ma
the
ma
tisch

cen
trum

AFDELING INFORMATICA IW 35/75 APRIL

L. AMMERAAL

ON THE DESIGN OF PROGRAMMING LANGUAGES INCLUDING MINI ALGOL 68

amsterdam 1975

stichting
mathematisch
centrum

AFDELING INFORMATICA IW 35/75 APRIL

L. AMMERAAL

ON THE DESIGN OF PROGRAMMING LANGUAGES INCLUDING MINI ALGOL 68

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

AMS(MOS) subject classification scheme (1970): 68A00, 68A30

ACM -Computing Reviews- Category: 4.22

On the Design of Programming Languages Including Mini ALGOL 68

by

L. Ammeraal †)

ABSTRACT

Some general characteristics of ALGOL-like programming languages are introduced. It is discussed what kind of language concepts are useful enough for most users to justify their presence in new languages. As an illustration, Mini ALGOL 68 is proposed as a modest successor of ALGOL 60.

KEY WORDS & PHRASES: *Language design, programming, ALGOL 68.*

---

†) Mathematical Centre, Amsterdam

SOME REMARKS ON THE INTRODUCTION OF NEW LANGUAGE CONCEPTS

When a new programming language is presented, most of us are only interested in the question whether the set of new language concepts contains the things that we consider useful in a language. If the language also offers a number of features that we do not need ourselves, we, as "humble programmers", usually assume that they will be useful to others. Sometimes we even learn those new features eagerly and then teach them to others without knowing their merits from our own practical experience. The following three considerations justify a less tolerant attitude towards new languages.

First, unnecessary language elements are undesirable from an educational point of view. The subject-matter for students should consist of useful and interesting things. Special care should be taken to avoid teaching the wrong programming habits as a consequence of inappropriate tools in a language.

Secondly, a language should be well implementable and its availability should not be limited to users of large computers. As a companion to the definition of the language, a fast and reliable compiler is much more wanted than a clever doctoral thesis on some advanced implementation topic.

The third argument has to do with style. Useless things should be absent in a programming language, even if they do not harm anybody. Their presence shows the same bad taste as a number of unused buttons for airconditoning in a motor-car whose driver always prefers to open the window a little bit for fresh air.

SOME CHARACTERISTICS TO CLASSIFY LANGUAGES.

The idea of choosing only a small number of mutually independent elementary language concepts, which can be used to build more complex constructs, was introduced by VAN WIJNGAARDEN and called *"orthogonal design"* [1]. There is a strong relationship between this idea and the introduction of the terms *width* for the number of elementary language concepts, and *depth* (or *profundity*) for the amount of more complex consequences that are immediately implied by them. A classical example of a profound language aspect is the use of recursive procedures. Profound language properties are easily overlooked at first sight because they are hardly mentioned in the language definition and may even be discovered later on. A language element that looks very simple may have such profound implications that is seems wise to abolish it. A well-known example of such a "harmful" element is the "goto statement" [2]. In connection with this, it makes sense to mention a third characteristic of language concepts, viz. the *level*: the more a language concept is suited as a tool for our process of abstract thinking, the "higher" is its level. We call the level low if the concept is closely related to the construction of a machine. The adjectives high and low are frequently used for a language as a whole, e.g. to compare ALGOL-like languages with assembler languages. However, high-level languages may contain low-level elements. These elements may have been included on purpose, as an attempt to give the programmer a better grip of the facilities offered by the machine. Typical low-level elements are the DEFINED-attribute in PL/I and bits and bytes structures in ALGOL 68. In environments where more attention is paid to the design of general, reliable and machine-independent algorithms than to the exploitation of a particular piece of hardware, low-level language elements are not popular. Low-level elements may also exist in new languages for historical and conservative reasons, as a consequence of the designers' lack of courage to reject such an inheritance from preceding languages. Descended from a branch instruction in machine language, the goto statement is such a typical low-level element, which has been maintained even in ALGOL 68.

Inspired by Dijkstra's critical arguments against the goto statement, WULF proposed to consider the global variable harmful [3]. In this case, however, it should be kept in mind that not all tools that are dangerous should be considered harmful. A butcher will not follow the advice to replace a sharp knife by a blunt one, although he will admit that the latter is less dangerous. Similarly, global variables and, in particular, functions with side-effects, though dangerous, can be used as very powerful tools and should not be abolished as long as no satisfactory other means are given to replace them.

How wide and how deep a language should be depends on the kind of people and of machines that will work with it. It is not unreasonable to require that a high-level computer scientist should be familiar with a language as wide as PL/I or as profound as ALGOL 68. In most professions experts have to study several years and there is no reason why a computer scientist must be taught a programming language in only a week's time. On the other hand, only a small fraction of all computer users are computer scientists. There are a great many people who are working in completely different fields, such as e.g. chemical engineering, and who write computer programs from time to time, to solve their problems. They need a much simpler language than e.g. ALGOL 68 or PL/I. Theoretically, they could be taught only a well-chosen subset of such an extensive language and use the compiler for the full language. This philosophy, however, requires a compiler for the full language as well as a good teacher who is able to restrict himself. Such a compiler is more than most users need and there is a consequent danger that they will pay for things they do not use.

Mini ALGOL 68: A MODEST SUCCESSOR OF ALGOL 60

ALGOL 60 is a high-level language of moderate width. Most complaints from its users concern their implementations, and not the language itself. At those places where a good ALGOL 60 compiler is available, the language has proved to be very useful and convenient for a great variety of applications. Yet, fifteen years after the definition of this language, it is well-known from experience that, on the one hand, the language badly lacks a few simple extensions and that, on the other, some elements of the language are seldom used and can be considered superfluous. String and character handling facilities, e.g., would have made the language more appropriate for commercial applications. The <u>own</u> and <u>switch</u> concepts are examples of language elements that have not proved their right to exist. A very useful thing in ALGOL 60 is the conditional expression. This is a typical high-level language concept. It allows us to express ourselves in much the same way as we think and enables us to write things much more briefly, i.e. without repetitions of pieces of program text, than with only conditional statements. The following example shows this.

Suppose that we want to output the value of $p * i$ if $a[i] = x$, with the additional restriction that this test can only be made if $i \leq n$ and should be considered to fail if $i > n$. Otherwise we want to output the value of $(p+1) * q$. In ALGOL 60 this may be achieved by

```
output (if (if i≤n then a[i]=x else false)
        then p * i else (p+1) * q).
```

But for conditional expressions this could only be programmed in a considerably more laborious way. It is curious to observe that some newer programming languages such as PL/I and PASCAL lack conditional expressions. Using our terminology it can be said that these languages lack something in the depth dimension, which is available in ALGOL 60. ALGOL 68, on the other hand, has something more than ALGOL 60 in this direction (and so has Mini ALGOL 68), e.g. "unitary clauses" as an elegant generalization of "statements" and "expressions". By these we are allowed to write "statements", possibly enclosed by parenthesis, anywhere within "expressions", which provides us with a very powerful flow-of-control mechanism. Suppose, e.g., that we want to construct a loop with the test for termination placed neither at the beginning nor at the end, but somewhere in the middle, say between *part A* and *part B*. In the old days this was programmed as, e.g.,

```
again:  part A;
        if i > n then goto ready;
        part B;
        goto again;
ready: .
```

In ALGOL 68 this can be written as

```
while part A; i ≤ n do part B od.
```

We may conclude that, of all well-known languages, ALGOL 68 is probably the best candidate for programming without goto statements. However, ALGOL 68 is not only a language with fine profound properties, but it is also extremely wide, in our terminology. It offers too many facilities to be the optimum choice as a general-purpose language for everybody. A modest sublanguage of ALGOL 68 seems to be a better successor of ALGOL 60 in a number of situations. A proposal for such a sublanguage is Mini ALGOL 68. It has about the width of ALGOL 60 but is considerably more profound. The low-level concepts bits, bytes and gotos are not included in the language. The absence of structured values, united modes, heap generators, operator declarations, mode declarations, casts, flexible bounds, formats, completers and semaphores will probably disappoint those who are familiar with ALGOL 68. It would be a mistake, however, to conclude that Mini ALGOL 68 would hardly offer anything more than ALGOL 60. In addition to many useful ALGOL 60 elements, it offers the general concept of a unitary clause as mentioned before, the loop clause as an improvement of the for-statement, the case clause, variables to assign values of the modes char and string to, the improved parameter mechanism for procedures, the routine text as a special case of a unit and many other specific ALGOL 68 concepts. The following (nonsense) Mini ALGOL 68 program shows some possibilities concerning data types that exist neither in ALGOL 60, nor in PASCAL, SIMULA 67 and PL/I.

```
begin proc ([ ]int) [ ] int p;
      real pi 3 = pi/3;
      p := ([ ]int a) [ ] int:
           ([1:upb a] int b;
           for i to upb a do b[i]:=-a[i] od; b );
      # now a routine has been assigned to the variable p#
      [1:3] int x := (10,20,30);
      [1:3] int y := p(x); # yields (-10,-20,-30) #
      [1:3] proc (real) real q := (cos,sin,exp);
      print (q[1](pi3));
      # .5 (=cos(pi/3) is now written #
      q[1] := sqrt;
      print (q[1](25))
      # 5 (=sqrt(25)) is now written #
end
```

In most languages neither can a function yield an array, nor can elements of arrays be functions. In the program above these things happen to the "function" *r* and the "array" *q*, respectively. Further details about Mini ALGOL 68 can be found in the User's Guide [4]. A compiler [5] and a run-time-system for Mini ALGOL 68 were written by the author of this paper in about eight months, which indicates that implementing this language is an order of magnitude simpler than building an ALGOL 68 compiler.

A few choices with respect to the inclusion of certain concepts in Mini ALGOL 68 were made somewhat arbitrarily. E.g. the question can be raised if it was right to include modes beginning with an arbitrary number of refs. It was, however, not the intention of this paper to claim that Mini ALGOL 68 is better than any other sublanguage of ALGOL 68. Its main goal was to emphasize that we should think about the question what tools are useful in programming.

REFERENCES

[1] WIJNGAARDEN, A. VAN, *Orthogonal Design and Description of a Formal Language*, Mathematical Centre MR 76, Amsterdam, 1965.

[2] DIJKSTRA, E.W., *Goto Statement Considered Harmful*, CACM 11 (1968) 147-148.

[3] WULF, W. & SHAW, M., *Global Variable Considered Harmful*, SIGPLAN Notices, 1973, 28-34.

[4] AMMERAAL, L., *Mini ALGOL 68 User's Guide*, Mathematical Centre IW 32/75, Amsterdam, 1975.

[5] AMMERAAL, L., *An Implementation of an ALGOL 68 Sublanguage, Prepublication*, Mathematical Centre IW 31/75, Amsterdam, 1975.