

–contents of Programming Language Vision–

Notation
 Evolution
 Who is the Master?
 History and Change
 Typography

A Vision for Programming Language Design

The advantage of simple and clear language concepts
 is that their implementation in a compiler
 is easily and efficiently possible
 without causing unexpected overhead.

–Leo Geissmann, *PhD thesis*

Notation

One can observe the use of notation and admire the power that it brings to its users. What would integral calculus be without $\int_a^b f(x)dx$, set theory be without $\overline{A \cup B} \equiv \overline{A} \cap \overline{B}$, inorganic chemistry be without $2H + O = H_2O$, electromagnetism be without $\nabla \cdot E = 4\pi\rho$, predicate logic be without $\forall \exists \neg \wedge \vee$, music be without notes? Notation evolves through trial and error. What works survives. No committee decides. The inventors of notation are its users.

Evolution

Programming languages were designed, in the beginning, to be somewhere between machine code and mathematical notation. Lest this seem to be an inordinately unambitious goal, review the charter for the design of ALGOL 60.

1. *The new language should be as close as possible to standard mathematical notation and be readable with little further explanation.*
2. *It should be possible to use it for the description of numerical processes in publications.*
3. *The new language should be readily translatable into machine code by the machine itself.*

Except for the narrow focus on numerical processes, this charter still serves a half-century later. It is ironic that another goal of ALGOL60 was to avoid proliferation of languages by offering a standard universally applicable means for expressing programs. There are now hundreds of programming languages in use, each with one or more translators. What the authors did not anticipate was that, like other notations, programming languages would evolve by Darwinian rules. ALGOL 60 was influential but now is dead. Evolution is patient; it has only begun to work its magic.

Who is the Master?

It is the intention of this course to return programming languages to their users by providing those users with the capability for personal compiler implementation. I think Darwin would approve.

History and Change

In the beginning, each computer was programmed directly in its own instruction set. High level programming languages changed that. FORTRAN soon dominated scientific use and COBOL dominated commercial use. LISP became popular in academic environments. They are with us still. C++ has become the language of choice for system implementers (including compiler writers). JAVA, by virtue of its elegant integration of many features including parallel execution, networks and graphics, has dominated on the internet. There are many programming languages with specialized niches, such as M for scientific applications in MATLAB[®].

¹

We can agree that programming languages have provided an essential abstraction of the programming process. Most of us, understanding the trends in other scientific disciplines, would like even higher levels of abstraction. Agreement on what new features are needed and what old features can safely be discarded is hard to achieve. The discussion of programming languages often invokes partisan positions, perhaps because we programmers are so close to our languages; perhaps because “language user community” and “standards committee” are inherently political concepts; perhaps because our preferences are so deeply imbedded that words fail us.

One can observe that popular programming languages tend to grow, gathering constructs whenever the political climate is right. It is hardly ever time to throw things out; for every old feature there is a saving argument or a mass of entrenched use.

Typography

The need for convenient keyboard entry has restricted programming languages to a limited set of symbols such as that provided by ASCII or UNICODE. Neither approaches the richness of typeset scientific notation. This is not a fundamental constraint; programs can be typeset. ALGOL 60 introduced the concept of a *publication language*, a form of expression optimal for readability and transparently mapped to the *reference language*. Greek letters, subscripts and superscripts, and an open-ended set of parentheses were explicitly encouraged.

I propose that, in addition to translation to executable form, each compiler also provide for translation to publication form.

¹MATLAB is a registered trademark of The MathWorks, Inc.