

CS 403: A Brief (and Pretty Incomplete) History of Programming Languages

Stefan D. Bruda

Fall 2021



- “Prehistory”
- 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: The explosion of the World Wide Web
- The 21st century

- Cuneiform writing used in the **Babylon**, founded by Hammurabi around **1790 BC**

poems, stories, contracts, records, astronomy, **math**



Famous Babylonian math tablet (Plimpton 322) involving Pythagorean triples, $a^2 + b^2 = c^2$ – with a mistake! (or **bug**)

Weird math (base 60!)

two characters to express a (base-60) digit

decimal point not specified (must be figured out from context)

WRITTEN LANGUAGE TO DESCRIBE COMPUTATIONAL PROCEDURES



A cistern.

The length equals the height.

A certain volume of dirt has been excavated.

The cross-sectional area plus this volume comes to 110.

The length is 30. What is the width?

You should multiply the length, 30, by ...

— Translation by Donald Knuth

- No variables
- Instead, numbers serve as a running example of the procedure being described
 - “This is the procedure”
- Programming is among the earliest uses to which written language was put

Programming languages design has tried to get as close to that as possible from the very beginning. . .



- Abū 'Abdallāh Muḥammad ibn Mūsā al-Khwārizmī, or **Mohammed Al-Khorezmi** for short (Baghdad, 780–850 BC)

One little book: “The Compendious Book on Calculation by Completion and Balancing”

Compilation and extension of known rules for solving quadratic equations and other problems

Used as a mathematics text in Europe for **eight hundred years**

The book is considered the foundation of **algebra**

Invention of the notions of **algorithms** and **data structures**

- Other early algorithms:

Euclid (300 BC): an algorithm for computing the **GCD of two numbers**

Eratosthenes (about same time): one of the most efficient algorithms for finding small primes (the **sieve of Eratosthenes**)

Alexander de Villa Dei (1220 AD): Canto de Algorismo = algorithms in Latin verse

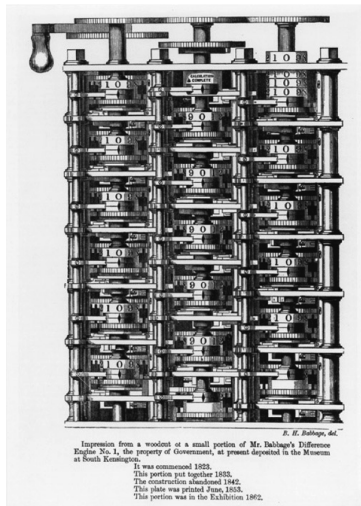
- Natural language (even poetry!) plus math rather than programming languages

- **Jacquard loom** (early 1800s) translated card patterns into cloth designs
- **Charles Babbage's Analytical Engine** (1830s & 40s)

First programmer: **Augusta Ada King, Countess of Lovelace** (today commonly known as **Ada Lovelace**)

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.

Programs were **punched cards** containing **data** and **operations**





- Harvard Mark I (1943) – Howard Aiken (IBM), Grace Hopper (Navy) → first electro-mechanical computer
Harvard Mark II: **First computer bug**
- ENIAC (1946) – Presper Eckert, John Mauchly (U. Penn.) → First electronic computer
- Programming was manual, with switches and cables
- **John von Neumann** led a team that built computers with stored programs and a central processor (as we know them today)
- **Konrad Zuse** designed the first programming language as we know it (**Plankalkul** = program calculus)

In Germany, in isolation because of the war; work finally published in 1972

Advanced data type features: floating point, arrays, records

Invariants for correctness

Rather cumbersome notation

$$A[7] := 5 \times B[6] \quad \rightarrow \quad \begin{array}{c|c} & 5 \quad * \quad B \quad \Rightarrow \quad A \\ V & 6 \quad 7 \quad (\text{subscripts}) \\ S & 1.n \quad 1.n \quad (\text{data types}) \end{array}$$

Never implemented

THE FIRST COMPUTER BUG!



92.

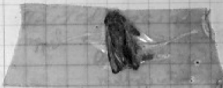
9/9

0800 Andam started { 1.2700 9.037 847 025
 1000 " stopped - andam ✓ 9.037 846 985 correct
 1300 (032) MP-MC 2.130476415 (2) 4.615925059(-2)
 (033) PRO 2 2.130476415
 correct 2.130676415

Relays 6-2 in 033 failed special speed test
 in relay 11,000 test.

Relays changed

1100 Started Cosine Tape (Sine check)
 1525 Started Multi-Adder Test.

1545  Relay #70 Panel F
 (moth) in relay.

First actual case of bug being found.

1650 Andam started.
 1700 closed down.



- **FORTRAN** (1957, John Backus)

FORMula TRANslator – designed for scientific programming

Many new features over time: FORTRAN II, FORTRAN IV, FORTRAN 66, FORTRAN 77 (structured programs, char's), Fortran 90 (arrays, modules), Fortran 2003 (objects), Fortran 2008 (concurrent programming)

Very efficient compilation into fast machine code

- **COBOL** (1960, Grace Hopper)

mathematical programs should be written in mathematical notation, data processing programs should be written in English statements — G. Hopper, 1953

Committee sponsored by US Department of Defence

Biggest contribution was the idea that programs should be written in a way that is easily understood

Adopted widely by businesses for record-keeping applications

Record structure, separation of data from execution part, versatile formatting of output using “pictures”

ANSI standards (1968, 1974, 1985)



```
IMPLICIT INTEGER (A-Z)
DIMENSION ORD(N),POPLST(2,20)
INTEGER X,XX,Z,ZZ,Y
INTEGER A(N)
NDEEP=0
U1=N
L1=1
DO 1 I=1,N
1  ORD(I)=I
2  IF (U1.LE.L1) RETURN
3  L=L1
  U=U1
4  P=L
  Q=U
  X=A(ORD(P))
  Z=A(ORD(Q))
  IF (X.LE.Z) GO TO 5 ....
5
```



ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-4381.

OBJECT-COMPUTER. IBM-4381.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 INPUT-FIELD.

05 INPUT-VALUE PIC 99 VALUE ZERO.

01 CALCULATION-FIELD.

05 SUM-VALUE PIC 9(03) VALUE ZERO.

05 AVERAGE-VALUE PIC 9(03)V99 VALUE ZERO.

01 OUTPUT-FIELD.

05 EDIT-FIELD PIC ZZ9.99 VALUE ZERO.

PROCEDURE DIVISION.

1000-MAIN.

PERFORM 2000-INPUT-ADD 10 TIMES.

DIVIDE 10 INTO SUM-VALUE GIVING AVERAGE-VALUE.

2000-INPUT-ADD. ...



- **Algol 60**

General, expressive language; most current imperatives are derivatives

Introduced many modern concepts

structured programming	reserved keywords	type declarations
recursion	stack dynamic arrays	call-by-value
user defined types	free-format	

Stack-based run time environment

Great success and also great failure (ahead of its time, too complex, lack of I/O, lack of support from IBM) → entrenchment of Fortran

- **LISP** (John McCarthy, MIT)

LISt Processing → the main data structure is the (singly linked) list

Untyped, messy language, but good for problems we solve by trial and error

(quick prototyping) → used in many AI applications

Historically inefficient on Von Neumann machines

Main processing unit: the recursive function → influenced the modern functional languages such as ML, Miranda, Haskell

Contemporary variants include Common Lisp, Scheme, Emacs Lisp



```
procedure Absmax(a) Size:(n, m) Result:(y) Subscripts:(i, k);
  value n, m;
  array a;
  integer n, m, i, k;
  real y;
comment The absolute greatest element of the matrix a, of size
  n by m is transferred to y, and the subscripts of this element
  to i and k;
begin integer p, q;
  y := 0; i := k := 1;
  for p:=1 step 1 until n do
    for q:=1 step 1 until m do
      if abs(a[p, q]) > y then
        begin y := abs(a[p, q]);
          i := p; k := q
        end
      end
    end
  end
end Absmax
```



```
(defun mapcar (fun list)
  "Applies FUN on every element of LIST and returns the
  list of results (iterative version)."
  (let ((results nil))
    (dolist (x list)
      (setq results (cons (apply #'fun x) results)))
    (reverse results)))

(defun mapcar (fun list)
  "Applies FUN on every element of LIST and returns the
  list of results (recursive version)."
  (cons (apply #'fun (car list))
        (mapcar fun (cdr list))))
```



- Hundreds of languages were developed
- **PL/1** (1964)
 - Combined features of FORTRAN, COBOL, Algol 60 and more!
 - Translators were slow, huge, and unreliable
 - Some say it was ahead of its time. . .
- **Algol 68** → still ahead of its time!
- **Simula** (or what would be called today Object-oriented Algol)
- **BASIC**
- etc.



- **Algol-W** then **Pascal** (Nicklaus Wirth and C.A.R.Hoare) → small, simple, efficient (reaction against the 60s), ideal for teaching
- **C** (Dennis Ritchie)
 - Constructed as a portable assembler to build Unix for various architectures
 - But also has modern features (structured programming, data structures, etc.)
 - The primary API for Unix (Mac OS, Linux, etc.) is still C!
- **Euclid** (University of Toronto, 1977)
 - Main goal → formal program verification
 - extends Pascal to include abstract data types
- **Scheme** (1978, MIT) → simplified, cleaner Lisp



```
#include <stdio.h>
main(t,_,a)
char*a;
{return!0<t?t<3?
main(-79,-13,a+
main(-87,1-_,
main(-86, 0, a+1 )
+a)):
1,
t<_?
main(t+1, _, a )
:3,
main ( -94, -27+t, a )
&&t == 2 ?_
<13 ?
main ( 2, _+1, "%s %d %d\n" )
:9:16:
t<0?
t<-72?
main( _, t,
"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{**+,/w{%/+,/w#q#n+,/#{l+,/n{n+,/+##n+,/#;\
#q#n+,/+k#;+,,/'r : 'd*'3,){w+K w'K:'+'e#';dq#'l q#'+d'K#!/+k#;\
q#+'r}eKK#}w'r}eKK{nl}"/#;#q#n'}{)#}w''){}{nl}'/'+#n';d}rw' i;# ){}{nl}!/n{n#'; \
r{#w'r nc{nl}"/#{l,+'K {rw' iK{;[{nl}]/w#q#\
\
n'wk nw' iwk{KK{nl}!/w{%/l##w#' i; :{nl}'/*{q#ld;r'}{nlwb!/*de}'c ;;\
{nl}-{}rw}'/+,}##'*)#nc,',#nw}'/k'd'+e}+;\
#'rdq#w! nr'/ ' ) }+}{rl#'{n' ' )# }'+}##(!/"/)
:
t<-50?
_==*a ?
putchar(31[a]):
main(-65,_,a+1)
:
main((a == '/') + t, _, a + 1 )
:
0<t?
main ( 2, 2 , "%s")
/*...*/
```



- **ML** → mostly functional language (like Lisp) with cleaner (math-like) syntax
- **Prolog** (Université Aix Marseille)
 - PROgrammation en LOGique / PROgramming in LOGic → describes the problem at hand as known facts and inference rules
 - Notable industrial uses: IBM Watson, Apache UIMA, and the... Buran spacecraft
- Objects 'r' Us:
 - Smalltalk** → the purest example of object-oriented language
 - C++** → extend a popular language (C) with strongly typed object system
 - Eiffel** → object-oriented Pascal



- 1990s

Java → eliminate the non-object-oriented features of C++

Haskell → purely functional programming language

```
quicksort [] = []
```

```
quicksort (x:xs) = quicksort [y|x <- xs, y < x] ++ [x] ++  
                    quicksort [y|x <- xs, y >= x]
```

- 2000s

Python → multi-paradigm language (procedural, object-oriented, functional, etc.)

Languages for the Web

- Java applets

- Languages within Web pages (PHP, server-side includes)

Emphasis on **cross-platform development**

- Develop on PC, run on cell phones, game consoles, and toasters



- Computing devices are ubiquitous, and so is the Internet and Web
- C and C++ are the most widely used system programming languages
- Java had peaked. . . and then came Android
- Most students learn C / C++ or Java
- Web 2.0 programming (server side over client side rendering)
- COBOL and Java are used for business applications
- Fortran is the main language on supercomputers
 - We already have Object-Oriented Fortran!
 - C++ is growing
- Several non-mainstream (but cleaner) languages rising (Ruby, Python, Haskell, Go, Swift, C#, Scratch)



- * Python has risen to be one of the most used languages
 - * Business Intelligence Applications
 - * Automation
 - * Quick Prototyping
- * Rust developed to deal with C++ shortcomings
- * Web programming with frameworks (e.g. React)
- * FORTRAN still main language for supercomputers
- * Matlab, and Python used as scientific programming languages
- * Scratch is the leading language for teaching
- * So much more...



- Strange languages definitely exist
- Case in point: **Brainf**k**

A Brainf**k program has an implicit byte pointer, called “**the pointer**”, which is free to move around within an array of 30,000 bytes, initially all set to zero

The pointer is initialized to point to the beginning of this array

The Brainf**k programming language consists of eight commands, each of which is represented as a single character

- > Increment the pointer
- < Decrement the pointer
- + Increment the byte at the pointer
- Decrement the byte at the pointer
- . Output the byte at the pointer
- , Input a byte and store it in the byte at the pointer
- [Jump past the matching] if the byte at the pointer is zero
-] Jump to the matching [

```
>+++++++ [ <+++++>- ] < . >+++++ [ <++++>- ] < + . ++++++ . . ++
+ . [-] >+++++++ [ <++++>- ] < . # >+++++++ [ <++++>- ] < . >+++
+++++ [ <++++>- ] < . +++ . ----- . ----- . [-] >+++++++ [ <++++
>- ] < + . [-] ++++++++ .
```

CHRONOLOGY

