The History Of Programming Languages

- Prehistory of programming languages
  - The story of the programmers of Babylon
  - The story of Mohammed Al-Khorezmi
  - The story of Augusta Ada, Countess of Lovelace
- Early programming languages
  - The story of the Plankalkül
  - The story of Fortran
  - The story of Lisp
  - The story of Algol
  - The story of Smalltalk
- Our languages
  - The story of Prolog
  - The story of ML
  - The story of Java

Babylon

- Cuneiform writing was used in the Babylon, founded by Hammurabi around 1790 BC
- Many Babylonian clay tablets survive:
  - poems and stories
  - contracts and records
  - astronomy
  - math, base 60

A famous Babylonian math tablet (Plimpton 322) involving Pythagorean triples, $a^2+b^2=c^2$ -- with a mistake!

Babylonian Numbers

- The two Babylonian digits for “1” and “10”, written together, signify a number base 60
- The exponent is not given; the reader must figure it out from the context

$$1{,}10 \quad = \quad \begin{array}{l} 1\times 60^{1}+10\times 60^{0}=70 \\ 1\times 60^{0}+10\times 60^{-1}=1\tfrac{1}{6} \\ 1\times 60^{i+1}+10\times 60^{i} \end{array}$$

A Babylonian Program

- 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 1,10.*
> *The length is 30. What is the width?*
> *You should multiply the length, 30, by …*
>
> *Translation by Donald Knuth*

Programming Language

- 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

- Prehistory of programming languages
  - The story of the programmers of Babylon
  - The story of Mohammed Al-Khorezmi
  - The story of Augusta Ada, Countess of Lovelace
- Early programming languages
  - The story of the Plankalkül
  - The story of Fortran
  - The story of Lisp
  - The story of Algol
  - The story of Smalltalk
- Our languages
  - The story of Prolog
  - The story of ML
  - The story of Java

Baghdad

- Near ancient Babylon
- Founded around 762
- A great center of scholarship, art and poetry
- 780-850: Mohammed Al-Khorezmi, a court mathematician, lived and wrote
- Two little books…

Algebra

- *Kitâ al-jabr wa'l-muqabâla*
- Translated into Latin, spread throughout Europe
- Used as a mathematics text in Europe for *eight hundred years*

Algorithms

- The original is lost
- Latin translation: *Algorthmi de numero Indorum*
- Algorithms for computing with Hindu numerals: base-10 positional system with 0
- A new technology (data structure and algorithms)
- Strongly influenced medieval European mathematics

Other Early Written Algorithms

- Euclid, 300 BC: an algorithm for computing the GCD of two numbers
- Alexander de Villa Dei, 1220 AD: *Canto de Algorismo*, algorithms in Latin verse
- Not programming languages: natural language (even poetry) plus mathematics

- Prehistory of programming languages
  - The story of the programmers of Babylon
  - The story of Mohammed Al-Khorezmi
  - The story of Augusta Ada, Countess of Lovelace
- Early programming languages
  - The story of the Plankalkül
  - The story of Fortran
  - The story of Lisp
  - The story of Algol
  - The story of Smalltalk
- Our languages
  - The story of Prolog
  - The story of ML
  - The story of Java

Augusta Ada

- Daughter of George Gordon, Lord Byron
- Early 1800’s in England (as elsewhere) women were generally denied education, especially math and science
- Ada studied math with a private tutor (as an antidote to feared Byronic tendencies)
- Married at 19 (Lady Lovelace), 3 children

Charles Babbage

- English mathematician
- Inventor of mechanical computers:
  - Difference Engine, construction started but not completed (until a 1991 reconstruction)
  - Analytical Engine, never built

> *I wish to God these calculations had been executed by steam!*
>
> *Charles Babbage, 1821*

Analytical Engine

- Processing unit (the Mill)
- Memory (the Store)
- Programmable (punched cards)
- Iteration, conditional branching, pipelining, many I/O devices

Sketch of the Analytical Engine

- A paper by Luigi Menabrea
- Published 1843
- Translated, with explanatory notes, by A.A.L.
- Algorithms in a real programming language: the machine language of punched cards for the Analytical Engine

Not Just For Numbers

The bounds of *arithmetic* were however outstepped the moment the idea of applying the cards had occurred; and the Analytical Engine does not occupy common ground with mere "calculating machines." … In enabling mechanism to combine together *general* symbols in successions of unlimited variety and extent, a uniting link is established between the operations of matter and the abstract mental processes of the *most abstract* branch of mathematical science.

A.A.L.

- Prehistory of programming languages
  - The story of the programmers of Babylon
  - The story of Mohammed Al-Khorezmi
  - The story of Augusta Ada, Countess of Lovelace
- Early programming languages
  - The story of the Plankalkül
  - The story of Fortran
  - The story of Lisp
  - The story of Algol
  - The story of Smalltalk
- Our languages
  - The story of Prolog
  - The story of ML
  - The story of Java

Konrad Zuse

- Built a mechanical computer in his parents' living room in Berlin in 1936: the Z1
- Metal strips and pins—very different from Babbage's wheelwork
- Programmable using punched tapes
- Binary floating point numbers with an explicit exponent

Early Development

- More computers:
  - Z2 experimented with relays for the ALU
  - Z3: all-relay technology (the first electronic programmable digital computer)
  - Z4: envisioned as a commercial system
- Most designs and prototypes destroyed in the war
- 1945: Zuse flees Berlin with wife and Z4

Plankalkül

- In 1945/46, Zuse completed the design of a programming language: the Plankalkül
- Many advanced ideas:
  - Assignment, expressions, subscripts
  - Constructed types: from primitive (bit) other types are constructed: integers, reals, arrays, etc.
  - Conditional execution, loops, subroutines
  - Assertions
- Many example programs: sorting, graphs, numeric algorithms, syntax analysis, chess

The Notation

- Main line with three underneath:
  - V: variable number
  - K: subscript
  - S: optional comment (showing types)
- **`V0[Z1]+=1`** looks like:

	V	Z	+ 1	$\Rightarrow$	V	Z
V	0	1			0	1
K						
S	$m \times 1 \cdot n$	$1 \cdot n$	$1 \cdot n$		$m \times 1 \cdot n$	$1 \cdot n$

Looks Influential…

- …but it was not: it was not published until 1972, and few people knew of it
- Never implemented: far beyond the state of the art in hardware or software at the time
- Many of Zuse's ideas were reinvented by others

- Prehistory of programming languages
  - The story of the programmers of Babylon
  - The story of Mohammed Al-Khorezmi
  - The story of Augusta Ada, Countess of Lovelace
- Early programming languages
  - The story of the Plankalkül
  - The story of Fortran
  - The story of Lisp
  - The story of Algol
  - The story of Smalltalk
- Our languages
  - The story of Prolog
  - The story of ML
  - The story of Java

The Labor Of Programming

- Programming has always been hard
- In the early days of large-scale digital computers, it was labor-intensive
- Hard to appreciate now, how much tedious work was involved then

The Good Old Days

> In the early years of programming languages, the most frequent phrase we heard was that the only way to program a computer was in octal. Of course a few years later a few people admitted that maybe you could use assembly language…. I have here a copy of the manual for Mark I. I think most of you would be totally flabbergasted if you were faced with programming a computer, using a Mark I manual. All it gives you are the codes. From there on you're on your own to write a program. We were not programmers in those days. The word had not yet come over from England. We were "coders."
>
> Rear Admiral Dr. Grace Murray Hopper

Wish List

- Floating point: coders had to keep track of the exponent manually (Babylonian style)
- Relative addressing: coders kept notebooks of subroutines, but the codes had to be adjusted by hand for the absolute addresses
- Array subscripting help
- Something easier to remember than octal opcodes

Early Aids

- Assemblers
- Programming tools:
  - Short Code, John Mauchly, 1949 (interpreted)
  - A-0, A-1, A-2, Grace Hopper, 1951-1953 (like macro libraries)
  - Speedcoding, John Backus, 1954 (interpreted)
- People began to see that saving programmer time was important

Fortran

- The first popular high-level programming language
- A team led by John Backus at IBM
- "The IBM Mathematical FORmula TRANSlating System: FORTRAN", 1954:
  - supposed to take six months -- took two years
  - supposed to eliminate coding errors and debugging
  - supposed to generate efficient code, comparable with hand-written code -- very successful at this
  - closely tied to the IBM 704 architecture

Separate Compilation

- First Fortran: no separate compilation
- Compiling “large” programs – a few hundred lines – was impractical, since compilation time approached 704 MTTF
- Fortran II added separate compilation
- Later Fortrans evolved with platform independence: no more **`PAUSE`** statement!

> I don’t know what the language of the year 2000 will look like, but I know it will be *called* FORTRAN.
> *C.A.R. Hoare,* 1982

Fortran's Influence

- Many languages followed, but all designers learned from Fortran
- Fortran team pioneered many techniques of scanning, parsing, register allocation, code generation, and optimization

John Backus

- Many contributions to programming languages: Fortran, Algol 58 and 60, BNF, FP (a purely functional language)

> My point is this: while it was perhaps natural and inevitable that languages like FORTRAN and its successors should have developed out of the concept of the von Neumann computer as they did, the fact that such languages have dominated our thinking for twenty years is unfortunate. It is unfortunate because their long-standing familiarity will make it hard for us to understand and adopt new programming styles which one day will offer far greater intellectual and computation power.
>
> *John Backus, 1978*

- Prehistory of programming languages
  - The story of the programmers of Babylon
  - The story of Mohammed Al-Khorezmi
  - The story of Augusta Ada, Countess of Lovelace
- Early programming languages
  - The story of the Plankalkül
  - The story of Fortran
  - The story of Lisp
  - The story of Algol
  - The story of Smalltalk
- Our languages
  - The story of Prolog
  - The story of ML
  - The story of Java

Lisp

- AI conference at Dartmouth, 1956: McCarthy, Minsky, Newell, Simon
- Newell, Shaw and Simon demonstrate Logic Theorist, a reasoning program written in IPL (Information Processing Language)
- IPL had support for linked lists, and caught McCarthy's attention
- He wanted a language for AI projects, but not IPL: too low-level and machine-specific

Early AI Language Efforts

- An IBM group (consulting McCarthy) developed FLPL: Fortran List Processing Language
- McCarthy had a wish list, developed while writing AI programs (chess and differential calculus)
  - Conditional expressions
  - Recursion
  - Higher-order functions (like ML's **`map`**)
  - Garbage collection
- FLPL wasn't the answer for McCarthy's group at MIT in 1958…

Lisp's Unusual Syntax

- A Lisp program is a list representing an AST:
```
(+ a (* b c))
```
- The plan was to use some Fortran-like notation
- But McCarthy wrote a paper showing a simple Lisp interpreter in Lisp: a function called **`eval`**
- To avoid syntax issues, he used the list-AST form, both for **`eval`**'s input and for **`eval`** itself
- This **`eval`**, hand-translated into assembly language, became the first implementation of Lisp

Lisp's Unusual Syntax

- The group never gave up the idea of compiling from some Fortran-like syntax
- But they never got around to it either
- In later years, people often tried to compile Lisp from a Fortran- or Algol-like syntax
- None of them caught on
- There are advantages to having programs and data use the same syntax, as we saw with Prolog

Lisp Evolution

- Quickly became, and remains, the most popular language for AI applications
- Before 1980: many dialects in use:
  - Each AI research group had its own dialect
  - In the 1970's, a number of Lisp machines were developed, each with its own dialect
- Today: some standardization:
  - Common Lisp: a large language and API
  - Scheme: a smaller and simpler dialect

Lisp Influence

- The second-oldest general-purpose programming language still in use
- Some ideas, like the conditional expression and recursion, were adopted by Algol and later by many other imperative languages
- The function-oriented approach influenced modern functional languages like ML
- Garbage collection is increasingly common in many different language families

- Prehistory of programming languages
  - The story of the programmers of Babylon
  - The story of Mohammed Al-Khorezmi
  - The story of Augusta Ada, Countess of Lovelace
- Early programming languages
  - The story of the Plankalkül
  - The story of Fortran
  - The story of Lisp
  - The story of Algol
  - The story of Smalltalk
- Our languages
  - The story of Prolog
  - The story of ML
  - The story of Java

Algol

- In 1957, languages were proliferating
  - In the US, computer manufacturers were developing platform-specific languages like IBM's Fortran
  - In Europe, a number of languages had been designed by different research groups: Plankalkül and others
- Algol was intended to stop this proliferation
  - It would be the one universal, international, machine-independent language for expressing scientific algorithms
- In 1958, an international committee (!) was formed to come up with the design

The Algols

- Eventually, three major designs: Algol 58, Algol 60, and Algol 68
- Developed by increasingly large (!) international committees

The Good News

- Virtually all languages after 1958 used ideas pioneered by the Algol designs:
  - Compound statements: **`begin`** *statements* **`end`**
  - Free-format lexical structure
  - BNF definition of syntax
  - Local variables with block scope
  - Static typing with explicit type declarations
  - Nested if-then-else
  - Call by value (and call by name)
  - Recursive subroutines and conditional expressions (ex Lisp)
  - Dynamic arrays
  - First-class procedures
  - User-defined operators

Issue: Phrase-Level Control

- Early languages used label-oriented control:

```
GO TO 27
IF (A-B) 5,6,7
```

- Algol languages had good phrase-level control, like the **`if`** and **`while`** we saw in Java, plus **`switch`**, **`for`**, **`until`**, etc.
- A debate about the relative merits began to heat up
- Edsgar Dijkstra's famous letter in 1968, "Go to statement considered harmful," proposed eliminating label-oriented control completely

Structured Programming

- Using phrase-level control instead of labels was called *structured programming*
- There was a long debate: many programmers found it difficult at first to do without labels
- Now, the revolution is over:
  - Some languages (like Java) eliminated **`go to`**
  - Others (like C++) still have it
  - But programmers rarely use it, even when permitted
- The revolution was triggered (or at least fueled) by the Algol designs

Issue: Orthogonality

- The Algol designs avoided special cases:
  - Free-formal lexical structure
  - No abritrary limits:
    - Any number of characters in a name
    - Any number of dimensions for an array
  - And *orthogonality*: every meaningful combination of primitive concepts is legal—no special forbidden combinations to remember

Example

	Integers	Arrays	Procedures
Passing as a parameter			
Storing in a variable			
Storing in an array			
Returning from a procedure			

- Each combination *not* permitted is a special case that must be remembered by the programmer
- By Algol 68, all combinations above are legal
- Just a sample of its orthogonality—few modern languages take this principle as far as Algol

The Bad News

- The Algol languages were not as widely used as had been hoped
  - Algol 58, extended to Jovial
  - Algol 60 used for publication of algorithms, and implemented and used fairly widely outside U.S.
- Some possible reasons:
  - They neglected I/O
  - They were considered complicated and difficult to learn
  - They included a few mistakes, like by-name parameters
  - They had no corporate sponsor (IBM chose to stick with Fortran)

Before Smalltalk: Simula

- Kristen Nygaard and Ole-Johan Dahl, Norwegian Computing Center, 1961
- Simula I: an special-purpose Algol extension for programming simulations: airplanes at an airport, customers at a bank, etc.
- Simula 67: a general-purpose language with classes, objects, inheritance
- Co-routines rather than methods

Smalltalk

- Alan Kay, Xerox PARC, 1972
- Inspired by Simula, Sketchpad, Logo, cellular biology, etc.
- Smalltalk is more object-oriented than most of its more popular descendants
- Everything is an object: variables, constants, activation records, classes, etc.
- All computation is performed by objects sending and receiving messages: **`1+2*3`**

A Design Philosophy

- Commit to a few simple ideas, then find the most elegant language design from there:
  - Lists, recursion, **`eval`**: Lisp
  - Objects, message-passing: Smalltalk
  - Resolution-based inference: Prolog
- Hallmarks:
  - Initial implementation is easy
  - Easy to modify the language
  - Programming feels like custom language design

Smalltalk's Influence

- The Simula languages and Smalltalk inspired a generation of object-oriented languages
- Smalltalk still has a small but active user community
- Most later OO languages concentrate more on runtime efficiency:
  - Most use static typing (Smalltalk uses dynamic)
  - Most include non-object primitive types as well as objects

- Prehistory of programming languages
  - The story of the programmers of Babylon
  - The story of Mohammed Al-Khorezmi
  - The story of Augusta Ada, Countess of Lovelace
- Early programming languages
  - The story of the Plankalkül
  - The story of Fortran
  - The story of Lisp
  - The story of Algol
  - The story of Smalltalk
- Our languages
  - The story of Prolog
  - The story of ML
  - The story of Java

Prolog

- Alan Robinson, 1965: resolution-based theorem proving
  - Resolution is the basic Prolog step
  - But Prolog did not follow easily or immediately
- Robert Kowalski, Edinburgh, 1971: an efficient resolution-based technique, SL-resolution
- Alain Colmerauer and Philippe Roussel, Marseilles, 1972: Prolog (*programmation en logique*)
  - For the automated deduction part of an AI project in natural language understanding

Prolog Evolution

- 1973 version:
  - Eliminated special backtracking controls (introducing the cut operation instead)
  - Eliminated occurs-check
- David Warren, 1977: efficient compiled Prolog, the Warren Abstract Machine
- (For many languages—Smalltalk, Prolog, ML—techniques for efficient compilation were critical contributions)
- ISO standards, 1995 and 2000

- Prehistory of programming languages
  - The story of the programmers of Babylon
  - The story of Mohammed Al-Khorezmi
  - The story of Augusta Ada, Countess of Lovelace
- Early programming languages
  - The story of the Plankalkül
  - The story of Fortran
  - The story of Lisp
  - The story of Algol
  - The story of Smalltalk
- Our languages
  - The story of Prolog
  - The story of ML
  - The story of Java

ML

- Robin Milner, Edinburgh, 1974
- LCF: a tool for developing machine-assisted construction of formal logical proofs
- ML was designed as the implementation language for LCF
- Strong typing, parametric polymorphism, and type inference were in the first designs
- Remained closely tied to LCF development for several years

Issue: Formal Semantics

- The definition of Standard ML includes a formal semantics (a natural semantics)
- This was part of the initial design, not (as is more common) added after implementation
- Fits with the intended application: to trust the proofs produced by LCF, you must trust the language in which LCF is implemented

ML Evolution

- Luca Cardelli, 1980: efficient compiled ML
- 1983: draft standard ML published
- Additions: pattern-matching, modules, named records, exception handling, streams
- Dialects:
  - Standard ML (SML), the one we used
  - Lazy ML: ML with lazy evaluation strategy
  - Caml: An ML dialect that diverged before the addition of modules
  - OCaml: Caml with object-oriented constructs
  - F#: Commercial OCaml in Visual Studio 2010

- Prehistory of programming languages
  - The story of the programmers of Babylon
  - The story of Mohammed Al-Khorezmi
  - The story of Augusta Ada, Countess of Lovelace
- Early programming languages
  - The story of the Plankalkül
  - The story of Fortran
  - The story of Lisp
  - The story of Algol
  - The story of Smalltalk
- Our languages
  - The story of Prolog
  - The story of ML
  - The story of Java

A Long Lineage

Algol 60

CPL

An even larger language than Algol 60, adding features for business data processing.
Christopher Strachey et. al., 1962-1966

BCPL

“Basic CPL.” Vastly simplified. Typeless: manipulates untyped machine words. Introduced the C-family array idea: **`A[I]`***, written in BCPL as* **`A!I`***, is the same as a reference to the word at address* **`A+I`***.*
Martin Richards (a student of Strachey), 1967

B

*An even simpler language, developed for systems programming for the first Unix systems at Bell Labs. Included compound assignments (***`a+=b`***), borrowed from Algol 68.*
Ken Thompson, 1969

A Long Lineage, Continued

B

C

Extension of B (originally, "NB") to take advantage of more hardware (PDP-11). Type system, macro preprocessor, I/O library, etc. Used to reimplement the Unix kernel, and spread widely with Unix.
Dennis Ritchie et. al., 1971-1973

C++

Originally a C preprocessor adding object-oriented features to C: "C with Classes". Added dynamic dispatch, overloaded operators and function names, multiple inheritance, templates, exception handling. Became and remains one of the most widely used languages.
Bjarne Stroustrup, 1984

Java

Java

- James Gosling, Sun Microsystems
- 1991: Oak: a language for ubiquitous computers in networked consumer technology
  - Like C++, but smaller and simpler
  - More secure and strongly typed
  - More platform independent
- 1995: renamed Java, retargeted for the Web
  - Incorporated into web browsers
  - Platform-independent active content for web pages

Nonlinear Lineage

- Not just a straight line from CPL
- Java also has:
  - Garbage collection (ex Lisp)
  - Concurrency (ex Mesa)
  - Packages (ex Modula)
- But nothing new: it was intended to be a production language, not a research language

Conclusion: The Honor Roll

- Some programming language pioneers who have won the Turing award:
  - Alan Perlis, John McCarthy, Edsger Dijkstra, Donald Knuth, Dana Scott, John Backus, Robert Floyd, Kenneth Iverson, C.A.R. Hoare, Dennis Ritchie, Niklaus Wirth, John Cocke, Robin Milner, Kristen Nygaard, Ole-Johan Dahl, Alan Kay, Peter Naur, Frances Allen, and Barbara Liskov
- These very bright people had to work very hard on things that now seem easy, such as:
  - Local variables with block scope
  - Using phrase-level control instead of `go to`
- Before becoming perfectly obvious to everyone, these things were unknown and unguessed

Conclusion

- Is the evolution of programming languages nearly done, or have we as far again to go?
- Maybe all the important discoveries have been made, and language evolution will now slow and converge
- Or maybe we will have the pleasure of seeing new ideas, now unknown and unguessed, become perfectly obvious to everyone
- Enjoy!