Preface

Programming languages have been the media for communicating our requirements to computers throughout the history of digital data processing. The basic goals for these languages, and for the processors that produce the corresponding executable programs, have changed surprisingly little over the years.

On the user's side of the user-computer interface, the fundamental goal for the past quarter century has been to make powerful computational facilities accessible to computer users, including those whose primary interest is not data processing. At one extreme, machine language coding is too arduous to be practical. At the other extreme, the inherent ambiguities of natural language make precise expression of complex algorithms exceedingly difficult. Between the extremes, language designers are confronted with a variety of desires and requirements. High-level languages are desirable for professional programmers to increase their productivity as well as the readability and quality of their programs, but these users continue to express concern about losing direct control over machine facilities. Easy-to-learn languages may be attractive to the beginning programmer but may be too limiting for large, complex programs. And the user who has acquired skill in one language may resist learning a new language that may be much more suitable for his applications.

On the computer's side of the interface, program execution speed was always a significant concern. Indeed, a feared sacrifice in speed may have accounted for some early resistance to high-level languages. Although compilers were demonstrated to incur only limited penalties in execution speed, efforts have continued throughout the history of high-level languages to further reduce that cost. Despite dramatic increases in processor speed, optimization of execution time continues to be important because of the existence of huge programs and because of real-time applications. Equally important, the trend toward higher-level languages increases the burden on compilers to produce efficient code. Space optimization also continues to be significant despite the availability of cheap, large main memories. One reason is the limited space available in the small processors used, for example, in networks. Space is important also because of its relationship to speed; very large programs overflow main storage and are forced onto slower secondary storage media, reducing overall execution speed.

The papers in this issue of the *IBM Journal of Research and Development* sample some current results in the elusive search for an appropriate avenue of communication between creative, imaginative, but dissimilar human beings and tremendously fast but inflexible computer hardware. Four of the papers are on the language side of the interface and six on the processor side.

Most but not all of the emphasis in the papers on the processor side is on optimization of the object programs produced by compilers. The first paper, by Scarborough and Kolsky, reports some impressive improvements in the output of an already good optimizing compiler. The new optimization techniques resulted not from theoretical considerations but from careful examination of the output of the existing compiler.

The next paper, by Boyle *et al.*, describes the optimization techniques used in a compiler designed to produce code for several different machines. The optimization methods are intended primarily to save space. In successive versions of this processor, additional optimizations were incorporated, providing some indication of the effectiveness of specific optimization methods.

The paper by Marks also reports new results in space optimization. In this case the most significant results are achieved through use of a software interpreter created dynamically for an individual program.

The final paper dealing exclusively with optimization is by Cocke and Markstein. This Communication reports a

strength reduction optimization for division and modulo operations as applied in accessing arrays in a multilevel store.

An experimental compiler is described in the paper by Allen *et al.* The objective of this work is to produce a compiler skeleton that can be tailored to a wide class of programming languages, as well as to different target machines. The compilers can produce highly optimized output if desired. To achieve this flexibility, a fundamentally different approach to compiler design has been devised.

Extensions to PASCAL, previously reported by Lafuente and Gries, were intended to facilitate the definition of user-computer interactions to take place via an alphanumeric display terminal. Implicit in that approach was the requirement of processing nonprocedurally specified rules governing complex interdependencies in these interactions. The current paper describes techniques to cope with this problem, a strategy which is applicable to a variety of interactive systems.

The first paper in the language group, by Denil, describes a system designed specifically for small businessmen. The user is provided with application programs, which he can tailor, if needed, to his particular requirements. Essential information about the program is communicated to the user via a display screen, and the user is guided by the system in modifying his program.

Another specialized language is that described by Sauer *et al.*; its purpose is to produce queuing network models. The current system provides significantly extended capabilities over an earlier one by the same researchers.

Becerril *et al.* introduce the concept of g-chains. One result is that they have been able to extend and formalize work originally reported by Urschler on the automatic structuring of programs.

Finally, Lomet describes a data definition facility that enables users to define their data in a way that is consistent with a value-oriented storage model used for the primitive data of the base language. This value-oriented storage model is a refinement of the model that Lomet believes intuitively underlies such familiar languages as FORTRAN and PL/I.

The order chosen for presenting the papers was influenced somewhat by the amount of specialized knowledge required of the reader. Thus, the paper by Scarborough and Kolsky provides entry into the following papers related to optimization. The final two contributions in the issue are somewhat more theoretical.

S. S. Husson
Editor