History of programming languages

The first high-level programming language was Plankalkül, created by Konrad Zuse between 1942 and 1945[1]. The first high-level language to have an associated compiler, was created by Corrado Böhm in 1951, for his PhD thesis (http://e-collection.library.ethz.ch/eserv/eth:32719/eth-32719-02.pdf). The first commercially available language was FORTRAN (FORmula TRANslation); developed in 1956 (first manual appeared in 1956, but first developed in 1954) by the team of John Backus at IBM.

When FORTRAN was first introduced it was treated with suspicion because of the belief that programs compiled from high-level language would be less efficient than those written directly in machine code. FORTRAN became popular because it provided a means of porting existing code to new computers, in a hardware market that was rapidly evolving. FORTRAN eventually became known for its efficiency. Over the years, FORTRAN had been updated, with standards released for FORTRAN-66, FORTRAN-77 and FORTRAN-92.

Contents

Early history

During a nine-month period in 1842–1843, Ada Lovelace translated the memoir of Italian mathematician Luigi Menabrea about Charles Babbage's newest proposed machine, the analytical engine. With the article she appended a set of notes which specified in complete detail a method for calculating Bernoulli numbers with the engine, recognized by some historians as the world's first computer program.[2]

The first computer codes were specialized for their applications. In the first decades of the 20th century, numerical calculations were based on decimal numbers. Eventually it was realized that logic could be represented with numbers, not only with words. For example, Alonzo Church was able to express the lambda calculus in a formulatic way. The Turing machine was an abstraction of the operation of a tape-marking machine, for example, in use at the telephone companies. Turing machines set the basis for storage of programs as data in the von Neumann architecture of computers by representing a machine through a finite number. However, unlike the lambda calculus, Turing's code does not serve well as a basis for higher-level languages—its principal use is in rigorous analyses of algorithmic complexity.

To some people, what was the first modern programming language depends on how much power and human-readability is required before the status of "programming language" is granted. Jacquard Looms and Charles Babbage's Difference Engine both had simple, extremely limited languages for describing the actions that these machines should perform.

First programming languages

In the 1940s, the first recognizably modern electrically powered computers were created. The limited speed and memory capacity forced programmers to write hand tuned assembly language programs. It was eventually realized that programming in assembly language required a great deal of intellectual effort.

The first programming languages designed to communicate instructions to a computer were written in the 1950s. An early high-level programming language to be designed for a computer was Plankalkül, developed by the Germans for Z1 by Konrad Zuse between 1943 and 1945. However, it was not implemented until 1998 and 2000.[3]

John Mauchly's Short Code, proposed in 1949, was one of the first high-level languages ever developed for an electronic computer.[4] Unlike machine code, Short Code statements represented mathematical expressions in understandable form. However, the program had to be translated into machine code every time it ran, making the process much slower than running the equivalent machine code.

At the University of Manchester, Alick Glennie developed Autocode in the early 1950s, with the second iteration developed for the Mark 1 by R. A. Brooker in 1954, known as the "Mark 1 Autocode". Brooker also developed an autocode for the Ferranti Mercury in the 1950s in conjunction with the University of Manchester. The version for the EDSAC 2 was devised by D. F. Hartley of University of Cambridge

Mathematical Laboratory in 1961. Known as EDSAC 2 Autocode, it was a straight development from Mercury Autocode adapted for local circumstances, and was noted for its object code optimisation and source-language diagnostics which were advanced for the time. A contemporary but separate thread of development, Atlas Autocode was developed for the University of Manchester Atlas 1 machine.

In 1954, language FORTRAN was invented at IBM by a team led by John Backus; it was the first widely used high level general purpose programming language to have a functional implementation, as opposed to just a design on paper.[5][6] It is still a popular language for high-performance computing[7] and is used for programs that benchmark and rank the world's fastest supercomputers.[8]

Another early programming language was devised by Grace Hopper in the US, called FLOW-MATIC. It was developed for the UNIVAC I at Remington Rand during the period from 1955 until 1959. Hopper found that business data processing customers were uncomfortable with mathematical notation, and in early 1955, she and her team wrote a specification for an English programming language and implemented a prototype.[9] The FLOW-MATIC compiler became publicly available in early 1958 and was substantially complete in 1959.[10] Flow-Matic was a major influence in the design of COBOL, since only it and its direct descendent AIMACO were in actual use at the time.[11]

Other languages still in use today include LISP (1958), invented by John McCarthy and COBOL (1959), created by the Short Range Committee. Another milestone in the late 1950s was the publication, by a committee of American and European computer scientists, of "a new language for algorithms"; the *ALGOL 60 Report* (the "**ALGO**rithmic **L**anguage"). This report consolidated many ideas circulating at the time and featured three key language innovations:

- nested block structure: code sequences and associated declarations could be grouped into blocks without having to be turned into separate, explicitly named procedures;
- lexical scoping: a block could have its own private variables, procedures and functions, invisible to code outside that block, that is, information hiding.

Another innovation, related to this, was in how the language was described:

- a mathematically exact notation, Backus–Naur form (BNF), was used to describe the language's syntax. Nearly all subsequent programming languages have used a variant of BNF to describe the context-free portion of their syntax.

Algol 60 was particularly influential in the design of later languages, some of which soon became more popular. The Burroughs large systems were designed to be programmed in an extended subset of Algol.

Algol's key ideas were continued, producing ALGOL 68:

- syntax and semantics became even more orthogonal, with anonymous routines, a recursive typing system with higher-order functions, etc.;
- not only the context-free part, but the full language syntax and semantics were defined formally, in terms of Van Wijngaarden grammar, a formalism designed specifically for this purpose.

Algol 68's many little-used language features (for example, concurrent and parallel blocks) and its complex system of syntactic shortcuts and automatic type coercions made it unpopular with implementers and gained it a reputation of being *difficult*. Niklaus Wirth actually walked out of the design committee to create the simpler Pascal language.

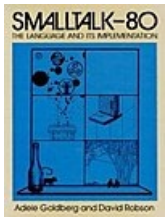
Some notable languages that were developed in this period include:

- 1951 – Regional Assembly Language
- 1952 – Autocode
- 1954 – IPL (forerunner to LISP)
- 1955 – FLOW-MATIC (led to COBOL)
- 1957 – FORTRAN (First compiler)
- 1957 – COMTRAN (precursor to COBOL)
- 1958 – LISP
- 1958 – ALGOL 58
- 1959 – FACT (forerunner to COBOL)
- 1959 – COBOL
- 1959 – RPG
- 1962 – APL
- 1962 – Simula
- 1962 – SNOBOL
- 1963 – CPL (forerunner to C)
- 1964 – Speakeasy (computational environment)
- 1964 – BASIC
- 1964 – PL/I
- 1966 – JOSS
- 1967 – BCPL (forerunner to C)

Fortran

Establishing fundamental paradigms

The period from the late 1960s to the late 1970s brought a major flowering of programming languages. Most of the major language paradigms now in use were invented in this period:



Smalltalk

- **Speakeasy (computational environment)**, developed in 1964 at Argonne National Laboratory (ANL) by Stanley Cohen, is an OOPS (object-oriented programming, much like the later MATLAB, IDL (programming language) and Mathematica) numerical package. Speakeasy has a clear Fortran foundation syntax. It first addressed efficient physics computation internally at ANL, was modified for research use (as "Modeleasy") for the Federal Reserve Board in the early 1970s and then was made available commercially; Speakeasy and Modeleasy are still in use currently.
- **Simula**, invented in the late 1960s by Nygaard and Dahl as a superset of Algol 60, was the first language designed to support object-oriented programming.
- **C**, an early systems programming language, was developed by Dennis Ritchie and Ken Thompson at Bell Labs between 1969 and 1973.
- **Smalltalk** (mid-1970s) provided a complete ground-up design of an object-oriented language.
- **Prolog**, designed in 1972 by Colmerauer, Roussel, and Kowalski, was the first logic programming language.

- **ML** built a polymorphic type system (invented by Robin Milner in 1973) on top of Lisp,[12] pioneering statically typed functional programming languages.

Each of these languages spawned an entire family of descendants, and most modern languages count at least one of them in their ancestry.


Scheme

The 1960s and 1970s also saw considerable debate over the merits of "structured programming", which essentially meant programming without the use of "goto". A significant fraction of programmers believed that, even in languages that provide "goto", it is bad programming style to use it except in rare circumstances. This debate was closely related to language design: some languages did not include a "goto" at all, which forced structured programming on the programmer.

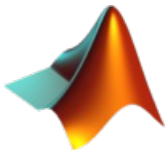
To provide even faster compile times, some languages were structured for "one-pass compilers" which expect subordinate routines to be defined first, as with Pascal, where the main routine, or driver function, is the final section of the program listing.

Some notable languages that were developed in this period include:

- 1968 – Logo
- 1969 – B (forerunner to C)
- 1970 – Pascal
- 1970 – Forth
- 1972 – C
- 1972 – Smalltalk
- 1972 – Prolog
- 1973 – ML
- 1975 – Scheme
- 1978 – SQL (a query language, later extended)

1980s: consolidation, modules, performance

The 1980s were years of relative consolidation in imperative languages. Rather than inventing new paradigms, all of these movements elaborated upon the ideas invented in the previous decade. C++ combined object-oriented and systems programming. The United States government standardized Ada, a systems programming language intended for use by defense contractors. In Japan and elsewhere, vast sums were spent investigating so-called fifth-generation programming languages that incorporated logic programming constructs. The functional languages community moved to standardize ML and Lisp. Research in Miranda, a functional language with lazy evaluation, began to take hold in this decade.


MATLAB

One important new trend in language design was an increased focus on programming for large-scale systems through the use of *modules*, or large-scale organizational units of code. Modula, Ada, and ML all developed notable module systems in the 1980s. Module systems were often wedded to generic programming constructs---generics being, in essence, parametrized modules (see also polymorphism in object-oriented programming).

Erlang

Although major new paradigms for imperative programming languages did not appear, many researchers expanded on the ideas of prior languages and adapted them to new contexts. For example, the languages of the Argus and Emerald systems adapted object-oriented programming to distributed systems.

The 1980s also brought advances in programming language implementation. The RISC movement in computer architecture postulated that hardware should be designed for compilers rather than for human assembly programmers. Aided by processor speed improvements that enabled increasingly aggressive compilation techniques, the RISC movement sparked greater interest in compilation technology for high-level languages.


Tcl

Language technology continued along these lines well into the 1990s.

Some notable languages that were developed in this period include:

- 1980 – C++ (as C with classes, renamed in 1983)
- 1983 – Ada
- 1984 – Common Lisp
- 1984 – MATLAB
- 1984 - dBase III, dBase III Plus (Clipper and FoxPro as FoxBASE, later developing into Visual FoxPro
- 1985 – Eiffel
- 1986 – Objective-C
- 1986 – LabVIEW (Visual Programming Language)
- 1986 – Erlang
- 1987 – Perl
- 1988 – Tcl
- 1988 – Wolfram Language (as part of Mathematica, only got a separate name in June 2013)
- 1989 – FL (Backus)

1990s: the Internet age

The rapid growth of the Internet in the mid-1990s was the next major historic event in programming languages. By opening up a radically new platform for computer systems, the Internet created an opportunity for new languages to be adopted. In particular, the JavaScript programming language rose to popularity because of its early integration with the Netscape Navigator web browser. Various other scripting languages achieved widespread use in developing customized applications for web servers such as PHP. The 1990s saw no fundamental novelty in imperative languages, but much recombination and maturation of old ideas. This era began the spread of functional languages. A

big driving philosophy was programmer productivity. Many "rapid application development" (RAD) languages emerged, which usually came with an IDE, garbage collection, and were descendants of older languages. All such languages were object-oriented. These included Object Pascal, Visual Basic, and Java. Java in particular received much attention.

More radical and innovative than the RAD languages were the new scripting languages. These did not directly descend from other languages and featured new syntaxes and more liberal incorporation of features. Many consider these scripting languages to be more productive than even the RAD languages, but often because of choices that make small programs simpler but large programs more difficult to write and maintain. Nevertheless, scripting languages came to be the most prominent ones used in connection with the Web.

Some notable languages that were developed in this period include:

- 1990 – Haskell
- 1991 – Python
- 1991 – Visual Basic
- 1993 – Lua
- 1993 – R
- 1994 – CLOS (part of ANSI Common Lisp)
- 1995 – Ruby
- 1995 – Ada 95
- 1995 – Java
- 1995 – Delphi (Object Pascal)
- 1995 – JavaScript
- 1995 – PHP
- 1997 – Rebol


Haskell

Lua

PHP

Rebol


D Programming Language

Groovy


Rust

Scratch

Swift

Current trends

Programming language evolution continues, in both industry and research. Some of the recent trends have included:

- Increasing support for functional programming in mainstream languages used commercially, including pure functional programming for making code easier to reason about and easier to parallelise (at both micro- and macro- levels)
- Constructs to support concurrent and distributed programming.
- Mechanisms for adding security and reliability verification to the language: extended static checking, dependent typing, information flow control, static thread safety.
- Alternative mechanisms for composability and modularity: mixins, traits, delegates, aspects.
- Component-oriented software development.
- Metaprogramming, reflection or access to the abstract syntax tree
  - AOP or Aspect Oriented Programming allowing developers to insert code in another module or class at "join points"
  - Domain specific languages and code generation
    - XML for graphical interface (XUL, XAML)
- Increased interest in distribution and mobility.
- Integration with databases, including XML and relational databases.
- Open source as a developmental philosophy for languages, including the GNU Compiler Collection and languages such as Python, Ruby, and Scala.
- Massively parallel languages for coding 2000 processor GPU graphics processing units and supercomputer arrays including OpenCL
- Early research into (as-yet-unimplementable) quantum computing programming languages
- More interest in visual programming languages like Scratch

Some notable languages developed during this period include:

- 2000 – ActionScript
- 2001 – C#
- 2001 – D
- 2002 – Scratch
- 2003 – Groovy
- 2003 – Scala
- 2005 – F#
- 2006 – PowerShell
- 2007 – Clojure
- 2009 – Go
- 2010 – Rust
- 2011 – Dart
- 2011 – Kotlin
- 2011 – Red
- 2011 - Elixir
- 2012 – Julia
- 2014 – Swift
- 2016 – Ring [13][14]

Prominent people

Some key people who helped develop programming languages:

- Alan Cooper, developer of Visual Basic.
- Alan Kay, pioneering work on object-oriented programming, and originator of Smalltalk.
- Anders Hejlsberg, developer of Turbo Pascal, Delphi, C#, and TypeScript.
- Bertrand Meyer, inventor of Eiffel.
- Bjarne Stroustrup, developer of C++.
- Brian Kernighan, co-author of the first book on the C programming language with Dennis Ritchie, coauthor of the AWK and AMPL programming languages.

- Chris Lattner, creator of Swift and LLVM.
- Dennis Ritchie, inventor of C. Unix Operating System, Plan 9 Operating System.
- Grace Hopper, first to use the term compiler and developer of Flow-Matic, influenced development of COBOL. Popularized machine-independent programming languages and the term "debugging".
- Guido van Rossum, creator of Python.
- James Gosling, lead developer of Java and its precursor, Oak.
- Jean Ichbiah, chief designer of Ada, Ada 83.
- Jean-Yves Girard, co-inventor of the polymorphic lambda calculus (System F).
- Jeff Bezanson, main designer, and one of the core developers of Julia.
- Joe Armstrong, creator of Erlang.
- John Backus, inventor of Fortran and cooperated in the design of ALGOL 58 and ALGOL 60.
- John C. Reynolds, co-inventor of the polymorphic lambda calculus (System F).
- John McCarthy, inventor of LISP.
- John von Neumann, originator of the operating system concept.
- Graydon Hoare, inventor of Rust.
- Ken Thompson, inventor of B, Go Programming Language, Inferno Programming Language, and Unix Operating System co-author.
- Kenneth E. Iverson, developer of APL, and co-developer of J along with Roger Hui.
- Konrad Zuse, designed the first high-level programming language, Plankalkül (which influenced ALGOL 58[15]).
- Kristen Nygaard, pioneered object-oriented programming, co-invented Simula.
- Larry Wall, creator of the Perl programming language (see Perl and Perl 6).
- Martin Odersky, creator of Scala, and previously a contributor to the design of Java.
- Nathaniel Rochester, inventor of first assembler (IBM 701).
- Niklaus Wirth, inventor of Pascal, Modula and Oberon.
- Ole-Johan Dahl, pioneered object-oriented programming, co-invented Simula.
- Rasmus Lerdorf, creator of PHP
- Rich Hickey, creator of Clojure.
- Robin Milner, inventor of ML, and sharing credit for Hindley–Milner polymorphic type inference.
- Stephen Wolfram, creator of Mathematica.
- Tom Love and Brad Cox, creator of Objective-C.
- Walter Bright, creator of D.
- Yukihiro Matsumoto, creator of Ruby.


Anders Hejlsberg


Yukihiro Matsumoto


Grace M. Hopper


Bjarne Stroustrup


Niklaus Wirth

See also

- ACM
- SIGPLAN
- History of Programming Languages Conference
- History of compiler writing
- History of computing hardware
- Programming language
- Timeline of computing
- Timeline of programming languages
- List of programming languages
- List of programmers

References

1. Knuth, Donald E.; Pardo, Luis Trabb. "Early development of programming languages". *Encyclopedia of Computer Science and Technology*. Marcel Dekker. **7**: 419–493.
2. J. Fuegi and J. Francis (October–December 2003), "Lovelace & Babbage and the creation of the 1843 'notes'", *Annals of the History of Computing*, **25** (4): 16, 19, 25, doi:10.1109/MAHC.2003.1253887 (https://doi.org/10.1109%2FMAHC.2003.1253887)
3. Rojas, Raúl, et al. (2000). "Plankalkül: The First High-Level Programming Language and its Implementation". Institut frame Informatik, Freie Universität Berlin, Technical Report B-3/2000. (full text) (ftp://ftp.mi.fu-berlin.de/pub/reports/TR-B-00-03.pdf)
4. Sebesta, W.S. (2006). *Concepts of Programming Languages*. p. 44. ISBN 0-321-33025-0.
5. "Fortran creator John Backus dies - Tech and gadgets- msnbc.com" (http://www.msnbc.msn.com/id/17704662/). MSNBC. 2007-03-20. Retrieved 2010-04-25.
6. "CSC-302 99S : Class 02: A Brief History of Programming Languages" (http://www.math.grin.edu/~rebelsky/Courses/CS302/99S/Outlines/outline.02.html). Math.grin.edu. Retrieved 2010-04-25.
7. Eugene Loh (18 June 2010). "The Ideal HPC Programming Language" (http://queue.acm.org/detail.cfm?id=1820518). *Queue*. Association of Computing Machines. **8** (6).
8. "HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers" (http://www.netlib.org/benchmark/hpl). Retrieved 2015-02-21.
9. Hopper (1978) p. 16.
10. Sammet (1969) p. 316
11. Sammet (1978) p. 204.
12. Gordon, Michael J. C. (1996). "From LCF to HOL: a short history" (http://www.cl.cam.ac.uk/~mjcg/papers/HolHistory.pdf) (PDF). p. 3. Retrieved 2015-05-04. "Edinburgh LCF, including the ML interpreter, was implemented in Lisp."
13. Rubin Liu (28 February 2018). "The evolution of the Ring programming language, Ring in Top 50 programming languages according to TIOBE Index" (https://www.codeproject.com/Articles/1223114/The-evolution-of-the-Ring-programming-language). *codeproject.com*. Code_Project.

14. TIOBE (2 March 2018). "TIOBE Index, Ring in Top 50 programming languages according to TIOBE Index" (https://www.tiobe.com/tiobe-index/). *www.tiobe.com*. TIOBE_index.
15. Rojas, Raúl; Hashagen, Ulf (2002). *The First Computers: History and Architectures* (https://books.google.com/books?id=nDWPW9uwZPAC&pg=PA292&dq=algol-68+konrad+zuse). MIT Press. p. 292. ISBN 978-0262681377. Retrieved October 25, 2013.

Further reading

- Rosen, Saul, (editor), *Programming Systems and Languages*, McGraw-Hill, 1967.
- Sammet, Jean E., *Programming Languages: History and Fundamentals*, Prentice-Hall, 1969.
- Sammet, Jean E. (July 1972). "Programming Languages: History and Future". *Communications of the ACM*. **15** (7): 601–610. doi:10.1145/361454.361485 (https://doi.org/10.1145%2F361454.361485).
- Richard L. Wexelblat (ed.): *History of Programming Languages*, Academic Press 1981.
- Thomas J. Bergin and Richard G. Gibson (eds.): *History of Programming Languages*, Addison Wesley, 1996.

External links

- History and evolution of programming languages (http://www.scriptol.com/programming/history.php)
- Graph of programming language history (http://www.levenez.com/lang/history.html)