PASCAL for Operating Software?
A Critical Examination

by C. E. Prael
2106 Monterey Ave.
Menlo Park, CA 94025

Abstract

The general status of PASCAL in the commercial field is described. Various characteristics of the language are examined critically in the context of operating software and utility program implementations.

Introduction

The programming language PASCAL has been the subject of considerable attention in the last few years. It should be noted that for a substantial majority of computer science students, PASCAL is the primary or only high level language with which there is a working familiarity. The language appears to be taught very extensively and to be used very heavily in student's project implementations. With such a large primary experience background, it is not surprising to find that PASCAL is so often preferred by younger project staff as an implementation language.

PASCAL has many attractions as a programming language. Its descent from ALGOL has resulted in sound structuring of programs and a useful array of active constructs. The information structure concepts were quite novel when PASCAL was designed. Further, these concepts are singularly attractive when viewed as 'Ding an Sich'.

In a commercially oriented operating software environment, the need is for a tool to facilitate the development of products. The criteria one should apply are: 1) The language should be the simplest that will do the job; 2) It should implement readily and impose low maintenance requirements; 3) It should facilitate the design and manipulation of the information structures used in developing operating software. As will be seen below, PASCAL fails to meet these criteria to a significant extent. But it should be noted that many, if not most, of the faults found with the language are a matter of the application of PASCAL to situations for which it was clearly not intended.

Information Primitives

The information primitives are the first elements one usually considers in a language. In PASCAL one is immediately confronted with a number of deficiencies.

Integral Data

Typically, the operating software implementor wants to be able to specify and use integral data elements in a number of 'word' sizes. Space is always at a premium, no matter how much memory is available, and the ability to pack various sized integers and manipulate them all as integers is virtually mandatory. PASCAL effectively limits one to the native memory word size.

Real Data

Although the real, or floating point, data type is occasionally encountered in operating software, the frequency and nature of the occurrence are such as to make the type quite useless. The real data type is, thus, excess baggage which merely complicates the language implementation and maintenance without sufficient return.

Char Data

Typically, the designer prefers to establish a single format for strings and apply it uniformly to all languages and processes that a system supports. The convenient time to do this is in the implementation of the implementation language. By restricting the textual data type to a string element, PASCAL forces the string format definition to be specified in the definition of each program module in a system. At best this is an inconvenience. But inconveniences of this type usually translate into frequently recurring bugs or incompatible system components.

'rogram Declaration

In the typical operating software situation one is dealing with fairly large program modules. These are much more manageable if they are broken down into a number of submodules which can be translated, linked and debugged separately. In fact, subdivided implementation is one of the most basic tools of the operating software implementor. PASCAL's total absence of any provision for subdivided implementation is its greatest single impediment to use in the operating software environment. While PASCAL does provide for declared subdivision with the 'procedure' and 'function' declarations, it tends to be extremely wasteful to have to edit and translate 1000 or more lines of program to correct a problem in a 20 or 30 line subprogram. The result is that PASCAL can only be reasonably used to implement programs so small that they can be of little use to the software engineer.

There are two problems with the manner in which PASCAL implements 'procedures' and 'functions'. First, PASCAL requires that one embed the code of a subprogram in the calling program. This requirement is the main source of the problem mentioned in the preceding paragraph. It also tends to reduce clarity in the source code, even if sound design methodology is followed. The second problem is in differentiating between a 'procedure' subroutine and a 'function' subroutine in the declaration of the subroutine. This failing is common to most higher level languages. The only difference between a 'procedure' and a 'function' is and should be

in the invocation which is not intrinsic to the procedure. This view of subroutines may seem quite radical, but the diferentiation is primarily a convenience to the implementor and needlessly restrictive to the user of the language.

There are other less substantial out undesirable aspects to the declaration division of PASCAL. The only utility to be found in the requirement that all line labels be explicitly declared is to raise the user's awareness of his having broken a rule of structured programming. This is not too useful to the experienced implementor. The requirement to declare and label all constants used in a program is aesthetically and practically unsound and restrictive. Certainly, one wants to be able to label constants, since this may enhance program clarity. But there are also many circumstances in which using a label would detract from program clarity. This feature appears to be an attempt to substitute for good judgment on the part of the user. The subject of information structures is discussed separately.

Action Declaration

The operator precedence defined for expressions is not well designed. While one rarely finds a sufficiently sophisticated treatment of operator precedence, PASCAL's lack of differentiation between arithmetic, relational, and logical operator types is an extreme case. One would strongly prefer to see the three operator types separated by precedence. This would make the use of parentheses optional in most mixed mode expressions. In PASCAL parentheses are required in any mixed mode expression if it is to be parsed correctly. Since mixed mode expressions are particularly common in operating software, the feature becomes a cause of needless verbosity.

The organizational statement constructs, (WHILE, CASE, etc.) are conventional and well proven. The provision of the GOTO construct requires some consideration. In the conventions of pure structured programming, the GOTO is, of course, not acceptable. However, the real world requires some means of departing from the normal processing flow when an exception condition is encountered. The proper question thus is the suitability of a simple GOTO for this purpose. And the answer is no. At a minimum, one should require that the language provide explicit means for passing a description of the exception that caused the departure from normal flow. PASCAL's GOTO clearly fails this requirement.

Variable Typing

The item typing features are generally considered to be a principle selling point of PASCAL relative to other descendants of ALGOL. The arguement advanced is that this ability to classify operands is a powerful tool in reducing misreferencing in active statements. Yet many implementations fail to provide the type checking without which the argument is unjustified. Certainly, one can find a substantial aesthetic pleasure in these features (which

happens less often than it should), but their utility is very limited. This violates the simplicity criterion.

Information Structure Declarations

The method of declaring an array is refreshingly novel. It appears to have much to recommend it as a tool for teaching modern algebra. But for every day use, the complexity does not seem to be an improvement over the method first seen in FORTRAN, which has the twin advantages of simplicity and sufficiency. (An atractive alternative is to treat arrays as a case of a record declaration, but this too violates the simplicity requirement in use.)

The record is the most useful and frequently used information structure concept in the operating software field. The design of record formats and their relationships usually rules the quality of processes and the modification capacities of systems. The declaration of records is, thus, a question of particular concern. The method of declaring records is a natural extension of the typing features, and is quite reasonable for the environment established by the features. However, simpler and more flexible concepts can be found by eliminating the typing features. The apparent inability of PASCAL to treat a record as a field (or fields) of a record is also a handicap.

The set types feature is a nice extension of the item typing features. The philosophy behind the feature is, however subject to the same objections expressed on the whole body of typing features (see 5, above). The same criticism applies to the provision of the pointer type.

In general, on the question of variable management, one would strongly prefer to see the philosophy found in BLISS in an operating software language. While the BLISS philosophy may require that the programmer pay a little more attention to things, its greater power and flexibility is well worth the trouble. This preference seems even more desirable in the teaching application for which PASCAL was originally intended. The purpose of a higher level language should not be to insulate the implementor from his machine, but to facilitate access to and enable full use of the machine, while improving portability of the software.

Files in PASCAL

File access methods are an area in which the requirements in the operating software implementor are directly opposed to the requirements of the rest of the programming world. Most will probably find the file access methods of PASCAL too rudimentary for their needs. The implementor of operating software prefers to leave file access methods completely out of a language. There are significant reasons for this. First, one does not want one language to impose a single structure on a component which will probably have to support the requirements of a number of languages and facilities. Second, the file access subsystem is usually one

of the things one is implementing, so the implementation language should not impose the implementation of facilities which will not be used. This requirement tends more than any other to set operating software specialized languages apart from those used by the rest of the computing field.

Summary

As was noted in the introduction, PASCAL was never intended to be applied to the implementation of operating software. Since the intended area of application is quite distant from that of operating software, it is not surprising to find that the langauge is significantly deficient in this use. The alterations which would make PASCAL acceptable in this application are so extensive that the result would not be recognizable as a dialect of PASCAL. While this critique deals only with the original PASCAL, the various extensions which have been implemented generally fail, with some exceptions, to correct the problems discussed here.

Reference: K. Jensen and N. Wirth, PASCAL User Manual and Report, Springer-Verlag, New York, 1978

Christopher E. Prael
2106 Monterey Ave.
Menlo Park, CA 94025