NUANCE : الكِتابُ الخُوَرِزْمِ

NEWSLETTER FOR USERS OF ALGOL ON NOVA COMPUTERS AND ECLIPSES

NUANCE 1 1 DECEMBER 1975

---PREAMBLE---

A. van Roggen

In this first issue of NUANCE, a note on its origin, purpose, and suggested operating procedure is in order. Credit for starting the effort goes to the Chicago area users group (Capone), a center that spews out good ideas with machine gun regularity: they have their own newsletter, regular seminars, a "Help" group, and are more active in one month than the National group is in a year! To be specific: one of Capone's columnists, Larry Wygant, asked in "Tinker's Corner", whether other users of DGC's Algol had encountered the same problems that he faced every time he tried it. The response he got from a small number of regular users led to an exchange of comments and ideas, among which was the clearly felt need for an improvement in documentation and communication more than either the regular manuals or the National Users Group provided. From this first small subset of users, I was chosen to coordinate the NUANCEs, ostensibly because I had more experience. This seems hardly possible, because I have used Algol-60 for about three years only, less than one for the DG version. In any case, I think the newsletter is worth a considerable amount of effort, because it is useful for all parties involved: users will get improved documentation, firsthand listings of problems, solutions, and examples that work (or don't!), and a list of other users who can help with problems faster than the official DGC channels can respond. This holds for me too: although time will be spent on writing and editing, this is minuscule compared to the time wasted on identification of bugs and the trials to bypass them. The lack of Algol-Nova definitions does me in: trial and error has to replace logic. Moreover, I too have questions (e.g. Q6). Finally, I am convinced that DGC also will profit from seeing user's nuances in Algol (variations in NUANCES?), and not only with help in clarifying and improving their documentation. If there is active users feedback, DGC will have more incentive to develop new products and improve existing ones, thereby increasing sales. And here is where the circle closes, because the new products and "fixes" in old ones are just the things that help all users! The argument should make it clear that the effort, to be successful, requires inputs from all involved, and that these notes should not be written by a single person or small group only.

Now how to make it work in practice. I am trying to find as large a group of Algol users and potential users on Nova and Eclipse machines as possible. This mailing list will be weeded out by removal of the names of persons who have not responded. Any response, such as the simple request to remain on the list, will do, but fuller responses will be appreciated. Among these are: other names/addresses, questions for the Question/Answer section, description of problems - especially if you found an answer, general comments or specific applications of Algol (either Algol-60, 68, or Algol-Nova), educational material with or without example programs, language comparisons, guest editorials, or finally any suggestions on possible improvements of NUANCE. Contributions in these and other categories are essential: the newsletter cannot survive without them.

Contributions lead to contents. To be useful, the NUANCEs must be reader oriented, and have as large a basis as possible. Therefore, I will print all contributions received for this purpose, including critical ones. If authors insist on "verbatim ac literatim" treatment, fine, as long as the text contains the signature. To facilitate work for authors with little time, little English, or both, I will edit, quote, or translate as required. Let me know how you want your text treated. What I am leading to is probably the most important point of this lengthy preamble - next only to the request for contributions. The NUANCEs should be strongly oriented to technical aspects of Algol, algorithms, computer use, etc. This can be accomplished best if contributors, including myself and any contributors from DGC, write as individuals and not necessarily as representatives of the Institutions or Companies with which they are associated. Experience as "editor" in a different technical field has clearly shown me that it is much easier for authors to write from this default position; where needed, a waiver can be included without difficulty.

Editorial - A. van Roggen

It is with some trepidation that this last page of Nuance-1 is being filled. Although text "manipulation" is nothing new to us - some would substitute "text mauling" - Nuance also had to be formated and produced, and here scant guidance could be found. Who can predict whether the pages will be pleasing, or even readable? One thing is certain: we have found only a very small group of initial contributors for this issue, too small to keep Nuance going, if it is judged to be useful. Therefore, a large response is essential. As was pointed out in the Preamble, a second issue will be sent only to those who request it. Why not make your request an extended one? We hope that anyone who finds something of interest or learns something new, will also contribute. And institutes of learning can help through student projects, etc.

The shipping and distribution of this first issue is done by Tom Streck from the National Users Group, who volunteered the good services of his office. Many thanks! He, of all people, can find the largest group of Algol users.

Having written already too much in non-Algol for this issue, witness the Preamble, we relinquish the rest of this column for more useful material: Nuances are random variations on a single theme, contributed by a non-uniformly distributed set of random authors trying to get singular, non-random results.

Random Numbers

Random nubers are used very frequently in computation, mostly with a uniform distribution over the interval 0 to 1. Strangely enough, Algol does not have such a generator, but has RANDOM which makes a series of 16-bit "integers", which can be started with the procedure SEED, see Algol p.C-16 and Nuance-1-D5. The following real procedure RNDUNIF transforms two successive random words into a real number with uniform distribution in the required interval. A real number requires two computer words: bit 0 must be 0 (positive), bits 1 to 7 are 100R8 (exponent 0), bits 8 to 15 (the second byte) and 16 to 31 (the second word) are random bits for the mantissa. Because a theoretical argument can be made that the more significant bits in series like RANDOM are "more" random than the least significant bits, the first call to RANDOM is right shifted into the second byte, and then the first byte is set with OR. The second call is copied into the second word as the rest of the mantissa. Note the convenience of the bit manupulation and pointer procedures.

The series RANDOM is built in; I have not been very thorough in investigating its randomness (can anyone give information on this, or on the results of the RNDUNIF procedure?) but so far I have not had any problems with RNDUNIF. For other distributions, it can be used as the starting series. In use, RNDUNIF should be called with a first call like RNDUNIF(A), using a fixed integer A for debugging. Later A can be changed to a random integer, for example the seconds from STIME(A,A,A,A,A,A) which results in an integer from 0 to 59 for A.

N U A N C E,

Newsletter for Users of Algol on Nova Computers and Eclipses

A NEWSLETTER OF VARIABLE SIZE,
PUBLISHED AT RANDOM INTERVALS
DEPENDING ON THE RATE OF FEEDBACK
TO THE EDITOR.

Editor:
Dr. A. van Roggen
DUPONT Experimental Station,
Wilmington, Del. 19898
Tel: 302-772.2581

CONTRIBUTORS to NUANCE-1:

J. W. Schmitt,
DATAD Corp.
PO Box 547, Glen Ellyn
Ill. 60137
Tel: 312-671.3330

L. F. Wygant,
PELAM Inc.
PO Box 66261, AMF O'Hare
Chicago, Ill. 60666
Tel: 312-671-4510

This issue has been made possible by support from the following companies:

E. I. du Pont de Nemours & Co., Inc.,
manufacturer of novel scientific instruments which simplify material analysis.

Data General Corp.,
manufacturer of minicomputers with software that makes them look like maxi's.

IN COMING ISSUES OF NUANCE:

A History of Algol
CMM Procedures and List Processing
String Handling in Algol
Using RDOS Switches in Algol
Readers' responses
LFE and Algol Libraries

```
REAL PROCEDURE RNDUNIF(S);
VALUE S;INTEGER S;
COMMENT RETURNS UNIFORMLY DISTRIBUTED NUMBER
IN THE RANGE FROM ZERO TO ONE;
BEGIN
BASED INTEGER B;BASED REAL BR;
EXTERNAL INTEGER PROCEDURE RANDOM;
EXTERNAL PROCEDURE SEED;
POINTER P2;REAL A;P2:=ADDRESS(A);
IF S<>0 THEN SEED(S);
P2->B:=SHIFT((RANDOM),8) OR 40000R8;
(P2+1)->B:=RANDOM;RNDUNIF:=P2->BR;
END RNDUNIF;
```

???? QUESTIONS --????-- ANSWERS ????

How could anyone answer if he doesn't know in the first place and doesn't profess to know ?

Plato

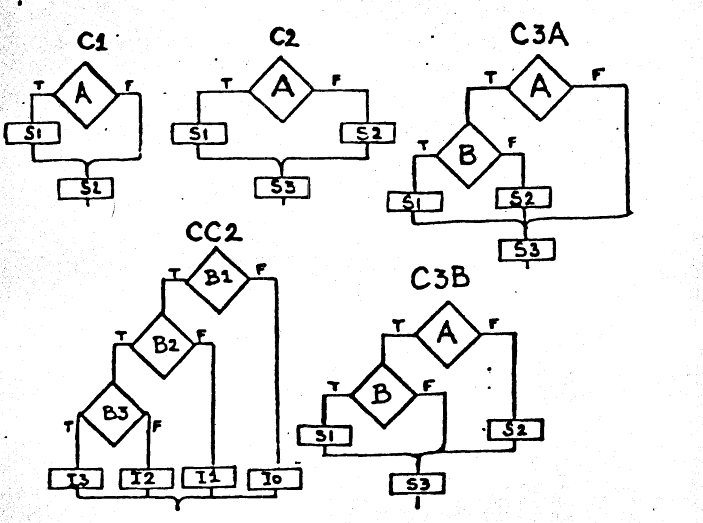
Q1: What are the current versions and tapes?
A: Algol version 2.03 uses paper tapes 88-34-5, 88-35-5, and 88-36-5 (the SV files), and 99-84-3, 99-28-4, 99-29-4, and 99-30-4 as library files. In addition you need one multiply/divide ,tape (the same as Fortran) such as HMPYD.LB, depending on hard- or software M/D, and Nova or Eclipse. The current manual is 93-52-5, with update 86-17-0 (Mar '75), and document 521 of the Algol 2.03 software update.

Q2: How can I get started with Algol on my RDOS system?
A: You will have to LOAD/D/V the SV tapes above, and to XFER the LB tapes. Then make an Algol source file with either EDIT or SPEED, and use the commands shown in the manual at the bottom of p.D9. There are shorter and easier ways to run, especially after you have built up a number of your own general procedures, by using the LFE facility. More on that later.

Q3: Are there any warnings or items to avoid for beginners?
A: If you are familiar with Algol-60 and new to Algol-Nova, beware of integer divisions, entier, and mod procedures which violate either Algol-60 or math definitions, or both, and do not work for negative integers. Programs with these operations or "call by name" need to be rewritten (and debugged!). If you are new to Algol, don't use pointers and based variables until you know how the program runs without them - or wait until you have read the next issue of NUANCE where they will be described more fully. In general, stay with the procedures mentioned in the first part of the Algol manual.

Q4: Is it possible to use octal numbers and arithmetic?
A: Yes, don't worry about the math, the machine will take care of that. Only the input and output are to be translated to octal rather than decimal. All input procedures will read e.g. 143R8 as 99, and for output, use OUTPUT with format e.g. ###R8 to print 143R8. The R8 can be used with P5, for example to increase the precision to a five-word number, either integer or real octal.

Q5: Is there a computed goto in Algol?
A: Yes, there are two: SWITCH, which is a standard Algol (see p.7-14), and the much handier subscripted label, a DGC extension (p.6-10). Use labels such as LB[1]: LB[2]:, and at the branch point compute an integer B. Then use GOTO LB[B] for a multiple choice branch.



Q6: Does anyone have a handler for a TI700 type terminal (upper, lower case, CR delay)? The NUANCE is being produced with it, but not on the NOVA, alas! [A. van Roggen].

Q7: What is the reason for the restriction in nested "IF" statements where only unconditional clauses are allowed after "IF", but any type clause after "ELSE"? [J. W. Schmitt].
A: This is a formal restriction in Algol, but can easily be circumvented. There are two forms of the conditional clause

C1: IF A THEN S1; S2;
C2: IF A THEN S1 ELSE S2; S3;

The logic diagrams are shown below. A nesting of C1, where the form C2 is substituted for S1 in clause C1, gives

C3: IF A THEN IF B THEN S1 ELSE S2; S3;

This statement can have two logic diagrams, which is the reason that it is disallowed in Algol. However, Algol first evaluates expressions in parentheses, therefore C3 can be written properly as

C3A: IF A THEN (IF B THEN S1 ELSE S2); S3;
C3B: IF A THEN (IF B THEN S1) ELSE S2; S3;

This is proper Algol and compiles and runs properly on the Nova. Nested statements with N "IF"s and N "ELSE"s can be written only in one way in a logic diagram, see statement CC1 in D12, this issue; the logic diagram is shown below. Although a compiler could be made to accept this, most Algol systems probably will reject these together with the case of unequal "IF"s and "ELSE"s. How to write it in proper Algol? Use either parentheses as in CC2 (or BEGIN END brackets), or use the device shown in CC5 which pivots the logic diagram.

END; END; END; END; END; END; END; END; END; END; END; END;

THE COVER

Algol, more than most computer languages, is an international language with a strong foundation on mathematics and logic. There is no better way to emphasize this than to allocate part of the cover header to quotes, statements, and other suitable material related to Algol, mathematics, or computers; preferably in the original language. For Nuance 1 I can think of no better one than the present cover which reads in both directions, either "Nuance: the book of Al-Chorismi", or "The book of Al-Chorismi : Nuance", thanks to a minor difference in English and Arabic. Moreover, the word "algorithm" derives from the Arabic, like "algebra" from a single work written about 850AD by Mohammad. His being a ubiquitous name at that time and place, it was common to use three space and time boundaries with the name: "father of Jafar", "son of Moses", and "the man from Chwarism", an oasis in the khanate Chorashmia (NW of Samarkand). This third boundary condition, in Arabic, using a form equivalent to "the Englishman", is Al-Chorism-i, the origin of algorithm. (In English transliterations, "kh" is often substituted for "ch", pronounced as in Loch Ness). Curiously, the word "algebra" comes from one word in the title of the book this early mathematician wrote: jabara = binding together, in relation to the rules of combining numbers and taking them apart.

Send in your favorite quote or artwork (in high contrast print), with a suitable explanation, and it will be used in the next issue!

ALGOL:

INTRODUCTORY OVERVIEW

J. W. Schmitt

The DGC implementation of Algol is basically a string handling language, where "string" is defined as a linear sequence of ASCII characters, and is thus suitable for text handling, list processing, etc. If you want to do anything other than pure numerical computation, try Algol. You might like it. You might even like it for your computational work, because the compiler is based on the algorithmic processing inherent in Algol-60, and DGC Algol has many powerful extensions. There are a few limitations too, such as the exclusion of complex data type. Otherwise, the compiler, coupled with RDOS and its features, offers capabilities which rival medium sized IBM-370 installations.

The following is an overview of some features available in DGC Algol. Only the highlights will be touched, but if there is sufficient reader interest, later issues can offer more details.

Algol is a freeform language: no coding sheets are required because the language and the compiler are column insensitive. The compiler treats the source program as one continuous character string. Extra blanks, carriage returns, tabs, etc. are ignored. Each program statement is terminated by a semicolon rather than a CR-LF, so statements may appear anywhere on one or many lines. Comments also can be imbedded anywhere in a statement. The two statements

```
IF M THEN GOTO STORE←DONE; Q:= ER/B;
```

are equivalent with the commented ones

```
IF M /*MEASUREMENT FINISHED*/ THEN GOTO STORE←DONE;
Q */SELECTIVITY*/ := ER /*RESONANT VOLTAGE*/ / B
/*BANDWIDTH*/;
```

The character "←", ASCII octal 137, really an underscore, serves as connector so that the reader will see two words whereas the compiler sees only one identifier. Identifiers are delimited by spaces, tabs, CR's, and operators (/, +, -, etc) and can be up to 32 characters long.

The source program may also read other source files on disk. For example, INCLUDE BYGOLLY; will cause the compiler to substitute instead of the statement itself, all the source text found in the disk file named BYGOLLY. This text will then be treated and compiled with the remainder of the program. A nice way to handle some modular programming! Procedures (subroutines) may be compiled separately and referenced as EXTERNAL, or may be compiled within the main program. All Algol procedures are fully recursive and re-entrant.

Algol data types include INTEGER (1 to 15 words precision), REAL (1 to 15 words), STRING (0 to 16283 characters long), BOOLEAN (1 word: true or false), POINTER (machine address, 1 word), and LABEL (1 word, a named place in the program). Data conversions between types is handled mostly automatically, and data may be scalar or array in shape. Arrays may have up to 64 dimensions, with positive or negative subscripts, and with full subscript bound testing.

Algol has a block program structure, resulting in dynamic allocation and freeing of memory storage during a run. This same block structure also controls the scope, i.e. the span of recognition, of program identifiers. The use of the same identifier name for different data elements in differing blocks of the program, need not cause conflicts, and facilitates team programming. Moreover, the dynamic allocation of memory allows arrays with computed dimensions to be allocated during execution. Core memory management goes via one or more stacks.

Pointers and based variables allow the programmer to manipulate machine addresses directly. The pointer gives the relative or absolute address, and an associated based variable (any data type and shape) defines how the data at that address is to be interpreted. This is a very efficient way to redefine data or to manipulate large assemblages. It forms the basis for efficient list processing.

A LITERAL in Algol can differ from a string constant. For example, a declaration LITERAL SAMMY(72); will cause the compiler to replace SAMMY with whatever is found in the parentheses, here an integer of value 72.

The programmer can define his own operators in Algol; these are special function procedures that receive two arguments and produce one result. For example S := A OP3 B; will perform operation OP3 on A and B and place the result in S. This can be done either with or without changing the values of A and or B. The normal set of mathematical functions is provided with the compiler. These functions accept either one scalar value or an expression which evaluates to a scalar value, and then perform the required operation to produce the result. Hyperbolic functions are not provided.

Among the most powerful non-math procedures provided are SHIFT and ROTATE to manipulate bits in a word, combined with logical procedures such as AND and OR, if necessary; ADDRESS to get actual machine addresses of a variable, SIZE, HBOUND, and LBOUND to give the number of elements in an array and the upper and lower index bounds of a dimension in the array; LENGTH and INDEX for finding the total number of characters and the position of a substring in a string. Then there is SUBSTR to extract or insert characters anywhere in a string, BYTE to find the numerical value of a specified byte, MEMORY to determine how many words are left in memory for the program, and a whole set of I/O routines, including OUTPUT to write data items via a picture format somewhat similar to a Cobol picture.

A disk file can be set to a specified byte POSITION, the FILESIZE can be retrieved, and the current FILEPOSITION determined. Disk file - names can be manipulated as string variables, and thus computed or looked up during program execution under RDOS. A given file may be RENAMEd or DELETEd. A file OPEN will create a new disk file, or optionally, give an error return when the file referenced does not exist. Command files from the CLI can be read by an Algol program using COMARG; one may CHAIN programs together and manage overlays by OVLOD and OVLAY.

The procedures for Cache Memory Management provide efficient disk file access with a great deal of transparancy when handling large files and data collections on disk.

Now let us look at the program structure and those features which are of help in modular programming. A simple program might be written as BEGIN INTEGER X; X:=7; END;. The words BEGIN and END bracket the program and tell the compiler where to start and stop, especially with regard to the scope of program identifiers. Here, X is the only identifier (program variable) and is declared to be an integer of default (1 word) precision. X is known between BEGIN and END, i.e. in the block where it is declared, but elsewhere is undefined.

This well defined scope of identifiers allows the following sort of programming technique. Programmer Brown can expand the example:

```
BEGIN    /*BROWNPROG*/    INTEGER    X;    X:=7;
OPEN(6,"$TTO"); WRITE(6,X); ...  ...  ; END;
```

Programmer Jones in the meantime can write a separate module:

```
BEGIN /*JONESPROG*/ STRING (16) X; X:="JONES";
.... END;
```

The second module can now be placed inside the first one, and the pair compiled without conflict. If the Jones module is stored on disk as JONESPROG.MD under RDOS, then merely adding one statement to Brown's module, will handle the replacement:

```
BEGIN /*BROWNPROG*/ INTEGER X; X:=7;
OPEN(6,"$TTO"); WRITE(6, X); INCLUDE JONESPROG.MD;
... END;
```

If Brown's program is compiled, the compiler will handle the following:

```
BEGIN /*BROWNPROG*/ INTEGER X; X:=7;
OPEN(6,"$TTO"); WRITE(6,X); BEGIN /*JONESPROG*/
STRING X; X:="JONES"; ... END; ... END;
```

There is no conflict between the two differing uses of X. X switches from an integer in the outer block (Brown's) to a string in the inner block from Jones. To make this useful, we still need interblock communication. This can be done by procedures, or by global identifiers. In the following, Z is global because it is not redefined in the inner block:

```
BEGIN /*BROWNPROG*/
INTEGER X;
STRING (16) Z; /*A NEW IDENTIFIER*/
X:=7; /*Z HERE UNDEFINED VALUE*/
OPEN(6,"$TTO");WRITE(6,X);
 BEGIN /*JONESPROG*/
 STRING (16) X; X:="JONES";
 Z:=X; /*GLOBAL Z ASSIGNED TO
       LOCAL X*/
 ... END; /*OF JONESPROG/*
 .... /*BACK IN OUTER BLOCK WHERE
NOW Z CONTAINS "JONES"*/
 ... END;
```

Since Algol is freeform, it makes no difference to the compiler which way the example was typed: in-line earlier, or as here on separate and mixed lines. Using indentation, etc. can make the documentation easier to read.

Now interblock communication via a procedure block. Procedures are much more general than subroutines (all subroutines are procedures, but there are procedures that are neither subroutine nor function), but can be thought of as a subroutine where the scope is concerned. A procedure follows the same rules as a BEGIN block for the scope of identifiers. In addition to communication through globals, a procedure may also communicate through its parameters (arguments). The parameters are a programmer specified list of identifiers within the procedure which are matched one for one with the arguments of the procedure call in the calling block. A parameter may be matched to any argument of proper data type, therefore different arguments may be used with the same procedure parameter at different places in a program, like calling SIN(X) and SIN(Y) at various locations. A global on the other hand, can only match itself from block to block.

If programmer Simon has written a procedure called SIMON.AL on disk, and that procedure has been compiled separately, we can reference it as follows:

```
BEGIN /*BROWNPROG*/ INTEGER X; EXTERNAL PROCEDURE
SIMON; X:=7; SIMON(X); ... END;
```

If we now tie together all these various and sundry rules about the scope of identifiers, interblock communication, the INCLUDE and LITERAL extensions of DSC, and the many available data types and shapes, remembering that Algol is fully reentrant and recursive, we see a very powerful means of handling modular programming and top-down programming, either by individuals or by teams.

Good definitions of the interfaces between a block and its outer world (either another block or the peripheral devices) will allow programming of even a large system without every participant having to know or care about the inner details of any program blocks except the one he is writing. As long as the interface requirements are met through globals and parameters, the nitty gritty inside of any other block becomes ignorable.

Flow control can be changed from strictly sequential execution by

1) GOTO JABBERWOCK; this will cause the program to make an unconditional jump to a location in the program labeled JABBERWOCK: . Subscripted labels such as L[1]: and TEMP[-8]: will allow computed GOTOs.

2) SIMON(2.5); This causes the program to execute procedure SIMON, using all identifiers within the scope with their current values, and then returns to the point in the program immediately following the procedure reference. Here it is the next statement, it could also be a containing statement if the procedure is a function.

3) FOR CV := <list> DO S; This permits repetetive execution of the statment or block S with the controlled variable CV set to values specified by <list>. For <list> one can use a simple list: V1, V2, ...Vn, which are assigned to CV from left to right until the list is exhausted. However, the FOR clause is very powerful, because each of the elements could be either a simple value, an expression yielding a value, a STEP .. UNTIL .. clause specifying a range of values, or a WHILE .. clause, causing repetition until a boolean value becomes false. All of the following examples are valid:

```
FOR N:= START STEP INCREMENT UNTIL MAX DO SUB7;
FOR N:= 17,468,1 STEP 1 UNTIL 10, 900 DO SUB8; FOR
N:= IF GLOOM THEN DOOM ELSE HAPPINESS DO PARTY;
FOR POTENTIAL:= VOLTAGE WHILE AMPERAGE < 75 DO
WATTS := 0.767 * POTENTIAL * AMPERAGE;
```

4) IF statements can be used to make logical decisions as to which blocks will or will not be executed.

The IF statement is one of the major features that makes Algol different from many other programming languages. Properly used, it can eliminate the need for most GOTOs, and make the program logic much clearer. This statement is of the form

```
IF BE THEN UC [ELSE AC];
```

Here BE is a boolean expression, UC an unconditional clause, AC is any clause, and the brackets enclose an optional portion of the statement. If BE evaluates to TRUE then UC will be executed and not AC (if present); if BE = FALSE, UC will not be executed but AC will be when present. In both cases, the following statement will be executed after the IF statement is done.

Suppose each day of the week is declared a boolean variable. Then

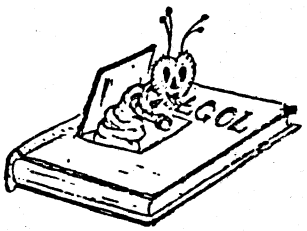
```
IF MONDAY THEN UC1 ELSE IF TUESDAY THEN UC2 ELSE
IF WEDNESDAY THEN UC3 ELSE .... IF SUNDAY THEN
UC7 ELSE AC; NXTST;
```

will cause the program control to jump down this set of conditionals until it hits the first day which is TRUE. It will then execute the corresponding UC, and finally jump to NXTST. If none of the days is TRUE, then AC will be executed, followed by NXTST.

Any IF statement can be reversed by using IF NOT to change flow of the program from UC to AC in a previous example.

The structure of the IF statement makes it very readable and therefore self documenting to a large extent. Furthermore, it can control very large program blocks through the BEGIN END brackets.

[Excerpted from the Capone Newsletter]



DOC BUGS CRAWLS OUT :

NOTES ON ALGOL DOCUMENTATION AND BUG REPORTS

S'il ne trouve pas d'erreur, il semble déçu. S'il en déniche une, il est furieux. P.Daninos.

D0 -
Documentation changes, additions, and corrections are grouped together with bug reports: these two are difficult to keep apart. One man's meat is another man's bug. Try roasted termites, a delicacy on Java. This column is meant for short notes only: more extensive items (see e.g. the Procedure List) will be handled separately. Items that already have been updated by DGC should not appear here. Cross references can be made, such as Nuance1-D0 to this general remark.

D1 - GTIME, STIME
The manual gives as offset for the year the number 1968 (p.9-30). Both procedures use 1900, so that 1975 should be entered as 75 in STIME, and returned as 75 in GTIME. Both procedures have to be used with an EXTERNAL declaration.

D2 - ALLOCATE, FREE
The allocation of storage words for a pointer does not have the scope of regular variables, which are dynamically assigned. Once allocated, the area remains even after exiting from the block. Therefore, FREE must be used, especially in recursive procedures, to make the allocated area avaalable again.

D3 - STRING1 < STRING2
Don't use this if you expect alphabetical order. Apparantly, this is only a direct call to STRCMP which first compares string length and only then compares values. It results in A < B < X < AA < AZ < ABCD. Extremely unuseful - I hope DGC will consider it an oversight and correct it.

D4 - Misprints, Algol Manual
1. The listings of descriptor 1 on p.9-36 and 9-37 do not match: is it time, address, or vice versa in words 3 and 4? Check before using this.
2. First example, p.6-6 has S as both array and real (?) variable.
3. The last sentence on p.5-4, change "keyboards" to "keywords".
4. The block with ENTIER, p.9-2, has a "(" missing.
5. On p.C-16, 31, 32, and 33 the non-Algol "◆◆" is used instead of "↑" for exponentiation.
6. On p.9-9, a ")" is missing in the block with ASCII

D5 - RANDOM
This procedure should be declared EXTERNAL: in the series it generates (p.C-16) the constants are A:=8197=20005R8 and C:=13849=33031R8.

D6 - ASCII and BYTE
These procedures have differing runtime descriptions (p.C-13 and 17) but are treated as identical (p.9-8). Why have two? Which one is "better" and what is the difference? I have not found any operational difference.

D7 - CLASSIFY
The corrected description on p.9-9 works indeed, except that it fails to assign the lowest range. I have no answer yet from the STR (software trouble report) whether this is a bug. Meanwhile, you could start the class table with "0 0 0" or such to give CTRL-A a class by itself (This is not seen by Algol anyway because it is intercepted by RDOS as interrupt).

D8 - LINEREAD
Incredibly, LINEREAD is terminated by not only CR - LF, but also by a null byte. Try to read Algol listing files, -.LS, which contain many null bytes. READ fails when a line starts with ", (believing that this is the string delimiter), LINEREAD fails because of the null byte, e.g. after the Algol generated linenumber. LINEREAD should terminate only on CR (or LF); a "BLOCKREAD" or such would be more suitable for a mixed set of delimiters. There are other problems with LINEREAD, see the comments from J. W. Schmitt elsewhere in this issue.

D9 - OUTPUT
The E format for numbers is always 0.nn..nnEnn, even when . E is specified. Specification of radix is allowed, e.g. R8.

D10 - INCLUDE
Bad code is generated when INCLUDE is the first statement after the program declarations: BEGIN INTEGER A; INCLUDE OPFIL.DC; A:=5; END;. Until this is fixed, use a dummy statement (;) before INCLUDE: BEGIN INTEGER A; ; INCLUDE .. etc.

D11 - Exponentiation (↑)
Like the errors in ENTIER, etc. where integer arithmetic fails, there are bugs in the ↑ operation. For example, <INT1> := <INT2>↑<INT3> is bad when INT2<0, and when INT2=2 while INT3>0 then INT1 is calculated as an odd number (with a few exceptions)! Until the mathbugs are fixed in the integers: ◆◆◆ BEWARE OF INTEGER ARITHMETIC ◆◆◆

D12 - Conditional Statements
This is an error message bug. Consider the following statements.

```
CC1: IF B1 THEN IF B2 THEN IF B3 THEN I4:=I3 ELSE
I4:=I2 ELSE I4:=I1 ELSE I4:=I0;
CC2: IF B1 THEN (IF B2 THEN (IF B3 THEN I4:=I3
ELSE I4:=I2) ELSE I4:=I1) ELSE I4:=I0;
CC3: I4:= IF B1 THEN IF B2 THEN IF B3 THEN I3 ELSE
I2 ELSE I1 ELSE I0;
CC4: I4:= IF B1 THEN (IF B2 THEN (IF B3 THEN I3
ELSE I2) ELSE I1) ELSE I0;
```

CC1 and CC3 violate Algol rules (p.6-10) but no error message is generated. Curiously, with I0=0, I1=1, etc and B1=B2=TRUE, B3=FALSE, CC1 gives I4=0 (wrong, as expected) but CC3 gives I4=2, the correct result for wrong coding! CC2 and CC4 show the proper coding; they run properly. The parentheses can be replaced with BEGIN END. To code this more elegantly without parentheses, use

```
CC5: I4:= IF NOT B1 THEN I0 ELSE IF NOT B2 THEN I1
ELSE IF NOT B3 THEN I2 ELSE I3;
```

See also Question Q7, this issue.

A. van Roggen on:

PROCEDURES
THAT ARE

RECURSIVE
RECURRECURSIVESIVE
RECRECURRECURSIVESIVEURSIVE
RECURECRECURRECURSIVESIVEURSIVERSIVE

Algol and several other "higher" languages (e.g. Lisp) have features not found in simple, straight-through languages such as Basic, Fortran, and Snobol. These features make programming much easier and faster, and usually result in shorter source programs, sometimes combined with faster program execution, at least in more complicated cases. One of these advanced features, recursion, will be discussed here.

A procedure - subroutine in Fortranese, the FNx functions in Basic - is recursive if it calls itself. Fortran and Basic specifically exclude such procedures, but in Algol it is a very powerful technique. Before discussing why and when it is useful, let us look at an example. I don't want to use the ubiquitous FACTORIAL(N)=N! which can be found in almost any textbook (e.g. Algol manual p.27), but a slightly more complex one: FIBONACCI(N). In the following, I will use various abbreviations, F(N), FIB(N), etc. to distinguish differing definitions. The Fibonacci numbers are integers of a sequence with F(0)=0, F(1)=1, F(2)=1, F(3)=2, etc. In general, F(N) = F(N-1) + F(N-2), thus each number is the sum of the two previous ones. Nowadays, nobody dares use a series unless it is relevant; this one is if you are a farmer or other generalized horticulturist. In a plant or tree where two branches come together, the sap flow below the axil is the sum of the flows from the two previous axils. If the only tree you recognize is a binary one (horticulturists would call it Arbor binaria), you will already know the use of the Fibonacci series in mathematics and list processing.

With the defining equation above, writing the procedure code with recursion is elementary, see FIB1 in the example below. The procedure heading, required by Algol for any procedure, is followed by a single statement. This statement sets the integer result of the procedure call, FIB1, equal to FIB1(N-1) + FIB1(N-2), unless N=1 or 0, when FIB1 is set equal to N. This "escape" clause is an essential element in recursive definitions. You see how nicely and smoothly these can be incorporated using Algol's conditional statements.

The general recipe for writing recursive procedures is to find a simple or trivial solution (here N<2), and work the recursion toward the trivial case. When you go the other way,DGC cannot sell you enough memory boards to find a solution! Here it happens for a call FIB1(-2), or any other negative argument. The computer attempts to solve FIB1(-3) and FIB1(-4), etc. In the main program of the example, a negative argument is used to quit (GOTO ZZ), and it also eliminates calls that would cause overflow for the single precision used (N>23).

In contrast to the simple recursive coding is that of the non-recursive one, FIB2. Here a loop is used with F as the new number, F1 and F2 as the previous ones. A little trick is needed to get the loop started correctly for N<2. This could have been done more laboriously with a conditional statement, as in FIB1, but this would make the coding even longer.

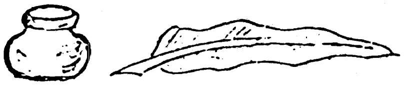
Which of the two is better, FIB1 or FIB2? As usual, it depends. FIB1 is very rapid to write, hardly needs a "test" program, but has more overhead than FIB2 because it has to build new blocks in memory when it calls the procedure for smaller arguments. Therefore, for a quick solution, FIB1 would be better, but for a permanent procedure, FIB2 is faster. Actually, both are slow and awful, and there is a much better solution which is shorter in coding than FIB2 and faster in running (on the average) than either FIB1 or FIB2, and which also takes care of negative arguments in the proper manner: FIB(-5)=5, FIB(-6)=-8, etc.

To help in getting responses to NUANCE, I am inviting (1) coding of a better Fibonacci procedure [All submitted solutions will be discussed or printed in NUANCE, together with mine. If a better one is sent in, I will update my Algol MATHLIB pronto!], (2) a description of your favorite recursive procedure , and (3) comments on recursion in general.

It is dangerous to generalize too much, but usually such simple algorithms as above are handled more efficiently without recursion by the computer, although, as indicated, not by the programmer! The situation changes drastically when the algorithm has a variable recursion, computed each time. String handling, such as parsing of text to separate words from a string, is the most common example. If all words were four letters long, language would suffer, but the computer program would be simplicity itself. Recursive programs for parsing have less overhead than non-recursive programs because not all possible decision branches need be written: a template will suffice. Recursive programs also are vastly easier to debug.

```
TYPE FIB.AL;ALGOL FIB ← RDOS commands
BEGIN
/* DECLARATIONS*/
INTEGER A,B,C;
LITERAL QU("<15>N: "),TB("<11>");

/*RECURSIVE PROC:*/
INTEGER PROCEDURE FIB1(N);          }  → { procedure
VALUE N;INTEGER N;                  }    { heading
FIB1:= IF N<2 THEN N ELSE           }  → { procedure
  FIB1(N-1)+FIB1(N-2);              }    { body

/*NON-RECURSIVE PROC:*/
INTEGER PROCEDURE FIB2(N);
VALUE N;INTEGER N;
BEGIN
INTEGER F,F1,F2,I;
F:=F1:=Ø;F2:=1;
FOR I:=1 STEP 1 UNTIL N DO
 BEGIN
 F:=F1+F2;F2:=F1;F1:=F;
 END;
FIB2:=F;
END FIB2;

/*START MAIN PROGRAM*/                      { simple interactive
OPEN(1,"$TTI");OPEN(2,"$TTO");              { question-answer
AA: WRITE(2,QU);READ(1,A);                  { program
IF A<Ø THEN GOTO ZZ ELSE IF A>23 THEN GOTO AA;
B:=FIB1(A);C:=FIB2(A);
WRITE(2,A,TB,B,TB,C);
GOTO AA;
ZZ: END

PROGRAM IS RELOCATABLE ← compiler message
R
.RLDR/M FIB ALGLIB;FIB ← RDOS commands

                          ~~~ user response
N: Ø
Ø          Ø         Ø
N: 2
2          1         1
N: 5
5          5         5
N: 8
8          21        21
N: 12
12         144       144
N: 2Ø
2Ø         6765      6765
N: 23
23         28657     28657
N: 3Ø

N: -2
R
```

LETTERS



LETTERS

THOUGHTS ON AN ALGOL NEWSLETTER

L. F. Wygant

"Tinker's Corner" is my monthly column in the Newsletter for the Chicago area Programmers of Novas and Eclipses (CAPONE). A column on Algol some time ago prompted a series of letters between Dr. van Roggen, myself, and several other Algol users. Now that Arend has taken on the editing, publishing, and writing much of, Nuancé, a true newsletter for the DGC Algol community, he has asked me to excerpt pertinent material from past Tinker's Corner atricles. Here they are, with updated comments interspersed.

June 1975:
Since I wrote last month about Fortran, I thought it would be nice this month to say something about Algol. Deep in my heart I know that there are just zillions of suppressed Algol freaks out there, using DGC equipment. Myself, I don't use Algol much because it does not work as well as Fortran, at least not on DGC machines, nor has it been given the system control features that Fortran has been given. However, we do have a gentleman named Joe Schmitt who swears by Algol and actually is able to make it run, bugs or no bugs!

I don't want to give the impression that I am trying to denigrate Algol. The above is more than anything else an attempt to start some dialogue on Algol, and possibly help out some people who are brave enough to try the language. I firmly believe that if Algol had received from DGC the same amount of attention as Fortran, it would be at least in the same class of usability that Fortran is. I would like to see that happen.

In fact, herewith I sollicit responses from anyone in the audience who is interested in seeing Algol as a usable language. Please write me and give me your opinion on how, and what kind of, support should be given.

July 1975:
Much to my surprise, I got some good responses to the Algol column last month. J. Mariott, of Ann Arbor, Mich, while not an Algol user, said "I feel that if a Company offers a language, they should support it to the fullest extent possible". Dave Busse, here in Chicago, has the most appropriate description of the DGC Algol compiler; he remarked on its "somewhat whimsical nature". Arend van Roggen called from Wilmington, Del; he uses Algol almost exclusively and has the good news that the latest version is relatively 'clean'. Arend also mentioned that he was very surprised when he found out that Algol does not support multitasking - the more so since he bought his machine explicitly for Algol! I can sympathize with him; I didn't find out either until I sat down to try the multitasking and found out it did not exist. Arend says he has been able to work around that problem by writing secondary tasks in assembler and using the '.XMT' and '.REC' to communicate between Algol and the assembler tasks, although that still leaves you with only one Algol task. Joe Schmitt, another heavy Algol man, has managed to get to a number of systems calls: this by going into assembly code at the intermediate stage after Algol has been translated to assembler. At any rate, there is some interest in Algol. There are enough of us who are interested, and we are going to start a round-robin letter to discuss Algol.

August 1975:
In the meantime, we are stil looking for people interested in some interaction on the subject of Algol. The Algol letter is now making the rounds. I would like to see the same sort of interest started in programming techniques, and in Fortran for commercial users; also, Jim LeRoy is attempting to get something going in the database area.
[Note: of these, only the Algol effort survives; the other three died because there were no Arend van Roggens and Joe Schmitts in their communities of interest. Nuance could die the same death unless you respond and support it.]

October 1975:
Given that there is a ratio of hundreds of users to only dozens of DGC software people, at the plant and in the field, it is inevitable that most problems will be detected first by users. Also, users are constantly working with myriads of diverse applications, while the DGC people are split between answering questions, maintaining old software, and developing new software at the plant. The DGC people almost never get involved more than superficially with any application programs. They will just not be able to fix any software bug unless a user tells them how the bug arises and behaves. You, the user, have the responsibility to yourself, to DGC, and to other users to report any error you happen to notice - preferably on a DGC Software Trouble Report (STR) form, with all pertinent background information. If you don't do your part, they cannot do theirs.
[Note: This has special pertinence to Algol users, since very few Algol problems seem to find their way back to Southboro.]

Now we come to information exchange. Almost anyone is willing to give someone else a break and mention a specific problem that they know of, if they know that that person has an interest in the problem. The most basic difficulty in the minicomputer world is that people who do have such valuable facts to share don't see each other very regularly. Most are in small shops, each of which is an island in itself.
[Note: Algol users are even more widespread, being a subset of DGC users in several continents. This makes Nuance all the more important as a means of information dissemination.]

The purpose of a users group is *contact*! Not slavering over the latest machines, or lusting after the newest software, or drinking beer and feeling loved, but contact! Personal interaction with other poor souls in the same boat as yourself. If I know a few wrinkles, and you know a few, and George knows a few too, then we all can get together and talk, and each will know three times as much (well, twice, anyway). If each of us would write it up for the newsletter, then everyone will know [and we won't be just passively picking Joe's or Arend's brains].

But, there is no such thing as a free lunch. It only works if people make it work. If you want to see lots of good articles in the newsletter, you have to encourage people to write them. What better way than to start it yourself by writing a short note or question on a subject you are interested in? Anyone who is only giving will soon grow bored and quit; anyone who is only taking is dependent on random handouts. Make an exchange, and all parties are helped and happy. Nuance is an unique forum for such an exchange.

[Note: The gist of all this is that Nuance has been a long time coming. Dr. van Roggen has undertaken to transform a disconnected correspondence among Algolisti (if I may borrow his word) into an orderly and informative newsletter. He has voluntarily taken on himself the job of creating, editing, and publishing. We Algol users who benefit from his work can do no less than contribute to his effort by writing articles for Nuance, or at least send our letters of thanks to Arend. For myself, I want to publicly thank him. In starting Nuance, he has done what few of us have the courage, public spirit, and responsibility to attempt.]

DGC's EXTENDED ALGOL AND A USERS GROUP

J. W. Schmitt

DGC's Extended Algol-60 is an impressive product. We use it with good success for string handling, text processing, and list processing. One must always keep in mind, however, that it is not Algol-60, but rather an Algol-60 like language with some liberal borrowings from PL/1. The point: programs written in DGC's Algol are not transferable to other manufacturer's machines, or vice-versa, without translation.

While we have had good success with Algol, we have also had some severe frustrations. Some of these come from bugs, which DGC has been good in fixing, while more misery comes from what the documentation does not say.

The Algol documentation which does exist is generally correct, concise, well written, and - alas - incomplete. Where for example, does it tell that the LINEREAD and LINEWRITE procedures are limited to handling 132 characters at a time? Where does it tell (other than the notification sent to Algol subscribers after a complaint) that an attempt to LINEREAD a few characters into a 40 word area allocated at the beginning of a program will cause a program abort with misleading error message because RDOS insists on having at least 67 words available below the lower bound of the operating system in core? Where does it tell one what happens when -1 is fed to SUBSTR as the length? Try that for some really challenging detective work. We wiped out our master device with it!! Apparently, SUBSTR did no sign checking and wrote a substring of 65536 characters all over the place. Or did it?? We couldn't really say what happened except that it took five hours to rebuild our disks.

An Algol users group, through a newsletter and other publications, can serve several important functions in addition to pleasant chit chat:

1) Bug bulletins. Bugs reported to DGC usually get fixed, but it does take time. A quick "hot line" notice that a bug exists, could save a lot of trouble for a lot of people. DGC, like everyone else, is not going to spend its own time and money publicizing faults in its own products. An independent users group could fill this gap, as long as it maintains a sense of balance and fairness. If bug bulletins merely deteriorate into a way to take potshots at someone else's work, then we don't need them. But if they fairly and accurately report real bugs - and promptly follow up when the bug is fixed - they can serve a real purpose and would be worth paying for.

2) Better documentation. We have spent a good part of the last year writing test programs to determine what various runtime procedures do and don't do. [Did you know that LINEREAD unconditionally expunges all linefeed characters, that LINEWRITE sometimes (!) changes a carriage return to a carriage return with linefeed, that OPEN(N,"$LPT") always generates a form feed, or that the first byte in a disk file is byte 0 rather than 1? Do you know what happens to BEGIN INTEGER ARRAY KK[A:B] when B<A during a run? None of this is documented]. It has cost us a great deal of time and money to find out. The users group and DGC together should produce more thorough and extensive documentation, especially on the runtime procedures. Only then will we not have to repeat the detective work again and again. Appendix C of the Algol Manual contains a wealth of procedures without telling how to use them, e.g. OVLAY, OVLOD. Page C10 provides the only clue, and not a very comprehensive one.

3) Application program library. Some of us find ourselves wanting to do the same thing over and over again in many programs. Writing more of the same code every time becomes a nuisance. Using INCLUDE to bring in pre-written source modules is one answer - a good one, thanks to DGC. A library of external procedures is another possibility. Dr. van Roggen at DuPont, for one, wonders about compiler default channels for $TTI and $TTO. On the other hand, default channels may get in the way at times, e.g. in programs using no terminal. Would it help to have a procedure CONSOPEN(KEYB,1,SCREEN,2) so that ensuing refernces to KEYB and SCREEN have the desired results? But should every user write his own CONSOPEN? Perhaps a MACROPEN would be better, such as MACROPEN(F1,V1,F2,V2,...) where the F's are associated with the program variables V, and channel numbers are assigned automatically and invisibly. Who is to write such routines? Who is to verify them and distribute them? It all takes time and money. Can a users group distribute such a library at cost? Should contributors be paid for their efforts? Free programs, as we often learn, are worth about what we pay for them! I don't have any easy answers, but the subject is worth some discussion. Otherwise, each and everyone of us will forever be reinventing the wheel.

4) Information exchange. The above deals with information exchange, butonly in limited areas. Exchange at a higher level is useful too. Who has test timings on run-time efficiencies? How much time, if any, do contiguous files save? Who knows the best techniques for list processing? [See Nuance2 for at least one type of list handling with CMM. (Ed)] Is there a package for application X? Has anyone four or more terminals running concurrently under Algol, and how? A newsletter with such information might save a lot of time and money for all of us. Who will pay for producing such a letter?

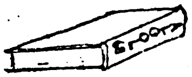
Much good work in these areas can be done if we organize it well, manage it competently, and somehow get the beneficiaries to pay the tariff. Who is willing to work on it?

END; END;

/* For discussion of this procedure, see ASSEMBLY LANGUAGE PROCEDURES, p. 10 */

```
; PROCEDURE TME(H,M,S);
        .TITL   TME
        .EXTU
        .ENT    TME
        .ZREL
.LP:    LP+200
        .NREL
        .TXTM   1
TME:    JSR     @SAVE
        FS0
        1B7+3
        SP+0                            ;H
        001021                          ;INTEGER PARAMETER
        SP+1                            ;M
        001021                          ;INTEGER PARAMETER
        SP+2                            ;S
        001021                          ;INTEGER PARAMETER
; INTEGER H,M,S;
; BEGIN
        .SYSTM    system call .GTOD gives seconds in AC0,
        .GTOD       min in AC1, and hrs in AC2.
added:  JMP     .+1     no error return
        STA     0,S+2,3     sec to parameter 2
        STA     1,S+1,3     min to parameter 1
        STA     2,S+0,3     hrs to parameter 0
; END TME;
        JSR     @RETURN
FS0=    3
L=      -200
S=      -167  stack displacement, start of parameters
SP=     1B0
LP:
        .END
```

You find not the apostrophus, and so miss the accent; let me supervise the cansonet. Here are only numbers ratified; but for the elegancy, facility, and golden cadence of poesy, caret.
Shakespeare.



REVIEWS

E. W. Dijkstra: "A PRIMER OF ALGOL-60 PROGRAMMING"
Acad. Press, NY, 1962 (114 pages).

This is a small book, but valuable for both Algol novices and for expert Algolisti. For the former group, the text gives a concise and clear explanation of Algol features, and how to write programs in Algol. Novices should not read the Appendix (the official definition of Algol-60), which is meant for more advanced users – who will not read it either, but just nibble the specific parts needed each time. They will find it essential in their program writing. A liberal use of the Appendix will decrease the debugging time enormously, at least in Algol-60. For input and output procedures, and the extensions that each installation has, the manufacturer's books must be consulted. And here is hoping that Algol-Nova also will get its definitions, with a similar increase in program writing efficiency! Meanwhile, for DGC machines, use manual 93-52-5, and such tables and information as can be found in Nuances. [A. van Roggen].

DGC: "ALGOL CACHE MEMORY MANAGEMENT" (Application note No. 17-16-0), Aug. 1974, 20 p.

DGC's manuals usually are relatively well-written, perhaps second only to IBM's. This manual is a sorry exception. It is as bad, in its own way, as the "Absolute Disk Editor" manual. The introduction starts out on a humorous note, saying that extended Algol without CMM (Cache Memory Management) will handle efficiently most scientific and business programming, and then advances to hilarity by noting that CMM is used for handling "large data bases", defined as three or more times larger than core. Good grief! Three or more times 32K a large database? There are samples of the calls for each procedure involved, but only one real example – the last in the manual, showing how to decide whether to call HASHWRITE or not (with rather poor logic, by the way). There are no sample programs, and there is no meaningful discussion of the use of CMM procedures. DGC standard nomenclature is often violated, e.g. using "file number" instead of "channel". On the whole, this manual approaches, from the downward side, a programmer's notes to himself about something he already understands thoroughly ... a memoranium with a cover on it. Hardly suitable as a manual. [L. F. Wygant].

M. Levinson, R.G. Ward, J.W. Webb: "THE SETTLEMENT OF POLYNESIA", Un. of Minnesota Press 1973.

This is an enormously fascinating book. Anyone who has ever flown over tropical oceans with white-rimmed islands, hardly discernible in the distance, even from a vantage point among the clouds, can hardly escape the question how sparsely scattered islands such as Polynesia could have been settled in previous centuries. This question itself has been "settled" experimentally by Heyerdahl on a one-way trip by raft from Peru to Tuamoto. And now there is this book with more evidence – and the reason for a review in NUANCE – from a computer model in Algol, with much of the source program in an Appendix. Basically, the trajectory of the rafts is calculated from wind and water current data; if the day's trajectory intersects the sighting circle of an island (and not necessarily the first one that day!), landfall is assumed. The huge amount of data for wind and current were stored on magtape; the raw data were gathered mainly from the British Navy. What a job that must have been: transcribing it probably takes as long as Heyerdahl's trip. However, the computer made up for this, and many "trips" were made, giving good statistics on the problem, and some interesting conclusions. The program has interesting and at least for a scientific programmer, rather gruesome procedures. For example, the procedure TERMINATE has a declaration SWITCH ALTERNATIVE:= CREWPERISH, OUTOFBOUNDS, GALE. The book is worth reading for the programming as well as for the main contents: the plastic overlay with the continents and islands on it makes the comparison between various assumptions in the model extremely easy and clear, and the program documentation is excellent. [A. van Roggen].

ASSEMBLY LANGUAGE PROCEDURES

A. van Roggen

In many cases, Algol, like most higher languages, must be complemented by assembly language patches. For example, interrupt handling of various interfaces, and program loops that have to be repeated often and hold up the program execution, are candidates for assembly programs. Unfortunately, system calls are also on this list. Assuming an elementary knowledge of ASM, a rapid method to make ASM patches in Algol programs is as follows: Make the required operation a procedure, and assign the parameters, and write this in standard Algol. For example, to duplicate part of the GTIME procedure, write the new procedure in a file called TME.AL, which contains PROCEDURE TME(H<M<S); INTEGER H,M,S; BEGIN END TME;. This procedure, with the hour, min, and sec parameters, just exactly does nothing, except set up the required frame in ASM. Compile TME.AL with the RDOS command ALGOL/S TME, and presto, you have a file TME.SR with the frame of the procedure in ASM. Now add with an editor in TME.SR the required coding, here shown at the bracket, , and compile with ASM TME to get the required TME.RB. In the main program, TME should be declared EXTERNAL, and the LOAD command should have TME follow the name of the main program, before the library name. The systems call shown here is ridiculously simple, just good as example, because all the work was done by DGC, and the only thing remaining is to pump the the parameters in and out of the registers. See the RDOS book for systems calls, and Algol Appendix B and C for some scanty tips on parameter and literal use, and perhaps half an example. In general, it is best to write as much as possible in Algol, and then patch the coding in the SR file; experience will later help in reducing the ASM coding.

/ See p. 9 for the coding of TME */*

AN ALGOL PROCEDURE LIST

A. VAN ROGGEN

Here is the cate-log of her conditions.
Imprimis, she can FETCH and CARRY ...
Shakespeare.

The new Algol manual has a large number of corrections and improvements over the old one, but it lacks one vital piece of information: a systematic listing of all available procedures. Perhaps this is typical for the new DGC manuals (the CLI book does not have an alphabetic listing or index of the CLI commands either), but at least the Algol runtime routines are listed in the index in one place. For procedures it is merely convenient to know where they are described in the book, but it is essential to know what type they have, what arguments are needed and in which order, etc. This information is mostly lacking even in the descriptions, and certainly is not readily available. I have tried to compile all those I could find in the book, and have attached what I think are the proper attributes.

Because I have not used all the procedures in a variety of conditions, I had to guess at some of the entries, especially where expressions can be used as formal parameter and where not. There may thus be errors in this list. Please report all those that you can find, so that later an updated list can be published. Perhaps the list can be made more useful by increasing the abbreviations, such as the use of "C" instead of "I" where a channel is indicated? Should the run time routines be included? Suggestions are welcome.

***** CAPTION TO ALGOL PROCEDURE LIST *****

PROCEDURE TYPES:

B	BOOLEAN
I	INTEGER
LB	LABEL
O	OPERATOR
P	POINTER
R	REAL
S	STRING
-	TYPELESS
*	USE EXTERNAL DECLARATION

FORMAL ARGUMENTS:

A	ARRAY
B	BOOLEAN
I	INTEGER
L	LABEL
LT	LITERAL
P	POINTER
R	REAL
S	STRING
N	I!R!LT!EXPRESSION
V	N!L!P!B!S
-X	ANY X
BX	BASED X
!	LOGICAL OR
X..X	LIST X,X,X,...

*** DATA GENERAL ALGOL PROCEDURES ***

REVISION OF 10:57 TUE 18 NOV 1975

PROCEDURE	TYPE	ARGUMENTS	COMMENTS
/*...*/	-	-	COMMENT STATEMENT
ABS	R	N	
ACCESS	-	I,S,P,I	CMM
ADDRESS	P	V	
ALLOCATE	-	P,I	
APPEND	-	I,S,L	
ARCTAN	R	N	
ASCII	I	V,I	
BUFFER	-	P,I	CMM
BYTE	I	V,I	=ASCII
BYTEREAD	-	I,P,I,L	
BYTEWRITE	-	I,P,I,L	
CHAIN	-	S	
CLASSIFY	I	I,P	
CLOSE	-	I	
COMARG	-	I,S,BA,L	FILE=COM.CM
COS	R	N	
DELETE	-	S	FILES
ENTIER	I	N	NOT MATH ENTIER
ERROR	-	S	
EXP	R	N	
FETCH	I	I,I	CMM
FILEPOSITION	-	I,I	
FILESIZE	-	I,I	
FIX	I	N	DFLT PREC
FLOAT	R	N	DFLT PREC
FLUSH	-	P	CMM
FREE	-	P	
GTIME	*-	I,I,I,I,I,I	
HASHREAD	-	I,I,P,I	CMM
HASHWRITE	-	P	CMM
HBOUND	I	-A,I	
INCLUDE	-	S	FILENAME
INDEX	I	S,S	
LBOUND	I	-A,I	
LENGTH	I	S	
LINEREAD	-	I,P,I,L	
LINEWRITE	-	I,P,I,L	
LN	R	N	
LONG	*-		FOR ERRORS
MEMORY	I		
MOD	*I	I,I	NOT MATH. MOD
NODEREAD	-	I,-A	CMM
NODESIZE	I	I	CMM
NODEWRITE	-	I,-A	CMM
OFFTRACE	*-		
ONTRACE	*-		
OPEN	-	I,S,L	
OUTPUT	-	I,S,V..V,L	
POSITION	-	I,I,L	
RANDOM	*I		
READ	-	I,V..V,L,L	
REM	*-	I,I,I,I	NOT MATH. REM
RENAME	-	S,S	FILES
ROTATE	I	I,I	
SEED	*-	I	FOR RANDOM
SETCURRENT	-	S,I	
SHIFT	I	I,I	
SHORT	*-		FOR ERRORS
SIGN	I	N	
SIN	R	N	
SIZE	I	S!A	
SQRT	R	N	
STASH	-	I,I,I	CMM
STIME	*-	I,I,I,I,I,I	
SUBSTR	S	S,I,I	
TAN	R	N	
TRACE	*-		
UMUL	-	I,I,I,I,I	
WORDREAD	-	I,I,P,I	CMM
WORDWRITE	-	I,I,P,I	CMM
WRITE	-	I,V..V,L	