Algorithms

J. G. HERRIOT, Editor

Algorithms Policy • Revised August, 1966

A contribution to the Algorithms Department should be in the form of an algorithm, a certification, or a remark. Contributions should be sent in duplicate to the editor, typewritten double spaced. Authors should carefully follow the style of this department with especial attention to indentation and completeness of references.

An algorithm must normally be written in the ALGOL 60 Reference Language [*Comm. ACM 6* (Jan. 1963), 1-17] or in ASA Standard FORTRAN or Basic FORTRAN [*Comm.* ACM 7 (Oct. 1964), 590-625]. Consideration will be given to algorithms written in other languages provided the language has been fully documented in the open literature and provided the author presents convincing arguments that his algorithm is best described in the chosen language and cannot be adequately described in either ALGOL 60 or FORTRAN.

An algorithm written in ALGOL 60 normally consists of a commented procedure declaration. It should be typewritten double spaced in capital and lower-case letters. Material to appear in **boldface** type should be underlined in black. Blue underlining may be used to indicate italic type, but this is usually best left to the Editor. An algorithm written in FORTRAN normally consists of a commented subprogram. It should be typewritten double spaced in the form normally used for FORTRAN or it should be in the form of a listing of a FORTRAN card deck together with a copy of the card deck. Each algorithm must be accompanied by a complete driver program in its language which generates test data, calls the procedure, and produces test answers. Moreover, selected previously obtained test answers should be given in comments in either the driver program or the algorithm. The driver program may be publishedwith the algorithm if it would be of major assistance to a user.

For ALGOL 60 programs, input and output should be achieved by procedure statements, using any of the following eleven procedures (whose body is not specified in ALGOL) [See "Report on Input Output Procedures for ALGOL 60," *Comm. ACM 7* (Oct. 1964), 628-629]:

insymbol	*inreal*	*outarray*	*ininteger*
outsymbol	*outreal*	*outboolean*	*outinteger*
length	*inarray*	*outstring*	

If only one channel is used by the program for output, it should be designated by 1 and similarly a single input channel should be designated by 2. Examples:

outstring (1, '*x*='); *outreal* (1,*x*);
for *i* := 1 **step** 1 **until** *n* **do** *outreal* (1,*A*[*i*]);
ininteger (2, *digit* [17]):

For FORTRAN programs, input and output should be achieved as described in the ASA preliminary report on FORTRAN and Basic FORTRAN.

It is intended that each published algorithm be well organized, clearly commented, syntactically correct, and a substantial contribution to the literature of Algorithms. It is necessary but not sufficient that a published algorithm operate on some machine and give correct answers. It must also communicate a method to the reader in a clear and unambiguous manner. All contributions will be refereed both by human beings and by an appropriate compiler. Authors should pay considerable attention to the correctness of their programs, since referees cannot be expected to debug them.

Certifications and remarks should add new information to that already published. Readers are especially encouraged to test and certify previously uncertified algorithms. Rewritten versions of previously published algorithms will be refereed as new contributions and should not be imbedded in certifications or remarks.

Galley proofs will be sent to authors; obviously rapid and careful proofreading is of paramount importance.

Although each algorithm has been tested by its author, no liability is assumed by the contributor, the editor, or the Association for Computing Machinery in connection therewith.

The reproduction of algorithms appearing in this department is explicitly permitted without any charge. When reproduction is for publication purposes, reference must be made to the algorithm author and to the *Communications* issue bearing the algorithm.—J.G.Herriot

ALGORITHM 309
GAMMA FUNCTION WITH ARBITRARY PRECISION [S14]

Antonino Machado Souza Filho and Georges Schwachheim (Recd. 12 Apr. 1966 and 14 Apr. 1967)
Centro Brasileiro de Pesquisas Fisicas, Rio de Janeiro, ZC82, Brazil

procedure *gamma*(*z*,*y*,*msize*,*error*);
  value *z*, *msize*; **real** *z*; **integer** *msize*; **label** *error*;
comment This procedure computes the value *y* of the gamma function for any real argument *z* for which the result can be represented within the computer, working with *msize* decimal digits. An exit is made thru the label *error* when the argument is a pole or is too large, while a zero result is returned when the argument is too small for a correct internal representation of the result.

This procedure is especially useful for variable field length computers and for double- or multiple-precision computations, when a simple power series algorithm is no longer applicable.

It computes the gamma function thru the Stirling asymptotic series for the logarithm of the gamma function with an argument increased by an appropriate integer to insure the required precision with the least computation work.

Negative arguments are reduced to positive ones by:

$$\Gamma(z) = \frac{\pi}{\sin(\pi z) \times \Gamma(1-z)}$$

This procedure is not recursive and uses no own variable. It was translated to Fortran II and run on an IBM 1620. The errors were at most of a few hundred units in the last digit of the mantissa, being due to the use of logarithms;

begin
  real procedure *loggamma* (*t*); **value** *t*; **real** *t*;
  comment The *loggamma* auxiliary procedure computes the logarithm of the gamma function of a positive argument *t*. If its argument is below a value *tmin*, *loggamma* first increases the argument by an integer value, using the relation:

$$\ln \Gamma(t) = \ln \Gamma(t+k) - \ln(\prod_{i=0}^{k-1} (t+i))$$

where $\ln \Gamma(t+k)$ is computed by the procedure *lgm*.

The formula we use for *tmin* is a rough empirical relation to minimise computation time.

Indeed an increase of *k* while decreasing the number of terms of the series, results in more computation for the factor $\ln (\prod_i (t+i))$;

  begin integer *tmin*;
    tmin := **if** *msize* ≥ 18 **then** *msize* − 10 **else** 7;
    if *t* > *tmin* **then** *loggamma* := *lgm*(*t*)
    else
    begin real *f*;
      f := *t*;
L: *t* := *t*+1;
      if *t* < *tmin* **then**
      begin *f* := *f* × *t*;
        go to *L*
      end;

```
    loggamma := lgm(t) − ln (f)
  end
end of procedure loggamma;
real procedure lgm(w);  value w;  real w;
comment This procedure evaluates the logarithm of the
  gamma function according to the Stirling asymptotic series:
```

$$\ln \Gamma(w) \simeq (w - \tfrac{1}{2}) \times \ln (w) - w + \ln \sqrt{2\pi} + \sum_i \frac{c_i}{z^{2i-1}}$$

```
  The coefficients c_i = B_2i/(2i(2i−1)), B_2i being the
  Bernoulli numbers, are rational numbers given here as irre-
  ducible fractions.
  Twenty terms are sufficient for a precision of up to 50
  decimal digits;
begin array c[1:20];  real w2, presum, const, den, sum;
  integer i;
  c[1]  := 1/12;            c[2]  := −1/360;
  c[3]  := 1/1260;          c[4]  := −1/1680;
  c[5]  := 1/1188;          c[6]  := −691/360360;
  c[7]  := 1/156;           c[8]  := −3617/122400;
  c[9]  := 43867/244188;    c[10] := −174611/125400;
  c[11] := 77683/5796;      c[12] := −236364091/1506960;
  c[13] := 657931/300;      c[14] := −3392780147/93960;
  c[15] := 1723168255201/2492028;
  c[16] := −7709321041217/505920;
  c[17] := 151628697551/396;
  c[18] := −26315271553053477373/2418179400;
  c[19] := 154210205991661/444;
  c[20] := −261082718496449122051/21106800;
  const := .91893853320467274178032973640561763986139747363778;
  comment const = ln√2π;
  den := w;  w2 := w × w;  presum := (w−.5) × ln(w) −
  w + const;
  for i := 1 step 1 until 20 do
  begin sum := presum + c[i]/den;
    if sum = presum then go to exit;
    den := den × w2;
    presum := sum
  end;
exit : lgm := sum
end of procedure lgm;
comment: main procedure gamma starts here;
real pi;
pi := 3.1415926535897932384626433832795028841971693993751;
comment argov, argund, lnunder are hardware dependent con-
  stants that are compared to the arguments of intermediate
  results, setting error exit or zero result to prevent exponent
  over or underflow. Should be replaced in the procedure by
  the appropriate numbers;
if z > argov then go to error else if z = entier (z) then
begin if z ≤ 0 then go to error;  y := 1;
  if z > 2 then
  begin loop: z := z − 1;  y := y × z;
    if z > 2 then go to loop
  end
end when z is integer
else if abs(z) < 10 ↑ (−msize) then y := 1/z
else if z < 0 then
begin if z < argund then y := 0
  else
  begin comment As the use of the sine subroutine for large
      arguments might introduce errors, some reductions of
      the argument are made before using it;
    Boolean procedure parity (m);  real m;
    begin integer j;
      j := entier(m);  parity := j = j ÷ 2 × 2
    end parity;
    real procedure decimal(x);  real x;
    begin integer n;
      n := x;
      decimal := abs(x−n)
    end decimal;
    real delta, ex;
    delta := decimal(z) × pi;
    ex := (if delta<10 ↑ (−msize/2) then − ln(decimal(z)) else
    ln(pi/(sin(delta)))) − loggamma(1−z);
    y := if ex < lnunder then 0 else
      if parity (z) then exp(ex) else
      −exp(ex)
  end
end when z is negative
else y := exp(loggamma(z))
end of procedure gamma
```

CHAPIN—Cont'd from p. 510

reasons for accepting or rejecting interrupts. Some data available from an interrupt may not be processable until certain other not yet complete processing work is finished. Some processing work may lack certain items of data required for it to be carried further. The decision table covering checking, accepting, and analyzing interrupts must include in its condition stub, or include by action linkage, provisions to discriminate among all such situations.

When interrupt timing is not controllable, then the parsing of the decision table must incorporate a status analyzer and recorder. This can be simplified by a series of pushdown stacks, with the actions of the analyzer and recorder decision table pushing these down or popping them up to establish changing sets of conditions to use in the subsequent interrupt checking, accepting, and analyzing decision table. Especially in real time environments, parsing the decision table in this manner is very helpful in providing accurate handling of the hundreds of different situations that can arise.

Conclusion

Much of the resistance to the acceptance of decision tables stems from their claimed cumbersomeness of use on large jobs, and from their claimed lack of flexibility. Parsing of decision tables using one or more of the techniques cited here can result in overcoming some of these claimed deficiencies.

Acknowledgment. The author acknowledges the contribution of Kenneth W. Kolence (CDC) to the author's thinking on reflecting data organization in decision tables.
RECEIVED JANUARY, 1967; REVISED MARCH, 1967

REFERENCES

1. CHAPIN, NED. A guide to decision table utilization. In *Data Processing* Vol. IX, DPMA, Parik Ridge Ill., 1967, pp. 327–329.
2. FISHER, D. L. Data, documentation, and decision tables. *Comm. ACM 9* (Jan. 1966), 26–31.
3. POLLACK, SOLOMON L. Analysis of the decision rules in decision tables. RM-3669-PR, The RAND Corp., Santa Monica, Calif., May 1963, 69 pp.
4. VEINOTT, SYRIL G. Programming decision tables in FORTRAN, COBOL, or ALGOL. *Comm. ACM 9* (Jan. 1966), 31–35.