Discussion

Session 3: 18 April 1962 (Morning)

The Chairman (Mr. A. Geary, *Vice-President of The British Computer Society*): Ladies and gentlemen, it is my privilege this morning to take the Chair for what I am sure will be a very interesting session. My first pleasure is to introduce Dr. Dijkstra to you. He is well known, and we have the opportunity this morning of seeing him as well as of hearing him. As we all know, he is from the Mathematical Centre in Amsterdam, and has collaborated to some extent with Professor van Wijngaarden. He is one of the authorities on ALGOL. We have warned him that there has been a certain amount of bias against ALGOL in England, in some quarters, and this morning he will have an opportunity to tell us precisely why ALGOL is a much better language than any other of those we may call substitutes.

His book is the second of the A.P.I.C. Studies in Data Processing, and is called *A Primer of ALGOL 60 Programming*. Two or three copies will be available for your inspection.

I have great pleasure in asking Dr. Dijkstra to speak to us on "Operating Experience with ALGOL 60."

(*Dr. Dijkstra then presented his paper*), *p. 125.*

Dr. R. D. Clippinger (*Honeywell Controls*): I notice that the Primer does not have such special information as the way to make use of subroutines. Is that correct? Does the supplement include such special features?

Dr. Dijkstra: The programming primer is not meant as a full guide to the use of our implementation. We have mentioned the special restriction of our implementation, and we have tried to be very careful to make a distinction between ALGOL 60 and what we could do with it. If somebody is going to use our systems, he just gets a couple of copies, probably in Dutch, telling him exactly what facilities are available. But the primer is continuously expanding.

Dr. Clippinger: In America we consider Europe to be the leaders in ALGOL, and we are interested in any appendices which could be used as reference material.

Dr. R. Taylor (*National Institute for Research in Nuclear Science*): I am interested in the difficulties of teaching programming. Were the students who picked up ALGOL in four days people with previous experience of machine-language programming, or complete beginners to programming of any kind?

Dr. Dijkstra: We had a couple of heads of department of other computation centres from other universities. On the other hand, we had people who did not know anything about computers at all, and who did not have any mathematical training but who wanted to learn about programming.

We also use the course as a short introduction to what can conveniently be done with a computer, and what can be expected from it. We had a very varied group of people. For that reason, where possible we gave an explanation of the syntactic rules of the language. We showed the kind of ambiguities you get if you have a language in which you omit multiplication signs. These ambiguities of the rules have to be solved for users in order to make them more tolerant.

Dr. Taylor: Have you thought of running different courses for the two different classes of people?

Dr. Dijkstra: No. We have one course only.

Mr. C. Strachey (*Consultant*): I should like to make two comments about embedding ALGOL into a system. As it stands ALGOL contains no mechanism for input-output; there must, therefore, have been some extensions to allow for the possibility of getting things in and out, especially as Dr. Dijkstra has ruled out the possibility of any machine-code procedures.

The second point concerns the business of incorporating a library of subroutines, and having to write the ALGOL program in this curious way inside their scopes. This is also part of the job of incorporating ALGOL into a system. Has Dr. Dijkstra contemplated introducing some form of formal editing facilities for joining his tapes together, or constructing programs out of bits of other programs, which one often has to do with large programs?

Dr. Dijkstra: The technique of embedding the program in a universe which contains *a priori* knowledge is completely fair. ALGOL 60 is described as a language to describe algorithms, and is not a language as a whole. If you have a piece of text it means something, thanks to your prior knowledge of the way to read it. The technique of ALGOL 60 is that if something is not defined in the block itself, it is defined in one of the surrounding blocks. If you wish, you can go further and think about it as a whole language embedded in a universe which tells you what "plus" means. If you have to define the process there is a level of description, and if you restrict yourself to a description of algorithms, as ALGOL 60 does, input-output does not come in. But if you want to use a machine, you are forced to go into a little more detail.

We feel that it was completely compatible with the spirit of the language to do this with the universe surrounding it. We do not actually embed it in a tape with a beginning round it and declarations round it. The translator has the prior knowledge as a series of identifiers of procedures which are contained in the library, and when it translates a program it just notes which of these library routines are used. Finally, when the object program is read in, the library passes through the tape reader and is read in selectively. You may have library routines, *A*, *B*, *C* and *D*. It may be that *A* uses *B* and *B* uses *D*, and *C* also uses *D*. The organization must be such that if the program called explicitly for library *A*, *B* and *D* are also read in, and if you call explicitly for *C*, *D* will also be read in. If you call explicitly for *A* and *C* you must see that *B* and *D* are called in only once, and if you have two routines—*E* and *F*—which call one another, mentioning one explicitly calls the other in. I hope that this more or less answers the question how to compose a program from different bodies and how it is organized. From its very structure it can detect what it needs.

Mr. Strachey: What facilities are there for making one's own library? If the library is made for you, you have no facilities for assembling except for those made for you by somebody, and presumably published in Dutch—to which I have no access.

Dr. Dijkstra: The possibilities of making one's own library are unlimited, provided you can convince those who can tell you how to do it that it is worth telling.

The following papers were then presented:

"Progress Report on the Elliott ALGOL 60 Compiler," by C. A. R. Hoare (Elliott Brothers Limited) p. 127.

"Implementation of ALGOL 60 for KDF9," by F. G. Duncan (English Electric Co.) p. 130.

"Operating Experience with the FORTRAN Language," by A. E. Glennie (A.W.R.E.) p. 132.

Mr. G. M. Davis (*English Electric Co.*): As Chairman of the Society's Committee on Standardization of Programming Languages, I would like to add a few words to what has already been said on this matter. The whole subject is quite difficult for a number of reasons, but I would not agree that the number of committees considering it is in itself a major cause of difficulty. This is the way in which standardization normally works. I can illustrate this by reference to the work on character sets and coding which has been going on for some years. This has involved at least three British Trade Associations and one international one, the British Standards Institution, the International Standards Organization and other national standardizing bodies such as the American Standards Association. These bodies all work well together, each one handling aspects relevant to its own position and problems with which it is particularly concerned, and preparing recommendations to put to larger bodies.

Things are moving well towards putting forward an international standard for character codes, so it cannot be said that the number of committees is the trouble. In the case I have quoted this method works well, because there is substantial common membership on the committees, and there are many invitations to members of one committee to sit in with another. That is something we have not yet established with the growing number of committees concerned with programming languages, and it is necessary to establish it if our system is to work well.

Once the character code standard is produced it will be a solid and comprehensible document, which will probably last for some time. It is a little questionable whether this would ever be true in the case of programming languages, but we are not yet at all clear as to what might be meant by standardization in this field. This is the point which the Society Committee is currently looking into, considering the possibilities and trying to reach reasonable conclusions upon which people might work. Our terms of reference are "To consider the desirability, feasibility, and possible scope of standardization in programming languages." Mr. d'Agapeyeff may have been a little over-enthusiastic when he suggested that we had reached a conclusion. We are still working towards it. The members of the Committee—looked at from the Chair—are highly creative, independent people and if we reach a conclusion it will certainly have already satisfied a range of conflicting interests.

I do not want to deal with the substance of the matter, because that might be prejudicing the conclusions that might be reached, but some things can be said. Firstly, about the form which standardization might take: there might be standardization of one language which will do everything, or one language with variations to meet different fields of application, or several languages to cover different fields of application. Apart from this, many useful things might be done, by international agreement, with an element of standardization. We might be able to agree standards of maintenance arrangements and the efficiency of such arrangements, for keeping languages up to date, dealing with modification and correcting errors, or for testing compilers over the field for which they are intended. We might also agree on the standardization of means of specifying and describing languages. Until this is done one cannot start the standardization of languages themselves.

It has been suggested that the standardization of languages might have the effect of stultifying their development and the development of computing equipment; there is no doubt that this is a real danger. However, some people have quoted the fact that Dr. Clippinger's Honeywell 1800 has the same code as the 800, as an argument against standardization, and this is not really relevant; FACT is in no sense a standard; it is a piece of equipment into which a lot of capital has been poured, and it therefore has its influence on other things, but this has nothing to do with standardization.

There are already powerful commercial interests moving towards standardization, and vague overall objections are not likely to be very effective. It would be better either to join in and try to influence the course of development, or to keep out of the way.

There are also genuine user interests, in avoiding capricious and vexing variations which are unhelpful to the great mass of people who want to program computers and want to exchange information. There is advantage in trying to avoid capriciousness. There are also advantages in trying to run jobs on a variety of machines from a centre; this is already possible to some extent with FORTRAN. International standardization might offer advantages in exchanging information, for example exchange of procedures written in ALGOL; such exchange is hindered by continuous development of the language; variations would better be introduced by periodical revision with constancy in between.

I should point out that what I have said represents my own views and not those of my Committee. I am glad to have this opportunity of raising the matter, since we have Dr. Clippinger here, who is Chairman of the International Standards Organization Working Group on the subject. In the work on character codes, there was some initial suspicion among the various bodies concerned, before they got to know each other; Dr. Clippinger might like to take the opportunity to say something to resolve any existing misunderstandings on the present subject.

Dr. Dijkstra: I am sorry that I did not mention that our group spent three days in April 1960 in Copenhagen, where we had some very inspiring discussions.

Mr. Hoare said that multiple error indication presented some difficulties. Our experience with ALGOL is that the first program of the user is full of clerical errors, but this kind of error decreases so fast that it is hardly worth while in subsequent programs continuing to check for it, after finding the first one, because there is a fair probability that it is the only one.

Mr. Glennie stated that the FORTRAN translator is slow, for instance, on account of optimization. At the Rome meeting last month a contributor showed a very elegant way to optimize exactly the situation required, but this can be very time-consuming. One reason why FORTRAN translation tends to be slow is the requirement that you can translate several things independently. This makes translation more difficult. I should like to know whether Mr. Glennie has any inside information about this fact.

I was somewhat shocked by the frequent mention, by Mr. Glennie and Mr. Hoare, of the general unreliability of these translators. I regard language as a tool. You must learn to love it, and it must be reliable. One reason why we did not have extensive optimization was that we were afraid that we could not guarantee its accuracy any longer. I strongly object to the allegation that the unreliability of the translator does not matter so much because it is highly improbable that it will hurt the common user. I should prefer to direct our attention to the *uncommon* user. In my opinion he is the man who matters.

Mr. Glennie: I seem to be the principal target of Dr. Dijkstra. I was stating the facts as I know them. I agree it is undesirable that a compiler should be unreliable. In this context, however, the benefits of optimization often outweigh the troubles we get in FORTRAN. The FORTRAN Compiler was written many years ago, basically before people like Dr. Dijkstra and the Copenhagen group began to study compilers in a theoretical manner. We cannot expect that in the future the experience of the past will be replicated. Within the next five years I expect that we shall have reliable economizers. I do not think it would be advisable to take Dr. Dijkstra's point too seriously.

Dr. Clippinger: With regard to standardization at an international level, it is worth noting that there are approximately 1,000 American standards, and there are apt to be a similar number of British standards, and standards of various other nationalities. There are approximately 100 international standards of all classes. There are fewer international standards because more people have to agree, and it takes longer to arrive at agreement. It is also more complex to arrive at agreement, and some international standards have taken a very long time to develop—some have taken twenty years.

I cannot predict how fast standardization will occur in the programming-languages field, or even whether it will occur. I cannot speak for my Committee because it has been in existence for only a very short time. It has only had one meeting. It has been doing some work in preparation for another meeting, but nobody could distill the essence of the opinions of the members of the Committee yet from such a short exposure.

I could speak a little more from the point of view of what is going on in the United States, but I do not think that this is the appropriate time. That is about all I can say—except that there is a meeting in Stockholm on 9 and 10 May, which will be attended by members from many countries. The agenda will include a discussion of such things as the survey of programming languages, which survey is far from complete; it is only just getting started. There will also be a discussion on the interaction between work on programming languages and coding character sets. Where this discussion will be I cannot predict.

I now direct myself to an entirely different point. The question of how much checking of a compiler is appropriate is very interesting. It is similar to the question of how much checking is appropriate in a machine. In the early days people attempted to construct machines without checking and they had a lot of trouble getting the machines to run long enough to get any useful work done, and to have a mean free period between errors that was great enough. They started to build in checking facilities, and some machines had a lot of checking—and that raised a new problem: they could not do any work at all, because they kept bumping into checks. This forced them to improve the machines until they began to work very well, and the mean free period between errors got very much longer until it reached the point where the operators asked themselves, "If we removed some of the checking equipment, would it not be better?"

Because some checking circuits fail occasionally, there has been a feed-back principle governing computers, and you arrive at a good balance between main and checking circuitry and the circuitry for getting the job done only as a result of determining how reliable the components are. A similar consideration applies in the case of the design of compilers. Certain kinds of checks are put in because one has certain habits and does things one did before, but when the compiler is used it is discovered that some of the built-in checks are completely inappropriate and should be removed. They provide information that does not have a value corresponding to the efforts it took to get the checks incorporated.

On the other hand, there is probably some irreducible minimum to the amount of checking to be done, and we shall not know for sure until we have more experience of what this amount is. But I think that it is somewhat less than in the second generation of FORTRAN compilers, and it may be a little more than Dr. Dijkstra has now. It is an interesting question.

Dr. Dijkstra: I thought that one of the main differences between the hardware and the software was that the hardware might be O.K. today and wrong tomorrow, whereas the software had the unique property of being independent of time. For that reason the analogy mentioned seems somewhat superficial.

Mr. Glennie: It did not seem to me that this comment was superficial. The programmer—the compiler of a program—by definition makes mistakes all the time, and therefore there should be checks.

Dr. Clippinger: Those two comments on my remarks seem to be diametrically opposed. Dr. Dijkstra said that the comments were inappropriate, because of the logic that if it is correct it is correct and there are no further errors to worry about, but Mr. Glennie said it was not those errors that we were talking about. He said that it was the errors of the human that were pertinent. Those errors will continue, so I think that my remark that we do not yet know how much checking is appropriate is still pertinent, in order to take care of the errors created by humans. I am sure that a certain amount will be appropriate in the future, and it is important to determine what is a reasonable amount.

Dr. Wilkes (*University Mathematical Laboratory, Cambridge*): The standardizing of programming languages is beginning to engage the attention of many people. It is, therefore, worth while to consider what type of progress we want to make in this field. I suggest that the most useful sort of standardization is that which concerns itself with saying exactly what is meant by a given article. It is, for example, useful to know exactly what is meant by a 2BA screw thread. Similarly, it will in the future be useful to know exactly what is meant by a given programming language, such as a particular edition of ALGOL, or a particular mark of FORTRAN. But to suggest that standardization should mean the selection of one particular language to be used on all occasions, in preference to all others, appears to me to betray a very superficial knowledge of the subject. It may be that no one in the programming field believes that this is what standardization should mean in relation to programming languages. If so, it would help to have a clear statement to that effect.

Mr. M. Woodger (*National Physical Laboratory*): Even if it is desirable to describe precisely a variety of programming languages, rather than a single universal language, will it not still be necessary to choose a single language in which to express these descriptions?

Mr. J. S. Hillmore (*Elliott Bros.* (*London*) *Ltd.*): In his talk Mr. Duncan explained that considerable effort had gone into enabling the KDF9 ALGOL compiler to accept procedure bodies written in User Code. On the other hand, Dr. Dijkstra has said that he does not think that ALGOL programs should contain anything but pure ALGOL, and has implied, to my mind, that even if the X.1 had a larger store he would not incorporate into his compiler the facility of dealing with machine-code blocks. Will the panel discuss this matter a little further and state whether they think that one should be able to put machine code into ALGOL, or whether ALGOL is ALGOL and nothing but ALGOL?

Mr. Duncan: The thing about implementing ALGOL by making it part of an automatic-coding system is that it should be possible to communicate with it. A means for this communication has been provided in the report, and we have used this means in the way that I have sketched.

Dr. Dijkstra: You have rightly interpreted my saying that even if we had a larger store we should have to deny the possibility of inserting machine code. The library procedures about which I have been talking are in machine code, but it is one of the unavoidable drawbacks that constructing them and putting them in the library is a somewhat elaborate procedure. It is so elaborate that it is a good brake to prevent machine code coming in again. The library contains an unlimited number of procedures. It is growing very fast. Many of its procedure functions can be written in ALGOL as well. They are put in the library first to take the burden from the programmer who wants to use them and, secondly, because they are considerably more efficient than anything the translator can make out of them.

Mr. Hoare: I should like to make clear our approach to the question of machine coding in ALGOL programs. We agree with Dr. Dijkstra mainly, but there is one point which is peculiarly relevant to our operating system. If a casual user is allowed to use a machine code in his programs he will, as likely as not, overwrite the translator, and this will cause far more trouble than any optimization he thinks he may have achieved by putting in machine-code instructions.

Mr. Duncan: We have a requirement for various kinds of input and output procedures. Our solution is a simple one, which makes it possible for us to read into an ALGOL program information prepared in rather peculiar forms, for reasons best known to the people who produce these data. The whole question is a little misunderstood. I do not think that we are encouraging people to use machine code unnecessarily; what I am saying is that all the functions and procedures in the library are either procedures with ALGOL bodies or procedures with machine-code bodies, constructed in accordance with the directions of the report. The report seems to be quitc adequate here, and the problem of communicating between the outside world and the ALGOL program is important. We want to be able to read various forms of data and have a fine control over the appearance of our results on the printed sheet. That is a perfectly legitimate requirement requested by our customers.

Mr. P. Wegner (*London School of Economics*): I should like to ask three questions of Dr. Dijkstra. The first concerns the relationship between ALGOL and FORTRAN. Dr. Dijkstra said in his paper that ALGOL was a more advanced language than FORTRAN. I strongly disagree with this statement.

Although ALGOL is, in certain respects, more elegant, FORTRAN is a more complete language since it includes input/output and system facilities. Moreover, FORTRAN has certain advantages over ALGOL even in the area of algorithmic procedures where ALGOL is strongest. As Mr. Glennie pointed out, FORTRAN permits the use of independently manipulatable subroutines. This is not possible in the case of ALGOL, since ALGOL procedures must be physically embedded in a context. The importance of facilities for contextually independent subroutines within a programming system cannot be underestimated. Such a facility permits a complex problem to be stated in terms of the solution of a number of independent subproblems—a procedure which is closely analogous to that of everyday problem solving. ALGOL does not permit the statement of a problem in terms of independent subproblems, since subproblems in ALGOL must be embedded in a context. This is a grave drawback of ALGOL, which becomes evident particularly when ALGOL is used for the statement and solution of large problems.

The second question is concerned with the relative importance of main memory and auxiliary storage. Dr. Dijkstra has said that he is adding additional units of 4K (4,000 words) to his basic 4K machine. If he had a choice between an additional 4K and two tape units, which would he prefer? Would the addition of a backing store make a great deal of difference to the implementation of his ALGOL compiler?

Finally, does Dr. Dijkstra advocate recursive procedures because he feels that it helps the user to love his tools? I feel that a reason of this kind is valid and important, and would agree with him that aesthetic considerations are among the most important in the formulation and implementation of a language.

Dr. Dijkstra: The main virtue of recursive procedures is that they make the tool more lovable for computers. A few weeks ago somebody used the phrase "ALGOL playboys" in a nasty fashion, and I was very angry. A Dutch philosopher wrote a big book called *Homo Lucidas*—the plain man—showing clearly that everything which, ages later, was regarded as of some significance, had started off as being 100% plain.

Fate has decided the question of the 4K memory for us. It was fairly easy to get another 4K added, whereas for the same machine a magnetic-tape backing store was a different matter. We could extend to a 12K memory, if tapes become available. It is a question of the lifetime of our installation. It is less powerful without a big store. One of the great advantages of extension is that it is very simple. You just change some contents of the storage allocation scheme and everything goes on fine again.

Similarly, if you start having a drum, or tapes, there is a new piece of software, which must also work properly. As for the backing store, I hope to move for the rest of my life in circles where it is not obligatory to use magnetic tape. I hope that drums will become available, which will serve for most applications.

I still do not understand the point about segmentation. The procedure in ALGOL 60 is as flexible as I can imagine. In my opinion retranslation of the whole lot should not pose a problem.

Mr. B. Randell (*Atomic Power Division, English Electric*): As a member of the group working on the KDF9 ALGOL Translator at Whetstone, I think I could throw some light on our reasons for allowing the inclusion of machine code in ALGOL programs. There are two very important advantages. It will save duplication of effort, and also allow us to put off the thorny problem of standardizing input and output. Once a system for including machine code in ALGOL is published the two groups in English Electric can go ahead with their respective translators, each with its separate field of application.

Somebody mentioned recursive procedures, and people playing with them. Most published examples show you how to do this—for instance, the famous factorial example—and it would be difficult to think of a less efficient procedure for factorial than this. However, if you have a procedure, written by somebody else, which uses, say, an integration routine, it would be possible to use this procedure, even within the integration routine, without having to delve into the procedure. That is the real benefit of recursive procedures.

Mr. F. S. Ellis (*Elliott Brothers*): Dr. Dijkstra has quoted certain ratios in connection with ALGOL programs and handwritten programs. Can he give the corresponding ratios in time involved in writing the handwritten programs and the ALGOL programs? Mr. Glennie quoted a figure of 3 to 1 in cost, but what about time? Do you more than make up for the difference in cost by the saving in time?

Dr. Dijkstra: The corresponding ratios in writing time might be 100 or 1,000. It goes up on an exponential scale in practice. That is why so few comparisons with run-time systems are available.

Mr. Glennie: The difficulty of answering questions as to the effect of ALGOL and FORTRAN on programming times is that there is no such thing as a standardized programmer. But I would say that you might expect to get a gain of 2 or 3 to 1 with FORTRAN with the average programmer, assuming the problems were defined.

The Chairman: I am sorry to have to draw this discussion to a close, but we have to meet again at 2 o'clock. I know that you would wish me to add your thanks to mine, to our four speakers. We are indebted to Dr. Dijkstra, Mr. Hoare, Mr. Duncan and Mr. Glennie for making this morning's session a great success. Thank you very much. (*The conference adjourned.*)

Computation of the latent roots of a Hessenberg matrix by Bairstow's method

By D. C. Handscomb

This paper gives the details of an algorithm, equivalent to Bairstow's process, for finding the real quadratic factors of the characteristic polynomial of a Hessenberg matrix, working from the matrix itself. This includes a method of removing known factors without introducing serious errors into the remaining roots.

Introduction

1. Bairstow's method of finding the real quadratic factors of a polynomial is well known (Wilkinson, 1959b), and can obviously be applied to the problem of determining the latent roots of a matrix from its characteristic polynomial. However, the roots of a polynomial are often very sensitive to errors in its coefficients, and it is therefore necessary to carry the calculation to much higher precision than is ultimately required.

We shall here describe an alternative direct method of finding the roots of a lower Hessenberg matrix $\{a_{ij}\}$, in which $a_{ij} = 0$ when $j > i + 1$. Wilkinson (1959a) has shown that any real matrix may be turned into this form, by similarity transformations, without great loss of accuracy.

We do not subject the matrix to any further transformation, so avoiding the inconvenience of multiple-precision arithmetic. On the other hand, unless the matrix is small or very sparse, this direct method calls for many more multiplications than are needed to solve the expanded polynomial.

The fact that we never construct a characteristic polynomial compels us to find a substitute for the usual method of dividing out each factor before attempting to find the next. The device which we use is in fact more accurate than the usual method, and could profitably be used on explicit polynomials also.

Bairstow's Process for Hessenberg Matrices

2. In order to clarify the subsequent removal of factors, we shall present Bairstow's process in terms of an algebraic congruence. If $F(\lambda)$ is the polynomial whose quadratic factors are sought, and $(\lambda^2 + p\lambda + q)$ is an approximation to such a factor, then we first find the coefficients of the congruence

$$F(\lambda) \equiv \{(\lambda a + b) + (\lambda^2 + p\lambda + q)(\lambda c + d)\} \quad \text{modulo } (\lambda^2 + p\lambda + q)^2. \quad (1)$$

In finding these coefficients, we may multiply $F(\lambda)$ by any constant scale factor, since all the expressions which we shall encounter are homogeneous in a, b, c, and d.

The second step is to replace p and q, respectively, by $(p + \delta p)$ and $(q + \delta q)$, where

$$\left.\begin{aligned} \delta p &= \frac{ad - bc}{d^2 - cdp + c^2q} \\ \text{and} \qquad \delta q &= \frac{acq + bd - bcp}{d^2 - cdp + c^2q} \end{aligned}\right\} . \quad (2)$$

Under appropriate conditions, these new values are then better approximations to the coefficients of the quadratic factor.

It may easily be shown that this is Bairstow's process. If we differentiate the congruence (1) with respect to p