Input and output for ALGOL 60 on KDF 9

By F. G. Duncan

1. Introduction

The absence from the ALGOL 60 Report (Naur, ed. 1960) of any explicit reference to the ideas of "data" or "results" for a program has led to the situation where almost every implementation has its own peculiar mechanism for input and output. The ALCOR group, for example, have two standard procedures built into their computers which when activated read and print a number in standard form. The MC-translator (Dijkstra, 1962) has similar provisions. The DASK user (Jensen *et al.*, 1960) has at his disposal a much more elaborate set of standard procedures which give very fine control over the appearance of the printed sheet of results. These DASK procedures, however, are not procedures in the strict ALGOL sense, since their treatment of parameters goes beyond that prescribed by the Report. The Elliott scheme (Hoare, 1962) provides input-output facilities of considerable power, but they are based on a structure which is additional to that of ALGOL itself. The system described by McCracken (1962) is likewise an extension of ALGOL 60. (Incidentally, the statement in the first sentence of his section 8.3 is not true; not all ALGOL input-output systems are based on the approach he describes.)

2. Principles of the KDF 9 system

For KDF 9 we have set out to keep entirely within the spirit and letter of the Report. The foundation of the scheme is "code procedures" with bodies in KDF 9 User Code. Code procedures are foreseen by the Report as follows:

(i) In the section on procedure declarations, a procedure body is defined syntactically thus:

⟨procedure body⟩::=⟨statement⟩|⟨code⟩ (5.4.1)

Semantically:

"5.4.6. Code as procedure body.

It is understood that the procedure body may be expressed in non-ALGOL language. Since it is intended that the use of this feature should be entirely a question of hardware representation, no further rules concerning this code language can be given within the reference language."

(ii) In the section on procedure statements:
"4.7.8 Procedure body expressed in Code.
The restrictions imposed on a procedure statement calling a procedure having its body expressed in non-ALGOL code evidently can only be derived from the characteristics of the code used and the intent of the user and thus fall outside the scope of the reference language."

(iii) Strings can be operated on only by procedures with bodies in non-ALGOL code (4.7.5.1).

The first point to notice is that it is only the procedure *body* which can be in non-ALGOL code; a code procedure has a normal ALGOL *heading*. Provided the *action* of a procedure is understood, it is of no consequence to the user whether the procedure body is in code or in ALGOL; the procedure heading contains its identifier, formal parameter list, and specification part. The call of a procedure is written in exactly the same way whether the body of the declaration is in code or in ALGOL. Thus no knowledge of "code" is required of KDF 9 ALGOL users.

On the other hand, in order to *write* code procedures one needs to know both KDF 9 User Code and ALGOL, but not—and this is the second point—any details of the inner workings of the ALGOL Compiler. In fact, this scheme is implemented with two entirely different compilers, so one cannot make use of the special properties of either. A code procedure can be accepted by either the Whetstone interpretive compiler or the Kidsgrove optimizing translator (see Duncan, 1962).

We have extended the User Code to allow references to ALGOL formal parameters within a code body, and both compilers have been provided with a mechanism for recognizing formal parameters and effecting actual-formal replacement.

Three forms of reference are permitted:

'*a*' : fetch the (current) value of *a* to the top cell of the nesting store; *a* is a formal parameter specified as a simple variable or array, called by name or value, or a string.

=*'a'* : assign the value in the top cell of the nesting store to *a*; *a* is specified as a simple variable called by name or value.

J '*a*' : jump to label *a*; *a* is specified as a label.

The actual-formal replacement mechanism is powerful enough to allow the call of expressions by name as in ALGOL proper (cf. example in Duncan, 1962).

The main restrictions on the code bodies are that formal parameters may not be specified as switches or procedures, and that communication between a code body and its ALGOL context is solely through the parameter list; there is no other access to non-local variables.

If this point of code procedures has been somewhat laboured, the excuse is that our input-output scheme is

founded on procedures, and is not a fixed part of the compiler. Consequently, the scheme can be extended or reduced without effect on the compiler, and this can be done by anyone with a knowledge of ALGOL and KDF 9 User Code. The procedures we provide with our ALGOL library, to be described later, are, we hope, adequate and convenient for most users; but those with special requirements have the means of setting up a scheme suited to their own needs. (A request for reading automatically generated tapes has already been made.)

3. KDF 9 input-output mechanisms

Before describing the procedures, the method of use of KDF 9 input–output equipment ("peripheral devices") should be sketched. (Fuller descriptions are available in English Electric reports.)

Whether on the single-program or time-sharing version of KDF 9, each program, including the ALGOL compiler and the object program, is controlled by a master program called the "director." The director is responsible, among other things, for controlling the operation of peripheral devices. In particular, it indicates to the operator which reader is to be used for reading in the program, and during execution it "allocates" to the program such peripheral devices as may be called for; in the case of magnetic-tape input it will search for a tape with a specified label ("identifier") and inform the program on which unit that tape is to be found.

It follows, therefore, that in using input-output devices on KDF 9 a program must:

(*a*) ask the director to "allocate" to it the appropriate devices;
(*b*) use these devices;
(*c*) after use, inform the director that these devices are no longer required.

In ALGOL terms, therefore, one must not only provide procedures for "actually doing the work," but also for obtaining and abandoning a particular device. Since there may be several devices of a particular kind, input-output procedures should normally have a parameter for the device number; but in special cases, for the convenience of the user, this may be absorbed, as it were, into the procedure identifier.

4. Standard format input-output procedures

There is clearly a need for simple procedures to read or print numbers without requiring a format to be specified.

The procedure for reading a single number in this way is substantially the same as that used in the compiler itself for reading an ALGOL number. It will read, for example, any of the numbers in section 2.5.2 of the Report, recognizing the end of a number by the appearance of some symbol other than a digit, sign, decimal point, or subscript. ten. It goes without saying that it checks that the combination of characters read is a meaningful number and within the range allowed by the implementation (for KDF 9, numbers of type **integer** have up to 40 bits, including sign; numbers of type **real** have an 8-bit exponent and 40-bit mantissa—the form required for floating-point operations).

The procedures for printing a single number give a number of type **integer** as a signed, 12-decimal-digit, integer, and a number of type **real** as a standard floating-decimal number of the form $c_{10}d$, where c is a signed number in the range $1 \leqslant c < 10$ to 11 decimal places, and d is a two-digit signed integer.

Each output number is terminated with a "new line" character. A sheet of results, therefore, has one number to a line. If it is required to read an output tape, the "new line" characters can serve as convenient separators.

With the input procedure existing both as an ordinary procedure (called by a statement) and as a function (called within an expression) we have available the input-output facilities of both the ALCOR-convention and the MC-translator.

5. Procedures allowing fine control over appearance of output

5.1. *The layout of a number*

The number of parameters needed to describe the appearance of a number on a printed sheet can easily be made to run into double figures.

For example: number of significant figures,
treatment of sign,
fixed or floating point, and if the latter,
treatment of exponent,
number of decimal places,
spacing of digits,
zero suppression,
termination, and so on.

However, a "picture" is more convenient to the user than a string of parameters.

We might have

'+*ddd*·*dddsddd*'

(where *d* stands for "digit" and *s* for "space").

Numbers printed with this format are:

+123·456 789
−001·234 567

Zero suppression is achieved by writing *n* as the leading digit:

'+*ndd*·*dddsddd*'

This gives: +123·456 789
−1·234 567
+0·123 456

Note that the sign is "shifted down" and that a non-significant zero in the units position is not suppressed. Other treatments of the sign are permitted: "−" causes + signs to be suppressed, and "±" causes the sign to appear always in the same column.

Returning to the example, one might want to print only a certain number of significant figures, say five:

'+*ndd*·*dd*0*s*000'

This gives:
+123·45
−12·345
+1·234 5
−0·123 45
−0·012 345
−0·001 234, etc.

For "floating-decimal" output, one would write a format with one digit before the decimal point and followed by an exponent part such as $_{10}+dd$ (it cannot have more than two digits).

One might wish to precede a number by a sequence of spaces, or to "terminate" it with, say, one or more "new line" characters, or a TAB character, or a semicolon, or a "new page" character. The layout can accommodate these requirements.

5.2. *Establishing a layout*

The scheme sketched above appears to be adequate for most situations, but it has been criticized on the ground that it needs too much writing—that it is absurd to write '*ddd . . .*' every time one writes an output statement. But, of course, one need not write '*ddd . . .*' every time. If a particular layout is standard in a program one can build it into a procedure thus:

procedure *OUT*(*x*); **value** *x*; **real** *x*;
write ('*ndd*·*ddddd*', *x*);

Then the statements *pi*:= 3·1415926536; *OUT* (*pi*);
would cause 3·14159 to be printed.

However, there is a strong case for separating the establishment of a layout from its use, not only for the programmer (as in the method just indicated), but for the computer at "run time." In our scheme this is achieved by means of a procedure for associating an integer with each format:

integer procedure *format* (*string*); **string** *string*;
comment *the value assigned to format defines the layout described by string in a form convenient for the write procedure*;
KDF 9 . . . . . ALGOL; . . .
*f*1 := *format* ('*dd*·*dddd*'); . . .
write (*f*1, *x*); *write* (*f*1, *y*); . . .

5.3. *Opening and closing "devices"*

The KDF 9 user may be concerned with monitor typewriter, paper-tape readers and punches, card readers and punches, printers, and magnetic-tape units in varying combinations. For ALGOL, each "device" has a two-digit number; the first digit represents the *class* of device (e.g. 2 for paper-tape reader), and the second is the programmer's "serial number within class"—the director determines which actual machine is given this number.

To get a device from the director, one should call the procedure "open":

procedure *open* (*device number*); **value** *device number*;
integer *device number*;
comment *makes the device available to the program, and reserves main store space for a double buffer. For an input device, fills the first half of the buffer*;
KDF 9 . . . ALGOL;

Magnetic tape is the exception to this rule (cf. section 3 above). A tape is obtained by writing the tape label as a string parameter for the tape-finding procedure.

To abandon a device to the director, one should call the procedure "close":

procedure *close* (*device number*); **value** *device number*;
integer *device number*;
comment *for an output device, puts out the current buffer contents and abandons the device*;
KDF 9 . . . ALGOL;

5.4. *The "read" and "write" procedures*

These are the procedures for "actually doing the work."

real procedure *read* (*device number*);
value *device number*;
integer *device number*;
comment *no format is specified—this is the same procedure as described in the previous section. It is called as a function designator to allow both reals and integers to be read by the same procedure. Thus*:
x := *read* (20) *and i* := *read* (20). *In the KDF 9 implementation no loss of digits occurs in transferring from* **real** *to* **integer** *if the number is within the limits specified*;
KDF 9 . . . ALGOL;

procedure *write* (*device number*, *layout*, *quantity*);
value *device number*, *layout*, *quantity*;
integer *device number*, *layout*; **real** *quantity*;
comment *layout is a value produced by a call of the integer procedure format. Quantity is an arithmetic expression. Transfer from* **integer** *to* **real** *is of course invoked automatically, according to the rules of ALGOL: in the KDF 9 implementation there is no loss of digits if the number is within the permitted range*;
KDF 9 . . . ALGOL;

6. Alarm printing

If the procedure "write" is presented with a number which cannot be printed within the specified layout, the current contents of the output buffer are printed, and then the offending quantity is printed according to the "standard" layout described earlier.

For example, suppose the layout '*ssssdddd*;' is used three times, then the layout '*ssssdddd*: *c*', and all this repeated. Since *c* means new line, the page layout is as follows:

1234; 5678; 9012; 3456;
7890; etc.

Now suppose the next number to be converted happens to be −1, instead of 1234.

The page now appears thus:

```
...1234;    5678;    9012;    3456;
..7890;
−1·00000000000₁₀ + 0
                     5678;    9012;
   3456;    7890;    1234;    5678; etc.
```

Note that the wrong number stands out from the correct results. After an alarm printing the normal printing is resumed in such a way that the grouping of the numbers on the page is as it would have been had there been no error.

7. Output to the different devices

The procedure "write," when sending output information to a printer, does as described for paper tape, it produces punchings which will give rise to the expected layout on paper. For magnetic tape, the output is as to the printer, since the printer can be run separately from magnetic tape.

Procedures for binary output to, and input from, magnetic tape are being provided.

8. The matrix scheme

Input and output procedures for matrices under control of the Denison Matrix Scheme (Denison, 1962) are quite distinct from the procedures described above. They are not discussed in the present paper.

9. A note on compatibility with other input-output systems

The facilities of the MC-translator and the ALCOR convention are already, as it were, contained within the KDF 9 scheme.

Programs written within Bottenbruch's system can be transferred simply for KDF 9 if one regards, e.g.

read (*a*, *b*, *c*, . . .);

as an abbreviation for, e.g.

a = *read* (20); *b*:= *read* (20); *c*:= *read* (20); . . .

(Bottenbruch states, section 39, that the ALGOL report allows code procedures to have variable numbers of parameters. Much as I would like to be able to interpret the Report in this way, it seems clear to me that the call of a code procedure must have the same number of parameters as its declaration—it is only the procedure body that is not in ALGOL!)

The scheme described by McCracken (1962), and the Elliott scheme (Hoare, 1962), would require transformation of a non-trivial nature.

Acknowledgements

The author would like to thank particularly Dr. Peter Naur and Professor W. L. van der Poel for helpful discussion and encouragement on this topic. (The DASK scheme was drawn upon shamelessly for the layout descriptions of section 5.1.) Discussion with Messrs. D. H. R. Huxtable, B. Randell, L. J. Russell, and L. R. Hodges, besides ensuring the feasibility of the scheme with the two different compilers, did much to improve it. Further discussion will no doubt allow further improvement.

The paper is published by permission of The English Electric Co. Ltd.

References

ALCOR Convention (1961). *Elektronische Rechenanlagen*, Vol. 3, p. 206.

Bottenbruch, H. (1962). "Structure and Use of ALGOL 60," *Journal of the ACM*, Vol. 9, No. 2, April 1962.

Denison, S. J. M. (1962). "A proposed ALGOL 60 matrix scheme," Paper presented at IFIP Congress 62, Munich, Sept. 1962.

Dijkstra, E. W. (1962). *A Primer of ALGOL 60 Programming*, Academic Press, London.

Duncan, F. G. (1962). "Implementation of ALGOL 60 for the English Electric KDF 9," *The Computer Journal*, Vol. 5, p. 130.

Hoare, C. A. R. (1962). "Report on the Elliott ALGOL Translator," *The Computer Journal*, Vol. 5, p. 127.

Jensen, J., Jensen, T., Mondrup, P., and Naur, P. (1960). *A Manual of the DASK ALGOL Language*, Regnecentralen, Copenhagen.

McCracken, D. (1962). *A Guide to ALGOL Programming*, John Wiley, New York.

Naur, ed. (1960). "Report on the Algorithmic Language ALGOL 60," Regnecentralen, Copenhagen; modified by Woodger, ed. (1962) "Supplement to the ALGOL 60 Report" in *ALGOL Bulletin* No. 15. Regnecentralen, Copenhagen. (Note: The Report as modified by the Supplement has appeared as an official IFIP publication, and is reproduced by permission on p. 349 of this issue of *The Computer Journal*.)