Masud Ahmad Malik

On the Perils of Programming

Viewed as an art and a science among professionals and academics, programming retains its significance in any teaching or training program in the field of computing.

With the unbelievable rise in personal computing and wide acceptance of inexpensive, user-friendly operating systems and productivity tools, programming may have lost its place as a major activity in the modern world of computing.

Gone are the days when learning a programming language would open up a well-paid and challenging career. Now careers in information technology (IT), are not necessarily built out of programming skills only. As a matter of fact, working knowledge of a programming language is not considered adequate and must be supplemented by familiarity with different working environments. Experts in noncomputing fields are becoming computer professionals of great potential, with expertise in productivity and profession-specific tools, because of their in-depth knowledge of the application area—"knowledge that was never easy to share with a team of programmers and analysts during a system development exercise in the past," they admit with a sigh of relief.

However, teaching programming and specific languages is not obsolete and may never be. Programming has been viewed as both an art and a science among professionals and academics, and it retains significant importance in any teaching or training program in the field of computing. Moreover, teaching programming concepts and style has gained increased importance, and programming languages have been relegated to the level of support tools in this kind of teaching.

Most CS teaching programs now include an introductory course on basic computer concepts, covering hardware and software, followed by teaching programming concepts built from algorithms and problem-solving techniques. Among the older generation of computer scientists, I still recall my introduction to computers and programming 30 years ago, with an IBM-arranged Fortran IV course at an engineering university in Pakistan. With little emphasis on basic computer concepts in general, all material related to the IBM 1130 being installed at that university.

It took academics and professionals alike many years before it was realized that a first course in programming should not be influenced by a specific programming language. This served the computing world well, because a gap in understanding would invariably result between the groups emerging out in the business and scientific areas, trained and taught in Cobol and Fortran, respectively. Scientists would not know about records and files or the immensely rich Cobol repertoire of loop structuring mechanisms, at least until the early 1970s, and Cobol programmers had little idea of separately compiled program modules and data sharing by parameter passing.

Computer manufacturers, with stakes high in this new emerging computer world, not only accelerated development of innovative hardware pieces, they also contributed and attempted to make problem-solving easier. IBM even perceived in the early 1960s (though not correctly) the emergence of a single unified world of computing in place of the visibly distinct business and scientific

Although Cobol offered less difficulty in comprehension to novices, language complexity and inefficient compilers made its teaching no easier.

streams of applications. This prompted IBM to plan and implement two major design efforts in early 1960s—a single machine and a single programming language for all applications. While the machine effort—the IBM 360 family—was a great success, the language design effort leading to PL/1 was a failure, partly due to its much-criticized complexity [2].

Now that computer science has emerged as a separate discipline in itself, a first course in programming usually aims at teaching basic programming concepts, data and control structure, and good programming style—a style that has grown out of unending discussions and arguments spread over numerous articles in literature, emphasizing goto-less programming, structured modular design, data abstraction, well-defined control structures, and the like.

The earlier languages, like Fortran and Cobol, made teaching of programming somewhat ill-disciplined, though not evident at that time, due to primitive control structures and lack of adequate debugging support from compilers running on machines with limited core memory. For example, no justifications could be offered to an inquisitive novice regarding why array subscripts must follow a strict syntax in Fortran, since 4*I was valid as subscript but I*4 was not. Moreover, processing of alphanumeric information—the strings in the current terminology—was a headache to teach or comprehend using the infamous format statements since a varying number of characters could be packed into variables. And of course, parameters in Fortran subprograms offered unlimited scope for blunders, since parameter checking was minimal. Thus parameter type mismatches could wreak havoc within a program, with no indications of illegal actions, except through an incorrect result noticeable only by a sharp-eyed programmer. Although Cobol offered less difficulty in comprehension to novices, language complexity and inefficient compilers made its teaching no easier.

The need for better and versatile languages lead to the emergence of many new programming languages in the 1960s. Two of these languages deserve a mention here; Algol [5] was the result of a joint U.S.-Europe design effort; PL/1 was IBM's dream child [4]. Each of these was conceived as the single universal programming language suitable for scientific as well as business applications. Both languages were complex and huge by design and failed to gain wide acceptance despite a PL/1 promotion by the giant IBM and the great innovativeness and versatility of Algol.

Also, debugging and program development was a painstaking task in the 1960s and 1970s, since users were kept at a distance from the machines and the only way to communicate with computers was via punched cards through a counter/window in the computer center. Cards were used to feed programs and data to machine, and correcting a statement meant replacing a card by a new punched card and resubmitting the whole properly arranged deck of hundreds of cards at a time for a rerun on the machine.

The complexity of PL/1 and Algol, and the subsequent failure to gain wide acceptance, lead to the belief that the goal of a single universal language was impractical, and simplicity in language design was essential for a language to become popular among users. This prompted the famous European language designer Niklaus Wirth [6] to announce Pascal language in 1971 as an ideal language to teach programming, and well-suited for general-purpose applications, with claims to simplicity, well-structured data and control structures, and efficient and strict-checking compiler implementation. Pascal amazed the computer industry by gaining worldwide acceptance in academic institutions as well as outside, and effectively replaced Fortran in academic programs all over the world in the 1970s.

The advent of PCs and wide noninstitutional use of computers has lead to an unforeseen level of software development activity since the early 1980s. All this has tremendously helped computing at large. With unfaltering support from technological breakthroughs in hardware development during the last three decades, and the availability of faster processors and high-volume storage media, the world of programming and programming languages has

changed dramatically.

Now, highly versatile languages like C, C++, Visual Basic, and Java offer unlimited flexibility and a tempting freedom from semantic rules to produce efficient programs that can easily get messy due to lack of compile-time checks for improper program behavior. Visual programming environments have made programming a fun and enjoyable creative activity full of pleasure if you have time and an innovative drive to "computerize," in most cases without any traces of the number-crunching and file-handling processes of the early years of programming.

Many pioneers deserve credit for contributing to the evolution of the present-day GUI-based program development environments. Not to ignore are language designers like Backus [1], who made high-level programming a viable alternative by outperforming early assemblers in the late 1950s; Wirth [6] and Hoare [3], who promoted simplicity in programming languages; software vendors like Borland, who produced inexpensive and highly efficient compilers; and the vision of user-friendliness prompted by Apple and Microsoft. C

References

1. Backus, J.W. The IBM 701 speedcoding system. *J. ACM 1* (1954).
2. Dijkstra, E.W. The humble programmer. *Commun. ACM 15*, 10 (Oct. 1972).
3. Hoare, C.A.R. The emperor's old clothes. *Commun. ACM 24*, 2 (Feb. 1981)
4. IBM. *The New Programming Language*. IBM U.K. Laboratory, 1964.
5. Naur, P. Report on the algorithmic language ALGOL 60. *Commun. ACM 3*, 5 (May 1960).
6. Wirth, N. The programming language Pascal. *Acta Infomatica 1*, 1 (1971).

Masud Ahmad Malik (masudmalik@hotmail.com) is a professor of computer science at Quaid-i-Azam University, Pakistan.