Algorithms

J. G. HERRIOT, Editor

ALGORITHM 335
A SET OF BASIC INPUT-OUTPUT PROCEDURES [I5]

R. De Vogelaere (Recd. 8 Sept. 1966 and 18 Nov. 1966; description revised 2 Nov. 1967)
Department of Mathematics and Computer Center, University of California, Berkeley, CA. 94720

By means of the primitives *insymbol, outsymbol* and *length*, as requested by this journal's Algorithms Policy [Comm. ACM 10 (Nov. 67), 729] a basic set of input-output procedures is defined aiming at quality and flexibility. *outreal*, for instance, is written as a derived procedure; it outputs using the fixed point or the floating point representation, and rounds properly. Variants can easily be written because of the explicit call of the procedures *decompose integer* and *decompose real*. The highly recommended practice of echoing input is made easy with one subset of derived procedures (*ioi, ior, iob, ioa*). The documentation of output in the form of equivalent ALGOL statements is also provided when use is made of the subset *oti, otr, otb, ota*. The Berkeley style of providing information on the form of output using prior calls of procedures such as *real format* is defined. A use of the parameter *outchannel* to provide information for simultaneous output to several channels is suggested. Interrelationship between the declared procedures is furnished in tabular form.

KEY WORDS AND PHRASES: input output, transput, input output procedures, input echo, quality output, decompose integer, decompose real, style, Berkeley style, procedures relationship, output documentation, equivalent ALGOL statements, ALGOL, ALGOL 60, integer format, real format, out integer, read real, input output Boolean, input output array, fixed point representation, floating point representation, output channel interpretation
CR CATEGORIES: 4.0, 4.41

1. Introduction

The reader will find below a set of basic input-output procedures. Let me state first some of the purposes for writing this set and give a general description and specific information about the procedures and their interrelationship.

In the October 1964 issue of the *Communications of the ACM* [1], a report on input-output procedures for Algol 60 was published. This report was prepared by a working group (WG 2.1) of the International Federation for Information Processing (IFIP/TC2) and approved by its Council.

The approved primitives were:

insymbol, outsymbol, length, inreal, outreal, inarray, outarray

In the examples the following derived procedures were defined:

outboolean, outstring, ininteger.

It is stated therein that "one needs, in practice, a fuller set of input-output procedures" and it is observed also that "different scheme of I/O procedures can be defined in it, largely by means of these primitives."

Since then, a few procedures have been published (see for instance [2, 3]) and the Algorithms Policy of this journal has requested [6] the use of the primitives of [1] and the use of *outboolean, outstring, ininteger* and *outinteger* for input-output.

The purpose of this algorithm is to present part of a consistent scheme of input-output procedures. The set uses as primitives, *insymbol, outsymbol*, and *outstring* (or equivalently *length*).

First *in integer, out integer, in real, out real, in Boolean, out Boolean* are derived. *in real* is related to [2]; *out integer* and *out real* call the more basic procedures *decompose integer* and *decompose real*. *out real* allows not only for floating point representation [3] but also for fixed point representation and for correct rounding.

Several sets of procedures, which point in several directions and which call the more basic ones, are then introduced. One set consists of parameterless input function designators akin to the **procedure** *read* of the Amsterdam Mathematisch Centrum. One set provides for echo of input to insure that the correct numbers have been read in—a practice which I recommend highly; it also provides for easy documentation of the output in the form of equivalent Algol statements. Another set with the same documentation feature is for output only; the last set outputs numbers, but no text.

It is not suggested that the set of procedures of this algorithm be used for quantity output. Its main purpose is for quality output.

2. General Description

2.1. The only primitives used are *insymbol, outsymbol*, and *length* (through *outstring*). *insymbol* and *outsymbol* assume that the value -1 is associated with the symbol carriage return-line feed (or new card), which is not a basic symbol of Algol 60. This is done in accordance with the convention of [1, Sec. 3]. *outstring* could have been avoided with some loss of clarity in the description of the procedures. *insymbol, outsymbol*, and *outstring* are defined in [1].

inreal and *outreal* are defined as in [2, 3] in terms of *insymbol, outsymbol*, and *outstring*. I do not believe that *inreal* and *outreal* should be primitives, firstly, because these procedures can be defined in terms of other primitives, and secondly, because many definitions will satisfy the requirements of [1]. On the other hand, the requirements set forth in [1] are most desirable.

in channel and *out channel* must be declared as integers and assigned a value in accordance with the requirements of *insymbol* and *outsymbol* [1].

I would like to observe in passing that the integer *out channel* cannot only be interpreted as identifying a single channel, but can also be interpreted as identifying a set of channels to all of which the output is to be sent. (If the binary representation of *out channel* is $\sum a[i] \times 2 \uparrow i$, the output is sent to channel i if $a[i] = 1$ and is not sent if $a[i] = 0$.) Although this is not yet implemented at Berkeley in this fashion, all output going to a terminal is now also sent to the printer. When time-sharing becomes widespread this interpretation will, I hope, be increasingly popular.

2.2. The more basic input-output procedures are *in integer*,

in real, and *in Boolean*; the first two use *in symbol* only through the integer procedure *symbol*.

symbol recognizes only the following basic symbols:

$$0|1|2|3|4|5|6|7|8|9|\cdot|-|+|_{10}|,|\sqcup$$

and carriage return-line feed (or new card).

in integer associates to the second parameter, which is of type integer, the next integer read from *channel* (the first parameter). Any number of consecutive spaces are ignored before the first digit; after the first digit, termination occurs with two consecutive spaces, a comma, or a carriage return-line feed. A comma before the first digit or sign, a period, $\langle_{10}\rangle$, or any other illegal symbol will call the procedure *error*.

in real associates to the second parameter, which is of type real, the next real number read from *channel* (the first parameter). Any number of consecutive spaces are ignored before the first digit, period, or $\langle_{10}\rangle$; after that, termination occurs with two consecutive spaces, a comma, or a carriage return-line feed. A comma before the first digit, sign, period, or $\langle_{10}\rangle$, or any other illegal symbol will call the procedure *error*. Communication between *in integer*, *in real*, and *in symbol* to take care of separation between integers or reals requires the nonlocals *z8100b* and *z8100bc*.

in Boolean associates to the second parameter, which is of type Boolean the next Boolean read from *channel* (the first parameter); any number of leading spaces or carriage returns-line feed are ignored; any illegal symbol will call the procedure *error*.

The procedure *error* has one parameter of type integer. It can be written according to the wishes of a user or of a group of users. An example with diagnostics in full is given below.

2.3. The more basic output procedures are *out integer*, *out real*, and *out Boolean*. The information on the form of the output can be given in various ways; the style used for these output procedures is what I will call the Berkeley style by contrast with the style used for output procedures at, for instance, the Amsterdam's Mathematisch Centrum or at Copenhagen's Regnecentralen. Call of these output procedures must be preceded by a call of corresponding procedures *integer format*, *real format* and *Boolean format*.

The only parameter of *integer format* determines the field width of any integer sent to the output channel. The parameters of *real format* are a Boolean, which determines when the value is **true** that fixed point representation is desired for the output of real numbers and when the value is **false** that floating point representation is desired. The second parameter determines the field width, the third parameter determines the number of decimal places and affects also the rounding of the number. The only parameter of *Boolean format* determines the field width.

The following decisions were made for *out integer*, *out real*, and *out Boolean:* If the field parameter is less than required, it is replaced by 20. The sign is outputed before the most significant digit if the number is negative. In floating point form, the first significant digit is immediately to the left of the decimal point. The exponent is replaced by four spaces if it is zero; otherwise the sign of the exponent is always outputed and the exponent is restricted to the interval -99 to 99.

If the user wishes to write variants of the Berkeley style, for instance if he wishes always to print the sign, or if he wishes to output it as the first character of the field, or if he wishes to output a space between every third or fifth digit, his task will be greatly eased by the introduction of the procedures *decompose integer* and *decompose real* which provide the basic information about an integer (its sign, the number of significant decimal digits, and the digits) or about a real (its sign, its size, the scale factor such that the scaled number has its first significant digit immediately to the left of the decimal point and the digits).

In *decompose real*, the size information determines if the number is too small; an integer declaration has been chosen instead of a Boolean to provide for the possibility of another test, which would determine if the number is too large. The rounding for reals is taken care of in *decompose real*.

Correct rounding is essential for a set of input-output procedures of quality. Although the point may be argued, I consider incorrect the output of 2 to two decimals as 1.99 unless computer or computations have only that precision. Examples:

real format (**true**, 5, 3); *out real* (1, 0.99099);

real format (**false**, 10, 2); *out real* (1, -0.99099);

will output

$0.991\text{–}9.91_{10} - 1.$

2.4. Four more sets of input-output procedures follow; these procedures do not require explicit calls of the format procedures:

read i, *read r*, *read b* are function designators without parameters which can be used to input respectively an integer, a real or a Boolean.

ioi, *ior*, *iob* are function designators and *ioa* is a procedure to input respectively an integer, a real, a Boolean or a real array and to output an equivalent ALGOL statement.

This style, which I have introduced to give the output in the form of parts of an ALGOL program in connection with the generation of the nonlinear equations satisfied by Runge-Kutta type methods (to be published elsewhere), can also be used to describe input and output within the conventions of the ALGOL language.

For *ioi*, *ior*, *iob*, the second parameter gives the string to be outputted; the others give the parameters corresponding to those of the format procedures. For *ioa*, the second and third parameters are the first and last subscript of the element of the one dimensional array to be read and the last parameters give the string to be outputted as well as the format information. Examples:

ior(*r*, *'time␣in␣minutes'*, **true**, 5, 2);

ioa(*a*, 1, 3, *'hippopotamus'*, **true**, 4, 1)

would output with appropriate input:

time in minutes := 21.05;

i := 1; **for** *hippopotamus* [*i*] := 15.1, 6.2, 7.0 **do** *i* := *i* + 1;

The next four procedures *oti*, *otr*, *otb*, and *ota* are for output only; the form of output is identical to that of *ioi*, *ior*, *iob*, and *ioa*.

The last four procedures *outi*, *outr*, *outb*, and *outa* are for output only. They output an integer, a real, a Boolean, or a sequence of reals, the format information being provided by the parameters of these procedures.

3. Specific Information About Procedures, Their Relationship, and the Nonlocal Parameters

To ease the local exchange of procedures and nonlocal identifiers of procedures between people at Berkeley, conventions have been introduced which are examplified in the procedures of this algorithm. All appropriate nonlocal identifiers are formed using as first symbols the letter *z* followed by a digit associated to the writer (I use 8) followed by 3 digits corresponding to the number of the procedure in which the nonlocal identifier is first used (my procedure *symbol* is number 100, *in integer* is number 101, etc.) followed by an ordinary identifier.

The following declarations must be made in the same block as that of this algorithm or in an outer block:

integer *in channel*, *out channel*, *z8106n*, *z8107n*, *z8107d*, *z8108n*;
Boolean *z8100b*, *z8100bc*, *z8107B*;
procedure *in symbol* (*channel*, *string*, *destination*); (see *Comm. ACM* 7 (Oct. 1964), 628–630)
procedure *out symbol* (*channel*, *string*, *destination*); (Idem)
procedure *out string* (*channel*, *string*); (Idem)

in channel and *out channel* must be assigned an appropriate value before a call of many of the input-output procedures (see Table I).

Table I indicates the relationship between the procedures and the nonlocal variables. Moreover, an explicit call of *out integer*, *out real*, and *out Boolean* requires a preceding call of the corresponding format procedure *integer formal*, *real format*, and *Boolean format*.

TABLE I. Relationship Between Procedures and Nonlocal Variables

File	Procedure	Number	*error*	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	*in channel*	*out channel*	file z8100						
																					z8100b	*z8100bc*	*z8106n*	*z8107n*	*z8107d*	*z8107B*	*z8108n*
	error		0																								
	in symbol	97		0																							
z8096	*out symbol*	98			0																						
	out string	99				0																					
	symbol	100	×	×			0												×		×	×					
	in integer	101	×	+			×	0											×		×	×					
z8100	*in real*	102	×	+			×		0										×		×	×					
	in Boolean	103	×	×						0																	
z8104	*decompose integer*	104									0																
	decompose real	105										0															
	integer format	106											0										×				
z8106	*real format*	107												0										×	×	×	
	Boolean format	108													0												×
z8106	*out integer*	109			×	×					×					0							×				
z8110	*out real*	110			×	×						×					0							×	×	×	
z8110	*out Boolean*	111			×	×												0									×
	read i	112	+	+			+	×											×		+	+					
z8112	*read r*	113	+	+					×										×		+	+					
	read b	114	+	+						×									×								
	ioi	115	+	+	+	×	+	×			+		+			×			×	×	+	+	+				
z8112	*ior*	116	+	+	+	×	+		×			+		+			×		×	×	+	+		+	+	+	
	iob	117	+	+	+	×				×					+			×	×	×								+
	ioa	118	+	+	+	×	+		×	×		+		+			×		×	×	+	+		+	+	+	
	oti	119			+	×					+		×			×				×			+				
z8119	*otr*	120			+	×						+		×						×				+	+	+	
	otb	121			+	×									×					×							+
	ota	122			+	×						+		×						×				+	+	+	
	outi	123			+	+					+		×			×				×			+				
z8119	*outr*	124			+	+						+		×			×			×				+	+	+	
	outb	125			+	+									×			×		×							+
	outa	126			+	+						+		×			×			×				+	+	+	

In Table I, each of the procedures is identified by a number. An × indicates that the procedure corresponding to the number in the same column or the nonlocal identifier on top of the same column is used explicitly (and perhaps also implicitly); + indicates that the corresponding procedure or identifier is used implicitly; 0 is placed in the column corresponding to the number of the procedure. Related procedures are grouped together in a file whose name appears in the first column. This information will be used in further publications.

The following declaration can be used for the **procedure** *error*:

```
procedure error (i); value i; integer i;
begin procedure nlcr; outsymbol (channel, ' ', -1);
      nlcr;
      if i = 8100 then out string (1,'a␣symbol␣is␣read␣which␣is␣not␣a␣digit␣·␣,␣
        -␣+␣10␣(space)␣carriage␣return-line␣feed') else
      if i = 810100 then out string (1,'while␣reading␣an␣integer,␣an␣illegal␣symbol␣
        is␣read␣before␣the␣first␣digit') else
      if i = 810101 then out string (1,'while␣reading␣an␣integer,␣an␣illegal␣symbol␣
        is␣read␣after␣the␣first␣digit') else
      if i = 810200 then out string (1,'while␣reading␣a␣real,␣an␣illegal␣symbol␣
        is␣read␣while␣reading␣the␣decimal␣fraction') else
      if i = 810201 then out string (1,'while␣reading␣a␣real,␣an␣illegal␣symbol␣is␣
        read␣before␣the␣first␣digit␣period␣or␣10') else
      if i = 810202 then out string (1,'while␣reading␣a␣real,␣an␣illegal␣symbol␣is␣
        read␣while␣reading␣the␣exponent␣part') else
      if i = 810203 then out string (1,'a␣real␣number␣is␣improperly␣terminated')
        else
      out string (1,'while␣reading␣a␣Boolean␣a␣symbol␣which␣is␣not␣true␣or␣false,
        is␣read␣before␣termination');
      nlcr
end error
```

Acknowledgment. The implementation of the procedures in this paper has been made possible by the existence of an ALGOL interpreter, which is the responsibility of many (see [4]). The editor, Q.E.D., used to prepare the program on the SDS 930, has been planned and implemented by Peter Deutsch and Butler Lampson. I especially thank Mr. Deutsch for the inclusion of requested features to copy part of a line until a given character noninclusive and to delete part of a line until a given character noninclusive. I thank my colleague R. S. Lehman for the use of his syntax checker and transliterator to BC-ALGOL.

Machine time for the preparation and implementation of the procedures and their tests was furnished by Project Genie of the Computer Center operating under Contract SD-185 with the Advanced Research Project Agency and by the Berkeley Campus Committee on Research.

REFERENCES

1. Report on input-output procedures for ALGOL 60. *Comm. ACM 7* (Oct. 1964), 628–630.
2. MCKEEMAN, W. M. Algorithm 239, Free Field Read. *Comm. ACM 7* (Aug. 1964), 481.
3. WIRTH, N. E. Algorithm 249, Outreal *n*. *Comm. ACM 8* (Feb. 1965), 104.
4. BC ALGOL Manual. U. of California, Computer Center, Berkeley, Oct. 1966 (Third Ed.).
5. ANGLUIN, D. C., DEUTSCH, L. P. Reference manual, Q.E.D., time-sharing editor. Doc. 30.60.30, Jan. 26, 1967, Contract SD-185, Office of the Secretary of Defense, ARPA, Washington, D. C.
6. Revised Algorithms Policy. *Comm. ACM 7* (Oct. 1964), 586.

```
integer procedure symbol(s); integer s;
comment symbol := s := the integer representation of the
  next symbol read, 0 to 9 for the integers, 10 for '·', 11 for
  '−', 12 for '+', 13 for '10', and 14 for ',' or for carriage
  return (or new card) represented by −1 when processed by
  in symbol or for two consecutive spaces when the nonlocal
  Boolean z8100b is false. When z8100b is true any number of
  consecutive spaces are ignored. Any other symbol will call a
  nonlocal procedure error with parameter equal to 8100;
begin
read: in symbol(in channel, '0123456789.−+10␣,', s);
  if s = −1 ∧ z8100bc then go to read;
  if s = 15 then
  begin
    if z8100b then go to read
    else in symbol(in channel, '0123456789.−+10␣,', s)
  end;
  if s = −1 ∨ s = 16 then symbol := s := 14
  else
  begin if s ≦ 0 then error(8100); symbol := s := s − 1 end
end symbol;
procedure in integer(channel, i); value channel;
  integer channel, i;
comment i := the next integer read from channel, any number of
  consecutive spaces are ignored before the first digit, after the
  digit termination occurs with two consecutive spaces, a comma
  or a carriage return, any illegal symbol will call a nonlocal
  procedure error with parameter equal to 8100 or 810100 or
  810101;
begin
  integer s; Boolean negative;
  negative := false; z8100b := z8100bc := true;
    in channel := channel;
  symbol(i); z8100bc := false;
  if i = 12 then symbol(i)
  else if i = 11 then begin negative := true; symbol(i) end;
  if i ≧ 10 then error(810100);
  z8100b := false;
L1: if symbol (s) < 10 then begin i := 10 × i + s; go to L1 end;
  if   s ≠ 14 then error(810101);
  if negative then i := −i
end in integer;
procedure in real(channel, r); value channel;
  integer channel; real r;
comment r := the next real number read from channel, any number
  of consecutive spaces are ignored before the first digit.
  After the first digit termination occurs with two consecutive
  spaces, a comma or a carriage return. Any illegal symbol will
  call a non local procedure error with paramater equal to 8100
  or 810200 or 810201 or 810202 or 810203. The main differences
  with ALGORITHM 239 of W. M. McKeeman [2] are the substi-
  tution of his integer procedure CHAR by symbol, the introduc-
  tion of the Boolean z8100b, the introduction of a parameter in
  the nonlocal procedure error and the change of type of a few
  declarations;
begin
  real sig, fp, d, ep, ip; integer esig, ch;
  real procedure unsigned integer;
  begin
    real u;
    u := ch;
K: if symbol(ch) < 10 then begin u := u × 10 + ch; go to K end;
      unsigned integer := u
  end unsigned integer;
  sig := 1.0; ep := fp := 0; z8100b := z8100bc := true;
  in channel := channel;
  symbol(ch); z8100bc := false;
  if ch = 12 then symbol(ch)
  else if ch = 11 then begin sig := −1.0; symbol(ch) end;
  z8100b := false;
  if ch ≦ 10 then
  begin
    ip := if ch < 10 then unsigned integer else 0;
    if ch = 10 then
    begin
      if symbol(ch) ≧ 10 then error(810200);
      fp := 0; d := 0.1;
```

```
M:   fp := fp + ch × d;  d := d × 0.1;
     if symbol(ch) < 10 then go to M
   end decimal fraction
  end decimal number
  else if ch = 13 then ip := 1
  else begin error(810201);  ip := 1 end;
  if ch = 13 then
  begin esig := 1;
    if symbol(ch) = 12 then symbol(ch)
    else if ch = 11 then begin esig := −1;  symbol(ch) end;
    if ch < 10 then ep := unsigned integer × esig
    else begin error(810202);  ep := 0 end
  end exponent part;
  if ch ≠ 14 then error(810203);
  r := sig × (ip+fp) × 10.0 ↑ ep
end in real;
procedure in Boolean(channel, b);  value channel;
  integer channel;  Boolean b;
comment b := the next Boolean read from channel, any number
  of spaces or carriage returns are ignored, any other symbol will
  call a nonlocal procedure error with parameter equal to 8103;
begin
  integer i;
L:in symbol(channel, 'true falseu', i);
  if i = 3 ∨ i = −1 then go to L;
  if i ≦ 0 then error(8103);
  b := i = 1
end in Boolean;
procedure decompose integer(i, negative, n of digits, digit);
  value i;  integer i, n of digits;  Boolean negative;
  integer array digit;
comment negative := i < 0, n of digits := the number of decimal
  digits of i (if i = 0 then n of digits := 0), digit [0: n of digits − 1]
  := the decimal digits of i starting from the right;
begin
  integer j;
  if i < 0 then begin negative := true;  i := −i end
  else negative := false;
  n of digits := 0;
L:
  if i > 0 then
  begin
    j := i ÷ 10;  digit[n of digits] := i − j × 10;
    n of digits := n of digits + 1;  i := j;  go to L
  end
end decompose integer;
procedure decompose real(r, max n of digits, negative, size, exponent,
  digit);
  value r;  integer max n of digits, size, exponent;  real r;
  Boolean negative;  integer array digit;
comment negative := r < 0, size := −1 if r is too small, i.e. is
  such that when abs(r) is multiplied repeatedly by 10 it does
  not become eventually larger than one, size := 0 otherwise,
  exponent := the power of 10 by which r is to be divided to ob-
  tain a number whose first significant digit is immediately to
  the left of the decimal point, digit [0: max n of digits − 1] :=
  the decimal digits of r starting with the first significant digit
  to the left;
begin
  integer i, k, m;
  Boolean procedure too small(r);  real r;
    too small := abs(r) < 2 ↑ (−127);
  comment this procedure should be replaced appropriately;
  negative := false;
  if too small (r) then
  begin size := 1;  go to end decompose end
  else size := 0;
  if r < 0 then begin negative := true;  r := −r end;
  if r < 1 then
  begin
    exponent := −1;
scale up: r := r × 10;
    if r < 1 then
    begin exponent := exponent − 1;  go to scale up end
  end
  else
  begin
    exponent := 0;
test:
  if r ≧ 10 then
    begin exponent := exponent + 1;  r := r × 0.1;
      go to test end
  end;
  m := max n of digits;
  r := r + 5 × 0.1 ↑ m;
  i := entier(r);
  if i = 10 then
  begin
    i := 1;  exponent := exponent + 1;  m := m + 1;  r := r/10
  end
  else if i = 0 then i := 1;
  digit[0] := i;
  for k := 1 step 1 until m − 1 do
  begin
    r := (r−i) × 10;  i := entier(r);
    i := digit[k] := if i ≦ 0 then 0 else if i = 10 then 9 else i
  end;
end decompose:
end decompose real;
procedure integer format(n);  integer n;  z8106n := n;
procedure real format(B, n, d);  integer n, d;  Boolean B;
begin
  z8107B := B;  z8107n := n;  z8107d := d
end real format;
procedure Boolean format(n);  integer n;  z8108n := n;
procedure out integer(channel, i);  value channel, i;
  integer channel, i;
comment the style of this procedure and of the out real and out
  Boolean procedures given below is what I will call the Berkeley
  style by contrast with that used for output procedures at the
  Amsterdam Mathematisch Centrum or at the Copenhagen
  Regnecentralen, for instance. It is characterized by the use of
  a field width parameter n and for real numbers, by the use of a
  parameter B which decides if the fixed point (value true)
  or the floating point representation (value false) is requested
  and by the number of digits d after the decimal point. The
  sign is outputed just before the most significant digit, if the
  number is negative. In floating point form the first significant
  digit is immediately to the left of the decimal point. If the
  field parameter is less than required, it is replaced by 20. These
  procedures pair with the corresponding input procedures if the
  field width is at least two units greater than required;
begin
  integer n of digits, j, k;  Boolean negative;
  integer array digit[0: 19];
  decompose integer(i, negative, n of digits, digit);
  if n of digits = 0 then
  begin n of digits := 1;  digit[0] := 0 end;
  j := n of digits + (if negative then 1 else 0);
  for k := (if j>z8106n then 19 else z8106n−1)
    step −1 until j do out string(channel, 'u');
  if negative then out string(channel, '−');
  for k := n of digits −1 step −1 until 0 do
    out symbol(channel, '0123456789', digit[k]+1)
end out integer;
```

```
procedure out real(channel, r); value channel, r;
  integer channel; real r;
comment this procedure outputs r properly rounded to channel
  using the Berkeley style. In this variant, the exponent part
  in the floating point form is replaced by 4 spaces if the exponent
  is zero. The sign of the exponent is always outputed, for com-
  patibility with in real. The exponent is restricted to the interval
  −99 to 99;
begin
  integer j, k, size, exponent; Boolean negative;
  integer array digit[0: z8107d+1+(if z8107B then
    entier(ln(abs(r)+1)×0.4343) else 0)];
  procedure out digit(d); integer d;
  begin
    out symbol(channel, '0123456789', d+1)
  end out digit;
  if z8107B then
  begin
    decompose real(r, if z8107d+exponent≦0 then 1 else 1+
    z8107d+ exponent, negative, size, exponent, digit);
    if size = −1 then
    begin
      exponent := if z8107d = 0 then 0 else −z8107d − 1;
      digit[0] := 0
    end
    else if z8107d = 0 ∧ exponent < 0 then
    begin exponent := 0; digit[0] := end;
    j := (if negative then 3 else 2) +
      (if z8107d = 0 then −1 else z8107d) +
      (if exponent ≧ 0 then exponent else −1);
    for k := (if j>z8107n then 19 else z8107n−1) step −1
      until j do out string(channel, '␣');
    if negative then out string (channel, '−');
    for k := 0 step 1 until exponent do
      out digit(digit[k]);
    if z8107d > 0 then
    begin
      out string(channel, '·');
      for k := exponent + 1 step 1 until exponent + z8107d do
      if k < 0 then out string(channel, '0') else out digit(digit[k])
    end
  end fixed point representation
  else
  begin
    decompose real(r, z8107d+1, negative, size, exponent, digit);
    if size = −1 then
    begin
      exponent := 0;
      for k := 0 step 1 until z8107d do digit[k] := 0
    end;
    j := 6 + (if z8107d=0 then −1 else z8107d)+
      (if negative then 1 else 0);
    for k := (if j>z8107n then 19 else z8107n−1)
      step −1 until j do
      out string(channel, '␣');
    if negative then out string(channel, '-');
    out digit (digit [0]);
    if z8107d ≠ 0 then out string(channel, '·');
    for k := 1 step 1 until z8107d do out digit(digit[k]);
    if exponent = 0 then out string(channel, '␣␣␣␣')
    else
    begin
      out string(channel, '₁₀');
      comment This procedure assumes that ₁₀ takes one space,
        if not, the preceding statement should be modified;
      if exponent ≧ 0 then out string(channel, '+')
      else
      begin out string(channel, '−');
        exponent := −exponent
      end;
      j := exponent ÷ 10;
      if j = 0 then out string(channel, '␣')
      else out digit(j);
      out digit(exponent−j×10)
    end
  end floating point representation
end out real;
procedure out Boolean(channel, b); value channel;
  integer channel; Boolean b;
begin
  integer k, j;
  j := if b then 4 else 5;
  comment this procedure assumes that true and false take
    respectively 4 and 5 spaces, if not the preceding statement
    should be modified;
  for k := (if j>z8108n then 19 else z8108n−1) step −1 until
    j do out string(channel, '␣');
  out symbol(channel, 'true false', j−3)
end out Boolean;
integer procedure read i;
begin
  integer i;
  in integer(in channel, i); read i := i
end read i;
real procedure read r;
begin
  real r;
  in real(in channel, r); read r := r
end read r;
Boolean procedure read b;
begin
  Boolean b;
  in Boolean(in channel, b); read b := b
end read b;
integer procedure ioi(i,s,n); string s; integer i, n;
comment this and the next 3 procedures input respectively an
  integer, a real number, a Boolean or a one dimensional array,
  they output an equivalent Algol statement;
begin
  out string(out channel, s); out string(out channel, '␣ := ␣');
  in integer(in channel, i); ioi := i;
  integer format(n); out integer(out channel, i);
  out string(out channel, ';␣')
end ioi;
real procedure ior(r, s, B, n, d);
  real r; string s; Boolean B; integer n, d;
begin
  out string(out channel, s);
  out string(out channel, '␣ := ␣');
  in real(in channel, r); ior := r;
  real format(B, n, d); out real(out channel, r);
  out string(out channel, ';␣')
end ior;
Boolean procedure iob(B, s, n); Boolean b; string s;
  integer n;
begin
  out string(out channel, s);
  out string(out channel, '␣ := ␣');
  in Boolean(in channel, B); iob := B;
  Boolean format(n); out Boolean(out channel, B);
  out string(out channel, ';␣')
end iob;
procedure ioa(a, l, u, s, B, n, d);
  integer l, u, n, d; array a; string s; Boolean B;
```

```
begin
  integer i;
  if l > u then go to end ioa;
  real format(B, n, d); oti(l, 'i', 3);
  out string(out channel, '␣for␣');
  out string(out channel, s);
  out string(out channel, '[i]␣ := ␣');
  for i := l step 1 until u do
  begin
    in real(in channel, a[i]); out real(out channel, a[i]);
    if i < u then out string(out channel, ',␣')
    else out string (out channel, '␣do␣i␣ := ␣i␣+␣1;␣')
  end;
end ioa:
end ioa;
procedure oti(i, s, n); value i, n; integer i, n; string s;
comment this and the following 3 procedures output Algol
  statements compatible with those of the input output procedures
  ioi, ior, iob, ioa;
begin
  out string(out channel, s);
  out string(out channel, '␣ := ␣');
  integer format(n); out integer(out channel, i);
  out string(out channel, ';␣')
end oti;
procedure otr(r, s, B, n, d);
  real r; string s; Boolean B; integer n, d;
begin
  out string(out channel, s);
  out string(out channel, '␣ := ␣');
  real format(B, n, d); out real(out channel, r);
  out string(out channel, ';␣')
end otr;
procedure otb(B, s, n); Boolean B; string s; integer n;
begin
  out string(out channel, s);
  out string(out channel, '␣ := ␣');
  Boolean format(n); out Boolean(out channel, B);
  out string (out channel, '; ␣')
end otb;
procedure ota(a, l, u, s, B, n, d);
  integer l, u, n, d; array a; string s; Boolean B;
begin
  integer i;
  if l > u then go to end ota;
  real format(B, n, d); oti(l, 'i', 3);
  out string(out channel, '␣for␣');
  out string(out channel, s);
  out string(out channel, '[i]␣ := ␣');
  for i := l step 1 until u do
  begin
    out real(out channel, a[i]);
    if i < u then out string(out channel, ',␣')
    else out string (out channel, '␣do␣i␣:=␣i␣+␣1;␣')
  end;
end ota:
end ota;
procedure outi(i, n); integer i, n;
comment this and the following 3 procedures output integers,
  real numbers, Booleans or one dimensional arrays using format
  as indicated in out integer;
begin
  integer format(n);
  out integer(out channel, i)
end outi;
procedure outr(r, B, n, d); real r; Boolean B; integer n, d;
begin
  real format(B, n, d);
  out real(out channel, r)
end outr;
procedure outb(B, n); Boolean b; integer n;
begin
  Boolean format(n);
  out Boolean(out channel, B)
end outb;
procedure outa(a, l, u, B, n, d); integer l, u, n, d; array a;
  Boolean B;
begin
  integer i;
  if l > u then go to end outa;
  real format(B, n, d);
  for i := l step 1 until u do out real(out channel, a[i]);
end outa:
end outa
```

REMARK ON ALGORITHM 217 [H]
MINIMUM EXCESS COST CURVE [William A. Briggs, *Comm. ACM 6* (Dec. 1963), 737]

JOHN F. MUTH (Recd. 26 Dec. 1967)
Michigan State University, East Lansing, MI 48823

KEY WORDS AND PHRASES: critical path scheduling, PERT, cost/time tradeoffs, network flows

CR CATEGORIES: 3.59, 5.41.

Algorithm 217 was transliterated into FORTRAN and successfully run on the CDC 3600 system at Indiana University after the following changes were made:

(1) In the first Boolean expression of the program the term:
$J[m] \geq J[m+1]$
was replaced by the term:
$(I[m] = I[m+1] \wedge J[m] \geq J[m+1])$

(2) The line:
*A*3: *labl*[*J*[*m*], 2] := *lex*;
was replaced by:
*A*3: *labl*[*J*[*m*], 3] := *lex*;

(3) In the statement labeled *B*1, the symbols:
[*m*,2] = 0
were replaced by:
f[*m*, 2] = 0

(4) Two statements before the statement labeled A was replaced by
*ntv*1 := *ntv* := *ord* := 0
where *ntv*1 was an additional integer variable. The third statement before *ANS* was replaced by:
ord := (*tb-node*[*sink*]) × *ntv*1 + *ord*; *ntv*1 := *ntv*;