Preface

The first APL workspace became available at IBM on November 27, 1966, making 1991 the twenty-fifth anniversary of APL. The *IBM Systems Journal* joins in the anniversary celebration by presenting 12 papers and one essay covering APL's history, implementation, and applications. We are indebted to R. P. Polivka of the IBM Data Systems Division in Poughkeepsie, New York, for his extensive contributions to the planning and development of this issue, including the solicitation of numerous papers and suggestions for referees. We also commend and thank J. McGrew of the IBM Application Solutions Division in Kingston, New York, for his considerable efforts in ensuring the proper appearance of APL throughout the issue.

The term APL is attributed to a suggestion made by A. Falkoff when a name was needed for the programming language that was to be built from the ideas in K. E. Iverson's 1962 book entitled *A Programming Language.* Today, after many generations of the language and implementations, APL2 is IBM's strategic interactive programming language, serving IBM and its customers in a wide range of applications and across a broad spectrum of implementations.

The papers and essay begin with a history and an introduction to APL, then progress through a set of papers on APL systems and a set on applications, ending with an exploration of the importance of symbols and a look forward from Iverson's unique vantage point.

The first paper, by Falkoff, traces the genealogy of the IBM family of APL systems. His perspective stems from his place as one of the first, foremost, and current advocates of APL. He describes the interplay between language constructs, implementation methods, and evolution for the breadth of IBM APL systems.

Brown and Crowder introduce the essential features of APL2, IBM's current APL offering. The authors show, through examples, the use of arrays and functions, and show how the arrays (APL's data structures) control the flow of execution of a program.

Programming languages exist in close association with the language environment designed for their use. In the first paper in the set on APL systems, Wheatley discusses the issue of connectivity among APL2, its environment, and other programming languages. From the point of view of APL2, there are three major facilities that permit communication beyond the APL workspace: system variables and functions, shared variables, and name association. Each is presented, along with the historical setting.

Most APL systems have depended on the storage management technique known as garbage collection. This strategy has become less effective as virtual and real storage have grown dramatically. APL2 Version 2 takes a new approach: a quickcell scheme for small data items, a variation of the buddy system, and a bit map scheme for large blocks of storage. Trimble shows how this provides a better means of storage management.

Jensen and Beaty present the results and the experience of building an X Window System** interface for APL2 (called APL2/X). They also present a C interface for all IBM APL2 systems (called APL2-to-C), which was created in order to support the APL2/X effort. Following an overview of the X Window System and the interface design criteria, the authors detail the APL2/X and APL2-to-C interfaces, concluding with a sample program.

The APL IL Interpreter Generator has contributed to the successful and rapid proliferation of APL systems. Alfonseca, Selby, and Wilks describe IL and the use of it to generate new APL interpreters. To date IL has been used to create nine IBM products, with as little as 13 person-weeks of effort.

APL, with its array orientation, would appear to be a natural candidate for use in parallel expression

and computing. Willhoft analyzes each APL construct for its potential parallelism. Through argument and studies of examples, he shows that APL has a high degree of parallelism, both in its constructs and in its common uses. The types of parallelism examined are data, algorithm, data flow, and task. Suggestions are made for improving the language and implementations to further increase parallelism.

Turning to papers on APL applications, Jordan and Friis describe the application of APL2 to music, both for building music software and as a musical notation. Examples are given that show how frequency, pitch, tempo, loudness, chords, and passages can be represented in APL. The authors claim that the iconic nature of APL2 is well suited to musical expression.

Verifying that an implementation of a new architecture indeed matches its functional specification usually involves the use of test generators. The IBM RISC System/6000* was tested in that way by the random test program generator (RTPG), built in APL for that purpose. Aharon, Bar-David, Dorfman, Gofman, Leibowitz, and Schwartzburd present the concepts and implementation of RTPG. They discuss the advantages of using an interactive language in test situations, where many changes are made with a need for rapid test creation, and the suitability of using APL to represent computer architectures.

Thomson describes the efforts of a group of academic and industrial statisticians in the United Kingdom, with the support of the British APL Association, to build on the popularity of APL for statistics and on its ability to express specifications of mathematical functions. They are creating the APL Statistics Library (ASL), which will contain standardized APL specifications of statistical functions. The author describes the philosophy of ASL code and documentation and illustrates how it provides a medium for algorithmic discussion among statisticians. The paper concludes with a demonstration of how advanced functions can be readily and reliably built using standardized ones from ASL.

Alfonseca summarizes his work on the application of APL to the fields of logic programming and artificial intelligence, neural networks, and object-oriented programming and hypertext. The paper argues that APL is applicable to a broad range of modern programming challenges.

Leaving the papers on APL applications, McIntyre describes APL from the perspective of symbolic languages throughout history, including our number system, many ancient written languages, and much of mathematics. He finds that APL, a symbolic language, is an "intellectual triumph." This paper grew out of an invited talk at APL83 in Washington, D.C., where the author presented a detailed history of the evolution of symbols.

The issue closes with an essay by Iverson that traces the development of his rationale for the APL notation, beginning with his original motive: creation of a tool for writing and teaching about data processing. Much of the essay is devoted to a discussion of the J language, which has evolved from his earlier work with APL. Iverson continues to pursue language styles and constructs that would be accessible to wide audiences.

There have been two changes to the form of the *Journal*. The first is the use of asterisks to signify a trademark or registered trademark. The appropriate designation for each term is shown just before the list of references in each paper. The second is the inclusion of the date on which the paper was accepted for publication by the editors (following editorial and peer review, and author revisions) and after which content would not have been materially changed. That date is shown just after the list of references in each paper.

The next issue of the *Journal* will contain several papers on the Optimization Subroutine Library (OSL) and others on such subjects as a portable model for the design of device drivers in OS/2*.

Gene F. Hoffnagle
Editor

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Massachusetts Institute of Technology.