The Elliott ALGOL input/output system

By C. A. R. Hoare

A description of the method of specifying input and output in ALGOL programs run on the National-Elliott 803 and the Elliott 503 digital computers.

Introduction

The first aim of the input/output system adopted in the Elliott implementation of ALGOL is simplicity. It is essential that an ALGOL programmer can learn to use the system with a minimum of instruction. The second aim is convenience of notation, and, in particular, the avoidance of unnecessary repetition of the words "print" and "read" in the case of multiple input and output. Thirdly, sufficient flexibility in format control must be allowed to satisfy the requirements of scientific users.

It soon becomes clear that ALGOL procedure facilities are insufficiently powerful to allow the fulfilment of all these aims. We have therefore introduced new forms of statement into the language, the **read** statement and the **print** statement. This, of course, may involve loss of compatibility, but we hope that this inconvenience will be outweighed by the other advantages of the proposed system.

Outline of the system

Print and read statements in Elliott ALGOL begin with the words **print** and **read** (printed in bold type here, but underlined in a handwritten program), followed by a list of variables or arithmetic expressions whose values are to be read in or put out; the successive operands of print and read statements are separated by commas, thus:

read $x, y, z;$
print $x - y + z, x \times y + 2 - z, -101 \times x;$

The data tape to be read in by the read statement may be punched thus:

```
30
20
11
```

and the results produced by the print statement would be:

```
   21
  591
—3030
```

However, if x, y and z had been declared as real variables, the results would have been printed differently:

```
   21·000000
  591·00000
−3030·0000
```

The rules for the punching of ALGOL input tapes are very simple; numbers may be written in any of the forms allowed in ALGOL, and followed by some character or string of characters not allowed in a number. A space or change to a new line will also terminate a number. The string of characters which terminates a number serves merely to separate that number from its successor, and is otherwise ignored.

Each number printed appears on a new line, with eight significant digits. In the case of integers, leading zeros are suppressed, and, if negative, the minus sign is floated; real numbers, however, always appear with their full set of digits, with a decimal point in the appropriate position. This style of printing is called the *freepoint format.*

Additional facilities

The last section gives the minimum necessary information to enable the ALGOL programmer to read in data and punch out results. It is obvious, however, that the programmer is going to require more than this basic minimum, and his first additional requirement is the output of alphabetic messages and headings. This may be done by enclosing the characters to be put out in the ALGOL string quotes, and putting the resulting string as one of the operands in a print statement. The Elliott hardware representations for string quotes are £ and ?. and an example of their use is:

```
print £ Statistical Analysis?,
      hbar, £ cm. = average height?,
      wbar, £ g   = average weight?,
      rho,  £     = correlation?;
```

This will produce output as follows:

```
Statistical Analysis
          5406 cm. = average height
          4312 g.  = average weight
      ·81604522    = correlation
```

The next requirement is the suppression of the change to a new line which normally occurs before each number in the output. The purpose is that when the print statement is obeyed repeatedly, the results will appear in a tabular form for easy comprehension. The desired effect may be achieved by use of the so-called *setting procedure "same line"*,* which should appear in the list of a print statement immediately before the first number to which it is to apply. For example, in the statement:

print n, same line, hbar, wbar, £   ?, rho;

the n will be printed as before on a new line; the other operands will appear on the same line, so that on repeated execution of the statement a table of the following form will be produced:

```
1     5406     4312     ·81604522
2     4980     3019     ·78344106
3     5199     3192     ·80463291
      .  .  .  .  .  .  .  .
```

* For the 803 this is written as one word "*sameline*."

Format setting procedures

The occurrence of *same line* in a print statement is in effect a procedure statement, which calls a standard procedure to set a marker to indicate that all numbers subsequently put out by the print statement shall not be preceded by a change to a new line. This marker is automatically cleared at the end of the print statement, so that the setting of a marker in one part of a program cannot possibly interfere with output of other parts of the program. The restriction of the scope of a setting procedure to the print statement in which it occurs is the standard rule in format control of the Elliott system, and applies to all the format-setting procedures described in this section.

The first requirement in format control is to change the number of digits put out, either to economize in column space and avoid spurious accuracy by reducing the number of digits, or to achieve greater range by increasing the number of digits. The method of controlling the number of digits is different for integers and real numbers. For integers we use the setting procedure *digits* (E), and for real numbers the setting procedure *freepoint* (E). In both cases E stands for an arithmetic expression specifying the number of digits to be printed, inclusive of suppressed zeroes in the case of integers. As in the case of *same line* the effect of these setting procedures extends to all numbers of the appropriate type occurring subsequently in the list of the same print statement; if two incompatible settings are encountered in the same list, the scope of the first one extends only as far as the second.

The second requirement of format control for real numbers is the alignment of the decimal point in tabular presentations, or the use of an exponent part to indicate the scale of a number. Elliott ALGOL provides the format-setting procedures *aligned* (E, F)—E digits before the point, F digits after—and *scaled* (E)—E significant digits followed by decimal exponent.

The third requirement of format control is change of input or output device. In the absence of contrary indication, all input is read from the first tape reader, and all output takes place on the first tape punch. To change this, the name of the required device should be written as a setting procedure in the list of a print or read statement; the accepted names are:

reader (n)
punch (n)
typewriter
lineprinter

Note that the use of a lineprinter involves no extra complication for the programmer, since the administration of buffer areas, etc., takes place behind the scenes.

There are a number of other convenient format-setting methods provided in Elliott ALGOL. They are fully described in the Elliott ALGOL programming guide, which may be obtained on request, and I shall not give a further description here. But it must be emphasized that the use of the Elliott system is possible even without any knowledge whatsoever of these methods of changing the format. This is made possible by the system of so-called *presumed settings* which apply to every number for which the programmer has not made an explicit contradictory setting statement. The effect of the presumed settings has been described in the first part of this paper.

Advanced facilities

It has hitherto been assumed that format-setting procedures will normally be used among the operands of a print statement, and that they will thereby be effectively quarantined from other print statements. It is, however, possible to put a format setting outside a print statement, in the ordinary text of the program, in which case it applies to *all* printing subsequently executed, except that which comes under the control of a contradictory format setting *inside* a print statement. This provides a method by which the programmer can, in effect, change the presumed settings, if he does not like those provided.

A second possibility is that of writing format specifications separately from the print statement which uses the format. This will appeal in particular to users of FORTRAN, and is sometimes convenient. The format specification should take the form of a procedure declaration, with the required format setting instructions occurring *outside* a print statement in the procedure body. When this procedure is called from inside a print statement, the required formats will apply to all subsequent numbers in the list, but the effects will be quarantined from other print statements in the same way as normal format settings.

Finally the Elliott system may be used to help in the printing of quantities other than reals, integers, and strings. The printing of sterling quantities, matrices, etc., will normally be achieved by writing a procedure (or subroutine). The aim in writing this subroutine is to allow the user to retain full or partial control over the format. This may be done by including the programmed output procedure among the list of a print statement, preceded by the specification of the required format, e.g.

```
print n, m, aligned (4, 3), outmatrix (A);
print quantity, order number, digits (4),
      outsterling (price);
```

In these examples it is assumed that *outmatrix* and *outsterling* are the names of the procedures which put out the elements of the array A, and the pounds, shillings and pence corresponding to the integer *price*. In the first example, the elements of A will be printed in the aligned format, with four digits before the point and three digits after. In the second case, *price* may be held as an integral number of pennies, with a maximum of £999 19s. 11d.; it is to be printed in pounds, shillings and pence. Note that shillings and pence will be printed to two significant digits in the normal way, so that the "*digits* (4)" will apply only to the £ section of the output.

Conclusion

This paper gives a brief outline of the input and output system available with Elliott ALGOL. It has been designed to be used very simply in simple cases, and yet to possess enough flexibility to cover the vast majority of more complicated cases.

The system will sometimes be unable to cater for the user with unusual requirements, and we therefore have made available some facility for inserting machine-code instructions in an ALGOL program. But we hope that this facility will only rarely have to be used.

Acknowledgement

This paper is published by kind permission of Elliott Brothers (London) Ltd.

References

HOARE, C. A. R. (1962). "Report on the Elliott ALGOL translator," *The Computer Journal*, Vol. 5, p. 127.
ELLIOTT BROTHERS (LONDON) LTD. (1962). *The Elliott ALGOL Programming Guide*.

Summary of discussion

Mr. C. A. R. Hoare: In reply to a question by **Mr. Buxton** (*Whitworth Gloster Aircraft*), Mr. Hoare said that format procedure identifiers such as "new line," "digits (*n*)," etc., were reserved identifiers and did not need to be underlined. Like other reserved identifiers, they could be declared for other purposes in any program.

Dr. D. W. Scott (*International General Electric, Paris*): Please describe methods for reading or printing:

(1) a single element of an array, *e.g.* $x[j]$; and

(2) all members of an array, *e.g.* $x[j]$, j := n **until** m.

Mr. C. A. R. Hoare:

(1) Include $x[j]$ in the list of a print or read statement;

(2) **for** j: = n **step until** m **do read** $x[j]$.

Mr. H. J. Richards (*IBM (U.K.) Ltd.*): Why include in ALGOL such obviously machine dependent words as "typewriter," "card reader"?

Why not

Read (*n*) *list*
Write (*n*) *list*

as in FORTRAN IV where *n* denotes a logical unit?

Mr. Hoare: The procedure declaration in ALGOL provides a very easy method of changing the meanings of reserved identifiers. To run a program which mentions a card punch on a machine with only a lineprinter, for example, write

begin procedure card *punch*; *lineprinter*, ⟨*program*⟩ **end**

Mr. B. R. Taylor (*Ministry of Aviation*): How easy is it to define the end of a data list, using a warning character punched on the data tape?

Mr. Hoare: A standard procedure "buffer" makes it possible at any time to test the last character read on any device. This will usually be the character which has terminated the last number input. For example, if the ends of the rows of a matrix (read on reader 1) are signalized by an asterisk, the statement

if *buffer* ('*', 1) **then go to** *end of row*,

may achieve the required effect.

Mr. R. M. Paine (*CEIR (U.K.) Ltd.*): (1) You may have a matrix of 30×5 but only wish to output 20×4. How do you specify these dimensions in your print statement?

(2) You may wish to combine elements of two matrices in printing—how do you specify this?

(3) How do you read in only selected fields from punched cards by statements in your read format?

I think you should provide at least as good facilities for input/output as exist in FORTRAN, which itself is by no means perfect in this respect. A single figure on a line is not good enough; quite complicated layout should be allowed and be easy to specify. The appearance of the printed page is important in many jobs.

Mr. Hoare: (1), (2), (3) All these tasks may be programmed very simply in ALGOL, using loops. (4) The use of "same-line" has been explained in the paper, and presents little difficulty.

I do not think there is any point in introducing too many new notations specially for input and output. The reason why FORTRAN input and output is so complicated is that FORTRAN attempts time sharing of on-line peripheral transfers.

Mr. F. G. Duncan (*in a subsequent written contribution*): In answer to these questions and remarks, the most important thing to bear in mind is that the KDF 9 system is based on procedures and is not a fixed part of either compiler. It follows, therefore, that requirements which are not met by our own "standard" set of procedures can be provided for by the user himself.

(1) The selection of the elements to be printed would be effected through ALGOL **for** statements. A whole matrix, on the other hand, would be printed by a simple statement. In our matrix scheme each matrix carries its dimensions as part of itself; therefore in a *print matrix* procedure statement no specification of dimensions is necessary.

(2) Again, this is a question of selecting, through ALGOL programming, the numbers to be printed. The problem introduces no difficulties as far as printing is concerned.

(3) The detailed answer to this rather vague question depends on many things, such as whether the format of the cards is determined by punchings on the cards themselves, or described in the program. The basic procedure is to read one number or field from a buffer; the card reader itself reads a whole card at a time. Thus the problem is the simple one of ignoring unwanted numbers or fields. If any number is liable to be divided between fields, an *ad hoc* procedure seems indicated.

Reference to the text of my paper will show that I agree entirely with Mr. Paine's closing remarks.

Mr. J. A. Fotheringham (*Ferranti Ltd.*): Do you have any trapping procedure in your input routines for spurious or illegal characters, so that the programmer can program his own warning characters?

Mr. Hoare: No, but there are ways of programming warning characters as indicated in the answer to an earlier question.

Mr. K. A. Redish (*University of Birmingham*): Since the dominant consideration in the design of ALGOL is that it should be universal, in some sense, will the two authors explain why they have not constructed identical conventions?

Mr. Hoare: It is certainly undesirable to have a divergence in methods of specifying input and output in ALGOL. Unfortunately, at the time when a conference of ALGOL implementors was arranged, it was found that we had adopted radically different approaches to the problem. We considered that the convenience of the user was the most important factor, while Mr. Duncan was more interested in keeping to the letter of the syntax of the ALGOL report.

The most striking advantage of our system is that input and output of several numbers may be specified without repeating the words "print" and "read." This is, of course, impossible if these are procedures, since an ALGOL procedure can only have a fixed number of parameters.

Mr. F. G. Duncan: It is indeed a great pity that the two systems have so little in common. The fundamental cause of the difference lies of course with the ALGOL report itself, which gives no guidance on questions of input and output. (Whether it should have said anything on this subject is another story.) On the other hand, it does suggest procedures with bodies in non-ALGOL language, and this is the basis of the KDF 9 scheme. We have not seen the need for any special structure for input–output statements; the ordinary ALGOL structure seems perfectly adequate. To this extent, therefore, we plead "not guilty." This, though, is not the whole point. Even if two implementors had agreed to use code procedures and keep within the ALGOL forms, I doubt whether, with different machines, they could write coded procedures with identical functions. Perhaps they could as far as format specifications are concerned, but some machines, like KDF 9, introduce the need for "hardware-oriented" procedures, such as those concerned with allocation of peripheral devices, which are meaningless for other machines.

There is a great diversity of input–output devices in the world today. There is no language for describing what they all do—even COBOL does not try to cope with curve plotters or knitting machines. Perhaps ALGOL is wise in saying nothing, for, as a certain Dutch professor has rightly said, "only by absolute silence can one preserve complete generality."

Mr. H. J. Richards (*IBM (U.K.) Ltd.*): Your format resembles very closely the COBOL "picture." I think the ALGOL school should at least study the work of the COBOL school for ideas in this area.

Mr. F. G. Duncan: In designing the KDF 9 formats, we have drawn mainly upon the work done for DASK. The COBOL report has not, to my knowledge and memory, provided us with any new ideas.

There must be many people working on ALGOL who have tried to get to grips with the COBOL report. I have been involved in some quite intensive work to see whether ALGOL needs to be extended to cope with so-called "commercial" problems, and I have consulted the COBOL report as part of this work.

One cannot take over the COBOL ideas into an ALGOL scheme for many reasons. In any case they would need to be extended in order to deal adequately with such notions as floating-point and significant figures, without which it is impossible to produce decently laid out results.

I sympathize with the questioner's concern that there is divergence between the ALGOL and COBOL "schools." I hope the day will come when this thoroughly artificial distinction between "scientific" and "business" programming languages is removed. It seems a long way off.

Mr. B. Randell (*Atomic Power Division, English Electric Co. Ltd.*) (*who has co-operated in the work described by Mr. Duncan*): An important feature of the system being implemented in KDF 9 ALGOL is that any user can extend the system in any way that he pleases by declaring new procedures. These procedures, whose bodies can be in ALGOL in User-code will work both on the program-testing compiler, which runs interpretively, and on the optimizing compiler, which translates ALGOL into machine code. It has been difficult enough to maintain absolute compatibility between two such different compilers, and the only way has been to remain absolutely within the rules of ALGOL 60.

Correspondence

To the Editor,
The Computer Journal.

Sir,

"Print-out of Algol Programs"

Professor E. W. Dijkstra referred, on page 126 of the July 1962 issue, to the MC Algol Flexowriter of which there are now fourteen examples in Europe. An editorial footnote refers to one at the Cambridge University Mathematical Laboratory. Your readers may be interested to know that this Observatory has a similar Flexowriter, with the same keyboard and coding, but with some additional facilities.

In considering how .to extend our facilities for originating and printing Algol programs, we are interested in the IBM type 72 rotating-head electric typewriter. The construction of this machine appears to be inherently suited to punched-tape input and output, and the design has the advantage that the fount of characters can be changed in a few seconds, by simply exchanging the rotating type-head. The corresponding disadvantage is that at present a whole new type-head is necessary even if only one character is to be changed.

Since an Algol type-head is not currently offered, it will be necessary to design one. We would be pleased to hear from anyone who has suggestions about this, or who would share in sponsoring the initial cost of the necessary pattern.

Yours faithfully,
PETER FELLGETT.

Royal Observatory,
Edinburgh 9.
21 November 1962