if x >= A[j] then begin
                                A[j+1] := x; goto Found
                        end else
                                A]j+1] := A[j];
                A[1] := x;
        Found:
                end
        end
end Sort
```

LISP (1960)

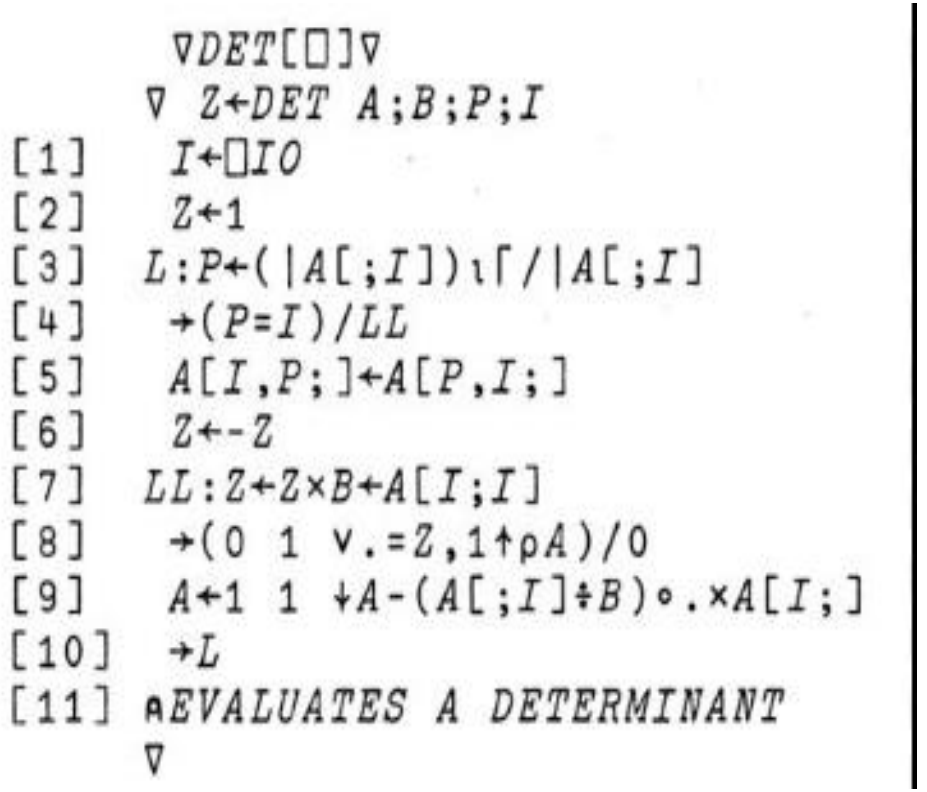
- a computer programming language developed about 1960 by John McCarthy at the Massachusetts Institute of Technology (MIT)
- LISP was founded on the mathematical theory of recursive functions (in which a function appears own definition).

```
(define (reduce f a x y b fx fy)
  (cond ((close-enough? a b) x)
        ((> fx fy)
         (let ((new (x-point a y)))
           (reduce f a new x y (f new) fx)))
        (else
         (let ((new (y-point x b)))
           (reduce f x y new b fy (f new))))))
```

SICP, Technical Report 735, 1983

APL (1960s)

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



```
      ∇DET[⎕]∇
      ∇ Z←DET A;B;P;I
[1]    I←⎕IO
[2]    Z←1
[3]   L:P←(|A[;I])⍳⌈/|A[;I]
[4]    →(P=I)/LL
[5]    A[I,P;]←A[P,I;]
[6]    Z←-Z
[7]   LL:Z←Z×B←A[I;I]
[8]    →(0 1 ∨.=Z,1↑⍴A)/0
[9]    A←1 1 ↓A-(A[;I]÷B)∘.×A[I;]
[10]   →L
[11]  ⍝EVALUATES A DETERMINANT
      ∇
```

The 1960s: An Explosion in Programming Languages

- The development of hundreds of programming languages
- PL/I designed in 1963-4
- Algol 68
- Simula
- BASIC

The 1970s: Simplicity, Abstraction, Study

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

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, non-integer array indices)

The 1980s: Consolidation and New Paradigms

- Ada
  - US Department of Defence
  - European team lead by Jean Ichbiah. (Sam Lomonaco was also on the ADA team :-)
- Functional programming
  - Scheme, ML, Haskell
- Logic programming
  - Prolog
- Object-oriented programming
  - Smalltalk, C++, Eiffel

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
- Industrial use continues to the present day

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 (i.e. threads)

The future

- In the 60's, the 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"
- Programming is not yet obsolete

Why Study Programming Languages

Why Study PLs?

- Increased ability to express ideas

  -The language in which they develop software places limits on the kinds of control structures, data structures, and abstractions they can use; thus, the forms of algorithms they can construct are likewise limited.

Why Study PLs?

- Improved background for choosing appropriate languages

  -If these programmers were familiar with a wider range of languages and language constructs, they would be better able to choose the language with the features that best address the problem

Why Study PLs?

- Increased ability to learn new languages

  - Once a thorough understanding of the fundamental concepts of languages is acquired, it becomes far easier to see how these concepts are incorporated into the design of the language being learned.

Why Study PLs?

- Better understanding of significance of implementation
  - An understanding of implementation issues leads to an understanding of why languages are designed the way they are

Why Study PLs?

- Better use of languages that are already known

  -Programmers can learn about previously unknown and unused parts of the languages they already use and begin to use those features.

Why Study PLs?

- Overall advancement of computing

  - The most popular languages are not always the best available.

  -Those in positions to choose languages were not sufficiently familiar with programming language concepts.

  -ALGOL 60 vs. Fortran

Attendance: