Standards

H. S. BRIGHT, EDITOR

EDITOR'S NOTE

The following communication has been received from Mr. C. E. Macon, Secretary, ASA Subcommittee X3-2 (Coded Character Sets and Input/Output Media).—H.S.B.

* * * * *

A paper entitled "Design of an Improved Transmission/Data Processing Code" by R. W. Bemer, H. J. Smith and F. A. Williams, Jr. appeared in *Communications of the ACM*, Volume 4, Number 5, May 1961. This paper, along with the Editor's Note, was discussed by the ASA X3-2 Committee at the regular meeting of June 28 and 29. It was the decision of the ASA X3-2 Committee that the following remarks be directed to your attention.

(1) The subject paper was one of a number of such papers by a variety of authors which have been received and discussed by the ASA X3-2 Committee.
(2) The objective of ASA X3-2 is to recommend a *single* coded character set to serve as the standard for *information interchange* between data processing systems and between such systems and associated equipment.
(3) Several man-years of research and study, expended over the past nine calendar months, have brought into being the coded character set currently being considered by ASA X3-2. This set differs considerably from that presented in the subject paper in many highly important points and is by no means a direct outgrowth of the material presented in the subject paper. The points with which the committee is at variance will not be covered in this letter but will be covered implicitly in the documentation supporting the official ASA X3-2 recommendation.

PRACNIQUES (Continued)

TAPE SPLITTING

When information must be written on magnetic tape and then immediately read—as in sorting, or when the information on a magnetic tape must be read several times—as in the manipulation of large matrices, considerable time may be spent waiting for tapes to rewind. Tape splitting eliminates waiting for rewind, at the expense of doubling the number of tape units required. The information formerly written on one reel of tape is now written on two; after half of the information is recorded on one reel, that reel begins rewinding, while the remainder of the information is recorded on a second reel.

This technique has been in use for some time (the author used it in an assembly program early in 1959), but it is not as well known as it should be.

By DONALD P. MOORE
Western Data Processing Center
University of California, Los Angeles

Algorithms

J. H. WEGSTEIN, Editor

ALGORITHM 70
INTERPOLATION BY AITKEN

CHARLES J. MIFSUD
General Electric Co., Bethesda, Md.

```
procedure AITKEN (x, f, n, X, F); real array x, f;
          integer n; real X, F;
comment If given x0, x1, . . . xn, n+1 abscissas and also given
  f(x0), f(x1), . . . f(xn), n+1 functional values, this procedure
  generates a Lagrange polynomial, F(X) of the nth degree so that
  F(xi) = f(xi). Hence, for any given value X, a functional value
  F(X) is generated. The procedure is good for either equal or
  unequal intervals of the xi. Aitken's interative scheme is used
  in the generation of F(X). Since the f array is used for tem-
  porary storage, as the calculation proceeds its original values
  are destroyed;
begin integer i, j, t;
  for j := 0 step 1 until n−1 do
    begin t := j+1
      for i := t step 1 until n do
        f[i] := ((X−x [j]) × f [i] − (X−x [i]) × f[j])/
                (x[i] −x[j])    end
      F := f [n]
end
```

ALGORITHM 71
PERMUTATION

R. R. COVEYOU AND J. G. SULLIVAN
Oak Ridge National Laboratory, Oak Ridge, Tenn.

```
procedure PERMUTATION (I, P, N);
  value I, N; integer N; integer array P; boolean I;
  comment This procedure produces all permutations of the
    integers from 0 thru N. Upon entry with I = false the pro-
    cedure initializes itself producing no permutation. Upon each
    successive entry into the procedure with I = true a new
    permutation is stored in P[0] thru P[N]. When the process has
    been exhausted a sentinel is set:
    P[0] : −1,
    N ≥ 0;
  begin
    integer i; own integer array x[0:N];
    if ¬ I then
    begin for i := 0 step 1 until N−1 do x[i] := 0;  x[N] := −1;
      go to E end;
    for i := N step −1 until 0 do begin if x[i]≠i then go to A;
      x[i] := 0 end;
    P[0] := −1;  go to E;
A:  x[i] := x[i]+1;  P[0] := 0;
    for i := 1 step 1 until N do
      begin P[i] := P[i−x[i]];  P[i−x[i]] := i end;
E:  end PERMUTATION
```

ALGORITHM 72
COMPOSITION GENERATOR

L. Hellerman and S. Ogden
IBM-Product Development Laboratory, Poughkeepsie, N. Y.

```
procedure comp (c, k); value k; integer array c;
  integer k;
comment Given a k-part composition c of the positive integer n,
  comp generates a consequent composition if there is one. If
  comp operates on each consequent composition after it is found,
  all compositions will be generated, provided that 1, 1, . . ., 1,
  n−k+1 is the initial c. If c is of the form n−k+1, 1, 1, . . . , 1,
  there is no consequent, and c will be replaced by a k vector of
  0's. Reference: John Riordan, An Introduction to Combi-
  natorial Analysis, John Wiley and Sons, Inc., New York, 1958,
  Chapter 6;
begin integer j; integer array d [1:k];
  if k = 1 then go to last;
  for j := 1 step 1 until k do d [j] := c [j] − 1;
test: if d[j]>0 then go to set;
      j := j−1;
      go to if j = 1 then last else test;
set:  d [j] := 0;
      d [j − 1] := d [j − 1] + 1;
      d [k] := c [j] − 2;
      for j := 1 step 1 until k do c [j] := d[j] + 1;
      go to exit;
last: for j := 1 step 1 until k do c [j] := 0;
exit: end comp
```

CERTIFICATION OF ALGORITHM 42
INVERT (T. C. Wood, *Comm. ACM*, Apr., 1961)

Anthony W. Knapp and Paul Shaman
Dartmouth College, Hanover, N. H.

INVERT was hand-coded for the LGP-30 using machine language and the 24.0 floating-point interpretive system, which carries 24 bits of significance for the fractional part of a number and five bits for the exponent. The following changes were found necessary:

(a)
```
if j = n+1 then a[i, j] := 1.0 else a[i, j] := 0.0;
  should be
if j = n+i then a[i, j] := 1.0 else a[i, j] := 0.0;
```

(b)
```
for k := j step 1 until 2 × n do
  a[i, k] := a[i, k]/a[i, j];
  should be
for k := 2 × n step −1 until i do
  a[i, k] := a[i, k]/a[i, i];
```

(c)
```
if l ≠ i then for k := 1 step 1 until 2 × n do
  a[l, k] := a[l, k] − a[i, k] × a[l, j];
  should be
if l ≠ i then for k := 2 × n step −1 until i do
  a[l, k] := a[l, k] − a[i, k] × a[l, i];
```

Given these changes, j becomes superfluous in the second i loop, and the other references to j may be changed to references to i.

INVERT obtained the following results:

The computer inverted a 17-by-17 matrix whose elements were integers less than ten in absolute value. When the matrix and its inverse were multiplied together, the largest nondiagonal element in the product was −.00003. Most nondiagonal elements were less than .00001 in absolute value.

INVERT was tested using finite segments of the Hilbert matrix. The following results were obtained in the 4 × 4 case:

16.005	−120.052	240.125	−140.082
−120.052	1200.584	−2701.407	1680.917
240.126	−2701.411	6483.401	−4202.217
−140.082	1680.920	−4202.219	2801.446

The exact inverse is:

16	−120	240	−140
−120	1200	−2700	1680
240	−2700	6480	−4200
−140	1680	−4200	2800

INVERT was also coded for the LGP-30 in machine language and the 24.1 extended range interpretive system. This system, which uses 30 significant bits for the fraction, obtained the following as the inverse of the 4 × 4 Hilbert matrix:

16.000	−120.001	240.001	−140.001
−120.001	1200.006	−2700.015	1680.010
240.001	−2700.016	6480.037	−4200.024
−140.001	1680.010	−4200.024	2800.016

The program coded in the 24.0 interpretive system successfully inverted a matrix consisting of ones on the minor diagonal and zeros everywhere else.

REMARK ON ALGORITHM 52
A SET OF TEST MATRICES (John R. Herndon, *Comm. ACM*, Apr. 1961)

G. H. Dubay
University of St. Thomas, Houston, Tex.

In the assignment statement

```
c := t×(t + 1)×(t + t − 5)/6;   (a)
```

the t is undefined. A suitable definition would be provided by preceding (a) with `t := n;` .

CERTIFICATION OF ALGORITHM 68
AUGMENTATION (H. G. Rice, *Comm. ACM*, Aug. 1961)

L. M. Breed
Stanford University, Stanford, Calif.

AUGMENTATION was transliterated into BALGOL for the Burroughs 220, and proved successful in a number of test cases. However, the following algorithm has exactly the same effect and is considerably simpler:

```
real procedure Aug(x, y); value x, y; integer x, y;
begin if x<0 then L : go to L else Aug := x+y end Aug
```

Contributions to this department must be in the form stated in the Algorithms Department policy statement (*Communications*, February, 1960) except that ALGOL 60 notation should be used (see *Communications*, May, 1960). Contributions should be sent in duplicate to J. H. Wegstein, Computation Laboratory, National Bureau of Standards, Washington 25, D. C. Algorithms should be in the Publication form of ALGOL 60 and written in a style patterned after the most recent algorithms appearing in this department.

Although each algorithm has been tested by its contributor, no warranty, express or implied, is made by the contributor, the editor, or the Association for Computing Machinery as to the accuracy and functioning of the algorithm and related algorithm material and no responsibility is assumed by the contributor, the editor, or the Association for Computing Machinery in connection therewith.