NAG

NEWSLETTER 2/81

```
(REF[,]COMPL a, BOOL preserve,
 REF[]COMPL val, REF[,]COMPL v
 NAGFAIL fail) VOID:
```

ALGOL 68

```
                (REF[,]COMPL a, BOOL preserve,
                 REF[]COMPL val, REF[,]COMPL vec,
                 NAGFAIL fail)VOID:
        BEGIN INT n1 = LWB val,
                  n2 = UPB val;
              REF[,]COMPL h = (preserve
                                 ! LOC[n1 : n2, n1 : n2]COMPL
                                 ! NIL
                                 );
              [n1 : n2]INT perm, ind;
              [n1 : n2]REAL scaling;
              F02BALANCE bal := (n1, n2, scaling);
              NAGFAIL fail1 = (INT n, []CHAR s)
                              VOID:
                              ([LWB s : UPB s]CHAR s1 := s;
                              fail(n, s1 RENAME "zzzzzz");
                              GOTO exit
                              );
              f01upb(a, bal, h, ind, vec, fail1);
              f02uab(bal, (preserve
                           ! h
                           ! a
                           ), val, NIL, perm, vec,
                     fail1
                    );
              IF vec :/=: NIL
              THEN f02upb(bal, NIL, NIL, 1,
                          vec, fail1
                          )
              FI;
          exit:
              SKIP
        END;
```

MARK 3

CONTENTS

Numerical Algorithms Group Limited October 1981

Editor: Miss Janet Bentley

Published by NAG Central Office

INTRODUCTION

Major events since the release of the last NAG Newsletter include the inauguration of an Association of NAG Users (NAGUA). Future issues of this Newsletter will include a NAGUA section and any members of the Association wishing to contribute to this section should address their correspondence to the Editor, at NAG Central Office. Plans have already been made for the 1982 meeting and these are outlined on page 23.

This year's lecture at the Annual General Meeting of NAG Limited was given by Professor C.A.R. Hoare of the University of Oxford. We are delighted to have the opportunity to publish the script of his lecture, entitled 'Is There a Mathematical Basis for Computer Programming?' as the feature article in this issue of the Newsletter.

In addition to regular articles, we include details of the Mark 3 Algol 68 Library, which has recently been completed, of Mark 8 Algol 60 and of the routines to be withdrawn from the Mark 9 FORTRAN Library.

If you have any comments or suggestions regarding the NAG Newsletter, or indeed would like to contribute an article, then please contact the Editor at NAG Central Office. We look forward to hearing from you.

ADDRESS FOR ENQUIRIES

Enquiries related to this Newsletter should be addressed to the Editor at NAG Central Office (address below). For further information about NAG and the NAG Library Service please contact:

The NAG Library Service Co-ordinator
NAG Central Office
7 Banbury Road
Oxford OX2 6NN
United Kingdom

Tel: National 0865 511245
International +44 865 511245

Telex: 83147 (ref: NAG)

or in North America,
The Company Secretary
NAG (USA) Inc.
1250 Grace Court
Downers Grove
Illinois 60516
U.S.A.

Tel: (312) 971 2337

Telex: 254708 (TELESERV DFLD NAG USA)

NAG LIBRARY SERVICE NEWS

Most NAG Library Service sites will by now have received the Mark 8 FORTRAN Library. The implementation review on page 19 gives an indication of when the remaining implementations are expected to be completed. The advent of this Mark of the Library has seen the addition of a new item in the NAG 'product range', namely the On-Line Information Supplement. We are delighted at the response we have had from sites and, since announcing its availability in July of this year, have received orders from 40 sites. Orders for the Library Service continue to arrive and 1981 has seen the installation of the Library at sites in four additional countries: Kuwait, Morocco, Zimbabwe and Qatar.

On-Line Information Supplement

This supplement is now available to all sites which have Mark 8 of the NAG FORTRAN Library. It contains information about the Library suitable for display at computer terminals or listing on line-printers. The cost of the Supplement is one-third of the FORTRAN-related licence fee and further details and order forms are readily available from NAG Central Office.

Graphical Supplement

The first implementations of this Supplement to the NAG FORTRAN Library are now nearing completion. As the different machine versions become available for distribution we shall contact all Library Service sites within that machine range with details of ordering procedures etc. Again the cost will normally be one-third of the FORTRAN-related licence fee.

DOCUMENTATION NEWS

Supplements to the Fortran Library

The On-Line Information Supplement to the FORTRAN Library Mark 8 was completed during the summer. This is derived from the Library Manual and describes the Purpose, Specification, Parameters and Error Indicators of each routine in the Mark 8 Library. It is suitable for display at computer terminals and for listing at line-printers.

The Graphical Supplement is a new product that has extended our use of Cuneiform and TSSD to meet the different demands of its Manual. TSSD is the typesetting program developed by Harwell and used by NAG in devising a single documentation data base intended for computer type setting. Cuneiform is a locally developed package, designed initially for extracting the on-line form from this data base. All the documents in the Graphics Manual, including the introductions, are typeset so that the Manual fits onto only 250 pages, and will be distributed in a single ring binder. An on-line form of this documentation will be supplied with the Graphical Supplement. Both Supplements are issued as separate products from the FORTRAN Library: for further details please contact the Library Service Co-ordinator at NAG Central Office.

DOCUMENTATION NEWS (CONTD)

Updates to NAG Library Manuals

Algol 60: the Update to Mark 8 contains 300 typewritten pages and was printed in April of this year. The cost of the update (code PAN1/ALM7) is £6.

Algol 68: the Update to Mark 3 contains some 1100 pages of new text and extends the Manual from 2 to 4 volumes. The update has been prepared and printed entirely in-house and is now being collated ready for distribution in November.

FORTRAN: preparation of the Update to Mark 9 is well under way. While this is much smaller than the Update to Mark 8, it is still a considerable work containing 15 new typeset documents, several new Chapter Introductions and a large number of corrections and improvements to individual pages in the existing manual. The On-Line Information Supplement will also be updated to Mark 9 and a FORTRAN Mini Manual Mark 9 is also being prepared.

New Products in Preparation

Genstat 4.03 has received sufficient interest from the German-speaking community for Rothamsted Experimental Station to have arranged translation of its manual into German. NAG are currently producing the manual on one of Central Office's new micro computers.

Microfiche

The FORTRAN Library Manual Mark 8 is now available on 34 microfiche, reduction ×24, at a cost of £18. FORTRAN Mark 7 is still available, on 31 fiche, at a cost of £13.

The Genstat 4.03 Manual is available on 8 fiche, at £8, and we have also recorded back issues of the Statistical Package Newsletters onto microfiche. Fiche containing Glim Newsletter issues 1-2, 3-4 and Genstat issues 1-6 are available at £1.50 each.

Nicola Bourdillon
NAG Central Office

IS THERE A MATHEMATICAL BASIS FOR COMPUTER PROGRAMMING?

C.A.R. HOARE

SEPTEMBER 1981

1. Introduction

A specification of a mechanism should in some sense be a full description of the intended observable behaviour of the mechanism. For any observation of the actual behaviour of the mechanism, it should be very clear whether the observation accords with the specification or not. For example, part of the specification of a command in a program may be that it increases the value of a program variable "x". The values of the variable before and after execution of the command can be observed and recorded on each occasion that it is executed, e.g.

x before	x after
3	4
97	137
31	12

The observations recorded in the first two rows of this table accord with the specification; but the third row does not.

An observation of the behaviour of a mechanism usually involves observation of the values of certain variables, to which it is convenient to ascribe special conventional variable names. For example, we shall use program variable names "x", "y", etc., to denote the values taken by these variables before execution of a command; and we shall use dashed variants "x'", "y'", etc., to denote the final values taken by these variables when the command terminates. We also introduce the Boolean variable "end", which takes the value true if the command terminates.

A specification can now be formulated as an assertion about the intended values which these special variables are allowed to take. For example, the specification given informally above can be formalised as the assertion

$$x < x' \ \& \ end.$$

For any observed values of "x", "x'" and "end", it is easy to determine whether they satisfy this specification or not.

A mechanism satisfies a specification if every possible observation of its possible behaviour satisfies the specification. The set of special variables which may occur in a specification is known as its alphabet. A specification may also contain other logical variables, either bound or free.

Following the lead of R.W. Floyd and E.C.R. Hehner, we propose to assign as the meaning of a command of a program the strongest specification which it is certain to satisfy. For example, let SKIP be the command which terminates successfully, and leaves the values of all variables unchanged. Its strongest specification is

$$x'=x \ \& \ y'=y \ \& \ end$$

Here and later, we assume that the alphabet of our specifications is the set {x,y,x',y',end}. The discussion can be readily adapted to different and larger alphabets.

The identification of a command with its strongest specification is justified by the principle of the identity of indiscernibles. If two mechanisms have the same specification, then every possible observation of every possible behaviour of either of them is also a possible observation of a possible behaviour of the other. There is no conceivable experiment that might distinguish between them. Furthermore, in every context in which we might use one of them, the other will be equally good. It is no counterargument to complain that one of the mechanisms might be cheaper or faster in execution. If we are interested in these additional properties, then we must be able to observe them, and we should introduce additional special variables into the alphabet of the specification to denote their values. Our present purpose is to discuss the logical properties of commands, and therefore we choose to ignore (for the time being) possible variations in efficiency or cost. In mathematics, the fact that $(a+b)^2$ may be easier to calculate than $a^2+2ab+b^2$ does not prevent us from regarding them as equal.

The identification of a piece of program with the strongest assertion which describes its possible behaviours may seem disconcerting at first: it has the consequence that a programming language is just a highly restricted notation for writing an assertion. The purpose of the restrictions is to permit the automatic implementation on a stored-program computer of a mechanism that exhibits the described behaviour. One particular restriction which we shall implicitly observe is that program texts may contain only undashed variables of their alphabet.

We shall now define the individual constructs of a simple programming notation, including only assignment, alternation, and recursion.

2. Assignment

Let P be an assertion, let x be an undashed variable, and let e be an expression containing only undashed variables. Then the notation

"x:=e;P"

is an assertion which is satisfied by a mechanism which first assigns the value of e to x and then behaves in a manner that satisfies P. An explicit assertion describing this behaviour can be constructed by transforming the assertion P. Let $P\left|{}^{x}_{e}\right.$ be formed from P by replacing every free occurrence of x by e. (If any free variables of e become identified with bound variables of P, the collision is averted by systematic change of the offending bound variables). Let De be an assertion which is true just when all the values of the operands of e are within the domain of their operators. Then we define

$$(x:=e;P) =_{df} \neg De \vee (P\left|{}^{x}_{e}\right.)$$

Examples of this rule are:

(1) (x:=x+y; SKIP) = (x'=x+y & y'=y & end)

(2) (y:=y/x; x:=x+y; SKIP) =

(x=0 ∨ x'=x+y/x & y'=y/x & end)

(3) (x:=1; y:=y/x; x:=x+y; SKIP) = (x'=1+y & y'=y & end)

Note that when the initial values of the variables are such that the expression is not defined, the specification takes the value <u>true</u>. This is the most useless specification of all, since it does not place any constraint on the final values of the variables, nor does it even state whether the command terminates. The specification <u>true</u> is the easiest one to meet, because it represents a prior decision to accept the product, without even looking at its behaviour. Such an indiscriminating customer is the only one who will be happy with a program which attempts to evaluate an undefined expression.

3. Alternation

Let P and Q be assertions, and let b and c be Boolean expressions, containing only undashed variables.

Then the notation.

(b → P □ c → Q)

is an assertion which is satisfied by a mechanism which first evaluates b and/or c. If either is undefined, or if both are false, the subsequent behaviour of the mechanism is arbitrary. If b is true and c false, the subsequent behaviour of the mechanism must satisfy P; if c is true and b false, the subsequent behaviour must satisfy Q; and in the final case, with both c and b true, the mechanism may satisfy either P or Q or both, i.e. it must satisfy their disjunction (P ∨ Q). This description of a mechanism is formalised by the definition of its behaviour:

$$(b \to P \,\Box\, c \to Q) =_{df} \neg Db \vee \neg Dc \vee \neg b \,\&\, \neg c \vee b \,\&\, P \vee c \,\&\, Q$$

Example

(x ≤ y → (x:=y; SKIP) □ y ≤ x → SKIP)

= (¬x≤y & ¬y≤x

∨ x≤y & x'=y & y'=y & end

∨ y≤x & x'=x & y'=y & end)

= (x'=max(x,y) & y'=y & end).

The definition given above can be readily extended to more than two alternatives. For one alternative, the definition is

$$(a \to P) =_{df} (a \to P \,\Box\, a \to P) = Da \vee \neg a \vee P$$

Other useful special cases are:

$$(\text{if } a \text{ } \underline{\text{then}} \text{ } P \text{ } \underline{\text{else}} \text{ } Q) =_{df} (a \to P \,\Box\, (\neg a) \to Q)$$

$$(P \vee Q) =_{df} = (\text{true} \to P \,\Box\, \text{true} \to Q)$$

(P ∨ Q) may be regarded as a non-deterministic program, which "decides arbitrarily" to behave either as described by P or as described by Q. However, the concept of non-determinism is one that has led to a lot of unnecessary confusion; by talking about specifications, we can avoid talking about some unobservable property of nondeterminism inherent in a mechanism.

4. Recursion

Let P be an assertion containing the variable p, which itself ranges over possible assertions. Thus if Q is some assertion, with the same alphabet, $P\left|{}^{p}_{Q}\right.$ is the result of replacing all occurrences of p in P by Q (with the usual change to bound variables if necessary). P may be regarded as describing the behaviour of an assembly, with a slot p into which a range of different components may be plugged. If the component is described by Q, then the result of plugging the component into the assembly will be described by $P\left|{}^{p}_{Q}\right.$. Now we shall define a mechanism which behaves like a call of a recursively defined procedure with name "p" and body P. The command will be written.

$$\mu p.P$$

The intention is that $\mu p.P$ will behave as specified by P, on the assumption that its component p does so as well. The effect we want is as if $\mu p.P$ were plugged into itself as its own component, i.e.

$$\mu p.P = (P\left|{}^{p}_{\mu p.P}\right.).$$

Such a self inserted assembly seems to be physically impossible; and even mathematically, the circularity of the definition seems quite improper. Scott has shown a way to solve these problems. It is both physically possible and mathematically valid to construct a series $P^0, P^1, \ldots$ of ever improving approximations to the required product. The first approximation P^0 is the wholly arbitrary process, satisfying specification true. The next approximation is formed by plugging the previous approximation as the component p in the assembly P, giving p^1. The whole series of approximations can be defined by induction:

$$P^0 = \text{true}$$

$$P^{n+1} = P\left|{}^{p}_{p^n}\right. \quad \text{for } n \geq 0$$

P^n is the assertion satisfied by a mechanism which behaves correctly up to recursion depth n, but breaks when required to recurse deeper; and a broken mechanism may do anything whatsoever.

Now what we really want is a mechanism which satisfies all these specifications, for all finite depths of recursion. This is the mechanism whose behaviour is described by the assertion:

$$\forall n \geq 0.P^n$$

The discussion so far deals with the general case of recursion, as featured in several well-known programming languages. Nevertheless, in all programs constructible in our little programming notations, the recursions reduce to iterations, which use the time dimension rather than a space dimension to create the illusion of boundlessness. This is illustrated by the following examples.

(1) $\mu p.(x:=x+1;p)$

$$p^0 = \text{true}$$

$$p^1 = \neg D(x+1) \vee \text{true}\left|{}^{x}_{x+1}\right.$$

$$p^n = \text{true} \quad \text{for all } n$$

therefore

$$\mu p.(x:=x+1;p) = \forall n \geq 0.\text{true} = \text{true}.$$

This is the most useless specification. The punishment for a non-terminating loop is the same as that for an undefined expression, - the mechanism may break, and do anything whatsoever.

(2) $\mu p.(x > 0 \rightarrow x:=x-1;\ p \ \square\ x=0 \rightarrow \text{SKIP})$

$$p^0 = \text{true}$$

$$p^1 = (x < 0 \vee x > 0 \ \&\ \text{true}\left|{}^{x}_{x-1}\right. \vee x=0 \ \&\ x'=x \ \&\ \text{end})$$

$$= (x < 0 \vee x \geq 1 \vee 0 \leq x < 1 \ \&\ x'=0 \ \&\ \text{end})$$

$$\ldots$$

$$p^n = (x < 0 \vee x \geq n \vee 0 \leq x \leq n \ \&\ x'=0 \ \&\ \text{end})$$

Therefore

$$(2) = x < 0 \vee x' = 0 \ \&\ \text{end}$$

Note how the condition under which the loop fails to terminate has emerged from our formal reasoning.

Of course, the reasoning is not entirely mechanical. The discovery of the particular finite approximations P^0, P^1, P^2, ... P^{157}, ... etc., can be accomplished formally, by simply expanding the definitions - although the size of the formulae will grow alarmingly. The discovery of a general formula P^n, containing "n" as a free logical variable, requires a little insight, and usually a proof by induction. For the example shown above, the required proof is:

(a) base step: $= P^0 = (x<0 \vee x\geq 0 \vee \ldots)$, which is trivially true.

(b) induction: $P\left|{}^{p}_{p^n}\right. = (x<0 \vee x>0 \;\&\; (P^n)\left|{}^{x}_{x-1}\right.$

$\vee x = 0 \;\&\; x' = 0 \;\&\; \text{end})$

$= (x<0 \vee x = 0 \;\&\; x' = 0 \;\&\; \text{end}$

$\vee x > 0 \;\&\; (x-1<0 \vee x-1\geq n \vee 0 \leq x-1<n$
$\&\; x' = 0 \;\&\; \text{end}))$

$= (x<0 \vee x \geq n+1 \vee 1\leq x, n+1 \;\&x'$
$= 0 \;\&\; \text{end})$

$= p^{n+1}$

5. The inverse calculus

The definitions given for our programming notations form the basis of a kind of calculus, which for any program permits its strongest specification to be derived; and all other assertions describing its potential behaviour follow logically from the strongest one. This calculus should be of direct use to maintenance programmers who are all too often given a program and asked to find out what it does. But this is the wrong way round. The specification should have come before the program, as in normal practice of a well established engineering discipline.

In our view, a program is just an assertion expressed in a highly restricted notation. So if S was the original specification, the programmer's task is to construct a program P of which he can prove:

$$P \Rightarrow S$$

This proof guarantees that every observable behaviour of P will satisfy the specification S. But to solve the problem of finding a suitable P we need a different calculus, which is a kind of inverse of the calculus presented here.

Consider the analogy provided by a more familiar calculus. Given an expression containing valuables x and y, the differential calculus enables us to formulate its derivative with respect to either x or y or both. For a wide class of initial expressions, differentiation is a purely formal process. However for practical purposes of science and engineering, we are more usually faced with the problem of finding an expression whose derivatives have certain desired properties, for example that they satisfy certain differential equations. Mathematicians have developed many ingenious strategies for solution of integral and differential equations, but in general their application requires good judgement, and there is no guarantee that a particular method will lead to a solution, or even that a solution exists. Nevertheless, when a solution has been found, its validity can be much more easily checked by using the simpler differential calculus. It is our hope that by using the calculus of specifications, the checking of properly annotated computer programs may become simpler than their original construction.

As an example of the kind of rule of the inverse calculus which might be of use to programmers, we shall suggest a technique for constructing a recursively defined program to meet a given specification R.

(1) Try to decide on upper bound on the depth of the recursion required. This will usually depend on the initial values of the program variables. Formulate the bound as an expression e, which maps these variables onto a natural number.

Prove: $De \vee R$.

(2) Let n be a fresh variable, and define:

$R(n) =_{df} e\geq n \vee R$.

Find a program P(p) with free variable p, such that you can prove:

$P(R(n)) \Rightarrow R(n+1)$

(3) Then the program you want is

$\mu p.P$.

6. Conclusion

We have propounded the view that a computer program is a mathematical formula whose properties can be explored by the normal methods of mathematical reasoning. It is thus feasible (and certainly desirable) to formulate accurate specifications of programs, and prove that they have been met. But much progress still needs to be made in developing the necessary mathematical theories and techniques, and in teaching them to programmers. Three obstacles must be overcome:

(1) Mathematicians are inexperienced and disinclined to deal with formulae that stretch over many thousands of lines.

(2) Mathematicians are classified as pure or applied. Applied mathematicians are interested in the continuum of real numbers, and their approximations; while pure mathematicians pride themselves on not being applied. Non-numerical computer programming is a direct application of the concepts and techniques of pure mathematics, and is therefore not appreciated by either class of mathematician.

(3) Mathematics has become a subject which students have to learn; computer programming is actually doing mathematics, though neither programmers nor mathematicians seem to realise it.

I hope this brief paper has made a start in removing these obstacles.

NAG ALGOL 68 LIBRARY, MARK 3

Scope of the Library

The Mark 3 Algol 68 Library contains roughly double the number of routines in Mark 2. It provides, with very limited exceptions, facilities equivalent to those of the Mark 5 FORTRAN Library, with additional material in some chapters based on Marks 6, 7 or 8 of the FORTRAN Library. Also included are some facilities not available in the other language versions of the Library. The new material in the Mark 3 Algol 68 Library is summarised below.

Mark 5 FORTRAN equivalents

To provide equivalence with Mark 5 FORTRAN, routines have been added in chapters A02 (complex arithmetic), C02 (zeros of polynomials), C05 (roots of one or more transcendental equations), C06 (summation of series), G01 (simple calculations on statistical data (but excluding routines for statistical distribution functions)), M01 (sorting), S (approximations of special functions (but excluding Fresnel integrals)) and X02 (machine constants).

In the F01/F04 (matrix operations/simultaneous linear equations) chapters, material has been added for least squares solutions. An extra routine to calculate the approximate pseudo (or generalised) inverse of A^TA has been included. (Routines for sparse matrices have been excluded.)

In the F01/F02 (matrix operations/eigenvalues and eigenvectors) chapters, general purpose and easy-to-use routines have been included for real and complex unsymmetric matrices (this covers all Mark 5 FORTRAN routines with some additions). Also included are routines for real symmetric (lower triangle only supplied), real band symmetric (two alternative modes of storage) and Hermitian matrices. In the real band symmetric case, a routine for inverse iteration has also been included. (Routines for the generalised $A-\lambda B$ eigenproblem have been excluded).

In the F05 (orthogonalisation) chapter, additional material for normalising the eigenvectors of real and complex matrices has also been included.

The X03 (inner products) chapter has been revised to provide inner-products of real and complex vectors and of rows and/or columns of symmetric and band symmetric real matrices. Both single and double length accumulation is possible. Also provided are facilities for double length arithmetic and conversion operations.

Mark 6 FORTRAN equivalents

In the E04 (minimising and maximising a function) chapter, routines are provided for unconstrained minimisation and minimising subject to simple bounds on variables (both general and easy-to-use routines). Also included is a routine for non-linear least squares minimisation. (Routines for non-linear constraints are excluded.)

NAG ALGOL 68 LIBRARY, MARK 3 (CONTD)

Mark 6 FORTRAN equivalents (contd)

In the G05 (random numbers) chapter, a basic generator uniformly distributed on (0,1) is provided. This may be initialised to a user specified value and the current generator state may be saved and restored by the user. Routines for generating pseudo-random numbers on a variety of continuous distributions are provided as are routines for initialising a reference vector for various discrete distributions (the reference vector can then be used to generate a pseudo-random integer).

Mark 7,8 FORTRAN equivalents

In the E02 (curve and surface fitting) chapter, routines additional to the Mark 5 FORTRAN equivalents are provided. General and easy-to-use routines for weighted least squares polynomial approximations to set(s) of data points are included (in the general case, there is an option to force the fit to contain a given polynomial factor). Service routines are provided to evaluate a polynomial given in Chebyshev series form and to determine the coefficients of its derivative and its indefinite integral. Also included is a routine for weighted least squares curve fitting by cubic splines, and service routines to evaluate a cubic spline from its B-spline representation and also to evaluate its first three derivatives and its indefinite integral. (Routines for surface fitting are not included.)

Extra, with no FORTRAN equivalents

A new A04 (extended arithmetic) chapter has been provided. This gives relational, arithmetic and conversion operators for multiple length integer and rational arithmetic.

A new T01 (vector and matrix operations - Torrix) chapter has also been provided. This provides a much generalised version of part of the F01 (matrix operations) chapter. Torrix is a system of operations on objects (vectors and matrices) of fairly general linear spaces. The underlying field is REAL. Extensions are available for generating complex vectors and matrices and real symmetric, real band symmetric and Hermitian matrices for manipulation by the Torrix operators.

Availability

The Mark 3 Algol 68 Library is currently available for ICL 1906A/S systems. Other implementations in progress include ICL 1900* and 2900, CDC, Telefunken TR440 and IBM/AMDAHL. A summary of the Library contents and details of costs are available on request from NAG Central Office.

ROUTINES TO BE WITHDRAWN FROM THE MARK 9 FORTRAN LIBRARY

20 routines will be withdrawn from the NAG FORTRAN Library at Mark 9. These impending withdrawals have been announced in the Mark 8 Library Manual, in particular in the document FORTRAN MK8 NEWS. Details are given below, together with brief comments on the reasons for withdrawal and on the choice of a replacement routine. Any users who are currently using a routine which will be withdrawn, should consider modifying their programs now, to use a replacement routine instead. Replacement routines are already in the library at Mark 8, and the relevant Chapter Introduction should be consulted.

Users who feel that they would be seriously inconvenienced by the withdrawal of a routine, may apply to their sites for a copy of the routine, but should note that NAG does not recommend this course of action and does not accept any responsibility for the withdrawn routine or offer any support for it.

Withdrawn routine	Replacement routine(s)	Comments
C05AAF C05ABF C05ACF	C05ADF	Improved algorithm, more reliable and robust routine. (Other new C05 routines, such as C05AGF, -AJF, -AXF or -AZF, provide additional new facilities.)
C06AAF C06ABF	C06ECF C06EAF	The new routines are not restricted to sequences whose lengths are powers of 2, and are in any case more efficient on most machines.
D01ACF D01AGF	D01BDF D01AJF	Improved algorithms. Both new routines are derived from 'QUADPACK'. 6 other 'QUADPACK' routines were included at Mark 8 and offer more specialized facilities for 1-dimensional integration.
D02ADF	D02HAF	Improved algorithm and re-designed software.
D02AFF	D02TGF	Improved software, designed in conjunction with the easy-to-use driver routines D02JAF and -JBF which handle a single equation or a linear system respectively.
E01ADF	E01BAF + E02BBF	E01BAF uses a more satisfactory form of cubic spline, and, in conjunction with E02BBF, is more efficient when the data is to be interpolated at several points.

ROUTINES TO BE WITHDRAWN FROM THE MARK 9 FORTRAN LIBRARY (CONTD)

Withdrawn routine	Replacement routine(s)	Comments
F01BHF	F02WAF or -WCF	More efficient algorithm and more flexible routine; amount of computation and amount of storage required may be reduced in many applications. The new routine F02WBF now handles the case m<n.
F01BJF	F01BWF	Better performance on paged machines.
F01BKF	F02WDF	The new routine uses the singular value decomposition for a more reliable determination of rank, in those cases where the QR-factorization has not established that the matrix is of full rank.
F01BMF, F03ALF	F01LBF	Better performance on paged machines.
F02BMF	F01BWF + F02BFF	The replacement routines have been shown to be more efficient.
F04AUF	F04JGF	These are companion routines to F01BKF and F02WDF (see above).
F04AVF	F04LDF	Better performance on paged machines.
F01ACF	G04AEF	The new routine offers more facilities and is designed to be consistent with other new G04 routines introduced at Mark 9.
H01AEF	H01BAF or H01ADF	H01AEF was sometimes unreliable. The new routine H01BAF is robust and stable; H01ADF may be faster.

The routine D02AGF will not after all be withdrawn at Mark 9, although its withdrawal has been announced. It allows the user to specify an interior matching point; this facility is not provided by the proposed replacement routine D02HBF, but can occasionally be very convenient. However in other circumstances users are strongly recommended to use D02HBF.

Jeremy Du Croz
NAG Central Office

MARK 8 ALGOL 60

The following 16 new primary routines are included in the NAG Algol 60 Library at Mark 8:

C05ADA	C05AXA	D01FBA	F02GJA
C05AGA	D01AHA	E01BAA	F02SDA
C05AJA	D01AJA	F01BRA	F04AXA
C05AVA	D01BCA	F01BSA	X02AGA

Users of the FORTRAN Library will be aware of the analogous FORTRAN routines: the 5 C05 routines find a simple zero of a single equation and D01AHA and D01AJA are adaptive quadrature routines replacing D01AGA. New linear algebra routines F01BRA, F01BSA and F04AXA address the problems of sparse, unsymmetric linear equations whilst F02GJA is an implementation of the QZ algorithm. E01BAA provides facilities for cubic spline interpolation.

David Sayers
NAG Central Office

A REVIEW OF NAG LIBRARY IMPLEMENTATIONS

Table 1 below summarises the implementation state of the NAG Algol 60 Library. Since the last issue of this newsletter (NL1/81), we are pleased to report that a new machine range version is available; an implementation of the Algol 60 Mark 7 Library for the Honeywell Level 66 (GCOS) systems, carried out at the University College of Wales at Aberystwyth. The first completed implementation of the Mark 8 Library, undertaken by the University of Sheffield, is about to be released.

COMPUTER SYSTEM	LANGUAGE	LIBRARY MARK NOW	LIBRARY MARK NEXT	LIBRARY MARK DUE	COMMENTS
BURROUGHS 6700	ALGOL 60	7	8	*	
CDC 6000/LOWER CYBER	ALGOL 60	7	8		ALGOL 4 COMPILER
CDC 7600 (LCM)	ALGOL 60	7	8		ALGOL 4 COMPILER
CDC 7600 (SCM)	ALGOL 60	7	8		ALGOL 4 COMPILER
DEC SYSTEM 10 (KA)	ALGOL 60	5	8	*	
DEC SYSTEM 10 (KI)	ALGOL 60	5	8	*	KL COMPATIBLE
DEC SYSTEM 20	ALGOL 60		8	*	
HONEYWELL GCOS	ALGOL 60	7	8?		
IBM 360/370 (H.E.P.)	ALGOL 60	7	8		D.P./DELFT COMPILER
IBM 360/370 (S.E.P.)	ALGOL 60	7	8		D.P./DELFT COMPILER
ICL 1900*	ALGOL 60	7	8		NON-1906A/S
ICL 1906A/S	ALGOL 60	7	8	OCT81	
ICL 2900(B)	ALGOL 60	7	8	*	VIA ICL VME/B A(EDIN)
PHILIPS 14/1800	ALGOL 60	6	7		EINDHOVEN COMPILER
TELEFUNKEN TR440	ALGOL 60	7	8		
UNIVAC 1100	ALGOL 60 D.P.	5	-		NU ALGOL DOUBLE PR.
UNIVAC 1100	ALGOL 60 S.P.	5	-		NU ALGOL SINGLE PR.

Table 1: Algol 60 Implementations

A REVIEW OF NAG LIBRARY IMPLEMENTATIONS (CONTD)

With the assembly of the Algol 68 Mark 3 Library now completed, it is appropriate that we include for the first time in this review, a summary of the current implementation status of that Library. Table 2 lists the Mark 2 implementations currently available and those Mark 3 implementations in progress.

COMPUTER SYSTEM	LANGUAGE	LIBRARY MARK NOW	LIBRARY MARK NEXT	LIBRARY MARK DUE	COMMENTS
CDC 7600/CYBER	ALGOL 68	2	3	*	
IBM 360/370	ALGOL 68		3	*	
ICL 1900*	ALGOL 68	2	3	*	NON-1906A/S
ICL 1906A/S	ALGOL 68	2	3	NOV81	
ICL 2900(B)	ALGOL 68		3	*	VME/B
TELEFUNKEN TR440	ALGOL 68	2	3		

Table 2: Algol 68 Implementations

As Table 3 shows, most FORTRAN implementations have now reached Mark 8. A newcomer to the list of available implementations is the CII Iris 80 Mark 8 Library completed earlier this year by the Oceanographic Centre at Brest in France. Another new machine range version is the Hewlett Packard 1000 implementation of Mark 8, performed jointly by NAG and M.A.F.F. Lowestoft personnel. Both single and double precision base versions of the Mark 9 FORTRAN Library have been assembled and IBM implementations of Mark 9 are underway. Some of the other implementations of Mark 9 will commence shortly; the rest will begin in due course. In general, where an implementation of the Mark 8 Library has been completed, priority is being given to implementing the Mark 1 Graphical Supplement before work on Mark 9 commences.

For further details of any specific implementation, please contact us at either of the addresses given at the front of this newsletter.

A REVIEW OF NAG LIBRARY IMPLEMENTATIONS (CONTD)

COMPUTER SYSTEM	LANGUAGE	LIBRARY MARK NOW	LIBRARY MARK NEXT	LIBRARY MARK DUE	COMMENTS
BURROUGHS 5700	FORTRAN S.P.	5	-		
BURROUGHS 6700	FORTRAN S.P.	7	8	NOV81	
CDC 3000L	FORTRAN S.P.	7	-		
CDC 6000/LOWER CYBER	FORTRAN S.P.	8	9		
CDC 7600 (LCM)	FORTRAN S.P.	8	9		
CDC 7600 (SCM)	FORTRAN S.P.	8	9		
CII IRIS 80	FORTRAN D.P.	8	9		
CRAY-1	FORTRAN S.P.	8	9		
DEC PDP 11 (H.F.P.)	FORTRAN D.P.	7	9		IV PLUS COMPILER
DEC SYSTEM 10 (KA)	FORTRAN S.P.	8	9		KA PROCESSOR
DEC SYSTEM 10/20	FORTRAN S.P.	8	9		KI PROCESSOR UPWARDS
DEC VAX11	FORTRAN D.P.	8	9		
GEC 4000	FORTRAN D.P.	8	9		
HARRIS VULCAN	FORTRAN D.P.	8	9		VIA HARRIS
HARRIS VULCAN	FORTRAN S.P.	8	-		VIA HARRIS
HEWLETT PACKARD 1000	FORTRAN S.P.	8	9?		
HEWLETT PACKARD 3000	FORTRAN D.P.	7	9		
HONEYWELL LEVEL 66	FORTRAN D.P.	8	9		GCOS
HONEYWELL MULTICS	FORTRAN D.P.	8	9		
IBM 360/370 (H.E.P.)	FORTRAN D.P.	8	9	*	G COMPILER
IBM 360/370 (S.E.P.)	FORTRAN D.P.	8	9	*	G COMPILER
IBM 360/370 (H.E.P.)	FORTRAN D.P.	8	9	*	H+(LEVEL Q) COMPILER
IBM 360/370 (S.E.P.)	FORTRAN D.P.	8	9	*	H+(LEVEL Q) COMPILER
IBM 360/370	FORTRAN S.P.	8	9		G COMPILER
IBM 360/370	FORTRAN S.P.	8	9		H+(LEVEL Q) COMPILER
IBM 360/370 (H.E.P.)	FORTRAN D.P.	8	9		WATFIV COMPILER
IBM 360/370 (S.E.P.)	FORTRAN D.P.	8	9		WATFIV COMPILER
ICL 1900*	FORTRAN S.P.	8	9		NON-1906A/S
ICL 1900*(G3&4)	FORTRAN S.P.	7	8?		SALFORD F77 COMPILER
ICL 1906A/S	FORTRAN S.P.	8	9		
ICL 1906A/S	FORTRAN S.P.	7	8	*	SALFORD F77 COMPILER
ICL 2900(B)	FORTRAN D.P.	8	9		VIA ICL VME/B F1 & FG
ICL 2900(K)	FORTRAN D.P.	8	9		VIA ICL VME/K F1 & FG
ICL SYSTEM 4	FORTRAN D.P.	8	9		
NORD 10/100	FORTRAN S.P.	8	9		
PERKIN ELMER 32	FORTRAN D.P.	6	8	*	
PHILIPS 14/1800	FORTRAN D.P.	7	8		
PRIME V MODE	FORTRAN D.P.	8	9		VIA PRIME (EUROPE)
SIEMENS BS2000	FORTRAN D.P.	7	8	*	
TELEFUNKEN TR440	FORTRAN S.P.	7	8	NOV81	
UNIVAC 1100	FORTRAN D.P.	8	9		ASCII FTN COMPILER
UNIVAC 1100	FORTRAN D.P.	8	9		FIELDATA E3 COMPILER
UNIVAC 1100	FORTRAN S.P.	8	9		ASCII FTN COMPILER
UNIVAC 1100	FORTRAN S.P.	8	9		FIELDATA E3 COMPILER
XEROX 530	FORTRAN D.P.	5	-		
XEROX SIGMA 6-7	FORTRAN D.P.		7	*	

Table 3: FORTRAN Implementations

Key: *H.E.P. Hardware Extended Precision* *D.P. Double Precision*
S.E.P. Software Extended Precision *H.F.P. Hardware Floating Point*
S.P. Single Precision *S.F.P. Software Floating Point*

[* *in DUE column indicates implementation in progress*]

Steve Hague
NAG Central Office

IN BRIEF

Annual General Meeting (NAG AGM/5)

The fifth annual general meeting of the Numerical Algorithms Group Limited took place on Friday, 18th September 1981 at Imperial College, London. Mr. J.G. Hayes of the National Physical Laboratory was re-elected to the NAG Executive. After completion of formal business, members heard an invited lecture by Professor C.A.R. Hoare, of the University of Oxford, entitled 'Is There a Mathematical Basis for Computer Programming?'.

Finite Element Library

NAG is shortly to take over the marketing and distribution of the Finite Element Library developed by the Science Research Council. This library of Finite Element subroutines and programs is designed as a development tool for those people wishing to experiment with the practicalities of solving partial differential equations. It addresses problems not currently covered by specialist Finite Element packages and provides a valuable teaching tool for those interested in the mathematics of the finite element method. The first machine versions of this library will be available early next year and more detailed information will be provided in the next issue of this Newsletter.

NAG(USA) Inc. Telex

NAG(USA) Incorporated can now be contacted via a telex service:

Telex Number: 254708
Ref: TELESERV DFLD

Message to be directed to 'Numerical Algorithms Group, Downers Grove'.

PERSONNEL

New Staff at Central Office

We are pleased to welcome one new member of staff at the NAG Central Office. Miss Fiona Byrne has recently taken over the post of secretary to the Implementation and Information Group.

NAG USERS ASSOCIATION

The inaugural meeting of the NAG Users Association was held at St. John's College, Oxford from 8th to 10th of April 1981. It attracted 120 users of NAG products, with representatives from 9 different countries. The meeting agreed a constitution and elected the following committee:

Dr. R.E. Huddleston (Chairman)
Sandia Laboratories, U.S.A.

Mr. G. Thackray (Secretary)
ICI Mond Division, U.K.

Dr. G.R. Field
University of London, U.K.

Dr. L. Graney
BP Research, Middlesex, U.K.

Mr. P. Endebrock
RRZN, Hanover, W. Germany

In addition there are two representatives from NAG (currently Dr. B. Ford and Miss J. Bentley).

The next meeting of the Association will again be held at St. John's College, Oxford, with lectures at the Mathematical Institute. The dates for the meeting are Wednesday, 29th September 1982 to Friday, 1st October 1982. Invitations and booking forms will be issued to all user sites during February 1982.

Any users wishing to make presentations at this meeting should write to the Chairman, c/o NAG Central Office, providing a title and brief abstract.

Sites which are not members of the Association will be given the opportunity to send a representative to the meeting. However, numbers are restricted and preference will be given to members. The registration fees for the meeting, for members and non-members, will be announced in the information to be circulated to user sites in February.

MEMBERSHIP OF THE NAG USERS ASSOCIATION

Why join the NAG Users Association? In joining any organization one must ask whether the expense - either in time or money - will be balanced by the return. Since the NAG Users Association (NAGUA) is not a charitable organization, we cannot offer you either tax benefits or that humanitarian feeling that comes from helping others.

What then can NAGUA offer you to make the fee for joining (*small*) and the expense of sending representatives to meetings (*much more expensive*) and possibly even the expense of real participation (*potentially very expensive*) worthwhile?

In my opinion, the whole question can be summed up in one word - COMMUNICATIONS.

At the present, you can certainly write a letter to NAG expressing a suggestion about their services or products and undoubtedly you will receive an acknowledgement of your communication; you are much less likely to receive a reply which gives you an indication of how NAG views your suggestion or the path they may take as a result of that suggestion. This is in no way due to NAG's unwillingness to communicate with you, but is instead indicative of how companies deal with individual inquiries. They wish to leave themselves the broadest possible path for the future.

Let us assume for the moment that NAGUA exists and has a broad base of members. You now have a forum for your NAG inquiries. If you have a problem, suggestion, or query and if your concern is shared by a number of other members of NAGUA, then your concern will receive a fuller reply from NAG. Additionally you may very well find that another member of the Users Association has solved your problem and is willing to share the work already completed. The Users Association will not only allow better communication with NAG, but with a large number of organizations which share common technical problems. It has been my experience that if you are able to visit NAG for a few days, they will take time to tell you how they work, where they want to go and what some of the problems are in getting there. The chances are that not many of you will have that opportunity. However, NAG has demonstrated that it is willing to discuss its goals in detail with an organization such as NAGUA. This sort of communication is in NAG's best interest since it receives many ideas from outside of its own organization. This sort of communication is in the best interest of the members of NAGUA since they can then influence NAG to implement their wishes.

So that is it - communicate through NAGUA - join.

Robert E. Huddleston
Sandia Laboratories, Livermore, CA, U.S.A.