DOI:10.1145/3447525

Niklaus Wirth

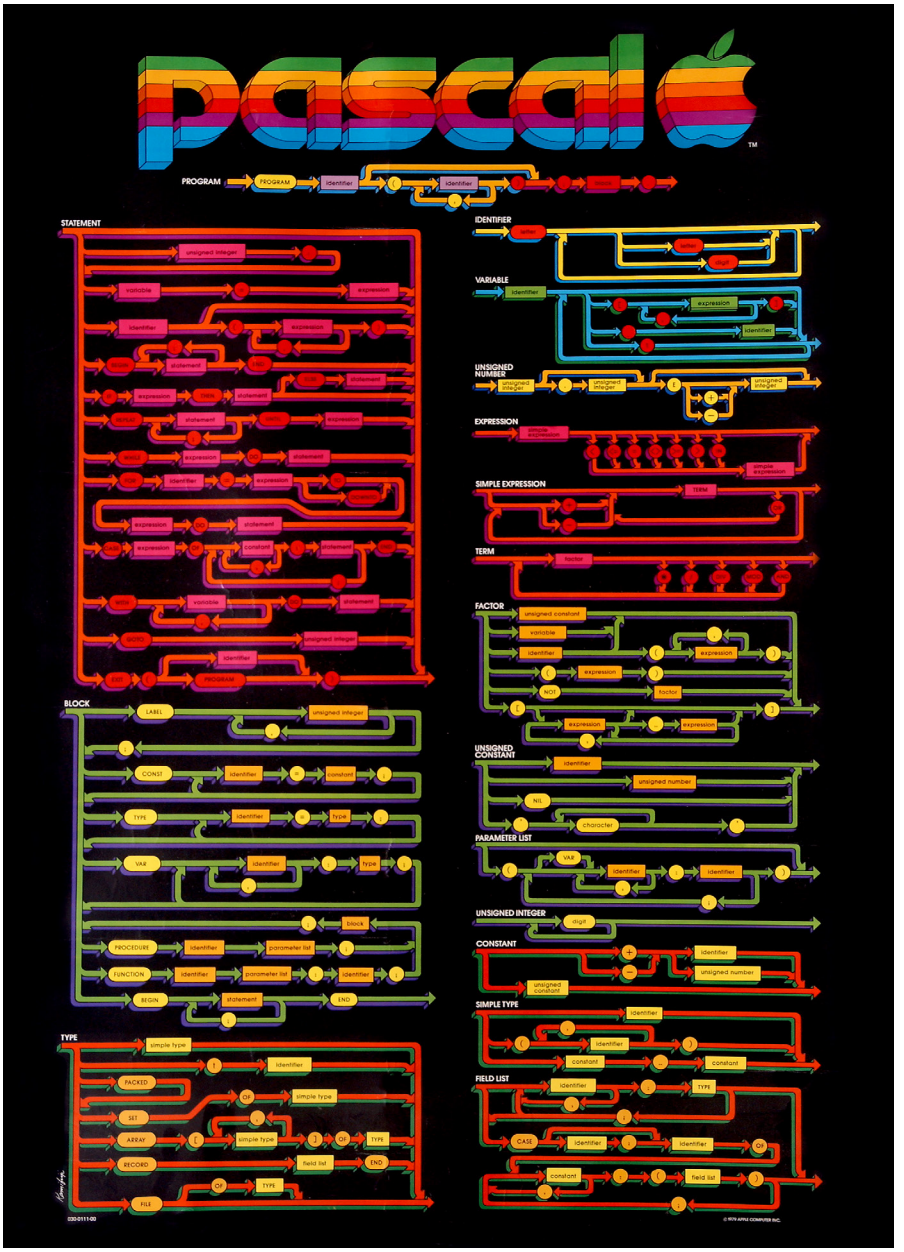
Viewpoint
50 Years of Pascal

The Pascal programming language creator Niklaus Wirth reflects on its origin, spread, and further development.

In the early 1960s, the languages Fortran (John Backus, IBM) for scientific, and Cobol (Jean Sammet, IBM, and DoD) for commercial applications dominated. Programs were written on paper, then punched on cards, and one waited a day for the results. Programming languages were recognized as essential aids and accelerators of the programming process.

In 1960, an international committee published the language Algol 60.[1] It was the first time a language was defined by concisely formulated constructs and by a precise, formal syntax. Two years later, it was recognized that a few corrections and improvements were needed. Mainly, however, the range of applications should be widened, because Algol 60 was intended for scientific calculations (numerical mathematics) only. Under the auspices of IFIP a Working Group (WG 2.1) was established to tackle this project.

The group consisted of about 40 members with almost the same number of opinions and views about what a successor of Algol should look like. There ensued many discussions, and on occasions the debates ended even bitterly. Early in 1964 I became a member, and soon was requested to prepare a concrete proposal. Two factions had developed in the committee. One of them aimed at a second, after Algol 60, milestone, a language with radically new, untested concepts and pervasive flexibility. It later became known as Algol 68. The other faction remained more modest and focused on realistic improvements of known concepts.



A poster of Pascal's syntax diagrams strongly identified with Pascal.

After all, time was pressing: PL/1 of IBM was about to appear. However, my proposal, although technically realistic, succumbed to the small majority that favored a milestone.

It is never sufficient to merely postulate a language on paper. A solid compiler also had to be built, which usually was a highly complex program. In this respect, large industrial firms had an advantage over our Working Group, which had to rely on enthusiasts at universities. I left the Group in 1966 and devoted myself together with a few doctoral students at Stanford University to the construction of a compiler for my proposal. The result was the language Algol W,[2] which after 1967 came into use at many locations on large IBM computers. It became quite successful. The milestone Algol 68 did appear and then sank quickly into obscurity under its own weight, although a few of its concepts did survive into subsequent languages.

But in my opinion Algol W was not perfectly satisfactory. It still contained too many compromises, having emerged from a committee. After my return to Switzerland, I designed a language after my own preferences: Pascal. Together with a few assistants, we wrote a user manual and constructed a compiler. In the course of it, we had a dire experience. We intended to describe the compiler in Pascal itself, then translate it manually to Fortran, and finally compile the former with the latter. This resulted in a great failure, because of the lack of data structures (records) in Fortran, which made the translation very cumbersome. After this unfortunate, expensive lesson, a second try succeeded, where in place of Fortran the local language Scallop (M. Engeli) was used.

Pascal

Like its precursor Algol 60, Pascal[2] featured a precise definition and a few lucid, basic elements. Its structure, the syntax, was formally defined in Extended BNF.[3] Statements described assignments of values to variables, and conditional and repeated execution. Additionally, there were procedures, and they were recursive. A significant extension were data types and structures: Its elementary data types were integers and real numbers, Boolean values, characters, and enumerations (of constants). The structures were arrays, records, files (sequences), and pointers. Procedures featured two kinds of parameters, value- and variable-parameters. Procedures could be used recursively. Most essential was the pervasive concept of data type: Every constant, variable, or function was of a fixed, static type. Thereby programs included much redundancy that a compiler could use for checking type consistency. This contributed to the detection of errors, and this before the program's execution.

Just as important as addition of features were deletions (with respect to Algol). As C.A.R. Hoare once remarked: A language is characterized not only by what it permits programmers to specify, but even more so by what it does not allow. In this vein, Algol's name parameter was omitted. It was rarely used, and caused considerable complications for a compiler. Also, Algol's own concept was deleted, which allowed local variables to be global, to "survive" the activation of the procedure to which it was declared local. Algol's for statement was drastically simplified, eliminating complex and hard to understand constructs. But the while and repeat statements were added for simple and transparent situations of repetition. Nevertheless, the controversial goto statement remained. I considered it too early for the programming community to swallow its absense. It would have been too detrimental for a general acceptance of Pascal.

Pascal was easy to teach, and it covered a wide spectrum of applications, which was a significant advantage over Algol, Fortran, and Cobol. The Pascal System was efficient, compact, and easy to use. The language was strongly influenced by the new discipline of structured programming, advocated primarily by E.W. Dijkstra to avert the threatening software crisis (1968).

Rapidly computers became faster, and therefore demands on applications grew, as well as those on programmers.

Pascal was published in 1970 and for the first time used in large courses at ETH Zurich on a grand scale. We had even defined a subset Pascal-S and built a smaller compiler, in order to save computing time and memory space on our large CDC computer, and to reduce the turnaround time for students. Back then, computing time and memory space were still scarce.

Pascal's Spread and Distribution

Soon Pascal became noticed at several universities, and interest rose for its use in classes. We received requests for possible help in implementing compilers for other large computers. It was my idea to postulate a hypothetical computer, which would be simple to realize on various other mainframes, and for which we would build a Pascal compiler at ETH. The hypothetical computer would be quickly implementable with relatively little effort using readily available tools (assemblers). Thus emerged the architecture Pascal-P (P for portable), and this technique proved to be extremely successful. The first clients came from Belfast (C.A.R. Hoare). Two assistants brought two heavy cartons of punched cards to Zurich, the compiler they had designed for their ICL computer. At the border, they were scrutinized, for there was the suspicion that the holes might contain secrets subject to custom fees. All this occurred without international project organizations, without bureaucracy and research budgets. It would be impossible today.

An interesting consequence of these developments was the emergence of user groups, mostly of young enthusiasts who wanted to promote and distribute Pascal. Their core resided under Andy Mickel in Minneapolis, where they regularly published a Pascal Newsletter. This movement contributed significantly to the rapid spread of Pascal.

Several years later the first microcomputers appeared on the market. These were small computers with a processor integrated on a single chip and with 8-bit data paths, affordable by private persons. It was recognized that Pascal was suitable for these processors, due to

its compact compiler that would fit into the small memory (64K). A group under Ken Bowles at the University of San Diego, and Philippe Kahn at Borland Inc. in Santa Cruz surrounded our compiler with a simple operating system, a text editor, and routines for error discovery and diagnostics. They sold this package for $50 on floppy disks (Turbo Pascal). Thereby Pascal spread immediately, particularly in schools, and it became the entry point for many to programming and computer science. Our Pascal manual became a best-seller.

This spreading did not remain restricted to America and Europe. Russia and China welcomed Pascal with enthusiasm. This I became aware of only later, during my first travels to China (1982) and Russia (1990), when I was presented with a copy of our manual written in (for me) illegible characters and symbols.

Pascal's Successors

But time did not stand still. Rapidly computers became faster, and therefore demands on applications grew, as well as those on programmers. No longer were programs developed by single persons. Now they were being built by teams. Constructs had to be offered by languages that supported teamwork. A single person could design a part of a system, called a module, and do this relatively independently of other modules. Modules would later be linked and loaded automatically. Already Fortran had offered this facility, but now a linker would have to verify the consistency of data types also across module boundaries. This was not a simple matter!

Modules with type consistency checking across boundaries were indeed the primary extension of Pascal's first successor Modula-2[4] (for modular language, 1979). It evolved from Pascal, but also from Mesa, a language developed at Xerox PARC for system programming, which itself originated from Pascal. Mesa, however, had grown too wildly and needed "taming." Modula-2 also included elements for system programming, which admitted constructs that depended on specific properties of a computer, as they were necessary for interfaces to peripheral devices or networks. This entailed sacrificing the essence of higher languages, namely machine-independent programming.

No reference to any computer or mechanism should be necessary to understand it.

Fortunately, however, such parts could now be localized in specific "low-level" modules, and thereby be properly isolated.

Apart from this, Modula contained constructs for programming concurrent processes (or quasiparallel threads). "Parallel programming" was the dominant theme of the 1970s. Overall, Modula-2 grew rather complex and became too complicated for my taste, and for teaching programming. An improvement and simplification appeared desirable.

From such deliberations emerged the language Oberon,[5] again after a sabbatical at Xerox PARC. No longer were mainframe computers in use, but powerful workstations with high-resolution displays and interactive usage. For this purpose, the language and interactive operating system Cedar had been developed at PARC. Once again, a drastic simplification and consolidation seemed desirable. So, an operating system, a compiler, and a text editor were programmed at ETH for Oberon. This was achieved by only two programmers—Wirth and Gutknecht—in their spare time over six months. Oberon was published in 1988. The language was influenced by the new discipline of object-oriented programming. However, no new features were introduced except type extension. Thereby for the first time a language was created that was not more complex, but rather simpler, yet even more powerful than its ancestor. A highly desirable goal had finally been reached.

Even today Oberon is successfully in use in many places. A breakthrough like Pascal's, however, did not occur. Complex, commercial systems are too widely used and entrenched. But it can be claimed that many of those languages, like Java (Sun Microsystems) and C# (Microsoft) have been strongly influenced by Oberon or Pascal.

Around 1995 electronic components that are dynamically reprogrammable at the gate level appeared on the market. These field programmable gate arrays (FPGA) can be configured into almost any digital circuit. The difference between hardware and software became increasingly diffuse. I developed the language Lola (logic language) with similar elements and the same structure as Oberon for describing digital circuits. Increasingly, circuits became specified by formal texts, replacing graphical circuit diagrams. This facilitates the common design of hardware and software, which has become increasingly important in practice.

Comments and Conclusion

The principal purpose of a higher-level language is to raise the level of abstraction from that of machine instructions. Examples are data structures vs. word arrays in memory, or conditional and repetitive statements vs. jump instructions. A perfect language should be defined in terms of mathematical logic, of axioms and rules of inference. *No reference to any computer or mechanism should be necessary to understand it.* This is the basis of portability. Algol's designers saw this goal; but it is most difficult to achieve without sacrificing power of expression. Yet, any new language must be measured on the degree to which it comes close to this goal. The sequence Pascal—Modula—Oberon is witness to my attempts to achieve it. Oberon is close to it. Yet, nothing is perfect. C

References

1. Naur, P. Revised report on the algorithmic language Algol 60. *Commun. ACM 6*, (Jan. 1963), 1–17.
2. Wirth, N. and Hoare, C.A.R. A contribution to the development of ALGOL. *Commun. ACM 9* (June 1966), 413–432.
3. Wirth, N. The programming langage Pascal. *Acta Informatica 1*, (1971), 35–63; https://doi.org/10.1007/BF00264291
4. Wirth, N. What can we do about the unnecessary diversity of notation for syntactic definitions? *Commun. ACM 20*, 11 (Nov. 1977).
5. Wirth, N. *Programming in Modula-2*. Springer-Verlag 1982.
6. Wirth, N. *The Programming Language Oberon. Software–Practice and Experience 18*, (Jul. 1988), 671–690; https://doi.org/10.1002/spe.4380180707

Niklaus Wirth (wirth@inf.ethz.ch) is a former Professor of Informatics at ETH Zurich, Switzerland.