Algorithms Supplement

Note by the Editor
An algorithm by L. J. Hazlewood of the The Computer Centre, University of Aston in Birmingham, has been accepted for publication in *The Computer Journal*. The title is 'An algorithm for cautious adaptive quadrature'. As an associated paper is to appear in the *Journal of the Institute of Mathematics and its Applications*, publication of the algorithm is being delayed so that it can appear at approximately the same time.

Algorithm 94
HANDLING DATES ON SMALL COMPUTERS

I. D. Hill
Clinical Research Centre
Harrow, Middlesex
HA1 3UJ

Author's note
The most convenient way of handling dates in computers is as integers representing the number of days since some fixed date. However a computer with 16-bit words has a maximum integer of 32,767—less than the number of days in 90 years, and thus inadequate. While the range of integers can be doubled by using the negative integers as well as the positive ones, this leads to difficulties in operations such as finding the day of the week (by finding the remainder on dividing by 7), and is not recommended.

It therefore becomes preferable, on such machines, to use integral valued 'real' numbers instead, but it is necessary to take care against rounding errors since using two words (32 bits) for floating point representation gives only about 7 decimal accuracy.

The routines presented are based on the method of Tantzen (1963) and are believed to be proof against all rounding errors. They refer to the Gregorian calendar only. Day 1 is taken as 14 September 1752 on which date Britain adopted this calendar. On this scale Day 1,000,000 will not be reached until 10 August 4490, so 7 decimal accuracy should be adequate for awhile yet.

FUNCTION DAY makes no test that its arguments are possible, and will accept the 34th day of the 15th month for example. If a test is needed the most convenient technique is as follows:

```
      A = DAY(ID, IM, IY)
      CALL DATT(A, JD, JM, JY)
      IF (ID .NE. JD .OR. IM .NE. JM) CALL FAULT



      FUNCTION DAY (IDAY, MONTH, IYEAR)
C
C        CONVERTS A GREGORIAN CALENDAR DATE TO THE
C        NUMBER OF DAYS SINCE 13TH SEPTEMBER 1752
C
      I = IYEAR
      M = MONTH - 3
      IF (M .GE. 0) GOTO 10
      M = M + 12
      I = I - 1
   10 K = I / 100 - 16
      I = MOD(I, 100)
      DAY = 36524.0 * FLOAT(K) + 365.0 * FLOAT(I) +
     *  FLOAT(K / 4 + I / 4 + (153 * M + 2) / 5 + IDAY) - 55714.0
      RETURN
      END
C
      SUBROUTINE DATT (D, IDAY, MONTH, IYEAR)
C
C        CONVERTS D, WHICH SHOULD BE POSITIVE AND INTEGRAL,
C        TO THE CORRESPONDING GREGORIAN DATE
C
      DD = D + 445711.0 / 8.0
      K = INT((4.0 * DD) / 146097.0)
      DD = DD - 36524.0 * FLOAT(K) - FLOAT(K / 4)
      I = INT((4.0 * DD) / 1461.0)
      DD = DD - 365.0 * FLOAT(I) - FLOAT(I / 4)
      I = 100 * (K + 16) + I
      K = INT(5.0 * DD - 1.875)
      M = K / 153
      K = (K - 153 * M + 5) / 5
      IF (M .GT. 9) GOTO 10
      MONTH = M + 3
      GOTO 20
   10 MONTH = M - 9
      I = I + 1
   20 IYEAR = I
      IDAY = K
      RETURN
      END
```

Reference

TANTZEN, R. G. (1963). Algorithm 199, conversions between calendar date and Julian day number, *CACM*, Vol. 8, p. 444.

Algorithm 95
GENERATING A PARITY TESTING TABLE

A. Colin Day
Computer Centre
University College London
London

Author's Note
What is the fastest way of testing the parity of a character (i.e. the number of bits set to 1 in it)? Given sufficient space, the answer is surely to use the character as an index to a table in which values such as 0 and 1 are stored to indicate even and odd parity respectively. Such a table will have the values 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, . . . (starting from the character with the binary value 0). Filling the table with its values is somewhat of a chore, as the pattern is rather subtle. An algorithm to construct the table may seem a trivial thing, but it may save time and aid accuracy.

The following ALGOL 60 procedure constructs a parity testing table in integer array *par*[0:*n*], using 0 to represent even parity and 1 for odd. *par*[0] is first initialised to zero. The remaining values are inserted in sequence from 1 to *n*. For any value *i*, the highest order bit set to 1 is stripped off by subtracting *power*, which always contains the appropriate power of 2. This leaves a value (say *j*) whose parity is already in the table. The parity of *i* is then the reverse of the parity of *j*, because of the bit which was stripped off. The reverse of a parity *k* is given by $1 - k$.

Note that if the requirement is not for a table of parities, but for a table giving the number of bits set to 1 (0, 1, 1, 2, 1, 2, 2, 3, 1, 2, . . .), this can be obtained simply by changing $1 - par[i - power]$ in the last assignment into $1 + par[i - power]$.

```
procedure parity(par, n);
 value n; integer n; integer array par;
 begin integer i, power;
  power := 1;
  par[0] := 0;
  for i := 1 step 1 until n do
  begin
   if i = power × 2 then power := i;
   par[i] := 1 − par[i − power]
  end
 end
```

Algorithms supplement—Statement of Policy

A contribution to the Supplement may consist of an Algorithm, a Note on a previous algorithm, or an item under the heading of Correspondence.

Because the aim is to facilitate the interchange of algorithms, these should normally be submitted in one of the standard high level programming languages, namely ALGOL 60 (1), ALGOL 68 (2),

Downloaded from https://academic.oup.com/comjnl/article/20/3/280/751979 by guest on 19 April 2024

FORTRAN (3), COBOL (4). In this case the algorithms must conform to the appropriate standard. If algorithms are submitted in other programming languages, the reference document for that language must be stated.

Algorithms must be self-contained. This means that an algorithm must consist of one or more complete segments, and that an algorithm must not use any non-local identifiers other than standard function names. COMMON areas are permitted in FORTRAN, but their use must be clearly described.

The algorithm must be written for publication in the appropriate reference language, and preceded by an appropriate Author's Note. It must be submitted in duplicate and be typewritten double-spaced. Where material is to appear in bold face it should be underlined in black. Where the appropriate character does not exist on a typewriter, it should be inserted neatly by hand in black and not be replaced by a similar composite character (e.g. $\leqslant$ should not be inserted as $\leq$).

An algorithm must be accompanied by a computer printout of a driver program testing it (possibly against test data) and producing test results. The machine, compiler and operating system used should be indicated. A computer readable copy of the algorithm, the test driver and any test data will be requested later, but should not be sent in the first instance. The Author's Note should include the theory of the method, with references, and also explain any tests used to verify the algorithm.

The algorithm must be syntactically correct, produce the results claimed and use computer resources as efficiently as possible. Constructions whose results may depend on the compiler used should be avoided (e.g. $v := x + f(x)$ where $f(b)$ is a function which alters the value of b). Comments should be used wherever appropriate to clarify the logic. Cases of failure should be clearly anticipated and handled. Approximate numerical constants must be given with as much accuracy as is appropriate. Numerical labels should be in ascending order.

Every effort is made to see that published algorithms are completely reliable. In particular all algorithms are submitted to independent referees and extensively checked. However, Notes or Correspondence which point out defects in or suggest improvements to previously published algorithms are welcomed. To help in preventing printing mistakes, galley proofs will be sent to authors where possible.

Whilst every effort is made to publish correct algorithms, no liability is assumed by any contributor, the Editor or *The Computer Journal* in connection therewith.

The copyright of all published algorithms remains with *The Computer Journal*. Nevertheless the reproduction of algorithms is explicitly permitted without charge provided that where the algorithm is used in another publication, reference is made to the author and to *The Computer Journal*.

In the event of the formation of a National Algorithm Library, all algorithms which have appeared in *The Computer Journal* will be made available to this Library.

References

1. PROGRAMMING LANGUAGE ALGOL, ISO/R/1538.
2. REVISED REPORT ON THE ALGORITHMIC LANGUAGE ALGOL 68. (1976). Edited by A. van Wijngaarden *et al*, Springer-Verlag.
3. PROGRAMMING LANGUAGE FORTRAN, ISO/R/1539.
4. PROGRAMMING LANGUAGE COBOL, ISO/R/1989.

Documents 1, 3 and 4 above may be obtained from: British Standards Institution, Sales Branch, 101 Pentonville Road, London N1.

Editors: P. A. Samet and A. C. Day, Computer Centre, University College London, 19 Gordon Street, London WC1H 0AH. Tel: 01-387 0858.

Downloaded from https://academic.oup.com/comjnl/article/20/3/280/751979 by guest on 19 April 2024