

Class 26: An Abbreviated History of Programming Languages

Held: Wednesday, 6 April 2011

Summary: Today we quickly consider the history of the development of programming languages. Our consideration will be both abbreviated and biased.

Related Pages:

- EBoard.

Notes:

- Sorry about neglecting to get a reading to you.
- Updates from Kington talk.
- Reading 1 for Friday: *The Story of Mel, a real programmer*, available at <http://www.wizzy.com/andyr/Mel.html>.
- Reading 2 for Friday: “The impact of software engineering research on modern programming languages”, available at <http://portal.acm.org/citation.cfm?doid=1101815.1101818>.
- I’ll reserve a few minutes at the start of class for questions on the exam.
- EC for today’s Escher talk.
- EC for tomorrow’s CS extra by DG and RC.
- EC for Friday’s CS table on Watson.

Overview:

- Exercise: Dating key ideas.
- An abbreviated history of programming languages.

Dating Ideas and Issues

- One odd or interesting aspect of computing is that many of us significantly misdate concepts, considering some new concepts old, and some new concepts new.
- We’ll spend a little time seeing when we think particular ideas were developed. Note that I don’t know all of the answers or dates, and that some of my knowledge is probably incorrect.
- **Concepts**
 - Computational devices
 - The general-purpose computer (as opposed to ...)
 - Punch cards
 - The programming language
 - The first implemented programming language

- Hypertext
- **Paradigms** (paradigm used; language implemented)
 - Imperative programming
 - Functional programming
 - Object-oriented programming
 - Logic programming
 - Parallel programming
- **Design Issues**
 - Garbage collection (automatic memory management)
 - Symbolic processing
 - Exceptions
 - Iterators
 - GUI languages

An Approximate History

This is clearly both a partial history, and only a partially correct history. However, it should be enough to get you thinking about some key issues. I present it, in part, to give you a sense of where I stand as compared to various other sources out there.

- Surprisingly (or perhaps not so surprisingly), there were a number of pre-computer ways to express computation. These include,
 - algorithms in mathematics (yes, mathematicians did need to be able to specify computation)
 - the lambda calculus (a formal system for describing functions),
 - Turing machines (weird objects designed to mimic computation),
 - Lady Augusta Ada, Countess Lovelace's language for programming the (nonexistent) analytical engine.
- Konrad Zuse developed a language he called the Plankalkul in 1945. While it was an interesting "high-level" language, it was not implemented until the late 20th century.
- Early computers had to be programmed in machine code (ugh) or as patch diagrams (more ugh).
 - Difficult to write programs.
 - Difficult to enter programs.
 - Difficult to check whether you got it right!
- However, many early computers soon included assembly languages and assemblers.
 - Disadvantage: every different computer had a different assembly language.
- Some also included expression interpreters.
- FORTRAN (for FORMula TRANslator) was the first high level language implemented. The team that designed and built FORTRAN was led by John Backus. Fortran was released in 1957, but design began much earlier (1954). A key goal of Fortran was efficiency, although portability was also a key issue.
- LISP (LISt Processing) was designed for symbolic processing. It was built by John McCarthy at MIT in 1958. LISP introduced symbolic computation and automatic memory management to the programming community.
- ALGOL-60 (ALGOrithmic Language) was designed by computer scientists (or those precursors to

computer scientist) primarily intended to provide a mechanism for expressing algorithms uniformly. The first report on Algol was issued in 1958, with the specs being revised in 1959 and 1960 (and, later in 1968).

- Algol is a primary ancestor of Pascal and C.
- It introduced block structure, compound statements, recursive procedure calls, nested if, loops, and arbitrary length identifiers.
- COBOL (Common Business-Oriented Language) was designed around 1960 for business applications, with a team led by (I think) Grace Murray Hopper. It is still used today, in primarily the same form. One goal of COBOL's design was for it to be readable by managers, so the syntax had very much of an English-like flavor.
 - While many computer scientists are taught that "Cobol is bad", note that Cobol introduced sophisticated records and had a very rich output system.
 - Note also that many modern scripting languages, like Hypertalk, Appletalk, and Lingo, strive for the same "English-like" flavor that Cobol is criticized for having.
- BASIC (Beginner's All-purpose Symbolic Instruction Code) was created by some professors at Dartmouth. It was one of the first languages designed for use on a time-sharing system, and one of the first designed for beginners.
- By this point, there were four core languages (Cobol, Fortran, Algol, Lisp) that represented two different paradigms (imperative, functional). IBM put them all together in PL/I and then extended the language further.
 - PL/I introduced concurrency and exceptions.
 - PL/I introduced the need for language wizards who understood the full manual set.
 - PL/I illustrated the problems with incorporating too much.
- Simula/67 was designed to simplify simulations. It introduced objects.
- Algol-68 and Algol-W were successors to Algol. Algol-68 was supposed to be Algol-64 (I think) but the report took four years to write. Algol-W was an offshoot designed by members of the committee dissatisfied with the complexity and time for creating the report.
- A number of important languages were designed and implemented in the 1970's.
 - Pascal in 1971, intended as a teaching language based on Algol.
 - C in 1972, intended as a lower-level language to support more portable operating system design.
 - Backus described FP, his functional programming language, in 1977.
 - Prolog was designed and implemented in the early 1970's.
 - ML was designed in the late 1970's.
 - Smalltalk was designed in the late 1970's as a more visual beginner's language and environment.
- The 1980's and 1990's gave rise to successor languages that extended these base languages, but did not clearly provide new paradigms or extensions.
 - ADA in 1980.
 - C++ in mid 1980's
 - Java in early 1990's
- As we've seen in our exploration of seven languages in sevenish weeks, people continue to design new languages.
 - Objects grew in popularity in the late 1980's and 1990's
 - Functional programming seems to be growing in popularity recently
 - Shriram Krishnamurthi tells me that it's stupid to retain our paradigmatic classification, given

that most languages now support multiple paradigms.