FEUER82
Feuer, A.R. and Gehani, N.H.; A Comparison of the Programming Languages C and Pascal; Computing Surveys 14, 1 (March 1982) pp. 73-92.

With any computer language, the design philosophy tends to affect the potential applications for which that language is suitable. This paper looks at two popular languages, C and Pascal, in an attempt to show how their differing histories and design philosophies affect their suitability for a variety of programming domains.

Historically, C was derived from B and BCPL, both of which are typeless languages typically used for implementing operating systems and language processors. A major advance of C was the introduction of typed variables into the language, while still allowing an efficient interface to the computer hardware.

Pascal, on the other hand, was influenced by ALGOL 60 and ALGOL W, and was designed as a direct response to the size and complexity of ALGOL 68. The major intents of Pascal led to the development of a high-level machine, from which it is very difficult to escape to the underlying hardware.

The design goals of the two languages were considerably different. Restrictions were built into Pascal to help in the development of reliable programs by enforcing a disciplined structure. By strongly enforcing these restrictions, Pascal is intended to help the programmer detect programming errors, as well as make it difficult to access memory locations outside of the program's data area. The permissiveness of C, however, was intended to allow a wide range of applications. The basic language was made small by omitting such features as input/output and string processing, since C was to be sufficiently flexible that these facilities, and others like them, could be built as required.

This difference in design philosophies is reflected in the formal definitions of the two languages. A precise specification of the Pascal syntax was available from the start, and an axiomatic definition for most of the semantics was not long in appearing. By contrast, there has yet to be published a complete definition of the C syntax, and only a denotational definition for the semantics of most of C (considered to be of less use in proving program correctness) has appeared.

The authors describe most of the features of C and Pascal, first by means of an illustrative example (a binary search), and the on a feature-by-feature basis. They group the features into five main categories (Data Types, Expressions, Statements, Routines and Program Structure, and Input/Output), and conclude the exposition of the features within each category with a discussion of the pros and cons of each. This section assumes that the reader is familiar with programming language concepts in general, but not necessarily with either C or Pascal, and is very well written.

The final section takes four different programming domains (Business Data Processing, Scientific Programming, Programming for Operating Systems, and Programming for System Utilities), and attempts to determine the suitability of C and Pascal to each of these domains by first defining what language features are useful for each of the domains, and then examining which of those features (if any) are present in each of the two languages.

The overall conclusion of the authors is that programs in Pascal tend to be more reliable than those written in C, primarily because of the much stronger typing present in Pascal. Additionally, because Pascal supports a richer set of data types, Pascal programs tend to be more readable and portable than C programs. However, because of the flexibility and lack of restrictions in C, it can be used in a larger variety of programming domains than can Pascal. They are, of course, careful to note that "just because a language can be used for a particular application does not mean that it should be".

All in all, this paper seems to provide an excellent example of how the design goals for a language must be kept in mind when evaluating that language.