Jean E. Sammet

History of IBM's Technical Contributions to High Level Programming Languages

This paper discusses IBM's technical contributions to high level programming languages from the viewpoint of specific languages and their contributions to the technology. The philosophy used in this paper is that it is the appropriate collection of features in a language which generally makes the contribution to the technology, rather than an individual feature. Those IBM languages deemed to have made major contributions are (in alphabetical order) APL, FORTRAN, GPSS, and PL/I. Smaller contributions (because of lesser general usage) have been made by Commercial Translator, CPS, FORMAC, QUIKTRAN, and SCRATCHPAD. Major contributions were made in the area of formal definition of languages, through the introduction of BNF (Backus-Naur Form) for defining language syntax and VDL (Vienna Definition Language) for semantics.

1. Introduction

This paper delineates some of IBM's technical contributions to high level programming languages, with some discussion of related work outside IBM to provide perspective. Section 2 provides a brief but broad chronological tracing of high level language developments, and Section 3 discusses the two very early IBM languages. Of all the other languages developed by IBM, four are major because they were widely used, as well as making significant technical contributions; they are described in Section 4. The four (listed alphabetically) are APL, FORTRAN, GPSS, and PL/I. Other languages (*e.g.*, FORMAC, Commercial Translator) had significant impact on the technology and/or on the development of other languages but never became major in their own right; those are discussed in Section 5. Still others made lesser conceptual contributions to the field (*e.g.*, COURSEWRITER) and are briefly mentioned in Section 7. Section 6 deals with formal definition methodology.

Aside from Section 2, which provides a very broad framework for the whole field of high level programming languages, this paper does not discuss the IBM languages in strict chronological order. The reason for this is that there was relatively little interdependence among the IBM languages. While each language used whatever was appropriate from existing or prior language technology both inside and outside IBM, there was no direct upward technical progression, as occurs in some other aspects of the computer field. Furthermore, the thrust of this paper is on each language as a unit, rather than its component elements. In most cases, it was really the proper technical packaging of ideas—some old, some new—in each language which made the contribution. Stated more explicitly, it is my view that the significant technical contributions made in the programming language area are by the cohesive combination of features in a language, and not particularly by an individual feature in a single language regardless of its novelty.

In addition to specific languages, there have been technical contributions from IBM in related fields. The whole subject of compilers is being covered in another paper [1] in this issue. However, methodologies for language definition are mentioned in Section 6 of this paper; the Vienna Definition Language is discussed in more detail in [2] in this issue.

The definition of the term "programming language" has been unclear and controversial almost from the beginning of computer activity. In this paper, "programming

language" is used as the equivalent of "high level language." The latter was defined in [3] to have the following four characteristics:

1. Knowledge of machine code is unnecessary.
2. There is good potential for converting a program written in high level language for one computer to run on another computer with minimal difficulty.
3. There is an instruction expansion, *i.e.*, a single statement in a high level language will produce many machine code instructions.
4. The notation for the language is problem oriented, *i.e.*, it is closer to the original conceptual statement of the problem than are machine instructions.

The last point in particular is meant to exclude from the high level language category any system involving fixed fields and fixed formats. This eliminates RPG and decision tables from the category of high level languages. It must be emphasized that this is a technical taxonomy of high level languages and not a value judgment of the usefulness of RPG or decision tables or any other tool or technique with which the user communicates with the computer. Also excluded are assembly languages (even with macros), languages for doing microprogramming, command languages, and text editors. Again, there is no value judgment intended here of the importance or value of these facilities but merely an attempt to keep within the definition. Finally, a significant set of subroutines added to an existing language, but without any change to the base language (*e.g.*, SLIP [4]), is not considered a new language.

The general area of application programming tools, some of which border on high level languages, is also outside the scope of this paper.

Because many of the source documents referred to in this paper are not publicly available, and because most of the languages mentioned in this paper have been described in the author's book [3], many citations to the book are used to provide an easily accessible source for the reader to find both a description of the language and the primary references.

Naturally, the comments in this paper and the value judgments expressed or implied are the personal views of the author and do not represent an official view of the IBM Corporation.

2. Broad chronology

In order to provide perspective on IBM's technical contributions to programming languages, we trace briefly their overall development. (A more detailed history covering the period through 1971 is given in [5].) Later sections of this paper discuss each of the specific IBM languages.

The earliest work known which legitimately fits the programming language definition given earlier is the "Plankalkül" by Zuse in Germany (1945). Unfortunately, this was not implemented. The next step was Short Code, suggested by J. Mauchly and implemented by others at Remington Rand UNIVAC (1949–50), and then the unnamed and unimplemented language developed by Rutishauser in Switzerland (1952). The Speedcoding system for the IBM 701 was developed by IBM in 1953 and is discussed in Section 3. In this time frame, Remington Rand produced the A-2 and A-3 systems (based on three-address pseudocodes to indicate mathematical operations), and the Boeing Company developed BACAIC for the 701. All these languages, plus others of that period, attempted to provide scientists and engineers with a notation slightly more natural to mathematics than machine code. Some permitted the users to write mathematical expressions in relatively normal format, but most did not. None of them had any significant or lasting effect on language development, and apparently minimal effect (if any) on people's thinking.

In May 1953 J. H. Laning, Jr., and N. Zierler at MIT had a system running on Whirlwind that appears to be the first system in the United States to permit the user to write expressions in a notation resembling normal mathematical format, *e.g.*,

$$c = 0.0052\ (a - y)/2ay,$$

$$y = 5y,$$

An excellent description of most of these early mathematical systems, plus others, appears in [6], and some of them are described briefly in [3], and even more briefly in [7].

In 1954, work on FORTRAN started, and the 704 compiler was released in April 1957. PRINT (described in Section 3) was actually finished before FORTRAN.

About the time the preliminary FORTRAN report was issued, a group at Remington Rand UNIVAC under Grace Hopper's direction began development of a system originally called AT-3 and later renamed MATH-MATIC. (John Backus says that he sent them a copy of his November 1954 preliminary FORTRAN report, but I cannot determine how much influence it had on MATH-MATIC. The preliminary FORTRAN specifications precede any language design documents on MATH-MATIC from Remington Rand UNIVAC.) MATH-MATIC (described in [3, 6]) was similar in spirit to FORTRAN, although different in syntax, and it is

not clear which system was actually running first. However, of all the parallel work going on in the mid-1950s, only FORTRAN has survived, and by 1957 there were the first glimmerings of significant practical usage of a high level language similar to those we know today.

While the main emphasis prior to 1958 was on the development of languages for scientific applications, the first English-like language for business data processing problems, FLOW-MATIC, was planned and implemented on the UNIVAC I under the direction of Grace Hopper at Remington Rand UNIVAC and released in 1958. (A description appears in [3].)

Activity on another language, APT, started in 1956 at MIT under Douglas T. Ross. APT was for numerical machine tool control, and hence was the first language for a specialized application area. (See [8] for full details on the early development.) APT (albeit modified over time) was still in use in 1980.

The years 1958 and 1959 were among the most prolific for the development of programming languages. The following events in universities and industrial organizations all occurred during those two years:

1. The development of the IAL (International Algebraic Language), which became known as ALGOL 58, and the publication of its definition [9]. IAL had a profound effect on the computing world, because of
   a. Its follow-on, ALGOL 60, which is clearly one of the most important languages ever developed, and
   b. The development of three languages based on the IAL specifications, namely, NELIAC, MAD, and CLIP (which eventually was the foundation for JOVIAL). All except CLIP became widely used.
2. The availability in early 1958 of a running version of IPL-V (a list processing language).
3. The start of work on the development of LISP, a list processing language which was intended for artificial intelligence applications.
4. The first implementation of COMIT, a string processing language.
5. The formation in May 1959 of the CODASYL (Conference On Data SYstems Languages) Short Range Committee, which developed COBOL, and the completion of the COBOL specifications.
6. The development and availability of language specifications for AIMACO, Commercial Translator (see Section 5), and FACT, all of which were for business data processing problems.
7. The availability of specifications for JOVIAL.

Of all these languages from 1958–59 and earlier, those that survive (albeit in modified form) and have significant usage in 1980 are ALGOL 60, APT, COBOL, FORTRAN, JOVIAL, and LISP. References and a discussion of all the languages named in this section can be found in [3]. A detailed history of the development of the six languages just cited is provided in [10].

A large impetus for most of this work was economic—even then programming costs were large, and any steps or tools which could reduce those costs were looked at favorably. However, the crucial issue often was whether any "slowdown" caused by these systems exceeded the overall savings in people's money or time; generally the answers favored the use of such systems.

The period 1960–1970 saw some maturation of the programming language field. The material for this period is taken almost verbatim from [5], copyright 1972, Association for Computing Machinery, by permission; descriptions of and references for the languages mentioned are in [3]. During this time the battle over the use of high level languages was clearly won, in the sense that machine coding had become the exception rather than the rule. (This comment is based only on the author's opinion and perception because there is simply no data to verify or contradict this statement.) Although the concept of developing systems programs by using high level languages was fairly well accepted, there was more machine coding of systems programs than of application programs. The use of powerful macro systems and "half way" languages such as PL/360 [11] provided some of the advantages of high level languages but made no attempt to be machine-independent.

The major new batch languages of this decade were ALGOL 60, COBOL, and PL/I, of which only the last two were extensively used in the United States. Although ALGOL 68 was defined, its implementation was just starting around 1970.

The advent of interactive programming in the mid-60s brought a host of on-line languages, starting with JOSS and later followed by BASIC, both of which became very widely used. Each had many imitators and extenders. APL\360 (see Section 4) was made available late in the 1960s and became popular among certain specific groups.

The development of high level languages for use in formula manipulation was triggered by FORMAC (see Section 5) and Formula ALGOL, although only the former was widely used. String processing and pattern matching became popular with the advent of SNOBOL.

The simulation languages GPSS (see Section 4) and SIMSCRIPT made computer simulation more accessible to most users and also encouraged the development of other

simulation languages. A number of other languages for specialized application areas (*e.g.*, civil engineering, equipment checkout) continued to be developed.

Perhaps one of the most important practical developments in this time period, although scorned by many theoreticians, was the development of official standards for FORTRAN and COBOL and the start of standardization for PL/I.

The period 1970-1980 involved relatively few significant pure language developments. Those few include: (1) the implementation and initial limited usage of ALGOL 68; (2) the implementation and heavy use of Pascal; (3) the massive effort by the Department of Defense to develop a single language—called Ada—for embedded computer systems [12, 13]; (4) the concept of data abstraction; and (5) concepts of functional programming by Backus [14], and (6) experimental languages such as CLU, EUCLID, and SETL. It is too early to tell which—if any—of these concepts and languages will have a fundamental effect on the computer field.

3. Very early IBM languages

The earliest IBM attempt at what was then considered a high level language was Speedcoding for the IBM 701 [15]. The motivation for its development was similar to that for all the early languages—to provide the user with a notation which was easier to use than raw machine code. Work was started on Speedcoding in January 1953 under the supervision of John Backus and the general direction of John Sheldon; the first official manual was dated September 1953.

The basic principle of Speedcoding was to create two sets of operations, the first category containing three addresses and the second set containing only one. The operations were not part of the hardware and were selected for their utility to the mathematician. Thus

523 SUBAB 100 200 300 TRPL 500

shows an instruction in location 523 which causes the computer to subtract the absolute value of the contents of location 200 from the value in 100 and to put the result in 300; then the computer tests the sign of the result in 300 and transfers control to 500 if it is positive.

Such a notation looks primitive by standards in use only a few years after its development, but it was actually used on many 701s and may have influenced the designers of the 704. However, Speedcoding had no lasting language effect.

The other early language of IBM was PRINT (PRe-edited INTerpretive System) [3]. It was designed under the leadership of Robert Bemer and was intended to meet the scientific computing needs of IBM 705 users. Since the 705 had been designed primarily for use in business data processing, PRINT was created to meet the needs of people who wished to use it for scientific computing. It provided a series of operation codes with variable fields (which could contain one to four variables, depending on the operation code).

Coding of the PRINT system started in February 1956, and the first customer tried it in July 1956. Thus, it was actually completed before FORTRAN. PRINT seems to have had no significant technical effect on other languages.

4. Major IBM languages

The question of which languages (of the many hundreds developed since 1950) might be construed as being "major" is highly controversial. The judgment used in this paper reflects that of a very well qualified group of individuals who served as the Program Committee for the ACM SIGPLAN History of Programming Languages Conference held in June 1978. To provide perspective for the categorizations made in this paper, we quote [10, p. *xviii*] the rationale used (in 1977) for selecting the languages for that conference and list the languages which were accepted. "The specific requirements were that the languages (1) were created and in use by 1967; (2) remain in use in 1977; and (3) have had considerable influence on the field of computing. (The cut-off date of 1967 was chosen to provide perspective from a distance of at least ten years.) The general criteria for choosing the languages are the following (not necessarily in order of importance, nor required to have each one apply to each language): Usage, influence on language design, overall impact on the environment, novelty (first of its kind), and uniqueness."

The program committee selected the following languages as satisfying the indicated criteria: ALGOL, APL, APT, BASIC, COBOL, FORTRAN, GPSS, JOSS, JOVIAL, LISP, PL/I, SIMULA, SNOBOL.

For two of these languages, FORTRAN and GPSS, the first version (and some of the later ones) were designed and implemented solely within IBM. A third, PL/I, involved a joint effort of IBM and SHARE for the language development, but the first commercial implementation was done entirely by IBM. APL was designed initially by K. Iverson at Harvard University, but its design was continued and modified after he joined IBM and was then implemented by IBM. And, of course, IBM employees contributed significantly to the external committee developments of ALGOL 58, ALGOL 60, and COBOL. The portion of this paper describing the four "major IBM languages" is based primarily on the papers prepared by the authors for the conference. (See [10, 16].)

Figure 1 Table of FORTRAN I statements for the IBM 704. The spacing is not significant.

Statement	*Normal sequencing*
$a = b$	Next executable statement
GO TO n	Statement n
GO TO n, (n_1, n_2, . . . , n_m)	Statement last assigned
ASSIGN i TO n	Next executable statement
GO TO (n_1, n_2, . . . , n_m), i	Statement n_i
IF (a) n_1, n_2, n_3	Statement n_1, n_2, n_3 as a less than, =, or greater than 0
SENSE LIGHT i	Next executable statement
IF (SENSE LIGHT i) n_1, n_2	Statement n_1, n_2 as Sense Light i ON or OFF
IF (SENSE SWITCH i) n_1, n_2	Statement n_1, n_2 as Sense Switch i DOWN or UP
IF ACCUMULATOR OVERFLOW n_1, n_2	Statement n_1, n_2 as Accumulator Overflow trigger ON or OFF
IF QUOTIENT OVERFLOW n_1, n_2	Statement n_1, n_2 as MQ Overflow trigger ON or OFF
IF DIVIDE CHECK n_1, n_2	Statement n_1, n_2 as Divide Check trigger ON or OFF
PAUSE or PAUSE n	Next executable statement
STOP or STOP n	Terminates program
DO n $i = m_1$, m_2 or DO n $i = m_1$, m_2, m_3	Next executable statement
CONTINUE	" " "
FORMAT (*Specification*)	Not executed
READ n, *list*	Next executable statement
READ INPUT TAPE i, n, *list*	" " "
PUNCH n, *list*	" " "
PRINT n, *list*	" " "
WRITE OUTPUT TAPE i, n, *list*	" " "
READ TAPE i, *list*	" " "
READ DRUM, i, j, *list*	" " "
WRITE TAPE i, *list*	" " "
WRITE DRUM i, j, *list*	" " "
END FILE i	" " "
REWIND i	" " "
BACKSPACE i	" " "
DIMENSION v, v, v, . . .	Not executed
EQUIVALENCE (a,b,c, . . .), (d,e,f, . .),	" "
FREQUENCY n (i,j, . . .), m(k,l, . . .), . . .	" "

• *FORTRAN*

As mentioned in Section 2, there were a number of early attempts at "automatic programming" systems (which is the term used in the 1950s for systems of this type). However, John Backus in [7, p. 26] states that "The Laning and Zierler system was . . . the world's first operating algebraic compiler, a rather elegant but simple one." He also describes the sequence of events which finally made it clear to him that the Laning and Zierler work did not influence FORTRAN. (Earlier, Backus had publicly stated his belief that the algebraic nature of FORTRAN was derived from the Laning and Zierler work.)

Backus indicates that in those days programmers were proud of their ability to use the computer efficiently. Many of them believed that any attempt to automate programming would lead to intolerable inefficiencies. It is important to understand that milieu because it significantly influenced the design philosophy for FORTRAN. Backus points out in [7, p. 27] that even at that time economic considerations made it clear that steps taken to reduce programming costs would be of value. He said that: "This economic factor was one of the prime motivations which led me to propose the FORTRAN project in a letter to my boss, Cuthbert Hurd, in late 1953."

Backus specifically identified [7, p. 28] the systems that existed at that time and their influence (almost none) on the development of FORTRAN. He then indicated that the 704 presented challenges to the designers of any new software system which attempted to simplify programming, because the 704 included floating point arithmetic and index registers whose simulation was part of the justification for some of the earlier software systems. He then said [7, p. 28]: "In view of the widespread skepticism about the possibility of producing efficient programs with an automatic programming system and the fact that inefficiencies could no longer be hidden, we were convinced that the kind of system we had in mind would be widely used only if we could demonstrate that it would produce programs almost as efficient as hand coded ones and do so on virtually every job." Consequently the de-

sign of the language as such was relatively straightforward, and the major emphasis was on a system which would produce efficient object code.

In order to provide an appropriate time sequence, note that a preliminary FORTRAN report was issued in November 1954, a reference manual was issued in October 1956, and a primer in early 1957. External professional publication occurred in 1957 in [17]. It is interesting to contrast the FORTRAN I statements with later and current versions of the language. Figure 1 shows all the FORTRAN I statements.

One of the interesting myths about FORTRAN that Backus puts to rest in his paper is the relation between the FORTRAN subscripts and the index registers on the 704. It had been widely thought for many years that only three subscripts were allowed in FORTRAN because there were only three index registers on the 704. Backus makes it clear [7, p. 34] that the reason for the limitation to three was the complexity of attempting to optimize index register allocation.

One of the most interesting (and novel) statements in FORTRAN I was FREQUENCY, which allowed the programmer to specify which path on a branch he thought was most likely. The compiler then attempted to optimize object code based on that information. It apparently worked well, but people usually did not have the correct data about the best branch and often used it in circumstances that made little difference; it was removed in FORTRAN IV. It should also be noted that FORTRAN I (and even FORTRAN II, discussed below) contained some machine-oriented statements, namely, references to tape/drum, sense switches, and even the accumulator (for overflow). Eventually these were dropped, when machine independence became more significant than it was in the early versions. The EQUIVALENCE statement in FORTRAN I provided a very early attempt to enable the programmer to use storage efficiently. The use of the loop statement DO became a model for all future languages (although most later languages significantly enhanced the FORTRAN loop statement capability).

FORTRAN II introduced two important concepts into the language—subroutines and the COMMON statement to permit communication among subroutines; it was also possible to link to assembly language programs. (Although subroutines had been in use for many years, their introduction into a high level language was an important step.)

Although neither FORTRAN I nor II had separate data declarations as such, FORTRAN I did use the concept indirectly, by specifying that variables whose names began with the letters I, J, K, L, M, or N were considered integers and all others were considered floating point.

A lot of cleanup and elimination of machine-dependent features was done in FORTRAN IV, and, along with FORTRAN II as a subset, it formed the basis for the first ANSI language standard(s) [18, 19].

In my own view, FORTRAN has probably had more impact on the computer field than any other single software development. Its major technical contribution was to demonstrate that efficient object code could be produced by a compiler; as a result, it became clear that productivity of programmers could be significantly improved. The fact that FORTRAN still exists in spite of more modern languages with newer concepts is testimony to the soundness of many of the original ideas (but also to inertia and investment in old programs). Naturally, since the first version of FORTRAN, a number of additions and changes have been made and presumably will continue to be made. But the basic framework was sound and enabled many people to build upon it.

This is a reasonable place to indicate some of the relationships between ALGOL and FORTRAN, particularly since Backus (of IBM) was one of the four U.S. representatives to the IAL (= ALGOL 58) activity; he also participated in the ALGOL 60 development. Since both FORTRAN I and II were released before the ALGOL 58 committee finished its language design [9], ALGOL 58 could not have any effect on FORTRAN. Space does not permit a discussion here of the effect FORTRAN had on ALGOL 58. In my view, neither of the ALGOL versions had any significant effect on FORTRAN IV.

GPSS (General Purpose Simulation System)

GPSS was developed by Geoffrey Gordon. He had worked on simulation studies in various places prior to joining IBM, including Bell Telephone Laboratories, where he participated in the development of a program called the Sequence Diagram Simulator, which was presented at a meeting of the IEEE in 1960. When he joined IBM in June 1960, it was in the Mathematics and Programming Department under D. V. Newton.

The work being done in that department involved queuing models. Gordon suggested developing a system description language based on the Sequence Diagram Simulator approach. He began writing a program implementing a block diagram language and chose a relatively simple table-oriented interpretive structure that would make changes easier. It was written using the SAP assembly language for the IBM 704. Gordon stated [20, p. 407]

that he was not aware of the work on SIMSCRIPT being done about the same time.

GPSS is a discrete simulation language, and the major characteristic that has made it useful twenty years after its initial development is that the block diagram portion of the language provides an excellent means of communication with systems people.

The most relevant similar development is that of SIMSCRIPT [21], which was also developed in the early 1960s, at the Rand Corporation. Unlike GPSS, SIMSCRIPT is a statement-oriented language. While it is not a direct extension of FORTRAN, the SIMSCRIPT "style" is very similar to that of FORTRAN, and many versions have been implemented by means of a preprocessor which translates the SIMSCRIPT program into FORTRAN.

While GPSS has been developed and improved in many versions, and continues to be in use in the early 1980s, it did not have a major impact on other language developments. I am not aware of any other major discrete simulation languages that have followed the block diagram conceptual approach. (Some of the continuous simulation languages involve block diagrams, but the need for that is more obvious.) The other discrete simulation languages have all tended to follow the statement language orientation, and a number of them have been based on ALGOL rather than a FORTRAN-like approach. In particular, SIMULA [22], which was originally intended as a major simulation language and then became more generally used, was certainly based on ALGOL.

- *APL*

The concepts of APL (A Programming Language) were defined by Kenneth E. Iverson in his 1962 book [23]. At that time he said [24, p. 345]: "The language is based on a consistent unification and extension of existing mathematical notations, and upon a systematic extension of a small set of basic arithmetic and logical operations to vectors, matrices and trees." It is perhaps an understatement to say that APL, as originally described in Iverson's book, was not received by the computing world with much enthusiasm. The combination of an unusual character set and concepts which were quite different from those of the more popular programming languages in the early 1960s tended to make a number of people say, and with some justification, that this was a "notation" rather than a programming language.

Iverson's original motivation was to provide a unifying method of describing and analyzing various concepts in data processing. In [25, p. 662] it is said that: "It is difficult to pinpoint the beginning, but it was probably early 1956," when Iverson was at Harvard. Shortly after Iverson joined IBM in 1960, Adin Falkoff started to work with him, and most of the work done on APL since then has been done jointly.

In the early 1960s, a formal description of the System/360 machine language was undertaken at IBM [26]. This was a significant use of APL, but not the first of its type, since Iverson's book contained a description of the 7090. A running version of a language based on Iverson's notation was developed on the IBM 1620 under the name PAT (Personalized Array Translator) [27], but this was hampered enormously by the need to use standard typewriter characters.

The breakthrough in demonstrating feasibility came with the development of a stand-alone interpretative system on the IBM 360/50 at the IBM Thomas J. Watson Research Center in the mid-1960s. Almost all of the language was implemented; however, the key breakthrough in my own view resulted from two factors. First, a SELECTRIC® printing element was designed to include almost all of the APL characters then in use, and the language which was implemented was in fact then restricted to those characters. Actually not much was left out, and Iverson states [25, p. 665] that some of the necessary changes "were beneficial, and many led to important generalizations." That paper describes several effects that the restriction to a linear 88-character set had on the language, *e.g.*, the notion of composite characters which are formed by striking one basic character over another.

The second breakthrough that helped make APL successful was that the implementation was relatively efficient. At that time (*i.e.*, the mid-1960s) many interactive systems tended to be quite inefficient in execution, partly because they had complex command languages. The ease with which users could communicate with the computer (because APL\360 was a stand-alone system) made it attractive to some users. That "ease of use" factor was something of a technical breakthrough, partly because there was no separate command language.

APL has demonstrated a flexibility for handling different types of problems that has been amazing to those people who view it as being primarily for mathematical problems. As well as by mathematicians and engineers, it has been found to be useful by administrators, secretaries, and people in business environments.

There are strong feelings about APL, both for and against. However, a large number of people who have not been personally trained and introduced to it by the developers have found it to be an extremely useful tool. That

pragmatic observation of usefulness would seem to outweigh emotional and even intellectual considerations from language designers and other computer scientists who may or may not feel that it is within the mainstream of language development. Its uniqueness results from (1) its very large and unusual character set and (2) the very large functional capability expressed by individual notations. For example, the notation for multiplication of matrices is shown below for FORTRAN and for APL.

FORTRAN	*APL*
DO 100 I = 1, M	$C \leftarrow B + .xA$
DO 100 J = 1, N	
C(I, J) = 0	
DO 100 K = 1, P	
100 C (I, J) = C(I, J) + A(I, K)*B(K, J)	

From a pure language viewpoint, APL introduced significant new features and concepts. For example, (1) because of the heavy emphasis on vector and matrix operations, there is little need to write the loops required in other languages to achieve equivalent results; (2) it provides a vast number of primitive operators, such as "Floor" and "Minimum"; and (3) the reduction operator allows brevity in applying other operators. However, in spite of (or perhaps because of) its uniqueness, it has had virtually no effect on other language design.

PL/I

Of all the languages developed within IBM, certainly the PL/I effort has to be considered the one with the largest and most grandiose goals. The motivation and background for PL/I stem from two somewhat different sources.

The first involved FORTRAN. By the early 1960s, improvements to the original FORTRAN had been made by delivering FORTRAN II and FORTRAN IV to customers. (FORTRAN III had been only an internal IBM development.) Considerations of extensions to FORTRAN IV were perennially considered. It was clear by the early 1960s that FORTRAN was extremely popular, but would not serve the needs of everyone.

To understand the second viewpoint, it is important to remember that in the early 1960s computer applications (as well as the machines and the programmers) tended to be thought of as either scientific (*i.e.*, engineering and mathematical) or commercial (*i.e.*, business data processing). One of the major objectives of the IBM System/360 was that it would be equally effective for both scientific and commercial applications.

The convergence of these two ideas, *i.e.*, to improve FORTRAN and to create a line of machines to be used across a broad set of applications, caused IBM and SHARE to jointly form the Advanced Language Development Committee of the SHARE FORTRAN project in October 1963, with Bruce Rosenblatt (Standard Oil of California) as Chairman and George Radin as Chairman of the IBM delegation. The group was supposed to specify a programming language which would meet the needs of the user classes indicated above, as well as the needs of systems programmers. Although I do not believe it was stated at that early stage, it was also expected that development of an effective "single" language would be beneficial to both IBM and its customers by eliminating the need for FORTRAN and COBOL.

The earliest significant attempt at a somewhat broad language had been JOVIAL, developed at the System Development Corporation in the early 1960s by Jules Schwartz and others [28]. JOVIAL not only had capabilities for the scientific programmer but also methods for specifying how data were to be allocated within the computer memory; it also introduced in a language the notation of a communications pool (COMPOOL) by which the same descriptions could be used by many programs.

But JOVIAL, which was being used almost exclusively for Air Force projects, did not really satisfy all the needs indicated above. The early orientation of the PL/I design was to provide appropriate extensions to FORTRAN. Although the project was originally referred to as FORTRAN VI, it rapidly became clear that it would be impossible to maintain upward compatibility with FORTRAN IV and also meet the objectives indicated for the design of PL/I.

With regard to the decision to abandon FORTRAN IV as a base, Radin says [29, p. 555]: "In retrospect, I believe the decision was correct. Its major drawback was that, by taking FORTRAN as a base, we could have gone, in an orderly way, from a well-defined language to an enhanced well-defined language. We could have spent our time designing the new features instead of redesigning existing features. By starting over, we were not required to live with many technical compromises, but the task was made much more difficult."

One aspect of the early development of PL/I was the rapidity with which it was supposed to be completed. According to Radin [29, p. 553], from a starting date of October 1963, they "were first informed that the language definition would have to be complete (and frozen) by December 1963. In December we were given until the first week in January 1964 and finally allowed to slip into late February." The objective of this tight time schedule, of course, was to make it feasible to introduce PL/I at the same time as the hardware which became known as the IBM System/360.

There is no space available here to trace the many and somewhat tortuous twists and turns taken to produce the "final PL/I." But PL/I made a number of significant contributions to the technology. For its time, PL/I was the culmination of the procedural line exemplified by ALGOL, COBOL, FORTRAN, JOVIAL, and others. It included almost all the good features from those languages, although generally in a different syntax, and included conceptual features from other languages (*e.g.*, pointers to allow list processing). Although other languages (*e.g.*, ALGOL) contained string variables, PL/I was the first language to provide operations on the strings, such as concatenate. PL/I was the first language to address itself seriously to the problems arising from the need to interact with an operating system. It provided more facilities for dealing with storage allocation, task management, and exception handling than any other language to that date. For example, the user can specify asynchronous execution of tasks and control their execution based on factors such as time delays or completion of another task. PL/I seems to have introduced the concept of generic functions, which means that a single function name can be used to cover a variety of input data types (*e.g.*, fixed or floating point numbers). It used the concept of default conditions very heavily; that eventually was viewed by some people as one of its weaknesses. In its attempt, and indeed its success, at being broad, PL/I became very large and in some cases produced surprising results. For example, many pages in the manual were needed to describe the interaction of implied conversions among variables of different types, and the object code did not always produce the results expected by the programmer's common sense.

PL/I was a major technical undertaking in its language design and implementation. It was the first significant multipurpose language and it introduced a large number of innovations.

5. IBM languages with significant technical impact but not wide usage

IBM developed a number of languages which made significant technical contributions but were not used as much as the ones discussed in Section 4. The most important of these (in my view) are Commercial Translator, FORMAC, QUIKTRAN, CPS, and SCRATCHPAD. Of these, FORMAC had the most effect for reasons to be discussed later. The other four languages had indirect or secondary effects.

- *Commercial Translator*

In order to understand the technical role played by Commercial Translator, it is helpful to understand its place in the environment. Around 1955, Grace Hopper and her department at Remington Rand UNIVAC developed a language for business data processing known originally as B-0 and later renamed FLOW-MATIC. (See description in [3, Chap. V].) They had preliminary specifications as early as January 1955, and the first implemented version distributed to customers was available early in 1958. FLOW-MATIC introduced two major concepts. One was the idea of using relatively long identifiers to get readability (*e.g.*, SOCIALSECURITY, INCOMETAX), along with English verbs for operations such as COUNT, INCREMENT, ADD, etc. The other major concept introduced by FLOW-MATIC was the combination of (1) separating the data description from the statements that were to operate on it and (2) the availability of a fairly flexible data format. Up until that time, all the high level languages had been for scientific computing, in which the types of data needed were integers and floating point numbers, with possible fixed radixpoint, complex numbers, and double precision also being used. But in all of those cases it was assumed that a number representation took an entire machine word (or two) regardless of what its maximum value actually was. Scientific users were not faced with the conceptual problem of data in which some elements might require only one character or even one bit (*e.g.*, distinguishing between male and female) or many computer words (*e.g.*, a person's address). FLOW-MATIC broke new ground in both of those areas, and IBM recognized that it needed to provide a comparable service to its business data processing customers.

As early as January 1958 there were some preliminary specifications for a language (eventually named Commercial Translator) to be used for business data processing; the work was done under the technical leadership of Roy Goldfinger, with Robert Bemer as the manager. Following the philosophy established in FLOW-MATIC, Commercial Translator was an English-like language, but it introduced several significant concepts. One was the use of formulas which were standard in scientific languages but were ostensibly new to the business data processing environment. A second key feature was the introduction of the IF . . . THEN facility, which had first appeared in ALGOL 58. The third idea, building on the data description facilities in FLOW-MATIC, was the concept of allowing several levels of data hierarchy. Finally, the PICTURE clause provided a succinct description of data characteristics such as alphabetic or numeric, placement of the decimal point, number of characters, etc.

But before IBM had implemented Commercial Translator, work on COBOL started outside of IBM. In May 1959 the Short Range Committee was chartered under CODASYL (Committee on Data Systems Languages), which was established under the auspices of the Department of Defense. The Short Range Committee consisted of representatives from six manufacturers, including

IBM, and three representatives from government organizations. The primary inputs to the work of the Short Range Committee were FLOW-MATIC, which had actually been in use for over a year, AIMACO (a modification of FLOW-MATIC developed by Air Force Air Material Command), and Commercial Translator [30], which existed as a set of unimplemented specifications. (Brief descriptions of these three languages, and references for them, are in [3, 31]. The history of COBOL is delineated in [31].) The important point is that Commercial Translator was one of the two major inputs to COBOL, and two of the key people on the Short Range Committee which designed COBOL were IBM employees, namely, William Selden and Gertrude Tierney. The significant new concepts of Commercial Translator indicated above were among those included in COBOL. Then, as work on Commercial Translator continued, as the Short Range Committee produced specifications for COBOL, and as Honeywell produced specifications for their own business data processing language known as FACT (see description in [3, Chap. V]), these three languages became interwined. Thus, in 1960 the Commercial Translator manual showed certain concepts that were obtained from COBOL and even from FACT, and, of course, COBOL and FACT were significantly influenced by ideas in Commercial Translator.

Commercial Translator was implemented on the IBM 7070, 7080, and 709/7090. The latter implementation was extremely efficient, which pleased customers, but COBOL prevailed. Thus, after the initial implementations on the 7070 and 7080, IBM discontinued further work on Commercial Translator except for those 7090 customers who insisted on it, and then dropped it completely when going to the System/360. The net result was that Commercial Translator had literally faded from any significant usage by the time the 360 was introduced. For one person's view of IBM's handling of this matter, see [32].

- *FORMAC*

The basic concepts of FORMAC (FORmula MAnipulation Compiler) were first developed by Jean E. Sammet (assisted by Robert G. Tobey) at IBM's Boston Advanced Programming Department in July 1962. I recognized that what was needed was a formal algebraic capability associated with an already existing numeric mathematical language; FORTRAN was the obvious choice. An internal memo describing the basic ideas was written on August 1, 1962, and a complete draft of language specifications was finished in December 1962; implementation design started shortly thereafter. The basic objective was to develop a practical system for performing formal mathematical manipulation on the IBM 7090/94. Originally FORMAC was intended only as an experiment, and there was no plan to make it available outside of IBM. In April 1964 the first complete version was successfully running after extensive testing, and papers on it appeared in the literature [33, 34]. As a result of pressure from numerous people who were interested in trying the system, and also as a way of obtaining feedback from users that would lead to better systems in the future, FORMAC was released in November 1964 but with no committed maintenance or support for it. Nevertheless, there were numerous users.

The use of a computer for performing formal algebraic manipulation went back to 1954, in which two Master's theses (one at MIT and one at Temple University—see references in [3]) involved programs to do formal differentiation. By the early 1960s, various programs were written to do formula manipulation for specialized purposes, *i.e.*, a particular group, such as astronomers or airplane designers, developed a set of routines to do the type of formula manipulation that was needed for their applications. But there was only one attempt at language development, namely ALGY [35], which was an interpretive system on the Philco 2000 computer. It allowed commands such as removing parentheses from an algebraic expression, substituting one or more expressions into another one, factoring a given expression with respect to a single variable. ALGY contained no arithmetic capability, nor was there any facility for loop control or control transfer. The primary objective of FORMAC resulted from the realization that when doing formula manipulation on a computer one needed input/output, numerical arithmetic, loop control, etc. From those needs and premises, the use of a language providing those facilities became necessary, and it seemed logical, as stated earlier, to build on an existing language which already contained the nonformula capabilities. Therefore, FORMAC was literally designed as a language extension of FORTRAN IV and was implemented by a preprocessor which used run-time subroutines to do the formula manipulation.

The conceptual contrast between FORTRAN and FORMAC is shown below.

FORTRAN	*FORMAC*
A = 5	
B = 3	
C = (A − B)*(A + B)	C = (A − B)*(A + B)
yields	yields
C ← 16	C ← $A^2 - B^2$

FORMAC also provided commands for formal differentiation, replacing variables with expressions, removing parentheses, evaluating expressions for specific numerical values, comparing expressions for equivalence or identity, etc. It simplified expressions automatically (as do most of the systems). For further descriptions of the

first FORMAC, see previous citations or [3, Chap. VII]. Eventually a version of FORMAC based on PL/I was developed for use on the System/360 and released, but again with no commitment for maintenance [36]. It was a significant improvement over the earlier version but did not introduce any major new concepts. Both versions were batch oriented but could be used interactively. PL/I-FORMAC remained in minor use in the early 1980s, even though far more sophisticated and better systems existed by then.

During the time that work was being done on the original 7090 FORMAC, a research effort was underway at Carnegie Tech (now Carnegie Mellon University) under the direction of Professor Alan Perlis to develop a system called Formula ALGOL [37]. Both of these efforts proceeded independently; in the very early stages neither was even aware of the other, and after that knowledge did become mutually available, each continued in its own direction. Formula ALGOL was never used extensively outside Carnegie Mellon University. Aside from the obvious differences arising from using ALGOL rather than FORTRAN as a base, there was one major conceptual difference in the two approaches. Formula ALGOL initially used very low level primitives from which the user could build up his own commands, whereas FORMAC used higher level commands.

FORMAC introduced and/or emphasized two major concepts—the desirability of a language for doing formula manipulation, rather than just a series of routines, and the concept of extending an existing language to provide this type of capability. Of the other two systems that appear to be in general use in 1980 one, REDUCE [38], has more or less followed the FORMAC language philosophy by adding capabilities to ALGOL, whereas MACSYMA has developed its own language [39]. Perhaps the most lasting value of FORMAC was its major role, along with Sammet's formation of the ACM Special Interest Group on Symbolic and Algebraic Manipulation (SIGSAM), in getting this technical field started.

- *SCRATCHPAD*

SCRATCHPAD is an experimental, LISP-based, symbolic mathematical system which runs under VM/370 at the IBM Thomas J. Watson Research Center. This work was started by James H. Griesmer in 1965, who was joined shortly after by Richard D. Jenks and later yet by David Y. Y. Yun. Systems outside IBM which were started around that time were REDUCE [38] and MACSYMA [39]. A description of SCRATCHPAD is in [40].

SCRATCHPAD shared with FORMAC (and other systems for symbolic computation) the philosophy that a language (and not just subroutines) was needed for dealing with symbolic mathematical problems. However, it differed from FORMAC in two major ways. One was that a major objective of SCRATCHPAD was to have the language be as natural for mathematicians as possible; thus it was designed *ab initio* and not based on any existing programming language. The SCRATCHPAD language is less procedural, allowing an intended computation to be described by a set of rewrite rules. Although SCRATCHPAD permitted two-dimensional input (for subscripts, superscripts, limits on summations and integrals), the actual input to the computer had to be linearized because standard equipment does not permit two-dimensional inputs; this in my view prevented SCRATCHPAD from meeting the objectives of naturalness to mathematicians.

The second major difference was that SCRATCHPAD was designed from the beginning to be interactive. Unlike normal numerical calculations which can effectively be done in batch mode, for many (although certainly not all) problems involving symbolic computation the user needs to see the formula displayed before knowing what to do next.

SCRATCHPAD provides a wide range of built-in symbolic facilities, which at present are exceeded only by those contained in MACSYMA. SCRATCHPAD facilities include differentiation, integration, polynomial factorization, solution of equations, APL array operations, formal manipulation of finite and infinite sequences and power series, and conversational "backtracking," which allows a user to return to a previous state in his computation.

SCRATCHPAD has never been released outside of IBM, and therefore its technical influence has been limited to an active publication plan in which numerous papers and talks have been given. People outside of IBM have come to the IBM Thomas J. Watson Research Center and successfully used the system for productive work.

- *QUIKTRAN*

Work on QUIKTRAN was started in IBM in 1961 by a group under the direction of John Morrissey. While their original objective was to improve user debugging facilities, this eventually took the form of a dedicated system which was essentially FORTRAN but with powerful debugging and terminal control facilities added. Two major constraints that the designers imposed upon themselves were to use only existing standard equipment (which turned out to be the 7040/44 computers and the 1050 terminal) and to use and stay consistent with an existing language (which turned out to be ANS Basic FORTRAN) defined in [18]. A first version was running in mid-1963. The best language reference is [41]. This system was eventually released for customer use.

The system was designed to handle most legitimate ANS Basic FORTRAN programs. The intent was to allow programs to be debugged using QUIKTRAN and then be compiled for production running on a regular compiler. QUIKTRAN introduced the concept of allowing either the COMMAND or the PROGRAM mode. In the former case each statement entered by the user was executed immediately, and the result was printed at the terminal; this was referred to as the "desk calculator" mode, and the statements were not retained by the system. In the PROGRAM mode, the statements were saved and executed only at the specific request of the user through some other commands. While this concept is very common now, it was either unique or relatively new at that time. JOSS [42] had the same capability, but it is not clear which had it first. In any case both systems were being developed at approximately the same time.

QUIKTRAN was significant from several viewpoints. It was the first on-line system using commercially available equipment (JOSS used the unique JOHNNIAC at the Rand Corporation). QUIKTRAN also retained compatibility with an existing major language and thus made it possible for a user to debug a program on-line and then use a regular FORTRAN compiler for batch production runs. JOSS, of course, and then later BASIC were unique interactive languages designed *ab initio* and each became widely used. There is no clear indication that QUIKTRAN had any long-lasting effect on software technology.

- *CPS (Conversational Programming System)*

CPS was a small, on-line, extended subset of PL/I. It was developed jointly by the Allen-Babcock Corporation (primarily by J. D. Babcock and P. R. DesJardins) and the IBM Corporation under the overall direction of Nathaniel Rochester with significant work by David A. Schroeder. The language is described in [43]; the work started early in 1965, and the initial version became operational in the fall of 1966. The system had two goals: One was to provide a language in the middle area between JOSS and QUIKTRAN. This really meant that it was to be as simple as possible for the terminal user, but the language was to have as much of the syntax of PL/I as possible. The second major objective was to investigate the effectiveness of microprogramming.

The system was originally implemented on a System 360/50 with a special read-only store which was used for special machine instructions to make the language interpreter more efficient. This was an important technical investigation at the time, and appears to be one of the early attempts at implementing software functions in hardware via microprogramming. However, some routines were also written to replace the microprograms and thus avoid the necessity for special hardware. A version of the system (called RUSH for Remote Use of Shared Hardware) was the basis for a commercial time-sharing service offered by the Allen-Babcock Corporation, while the system in use by IBM was called CPS. Starting from the same base the two systems added different facilities and eventually diverged significantly. CPS's primary technical contributions were (1) it was an early interactive subset of PL/I, and (2) some of the translator operations were put into microcode.

6. Formal definition methodology

In addition to specific language developments, there have been two major technical contributions made by IBM employees in the formal definition of programming languages. I think it is fair to say that the first of these may be one of the most significant contributions made to the computing field in general and certainly to the area of programming languages in particular. The two contributions are BNF (Backus-Naur, Backus Normal Form) for describing syntax and VDL (Vienna Definition Language) for defining semantics.

- *Backus-Naur Form (BNF)*

In 1959, John Backus presented a paper [44] in which he introduced to the computing field from the field of linguistics the concept of a metalanguage and showed how it could be used to define the syntax of a programming language, namely ALGOL 58. When the committee to develop ALGOL 60 met and Peter Naur of Denmark was appointed editor of the ALGOL 60 report [45], Naur chose to define ALGOL 60 using this metalanguage, which has been referred to since as "BNF" meaning either Backus-Naur Form (to recognize the contribution of Peter Naur) or Backus Normal Form. The idea was based on the productions of Emil Post, and from a vantage point of twenty years later, it seems very simple. However, the impact has been so profound that it is almost impossible to describe its importance. In my own view, this idea has generated much of the theoretical work in programming languages, and it has the practical advantage of providing a formal way of defining language syntax. The Short Range Committee which developed COBOL was unaware of this work by Backus and developed their own metalanguage in the summer of 1959 (although they called it a notation, not a metalanguage) and used it to define COBOL. (See [31] for a fuller discussion of this issue.) Since 1959, the syntax of most languages has been defined using one or the other of these notations, or a mixture of the two. For example, the metalanguage used in the early major PL/I manual [46] was primarily based on the COBOL metalanguage but contained many elements from BNF. By now BNF has become almost a generic term and, in fact, is sometimes incorrectly used as the name for any metalanguage, whether it is the original one described by Backus or not.

Considerable controversy arose about the use of BNF to describe ALGOL 60 from those people not "in the ALGOL community." Some people immediately regarded it as being an enormous contribution to the rigor of language definition, whereas others found it very difficult to learn and understand. Some people have said that the adoption of ALGOL 60 was severely hampered by the use of BNF; my own view is that its use brought ALGOL into the world in a technically rigorous and understandable fashion and in so doing caused enormous advances in the computing field.

• *Vienna Definition Language (VDL)*

It was recognized that a formal definition of the semantics of a programming language was an even larger problem than formally defining its syntax. (The "syntax" of a language indicates what legal expressions *may be written*, whereas the "semantics" specifies the *meaning* of what has been written. Thus, syntax would indicate that the expression A + B is legal, whereas the semantics would indicate what it meant, since there are other possible meanings than normal addition.)

Apparently the earliest large scale attempt at a formal definition of semantics was the work undertaken in the IBM Vienna Laboratory under the direction of Heinz Zemanek in the mid-1960s. The original ideas of a concrete syntax and an abstract syntax were developed by McCarthy [47], Elgot and Robinson [48], and Landin [49]. VDL (Vienna Definition Language), reduced to its simplest possible terms, simulated an abstract machine and defined the language in terms of the effect that statements in the language would have on this arbitrary machine. For a description of this method, see [50] or the more rigorous [51]. See also [2].

VDL was originally developed for PL/I and then applied to other languages (*e.g.*, ALGOL). At about the same time, work was being done at the IBM Hursley Laboratories to produce a semi-formal definition of PL/I. Some of the flavor of this approach is given in [52]. The use of VDL to define the syntax and semantics of PL/I was the first application of such formalisms to a large and complex language. However, VDL eventually proved too difficult and impractical for compiler writers to use in their development work. Nevertheless, because of the obvious value of a formal semantics definition, the ANSI standard for PL/I [53] was based on the VDL and Hursley approaches; the definition mechanism for the standard PL/I is described in [54]. Various other techniques for formally specifying semantics have been developed (see [55]), but none has received wide practical usage. The major contribution of the VDL effort was to provide a technique for formally defining the syntax and semantics of a complex language and to define PL/I using that technique.

7. Other languages and language activities

IBM has also developed a number of languages which are used in specialized application areas. Examples include COURSEWRITER (for Computer Assisted Instruction), ECAP (for circuit design), and MPSX (for mathematical programming). The whole field of languages for specialized application areas is larger than most people realize—the number of such languages has consistently been about half of all the high level languages developed in the U.S. (see [56–58] for backup data). However, the contributions of IBM in this area do not seem to be nearly as significant as those of the languages cited in earlier sections.

In addition to specific language development and language definition methods, a significant amount of work in compiler optimization has been done. This is being covered in this issue in the paper by F. E. Allen [1].

Summary

IBM and its employees have developed and implemented four major high level languages, APL, FORTRAN, GPSS, and PL/I, each of which made significant contributions to the field. Two languages for formal mathematical computation (FORMAC and SCRATCHPAD) have made important contributions, and two on-line languages (QUIKTRAN and CPS) were innovative at the time they were developed, although they had no long-lasting influence. One business data processing language (Commercial Translator) had a major technical impact via its contributions to COBOL. Numerous languages for specialized application areas have also been developed by IBM. People from IBM contributed to the development of the significant languages developed by interorganizational committees, namely, ALGOL 58, ALGOL 60, and COBOL.

Significant work in the development of formal methods for specifying high level language syntax and semantics was done by IBM employees. The former has had a profound impact on language development, and the latter had some impact on language definition technology.

It should be clear from all of the foregoing material that each of these IBM languages was developed independently of the others. Each built on the technical knowledge available to the developers at the time from within and outside IBM, and some of the individual languages had a progression of named improved versions—specifically APL, FORTRAN, GPSS, and FORMAC. However, the collective set of languages do not form a cohesive technology, because they deal with differing problems in different ways.

Acknowledgments

I would like to thank the following people, each of whom read at least two versions of the full paper: John Backus, Bernard Galler, Charles Gold, and John A. N. Lee. They made valuable suggestions for improvement, and any remaining deficiencies are mine.

References

1. F. E. Allen, "The History of Language Processor Technology in IBM," *IBM J. Res. Develop.* **25,** 535-548 (1981, this issue).
2. P. Lucas, "Formal Semantics of Programming Languages: VDL," *IBM J. Res. Develop.* **25,** 549-561 (1981, this issue).
3. J. E. Sammet, *Programming Languages: History and Fundamentals*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1969.
4. J. Weizenbaum, "Symmetric List Processor," *Commun. ACM* **6,** 524-544 (1963).
5. J. E. Sammet, "Programming Languages: History and Future," *Commun. ACM* **15,** 601-610 (1972).
6. D. Knuth and L. Trabb Pardo, "The Early Development of Programming Languages," *Encyclopedia of Science and Technology*, J. Belzer, A. G. Holzman, and A. Kent, Eds., Vol. 7, Marcel Dekker, Inc., New York, 1977, pp. 419-493. Also in *A History of Computing in the Twentieth Century*, N. Metropolis *et al.*, Eds., Academic Press, Inc., New York, 1980, pp. 197-274.
7. J. W. Backus, "The History of FORTRAN I, II, and III," *History of Programming Languages, ACM Monograph Series*, Academic Press, Inc., New York, 1981, pp. 25-45. Also in *Annals of the History of Computing* **1,** 21-37 (1979).
8. D. T. Ross, "Origins of the APT Language for Automatically Programmed Tools," *History of Programming Languages, ACM Monograph Series*, Academic Press, Inc., New York, 1981, pp. 279-338.
9. A. J. Perlis and K. Samelson, "Preliminary Report—International Algebraic Language," *Commun. ACM* **1,** 8-22 (1958).
10. *History of Programming Languages, ACM Monograph Series*, R. L. Wexelblat, Ed., Academic Press, Inc., New York, 1981.
11. N. Wirth, "PL360, A Programming Language for the 360 Computers," *J. ACM* **15,** 37-74 (1968).
12. "Preliminary ADA Reference Manual (Part A)" and "Rationale for the Design of the ADA Programming Language (Part B)," *ACM SIGPLAN Notices* **14,** No. 6 (1979).
13. *Reference Manual for the ADA Programming Language*, U.S. Department of Defense, Defense Advanced Research Projects Agency, Washington, DC, July 1980.
14. J. W. Backus, "Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs," *Commun. ACM* **21,** 613-641 (1978).
15. J. W. Backus, "The IBM 701 Speedcoding System," *J. ACM* **1,** 4-6 (1954).
16. "Preprints, ACM SIGPLAN History of Programming Languages Conference," *ACM SIGPLAN Notices* **13,** No. 8 (1978). (See also [10].)
17. J. W. Backus *et al.*, "The FORTRAN Automatic Coding System," *AFIPS Conf. Proc., Western Jt. Comput. Conf.* **11,** 188-198 (1957). (Also in *Programming Systems and Languages*, S. Rosen, Ed., McGraw-Hill Book Co., Inc., New York, 1967.)
18. *American National Standard Basic FORTRAN*, ANS X3.9-1966, American National Standards Institute, New York, 1966.
19. *American National Standard FORTRAN*, ANS X3.10-1966, American National Standards Institute, New York, 1966.
20. G. Gordon, "The Development of the General Purpose Simulation System (GPSS)," *History of Programming Languages, ACM Monograph Series*, Academic Press, Inc., New York, 1981, pp. 403-426.
21. H. M. Markowitz, B. Hausner, and H. W. Karr, *SIMSCRIPT—A Simulation Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1963.
22. O.-J. Dahl and K. Nygaard, "SIMULA—An ALGOL-Based Simulation Language," *Commun. ACM* **9,** 671-678 (1966).
23. K. E. Iverson, *A Programming Language*, John Wiley & Sons, Inc., New York, 1962.
24. K. E. Iverson, "A Programming Language," *AFIPS Conf. Proc., Spring Jt. Comput. Conf.* **21,** 345-351 (1962).
25. A. D. Falkoff and K. E. Iverson, "The Evolution of APL," *History of Programming Languages, ACM Monograph Series*, Academic Press, Inc., New York, 1981, pp. 661-674.
26. A. D. Falkoff, K. E. Iverson, and E. H. Sussenguth, "A Formal Description of System/360," *IBM Syst. J.* **3,** 198-262 (1964).
27. H. Hellerman, "Experimental Personalized Array Translator System," *Commun. ACM* **7,** 433-438 (1964).
28. J. Schwartz, "The Development of JOVIAL," *History of Programming Languages, ACM Monograph Series*, Academic Press, Inc., New York, 1981, pp. 203-214.
29. G. Radin, "The Early History of PL/I," *History of Programming Languages, ACM Monograph Series*, Academic Press, Inc., New York, 1981, pp. 551-575.
30. *General Information Manual: IBM Commercial Translator*, Order No. F28-8043, IBM Data Processing Division, White Plains, NY (1960).
31. J. E. Sammet, "The Early History of COBOL," *History of Programming Languages, ACM Monograph Series*, Academic Press, Inc., New York, 1981, pp. 199-243.
32. R. W. Bemer, "A View of the History of COBOL," *Honeywell Computer J.* **5,** 130-135 (1971).
33. E. R. Bond *et al.*, "FORMAC—An Experimental FORmula MAnipulation Compiler," *Proc. 19th National Conference*, Association for Computing Machinery, 1964, pp. K2.1-1-K2.1-11.
34. J. E. Sammet and E. Bond, "Introduction to FORMAC," *IEEE Trans. Electron. Computers* **EC-13,** 386-394 (1964).
35. M. D. Bernick, E. D. Callender, and J. R. Sanford, "ALGY—An Algebraic Manipulation Program," *AFIPS Conf. Proc., Western Jt. Comput. Conf.* **19,** 389-392 (1961).
36. *PL/I-FORMAC Interpreter*, IBM Corporation, Contributed Program Library #360D 03. 3.004, 1967, available through IBM branch offices.
37. A. J. Perlis and R. Iturriaga, "An Extension to ALGOL for Manipulating Formulae," *Commun. ACM* **7,** 127-130 (1964).
38. A. C. Hearn, "REDUCE 2, A System and Language for Algebraic Manipulation," *Proc. Second Symposium on Symbolic and Algebraic Manipulation*, Association for Computing Machinery, New York, March 1971.
39. J. Moses, "MACSYMA—The Fifth Year," *Proc. Eurosam Conf., ACM SIGSAM Bull.* **8,** 105-110 (1974).
40. R. D. Jenks, "The SCRATCHPAD Language," *Proceedings of the Symposium on Very High Level Languages, ACM SIGPLAN Notices* **9,** 101-111 (1974).
41. T. M. Dunn and J. H. Morrissey, "Remote Computing: An Experimental System, Part 1: External Specifications," *AFIPS Conf. Proc., Spring Jt. Comput. Conf.* **25,** 413-423 (1964).
42. J. C. Shaw, "JOSS: A Designer's View of an Experimental On-Line Computing System," *AFIPS Conf. Proc., Fall Jt. Comput. Conf.* **26,** Part 1, 455-464 (1964).
43. *Conversational Programming System*, IBM Corporation, Contributed Program Library #360D 03. 4. 016, 1967, available through IBM branch offices.
44. J. W. Backus, "The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-

GAMM Conference," *Proc. International Conference on Information Processing, UNESCO, Paris 1959*, Butterworths, London, 1960, pp. 125–132.
45. "Report on the Algorithmic Language ALGOL 60," *Commun. ACM* **3**, P. Naur, Ed., 299–314 (1960).
46. *IBM System/360 Operating System: PL/I Language Specifications*, Order No. C28-6571, IBM Data Processing Division, White Plains, NY (1966).
47. J. McCarthy, "A Formal Description of a Subset of ALGOL," *Formal Language Description Languages*, T. B. Steel, Jr., Ed., North-Holland Publishing Co., Amsterdam, The Netherlands, 1963, pp. 1–12.
48. C. C. Elgot and A. Robinson, "Random access stored-program machines. An approach to programming languages," *J. ACM* **11**, 365–399 (1964).
49. P. J. Landin, "A Correspondence between ALGOL 60 and Church's Lambda-Notation, Part I," *Commun. ACM* **8**, 89–101 (1965); "Part 2," *Commun. ACM* **8**, 158–165 (1965).
50. E. J. Neuhold, "The Formal Description of Programming Languages," *IBM Syst. J.* **10**, 86–112 (1971).
51. P. Lucas and K. Walk, "On the Formal Description of PL/I," *Annual Review in Automatic Programming* **6**, Part 3, 105–182 (1969).
52. D. Beech, "A Structural View of PL/I," *Computing Surv.* **2**, 33–64 (1970).
53. *American National Standard PL/I*, ANS X3.53–1976, American National Standards Institute, New York, 1976.
54. M. Marcotty and F. G. Sayward, "The Definition Mechanism for Standard PL/I," *IEEE Trans. Software Engineering* **SE-3**, 416–450 (1977).
55. M. Marcotty, H. F. Ledgard, and G. V. Bochmann, "A Sampler of Formal Definitions," *Computing Surv.* **8**, 191–276 (1976).
56. J. E. Sammet, "Roster of Programming Languages for 1976–1977," *ACM SIGPLAN Notices* **13**, 56–85 (1978).
57. J. E. Sammet, "Roster of Programming Languages for 1974–1975," *Commun. ACM* **19**, 655–669 (1976).
58. J. E. Sammet, "Roster of Programming Languages for 1973," *ACM Computing Reviews* **15**, 147–160 (1974).

Received March 18, 1980; revised August 5, 1980

The author is located at the IBM Federal Systems Division Headquarters, 6600 Rockledge Drive, Bethesda, Maryland 20034.