Issues in programming language design—
An overview*

by ANTHONY I. WASSERMAN
University of California at San Francisco**
San Francisco, California

The past few years have witnessed an increased understanding of the relationship between programming languages and problem solving. Programming is now understood to be a rather difficult task which requires the simultaneous application of principles, skills, and art.1,2,3 Computer scientists have recognized that the features of a programming language can have a significant effect upon the ease with which reliable programs can be developed. It has also been observed that certain languages and language features are particularly well suited for the use of systematic programming techniques, while others hinder or discourage such discipline.4,5,6 Of course, it is possible to write well-structured, clearly organized programs in any programming language, but such programs have often been the exception rather than the rule.

As a result of this work, there have been numerous developments in the general area of programming languages. Among these are the following:

(1) a significant number of new programming languages have been designed and/or implemented,7 with several developed principally to promote proper programming practices;8,9,10
(2) the general features of existing and proposed languages have been analyzed in an attempt to identify desirable characteristics of programming languages;11,12,13,14 strong criticism has been directed at those languages which do not appear to contain the requisite features for the systematic development of reliable software;15,16
(3) preprocessors have been implemented for several programming languages, thereby allowing programmers to use "structured programming" techniques;
(4) direct modifications have been designed and/or implemented for several programming languages, in order to enhance their suitability for program development;
(5) general design criteria for programming languages have been advanced, with attention focused on the need for a language to have a sound theoretical basis;17,18

Although the design goals for the individual language modifications and language developments vary considerably, there are a number of common objectives which can be identified. First, the value of linear flow of control was recognized, primarily for its value in program debugging and verification, and powerful control structures were proposed and added to promote such a flow.19,20,21,22 Second, the value of abstraction was recognized as a way to develop a representation of information which is more closely related to the application being programmed than exists in any programming language with a fixed number of data types.23,24 Third, the scope and binding of variables was studied as a technique which simplifies program verification and which reduces programming errors caused by side effects.25,26 Fourth, it was recognized that a language must be comprehensible, so that programs written in the language can be read and maintained. Fifth, efforts were made to limit the size of languages, in order to make them easier to implement and to make it possible for a programmer to thoroughly understand the tool. Finally, modular program structures were observed to make an important contribution to the production of large software systems.

These design objectives are reflected in a variety of decisions which are made in designing programming languages. Since the universe of design objectives is somewhat self-contradictory, as is immediately evident from a comparative analysis of languages, the language designer must consider the tradeoffs among the various possible features for a language, and give more emphasis to some of these objectives than others. It is agreed, however, that the language designer must have a thorough understanding of the goals of the language prior to commencing a specification of the syntax and semantics of the language.

Although there are a large number of closely related issues involved in the design of a language, much of the current work in language design is focused on three areas: language extensibility, data types and abstraction, and control structures.

Language extensibility refers to the ability of the programmer to modify the language being used, with the intent of extending the power of the language.25,27,28 A relatively small "base" language is defined, along with capabilities to add features such as new data types, new

* This work was supported in part by the Commonwealth Fund.
** Graduate Program in Medical Information Science. The author is also affiliated with the Computer Science Division, University of California at Berkeley.

297
operators, new syntax, and new control structures in order to enable the program to more closely correspond to the problem domain. The derived language or "task language" which thereby results can allow programs to be written in such a way that they are comprehensible to almost anyone familiar with the application area of the program. Persons working on the development of extensible languages recognize the computation of a high-level language which could evolve gracefully via packages of definitions. The availability of such packages for particular task areas could then greatly increase programmer productivity.

Data types and abstraction refers to the number of predefined data types which are available in a language, the means available for combining the primitive types to create more complex types, and the way in which new abstractions may be introduced into the language. The notion of an abstract data type has been advanced to define a class of abstract objects which is completely characterized by the operations available on those objects. The means by which a variable takes on a given type and has a value assigned to it are extremely important issues in language design.

Control structures are the means by which the order of execution of statements in a program is determined. While it has been formally shown that only sequential control, a conditional statement, and iterative control are necessary to describe any computation, it is also understood that restricted use of the go to statement may result in greater program clarity. Much of the work in control structures has dealt with the definition of mechanisms for conditional testing and iteration which reduce the need for the go to statement, and which produce dynamic program behavior closely resembling the static program structure. Because of the need to permit communication among tasks, various control structures have been proposed which permit coroutines, parallel processing, synchronization, and monitoring. A wide variety of proposals for improving control structures of existing and new languages have been suggested, including forms of nondeterministic control, and the relative power and merits of these alternatives have been discussed extensively.

Although these three issues are at the heart of much of the work on programming language design, there are a number of other issues which have received attention. First, the rapid growth of interactive systems and their use by non-programmers has identified a need for string processing facilities and exception handling capabilities. Second, the development of conversational programs for access to large data bases has focused attention on the need for capabilities in the area of data management and the need for more powerful input/output facilities. Third, there are standardization efforts in progress for a number of programming languages in order to improve program transferability. Fourth, research into program proving and verification has led to additional proposals for programming languages. Finally, there is also a need to simplify the task of program documentation, so that one can easily understand how a program works.

The language designer must then be able to synthesize all of these various concepts in such a way as to produce a language which is defined in a uniform way, which has a logical relationship between the syntax and the semantics, which allows an efficiently executing program to be produced, and which permits programmers to conceptualize a solution to a problem in a straightforward manner. The interrelationships among these design criteria are extremely complex, and it appears that it will be some time before a language emerges which can satisfy all of the needs of a broad class of programming applications.

Beyond that point, there is a number of political and economic issues which will affect the eventual acceptance of such a language. The primary determinants appear to be the support given to the language through implementation by major vendors of computer hardware and software, and the ease by which programmers and programming management can be attracted away from their present language and trained in the new language. Until then, most programmers will be left to work with tools which are now recognized to be somewhat inadequate for the effective solution of programming problems.

In conclusion, then, several key questions can be raised concerning the design of programming languages. How do we develop a programming mechanism which can accurately mirror logical thinking? Furthermore, how do we develop a tool which is suitable for stepwise refinement of the problem from an abstract form to its "elaborated" form in a "natural" way? Last, how then does such a language get introduced and accepted by the general programming community so that it raises the quality of software production? These are the main questions which underlie present research and development in the field of programming languages.

REFERENCES

Issues in Programming Language Design—An Overview


