More Comments on the Programming Language Pascal

O. Lecarme and P. Desjardins

Received June 17, 1974

Summary. A. N. Habermann recently published some “Critical comments on the programming language Pascal”. His reproaches are principally that numerous constructs are ill-defined, that there is “confusion” amongst ranges, types and structures, and that the \texttt{goto} statement should have been abolished. The present reply successively deals with points that are clearly refutable, those which are debatable and those which constitute valid criticism. Its principal aim is to encourage the reader to form his own opinion.

1. Introduction

We read with much interest and some miscontentment the paper “Critical comments on the programming language Pascal” by A. N. Habermann [6]. The interest was instigated by our current involvement with this language, and the miscontentment occasioned by the (sometimes unduly) strong attacks the paper makes on a language we like. We like it because, for three years now, we have been using it intensively, with complete success, as a support for teaching introductory and advanced courses in computer science (a first course on programming, and courses on data structures, compilers and operating systems design), and as an implementation tool for a compiler writing system [19] and for another Pascal compiler [2].

The aim of the present reply is to correct the unfavourable impression that readers, without any knowledge of Pascal, could get from Habermann’s criticisms. We will not follow his argumentation point by point, but rather classify the subject matter into four parts: clearly refutable points, points which are at least debatable, valid criticisms and finally misunderstandings and minor errors. Although this plan may occasionally cause some problems to the reader who tries to go through Habermann’s paper simultaneously, we believe it to be the only logical one. Thus, we strongly urge the reader to go through the whole of Habermann’s paper first, so that he can see the debate in the proper perspective.

Since all the key texts on Pascal can now easily be obtained, we encourage the reader to mould his own opinion by reading references [16, 12] and [25]. One should notice, however, that Habermann based his paper exclusively on the first version of the Revised Report [26] and only it (the Revised Report is now a part of [16]). The Axiomatic Description [12] was not available at the time he wrote his comments; this would allow for certain misinterpretations.

2. Refutable Points

In this section, we deal with points which are, in our opinion, clearly refutable, i.e. criticisms which resulted from a misinterpretation of the basic aims of Pascal
or a misunderstanding of some major aspects of the language itself. It is possible that some people might have preferred a different repartition of points between this section and the following one.

2.1. Useful Constructs Not Included in Pascal

Habermann suggests four such constructs, but it would be very easy to continue adding constructs to the language indefinitely: Pascal does not contain all the constructs which may be considered useful, nor even all those present in other programming languages. This is because creation of an endless list of constructs is clearly not the right direction to follow for the development of better programming languages. The most unfortunate attempt in this direction is that of PL/I [14], and even its most irreclaimable addicts and most enthusiastic eulogists always seem to find more constructs to incorporate in it [4, 13, 22].

In fact, one of the principal strengths of Pascal is that it is a simple and concise language, including only what is vital for reaching its aims. We remind the reader that there are only two of them: to allow the teaching of programming as a systematic discipline, and at the same time to be implementable in a reliable and efficient way. These objectives are precisely the most difficult ones to reach when using languages which try to incorporate all “useful constructs”. The author of Pascal has therefore severely restricted the number of facilities, and it is quite sure that almost everyone will find missing certain of his favourite constructs. Consequently, we find not valid as a whole the criticism that Pascal does not contain some feature or other. The individual importance of the specific “left out” constructs Habermann regrets constitutes yet another point, less decidable, which is deferred until Section 3.

2.2. An Exercise in Programming in Pascal

The simple exercise worked out for the reader by Habermann is supposed to prove that Pascal is a poor tool for teaching programming. All that such an example demonstrates is simply that it is possible to misuse Pascal, which is of course true for any tool. Consequently, we prefer to rework the part of the example which is given, to show that in actual Pascal no difficulties arise.

The problem is to compute prime numbers using the sieve of Eratosthenes. A comparison of the different algorithms available, even superficially, should be useful before trying to put down a solution [5, 28], but this precise algorithm is not so bad, and it has been particularly well investigated by Dijkstra [5], Hoare [9] and Wirth [16, 27].

Habermann chooses to represent the numbers between 2 and \( n \) by an array of integers, in which every element contains its own index: to remove a number from the sieve, one assigns a zero to the corresponding element. Although using the set structure of Pascal should be far better [9, 16], we shall use simply a Boolean array, not only because it seems more natural but because we will encounter the same problems with indexes as Habermann did. A natural way to start-off